НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

"На правах рукопису"
УДК 004.422.833

«До захисту допущено»
Завідувач кафедри
Коваль О.В. _____
(прізвище, ініціали) (підпис)
«_____» _____ 2018р.

Магістерська дисертація

зі спеціальності - 121 Інженерія програмного забезпечення
за спеціалізацією - Програмне забезпечення розподілених систем
на тему: «Застосування мікросервісної архітектури для побудови відмовостійкої хмарної системи»

Виконав: студент _6__курсу, групи_ТВ-71мп__
___Соломкін Максим Владиславович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник _к.т.н., доцент Смаковський Д.С._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент ст. викл. кафедри АЕС і ІТФ, к.т.н., доцент О.В. Баранюк _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2018

Національний технічний університет України
“ Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти другий, магістерський

зі спеціальності - 121 Інженерія програмного забезпечення
за спеціалізацією - Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
Коваль О.В. _____
(прізвище, ініціали) (підпис)
«____» _____ 2018р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Соломкіну Максиму Владиславовичу
(прізвище, ім’я, по батькові)

1. Тема дисертації Застосування мікросервісної архітектури для побудови відмовостійкої хмарної системи
Науковий керівник Смаковський Денис Сергійович, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 5 листопада 2018 року №4072
2. Строк подання студентом дисертації _____
3. Об’єкт дослідження розподілені системи та засоби автоматичного масштабування вузлів кластера
4. Предмет дослідження алгоритми для автоматичного масштабування у розподілених системах
5. Перелік питань, які потрібно розробити проаналізувати особливості проектування та розробки розподілених хмарних систем; проаналізувати використання мікросервісної архітектури та інструментів для автоматичного масштабування; розробити розподілену хмарну систему з використанням мікросервісних підходів; розробити інструменти для зчитування метрик сервісів та засоби автоматичного масштабування залежно від метрик сервісів

6. Перелік ілюстративного матеріалу мікросервісна система, засоби автоматичного масштабування, функції програмного забезпечення, структура програмного забезпечення, інтерфейс

7. Орієнтований перелік публікацій
1) Смаковський Д.С., Соломкін М.В. “Шаблони проектування для побудови мікросервісної архітектури”
2) Смаковський Д.С., Соломкін М.В. “Підходи до побудови відмовостійких розподілених систем”
3) Смаковський Д.С., Соломкін М.В. “Горизонтальне автоматичне масштабування сервісів на основі довжини черги у Kubernetes кластері”

8. Дата видачі завдання « 11 » вересня 2017 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз проблеми автоматизації рефакторингу програмного забезпечення	11.09.2017-12.11.2017	
2	Аналіз існуючих реалізацій та шляхів вирішення проблеми	13.11.2017-15.01.2018	
3	Аналіз існуючих механізмів генерації програмного коду за заданим шаблоном	16.01.2018-05.03.2018	
4	Аналіз метрик оцінки якості програмного коду	06.03.2018-27.06.2018	
5	Аналіз вимог до аналізатора програмного коду	28.06.2018-18.08.2018	
6	Моделювання схеми роботи майбутньої програми	19.08.2018-30.08.2018	
7	Розробка архітектури програмного забезпечення	01.09.2018-20.09.2018	
8	Розробка дизайну API	21.09.2018-10.10.2018	
9	Розробка програмного застосунку	11.10.2018-30.11.2018	
10	Оформлення документації	01.12.2018-07.12.2018	

Студент _____ Соломкін М.В.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Смаковський Д.С.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Структура й обсяг дипломної роботи.

Магістерська дисертація складається зі вступу, п'яти розділів, висновку, переліку посилань з 24 найменувань, 2 додатки, і містить 64 рисунки, 29 таблиць. Повний обсяг магістерської дисертації складає 115 сторінок, з яких перелік посилань займає 2 сторінок, додатки – 12 сторінок.

Актуальність теми. Хмарна інфраструктура дуже популярна і широко використовується багатьма компаніями тому що дозволяє економити як на обслуговуванні і персоналі, так і на інфраструктурі. Але, оскільки віртуальні машини у хмарній інфраструктурі не надійні, системи повинні бути спроектовані таким чином, щоб мінімізувати простої та перерви на обслуговування. Це зумовлює актуальність розробки розподілених хмарних систем, що максимально ефективно використовують хмарне середовище.

Мета дослідження полягає у визначенні способів підвищення надійності та стійкості розподілених хмарних систем на прикладі інтерактивної веб-системи для розширення словникового запасу іноземної мови.

Для досягнення поставленої задачі були сформульовані наступні **завдання дослідження**, що визначили логіку дослідження та його структуру:

— проаналізувати існуючі підходи до проектування розподілених хмарних систем;

— проаналізувати існуючі засоби для розгортки та забезпечення відмовостійкості хмарних систем;

— створити систему для вивчення нових слів, що спроектована для роботи у хмарній інфраструктурі та використовує переваги і мінімізує недоліки хмарного середовища;

— забезпечити засоби для автоматичного масштабування критичними для системи метриками.

Об’єктом дослідження є розподілені системи та засоби автоматичного масштабування вузлів кластера.

Предметом дослідження є алгоритми для автоматичного масштабування у розподілених системах.

Методи дослідження. Розв’язання поставлених задач виконувались засобами комп’ютерного моделювання, зокрема з використанням наступних методів:

— реалізація клієнта, що генерує навантаження на систему та імітує поведінку реального користувача;

— аналіз метрик кожного з сервісів для виявлення проблем

— аналіз запитів що виконуються довше за все за допомогою використання розподіленого трасування запитів.

Наукова новизна одержаних результатів. Найбільш суттєвими науковими результатами магістерської дисертації є удосконалення алгоритму автоматичного горизонтального масштабування у оркестраторі Kubernetes за рахунок застосування інформації про довжину черги, що дозволило збільшити ефективність застосування обчислювальних ресурсів у системі.

Практичне значення одержаних результатів роботи полягає в розробці розподіленої хмарної системи, що дозволяє вивчати слова іноземної мови, може бути розгорнута у Kubernetes кластері та динамічно адаптується під поточне навантаження системи за допомогою автоматичного масштабування екземплярів кожного з сервісів.

Ключові слова. ХМАРНА ІНФРАСТРУКТУРА, МІКРОСЕРВІСИ, REST, *KUBERNETES, МАСШТАБУВАННЯ, ВІДМОВОСТІЙКІСТЬ*.

ABSTRACT

The structure and volume of the thesis.

Master's thesis consists of an introduction, five chapters, conclusion, list of references with 24 titles, 2 annexes, and contains 64 figures, 29 tables. The full range of master's thesis is 115 pages with a list of links takes 2 pages, apps - 12 pages.

Topicality of the theme. The cloud infrastructure is very popular and is widely used by many companies, as it allows to save both on maintenance and staffing, and on infrastructure. But since virtual machines in a cloud infrastructure are not reliable, systems must be designed in such a way as to minimize downtime and service interruptions. This determines the urgency of the development of distributed cloud systems that efficiently use cloud environments.

The purpose and problems of research is to identify ways to increase the reliability and stability of distributed cloud systems, using the example of an interactive web system to expand the vocabulary of a foreign language.

To achieve this task the following objectives were formulated research study determined the logic and structure:

— analyze existing approaches to the design of distributed cloud systems;

— analyze the existing tools for scanning and providing fail-safe cloud systems;

— create a system for learning new words designed to work in cloud infrastructure and take advantage of and minimize the disadvantages of cloud environments;

— provide facilities for automatic scaling by metric critical for the system.

The object of the research is distributed systems and automated scaling of cluster nodes.

The subject of the research is the algorithms for automatic scaling in distributed systems.

Research methods. The solution of the set tasks was carried out by means of computer simulation, in particular using the following methods:

— the client that generates a load on the system and imitates the behavior of the real user;

— analysis of the metrics of each of the services to identify problems;

— analysis of requests that are performed the longest by using Distributed Query Tracing.

Scientific novelty of the results. The most significant scientific results of the master's thesis are the improvement of the horizontal automatic scaling algorithm Kubernetes orchestrator through the use of information about the length of the queue, which allowed to increase the efficiency of the use of computing resources in the system.

The practical significance of the results work is developed distributed cloud system that allows users to learn words of a foreign language, can be deployed in the Kubernetes cluster and dynamically adapts to the current system load by automatically scaling the instances of each service.

Keywords. CLOUD INFRASTRUCTURE, MICROSERVICES, REST, KUBERNETES, SCALING, FAULT-TOLERANCE.

ЗМІСТ

Перелік умовних позначень, символів, скорочень і термінів 11
Вступ 12
1 Задача побудови відмовостійкої хмарної системи 14
Висновки до розділу 1 15
2 Огляд існуючих програмних рішень для забезпечення відмовостійкості хмарних систем 16
2.1 Балансування навантаження 17
2.2 Масштабування і висока доступність 21
2.3 Масштабування баз даних 22
2.3.1 Партиціювання (partitioning) 22
2.3.2 Реплікація (replication) 23
2.3.3 Фрагментація (sharding) 24
2.4 Автоматичне відновлення даних 24
2.5 Шаблони розробки розподілених відмовостійких систем 25
2.5.1 Балансування на стороні клієнта (Client Side Load Balancing) 27
2.5.2 Реєстр сервісів (Service Discovery) 28
2.5.3 API Gateway 29
2.5.4 Circuit Breaker 30
2.5.5 Комбінація шаблонів 31
2.6 Забезпечення відмовостійкості 31
2.6.1 Apache Mesos 32
2.6.2 Kubernetes 33
2.6.3 Docker Swarm 35

2.6.4 Порівнення оркестраторів 35
Висновки до розділу 2 37
3 Методи реалізації системи 39
3.1 Середовище розробки IntelliJ IDEA 39
3.2 Інструменти для збірки проекту 40
3.3 Засоби розгортки програмного продукту 42
3.4 Технології для розробки інтерфейсу 43
3.4.1 Фреймворк Angular 6 44
3.4.2 Мова TypeScript 44
3.4.3 Структура інтерфейсу 45
3.5 Технології для розробки серверної частини 46
3.6 Взаємодія між клієнтом і сервером 47
Висновки до розділу 3 50
4 Опис програмної реалізації 51
4.1 Реалізація мікросервісних підходів 51
4.1.1 Реєстр сервісів 51
4.1.2 Сервер конфігурації 52
4.1.3 Взаємодія сервісів одне з одним 53
4.2 Точка доступу для повернення метрик сервісу 54
4.3 Робота з базою даних 55
4.3.1 Опис таблиць бази даних 55
4.3.2 Об’єктно-реляційне відображення даних у системі 58
4.4 Розгортання системи 61
4.5 Автоматичне масштабування за довільними метриками 63
4.6 Автентифікація користувача 69

4.7 Робота з зображеннями 71
4.8 Генерація вимови слів 72
4.9 Переклад слів 73
4.10 Тестування серверної частини 74
4.11 Документація API 75
4.12 Генерація шаблонного коду з Lombok 78
4.13 Веб клієнт 80
4.14 Взаємодія компонентів системи 83
4.15 Тренування вимови 84
4.16 Відстеження прогресу вивчення слів 84
Висновки до розділу 4 85
5 Методика роботи користувача 86
5.1 Системні вимоги 86
5.2 Взаємодія користувача с сервісом 86
Висновки до розділу 5 94
6 Стартап проект 95
6.1 Опис ідеї проекту 95
6.2 Технологічний аудит ідеї проекту 97
6.3 Аналіз ринкових можливостей запуску стартап-проекту 98
6.4 Розроблення ринкової стратегії проекту 106
6.5 Аналіз ринкових можливостей запуску стартап-проекту 108
Висновки до розділу 6 111
Висновки 112
Список використаних джерел 113
Додаток А 115
Додаток Б 126

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД	База даних
SQL	Structured query language — мова структурованих запитів
СУБД	Система керування базами даних
MVC	Model-View-Controller — Модель–вигляд–контролер
HTML	HyperText Markup Language — Мова розмітки гіпертекстових документів
IDE	Integrated development environment — Інтегроване середовище розробки
UI	User Interface — Інтерфейс користувача
JSON	JavaScript Object Notation
ORM	Object-relational mapping — Об’єктно-реляційна проекція
CRUD	Create read update delete (створення, зчитування, зміна і видалення)

ВСТУП

Ідея хмарних обчислень з'явилася ще в 1961 році, коли Джон Маккарті висловив припущення, що коли-небудь комп'ютерні обчислення будуть проводитися за допомогою "загальнонародних утиліт". Вважається, що ідеологія хмарних обчислень отримала популярність з 2007 року завдяки швидкому розвитку каналів зв'язку і стрімко зростаючим потребам користувачів.

Під хмарними обчисленнями зазвичай розуміється надання користувачу комп'ютерних ресурсів і потужностей у вигляді інтернет-сервісу. Таким чином, обчислювальні ресурси надаються користувачеві в "чистому" вигляді, і користувач може не знати які комп'ютери обробляють його запити або під керуванням якої операційної системи це відбувається.

Причини зростаючої популярності хмарних технологій зрозумілі: можливості їх застосування дуже різноманітні і дозволяють економити як на обслуговуванні і персоналі, так і на інфраструктурі. Апаратне забезпечення може бути значно спрощено при обробці даних і зберіганні інформації у віддалених центрах даних. Всі ці проблеми майже повністю перекладаються на провайдера послуг.

З підвищеним попитом на надійну та ефективну інфраструктуру, призначену для обслуговування критичних систем, терміни масштабованості та високої доступності не можуть бути більш актуальними. Хоча вирішення проблеми збільшення завантаження системи є головною задачею, зменшення простоїв і усунення окремих точок відмови є настільки ж важливими.

При створенні надійних інформаційних систем найчастіше пріоритетним завданням є мінімізація простоїв та перерв обслуговування. Незалежно від того, наскільки надійними є системи та програмне забезпечення, виникають проблеми, які можуть призвести до відмов програм або серверів.

Це обумовило створення відмовостійкої хмарної інтерактивної системи для вивчення слів іноземної мови, яка забезпечує надійну роботу у хмарному

розподіленому середовищі та автоматичне масштабування сервісів системи залежно від навантаження або довжини черги.

До основних функцій системи належать:

— балансування навантаження між всіма екземплярами сервісів;

— єдине місце для збереження конфігурацій сервісів;

— автоматичне масштабування залежно від рівня навантаження процесору або оперативної пам’яті;

— автоматичне масштабування залежно від довжини черги обробки задач;

— реєстрація та вхід користувачів;

— створення нових наборів слів для вивчення;

— додавання нових слів;

— автоматичне пропонування перекладу для нового слова на основі вже доданих слів та сервісу перекладу;

— додавання зображень до слова, що дозволяє створювати візуальні асоціації;

— пошук зображень для слова та можливість завантажити власне зображення;

— генерація вимови для слів;

— вивчення слів через написання;

— режим тренування вимови;

— режим вивчення слів у вигляді карток;

— відстеження прогресу навчання.

Записка містить 5 розділів. В першому розділі детально описується постановка задачі реалізації. У другому наводиться порівняльний аналіз існуючих інструментів та підходів до побудови розподілених відмовостійких систем та виявляється список необхідних функцій розроблюваної системи. У третьому розділі обґрунтовуються засоби, що були застосовані для розв'язання кожної підзадачі. Четвертий розділ є документальним супроводом програмного забезпечення, де описуються деталі реалізації. П’ятий розділ містить системні вимоги та приклади сценаріїв взаємодії користувача з системою.

1 ЗАДАЧА ПОБУДОВИ ВІДМОВОСТІЙКОЇ ХМАРНОЇ СИСТЕМИ

Метою дипломної роботи є розробка відмовостійкої хмарної системи, забезпечення автоматичного масштабування сервісів системи залежно від навантаження та аналіз інструментів для забезпечення відмовостійкості. Система є веб-сервісом, що призначений для розширення словникового запасу іноземної мови, тобто вивчення нових слів та закріплення вже вивчених.

При розробці та побудові сервісів, що мають велике навантаження, мають справу з двома значними проблемами: масштабованістю та надійністю. Більшість систем мають нерівномірне навантаження впродовж певних проміжків часу (рисунок 1.1). Система повинна розроблятися таким чином, щоб навіть під час тимчасових піків навантаження продовжувати працювати надійно.

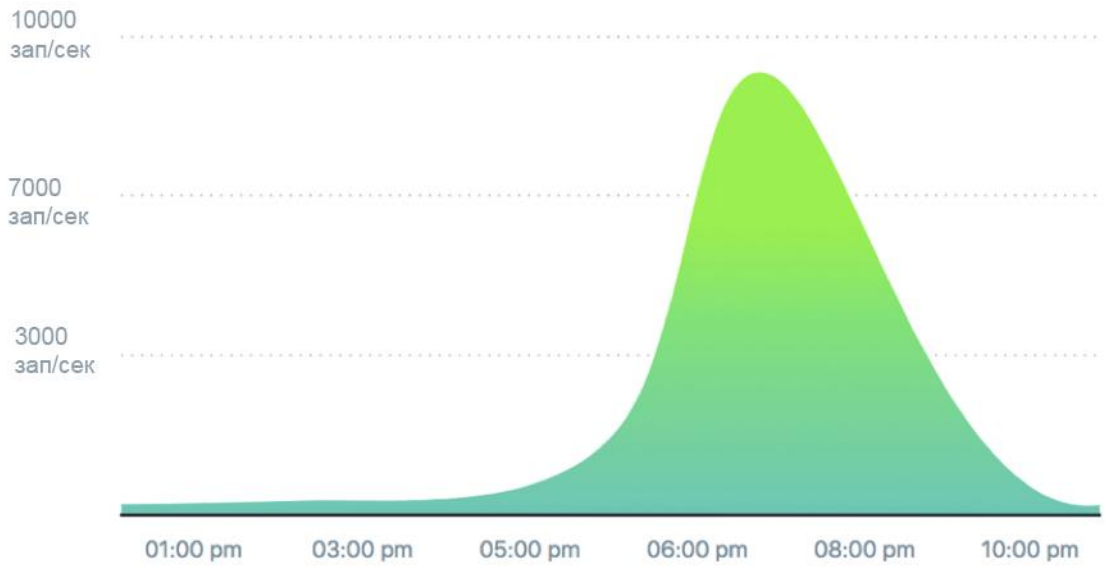

Рисунок 1.1 — Розподіл кількості запитів залежно від часу доби

Програмний продукт повинен забезпечити наступні можливості:

— єдиний шлюз для виконання всіх запитів;

— реєстрація екземплярів сервісів у загальному реєстрі;

— аналіз розподілених запитів та часу їх виконання;

— автоматичне масштабування сервісів за рівнем використання CPU або оперативної пам’яті;

— автоматичне масштабування сервісів залежно від довжини черги запитів;

— розгортка системи у хмарному середовищі;

— балансування навантаження на стороні кожного клієнта;

— повтор запитів у випадку помилки;

— дострокове переривання запитів у випадку відмови сервісу або довгого часу відповіді;

— точки для отримання інформації про стан кожного сервісу;

— точки для отримання інформації про поточні метрики кожного сервісу;

— інтеграція зі сторонніми хмарними сервісами;

— перегляд зареєстрованих екземплярів кожного сервісу;

— контроль стану кожного екземпляру сервісу та перезапуск у випадку невдачі перевірки життєвого стану.

Висновки до розділу 1

Задача побудови відмовостійкої хмарної системи полягає в тому, що система повинна надійно працювати в умовах хмарної інфраструктури та гарантувати толерантність до помилок різного рівня. Система повинна бути розроблена з можливістю масштабування як за метриками ресурсів, так і за метриками користувача.

2 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ДЛЯ ЗАБЕЗПЕЧЕННЯ ВІДМОВОСТІЙКОСТІ ХМАРНИХ СИСТЕМ

При створенні надійних виробничих систем найчастіше пріоритетним завданням є мінімізація простоїв та перерв обслуговування. Незалежно від того, наскільки надійними є системи та програмне забезпечення, виникають проблеми, які можуть призвести відмов програм або серверів.

Реалізація високої доступності інфраструктури є важливою стратегією зменшення впливу різноманітних подій на поведінку системи. Високо доступні системи можуть автоматично відновити роботу з сервером чи компонентом.

Однією з цілей високої доступності є усунення єдиних точок відмови у інфраструктурі. Єдина точка збою — це компонент інфраструктури, який може призвести до переривання роботи всієї системи, якщо він стане недоступним. Таким чином, будь-який компонент, який є необхідним для належної функціональності системи, який не має надмірності, вважається єдиною точкою збою.

Існує кілька компонентів, які необхідно ретельно враховувати для забезпечення високої доступності на практиці. Набагато більше, ніж застосування програмного забезпечення, висока доступність залежить від таких факторів, як:

— Середовище: якщо всі сервери розташовані в тій же географічній зоні, стан навколишнього середовища, такий як землетрус або затоплення, може призвести до знищення всієї системи. Надлишкові сервери в різних центрах обробки даних та географічних регіонах збільшать надійність.

— Обладнання: високодоступні сервери повинні бути стійкими до відключень живлення та апаратних збоїв, включаючи жорсткі диски та мережні інтерфейси.

— Програмне забезпечення: весь стек програмного забезпечення, включаючи операційну систему та саму програму, повинен бути підготовлений для обробки непередбачених збоїв, які, наприклад, можуть вимагати перезавантаження системи.

— Дані: втрата даних та непослідовність можуть бути викликані кількома факторами, і це не обмежується збоями жорсткого диска. Високо доступні системи повинні забезпечувати безпеку даних у випадку невдачі.

— Мережа: незаплановані перебої в мережі представляють ще одну можливу точку відмови для високодоступних систем. Важливо, що існує резервна мережева стратегія для можливих невдач.

Роздивимось підходи до побудови стійких систем на прикладному рівні хмарних систем.

2.1 Балансування навантаження

Балансування навантаження є ключовим компонентом стійких інфраструктур, який часто використовуються для підвищення продуктивності та надійності веб-сайтів, програм, баз даних та інших служб шляхом розподілу навантаження на декількох серверах.

Веб-інфраструктура без балансування навантаження (рисунок 2.1) навантаження може виглядати приблизно так:

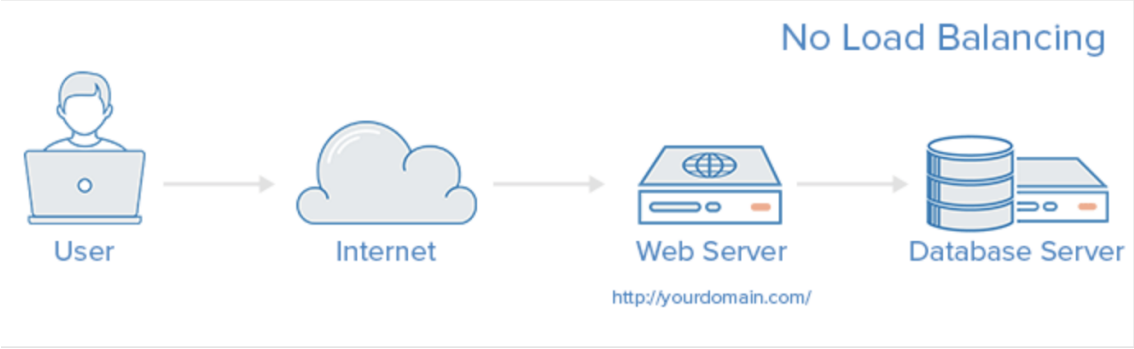

Рисунок 2.1 — Інфраструктура без балансування навантаження

У цьому прикладі користувач підключається безпосередньо до веб-сервера на сайті yourdomain.com. Якщо цей єдиний веб-сервер падає, користувач більше не

зможе отримати доступ до веб-сайту. Крім того, якщо багато користувачів намагаються одночасно отримати доступ до сервера і він не в змозі впоратися з завантаженням, то користувачі можуть довго чекати відповіді або взагалі не зможуть з'єднатися.

Цю єдину точку збою можна пом'якшити шляхом введення балансування навантаження та, принаймні, одного додаткового веб-сервера (рисунок 2.2). Як правило, всі сервери надаватимуть однакові відповіді, щоб користувачі отримували правильну відповідь незалежно від того, який сервер відповідає.

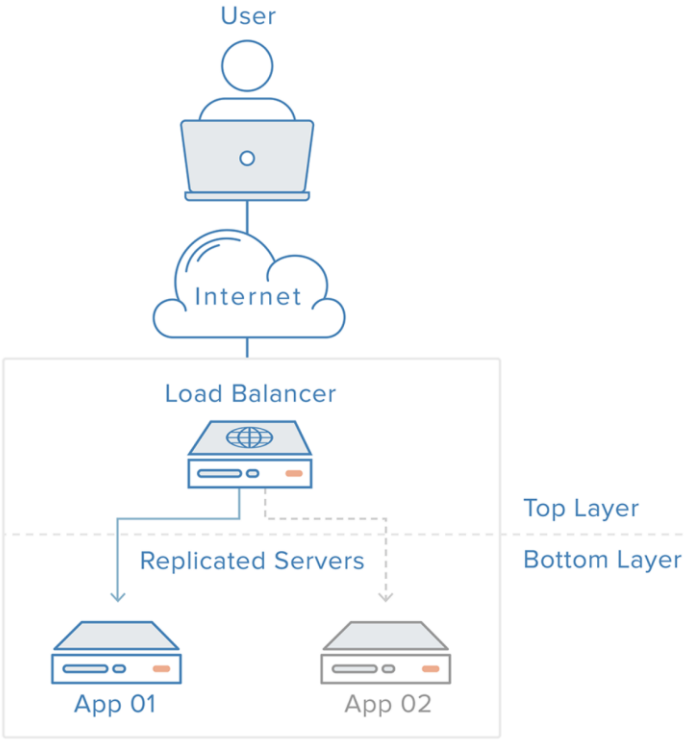

Рисунок 2.2 — Інфраструктура з балансуванням навантаження

У наведеному вище прикладі користувач доступається до балансувальника навантаження, який пересилає запит користувача на один з серверів, який потім безпосередньо відповідає на запит користувача.

Для усунення єдиних точок відмови, кожен рівень системи повинен бути готовим до надмірності. Наприклад, візьмемо інфраструктуру, що складається з двох ідентичних, надлишкових веб-серверів за балансуванням навантаження. Трафік, що надходить від клієнтів, буде рівномірно розподілений між веб-серверами, однак,

якщо один з серверів відмовить, балансування навантаження переспрямовуватиме весь трафік на решту доступних серверів.

У цьому випадку рівень веб-серверів не є однією точкою збою, оскільки:

— маються надлишкові компоненти для одного й того ж сервісу;

— механізм, що знаходиться перед рівнем веб-сервісів (балансування навантаження) здатний виявити збої в компонентах і адаптувати його поведінку для своєчасного відновлення.

За описаним сценарієм, що не є рідкістю в реальному житті, рівень балансування навантаження сам по собі залишається єдиною точкою відмови. Однак усунення єдиної точки відмови, що залишилася, може бути складним завданням; навіть якщо можна легко налаштувати додатковий балансувальник навантаження, щоб досягти надмірності, не існує очевидної точки над балансувальниками навантаження для виявлення та відновлення збою.

Тільки надлишковість не може гарантувати високу доступність. Потрібно встановити механізм для виявлення збоїв та виконання дій, коли один з компонентів інфраструктури стає недоступним.

Виявлення та відновлення збоїв для надлишкових систем можна реалізувати, використовуючи підхід “зверху донизу”: верхній шар стає відповідальним за відстеження та контроль збоїв шару, що знаходиться під ним. У попередньому прикладі сценарій балансування навантаження є верхнім шаром. Якщо один з веб-серверів (нижній рівень) стає недоступним, балансування навантаження зупинить переадресацію запитів для цього конкретного сервера.

Цей підхід має обмеження: в інфраструктурі буде місце, де верхній шар або неіснуючий, або недоступний, як у випадку з балансуванням навантаження. Створення сервісу виявлення несправностей для балансування навантаження на зовнішньому сервері просто створить нову точку відмови.

З таким сценарієм необхідний розподілений підхід. Кілька резервних вузлів повинні бути з’єднані разом як кластер, де кожен вузол повинен бути здатним однаково виявляти та відновлювати несправності (рисунок 2.3).

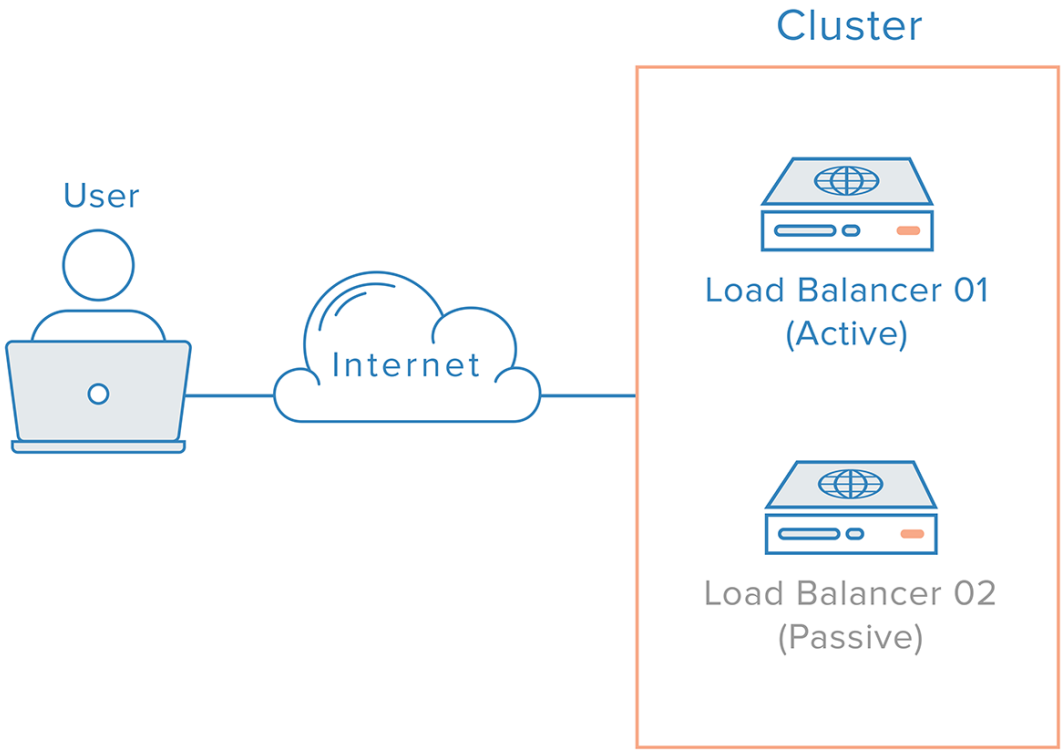

Рисунок 2.3 — Кластер балансувальників навантаження

Проте, для випадку з балансуванням навантаження, існує додаткове ускладнення через особливості роботи DNS серверів. Відновлення після несправності балансувальника навантаження зазвичай означає перехід до резервного балансувальника навантаження, що означає, що необхідно змінити DNS, щоб вказати ім'я домену на IP-адресу резервного балансувальника навантаження. Зміни, подібні до цього, можуть зайняти значну кількість часу для поширення в Інтернеті, що може призвести до серйозного простою цієї системи.

Можливим рішенням є використання балансування навантаження DNS за стратегією round-robin (по-черзі). Проте такий підхід не є надійним, оскільки він призводить до балансування на стороні клієнта.

Більш надійним та надійним рішенням є використання систем, що дозволяють гнучко переміщувати IP-адресу (рисунок 2.4). Ця функція називається Floating IP. Такі можливості надає Amazon Web Services (AWS), Digital Ocean та багато інших провайдерів. Зміна IP-адресу за потребою усуває проблеми, пов'язані з розповсюдженням та кешуванням, які властиві змінам DNS, шляхом надання

статичної IP-адреси, яку можна легко відновити, коли це буде потрібно. Ім'я домену може залишатися пов'язаним з тією ж IP-адресою, тоді як сама IP-адреса переміщується між серверами.

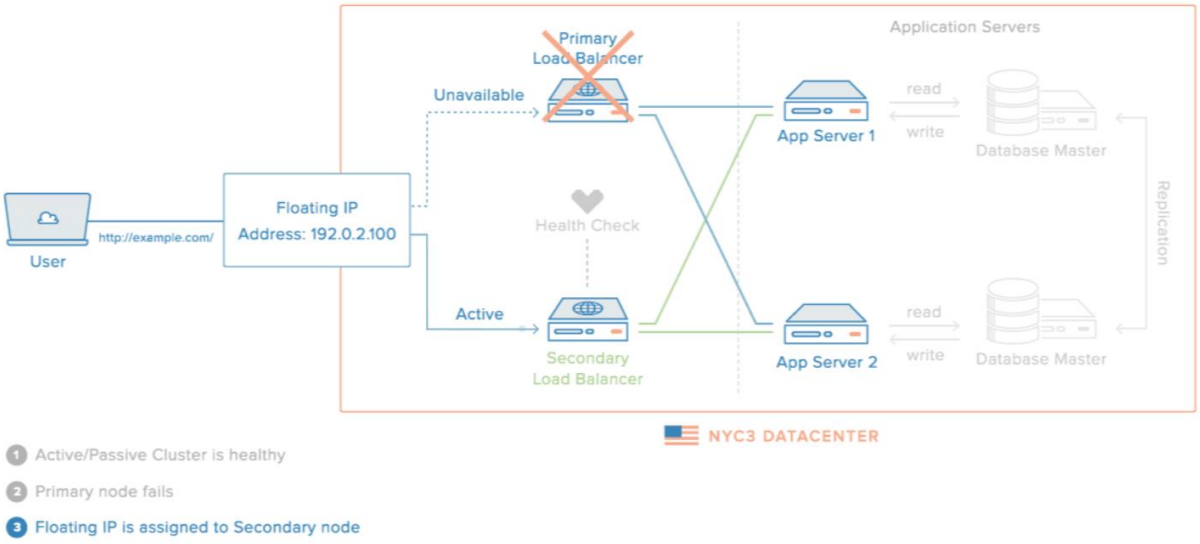

Рисунок 2.4 — Балансування за допомогою Floating IP

Кожен шар високо доступної системи буде мати різні потреби від програмного забезпечення та конфігурації. Однак на рівні програмних додатків балансувальники навантаження є важливим елементом програмного забезпечення для створення будь-якої високої доступності.

2.2 Масштабування і висока доступність

Масштабованість можна розглядати з кількох точок зору, але нас в основному цікавить виділення хмарних ресурсів за потребою. Можливість нарощувати ресурси (як вгору, так і вниз) означає поліпшену економічну ефективність для користувача і підвищену складність для розробника хмарних систем.

Масштабованість повинна забезпечуватися не тільки для самої системи (функціональне масштабування), але і для її пропускної здатності (масштабування навантаження). Ще одна ключова особливість хмарних систем — географічний розподіл даних (географічна масштабованість), яка дозволяє розташовувати дані та

ресурси в максимальній близькості до користувача завдяки групі центрів хмарного зберігання даних (шляхом міграції). Хмарна інфраструктура повинна забезпечувати і внутрішнє масштабування. Сервери та система зберігання повинні допускати зміни розміру без всяких наслідків для користувачів.

Система завжди повинна мати можливість відповісти на запит користувача. З урахуванням простоїв мережі, помилок користувачів та інших обставин виконання цієї умови надійним і детермінованим способом може виявитися дуже складною задачею.

2.3 Масштабування баз даних

Класична схема роботи програми з базою даних має один екземпляр сервісу і один екземпляр бази даних. Така схема роботи з БД не дозволяє масштабуватися та робить БД єдиною точкою відмови системи. В основі масштабування даних лежить той самий принцип, що і в основі масштабування Web додатків. Це поділ даних на групи і виділення їх на окремі сервера. Існує дві основні стратегії: реплікація і шардінг.

Описані нижче схеми масштабування застосовні як для реляційних баз даних, так і для NOSQL-сховищ. Зрозуміло, що у всіх баз даних і сховищ є своя специфіка, тому розглянемо тільки основні напрямки.

2.3.1 Партиціювання (partitioning)

Партиціювання — це розбиття таблиць, що містять велику кількість записів, на логічні частини за деякими обраним адміністратором критеріям. Партиціювання таблиць ділить весь обсяг операцій з обробки даних на кілька незалежних і паралельно виконуються потоків, що істотно прискорює роботу СУБД. Для правильного конфігурації параметрів партиціювання необхідно, щоб в кожному потоці було приблизно однакова кількість записів.

Наприклад, на новинних сайтах має сенс розбивати записи за датою публікації, так як свіжі новини на кілька порядків більш затребувані і частіше потрібна робота саме з ними, а не з усім архівом за роки існування новинного ресурсу.

2.3.2 Реплікація (replication)

Реплікація — це синхронне або асинхронне копіювання даних між декількома серверами (рисунок 2.5).

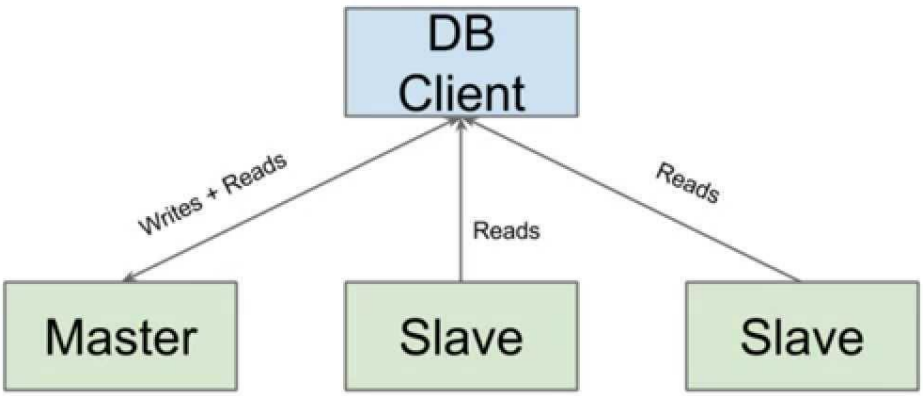

Рисунок 2.5 — Реплікація бази даних

Керуючі сервера називають майстрами (master), а ведені сервера — слейвами (slave). Майстри використовуються для зміни даних, а слейви — для зчитування. У класичній схемі реплікації зазвичай один майстер і кілька слейвів, так як в більшій частині веб-проектів операцій читання на кілька порядків більше, ніж операцій запису. Однак в більш складній схемі реплікації може бути і кілька майстрів.

Наприклад, створення декількох додаткових slave-серверів дозволяє зняти з основного сервера навантаження і підвищити загальну продуктивність системи, а також можна організувати слейви під конкретні ресурсомісткі завдання і таким чином, наприклад, спростити складання серйозних аналітичних звітів — використовуваний для цих цілей slave може бути навантажений на 100%, але на роботу інших користувачів додатку це не вплине.

2.3.3 Фрагментація (sharding)

Фрагментація — це прийом, який дозволяє розподіляти дані між різними фізичними серверами. Процес фрагментації передбачає рознесення даних між окремими фрагментами на основі якогось ключа фрагментації (рисунок 2.6). Об'єкти пов'язані однаковим значенням ключа фрагмента групуються в набір даних по заданому ключу, а цей набір зберігається в межах одного фізичного фрагмента. Це істотно полегшує обробку даних.

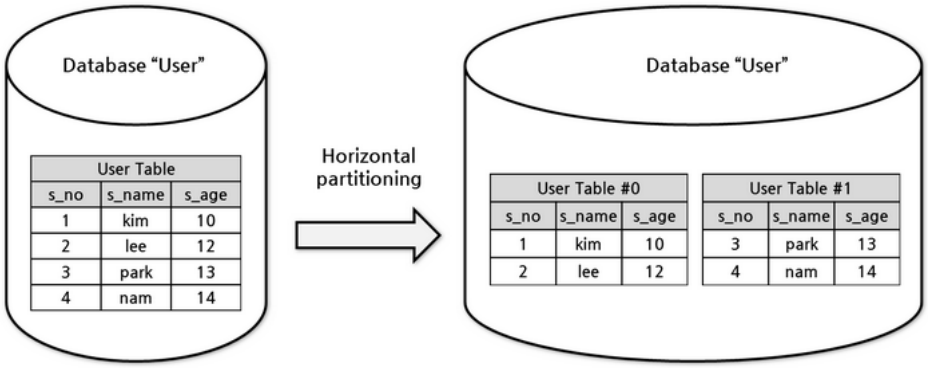

Рисунок 2.6 — Фрагментація бази даних

Наприклад, в системах типу соціальних мереж ключем для фрагментації може бути ID користувача, таким чином всі дані користувача будуть зберігатися і оброблятися на одному сервері, а не збиратися по частинах з декількох.

2.4 Автоматичне відновлення даних

Комбінація описаних стратегій масштабування дозволяє підвищити надійність та відмовостійкість систем. Після комбінації описаних стратегій всі дані розбиті на невеликі продубльовані частини та перенесені та різні сервера (рисунок 2.7).

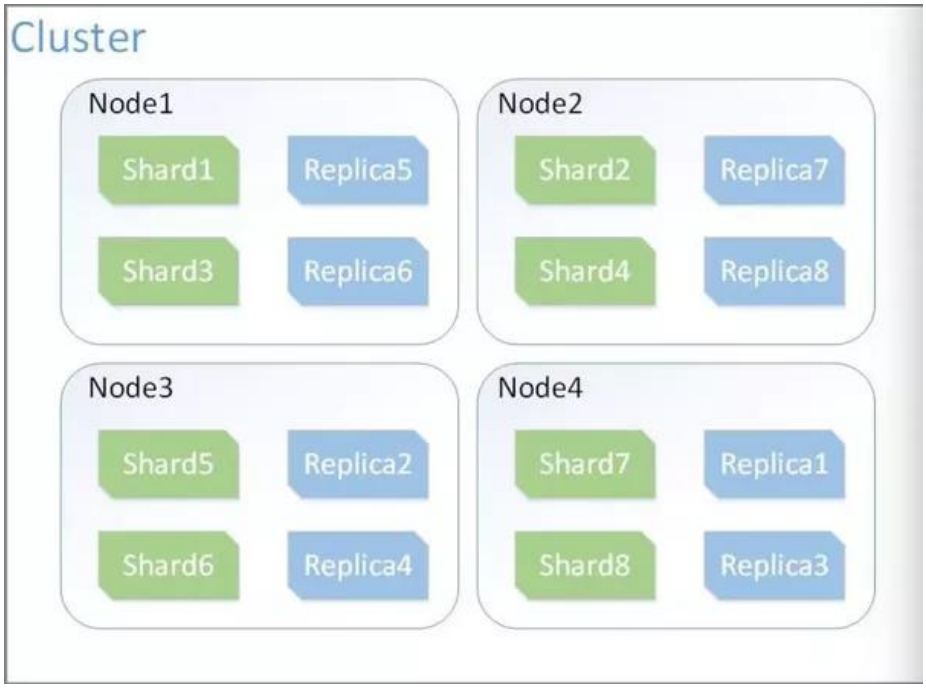

Рисунок 2.7 — Розміщення фрагментів по серверам

У випадку відмови одного з серверів можна ввести в дію новий сервер бази даних та перенести загублені фрагменті даних на новий сервіс БД. Більшість сучасних баз даних підтримують автоматичне фрагментування, реплікацію та відновлення стану після відмов.

2.5 Шаблони розробки розподілених відмовостійких систем

Для масштабування та надійності систем треба розбивати всі сервіси на маленькі частини та дублювати функціональність (рисунок 2.8). Такий тип архітектури називається мікросервісним. Ця архітектура полягає в тому, що система будується з декількох маленьких сервісів, кожен з яких добре виконує свою частину роботи. Для комунікації один з одним можуть виконуватися як синхронні REST запити [19], так і асинхронна обробка та відправка повідомлень за допомогою черг.

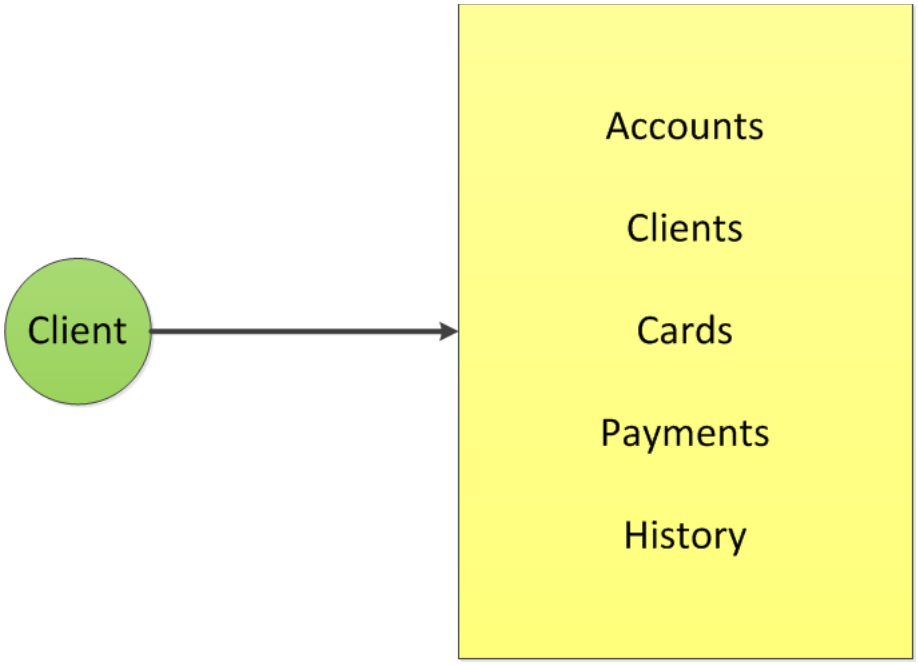

Рисунок 2.8 — Монолітна система

Мікросервісна система є розподіленою за своєю природою. Отже, система повинна бути розділена на логічні сервіси (рисунок 2.9). Мікросервісна система виділяє функціональні одиниці у окремі сервіси та масштабується через багатократний запуск необхідних функцій на декількох серверах

Розподілені сервіси забезпечують можливість масштабування самостійно. Кожен сервіс може бути розгорнутий окремо зі збільшенням кількості екземплярів тих сервісів, що потребують більшої обчислювальної потужності (наприклад, для підтримки більшої кількості запитів). Це називається горизонтальним масштабуванням.

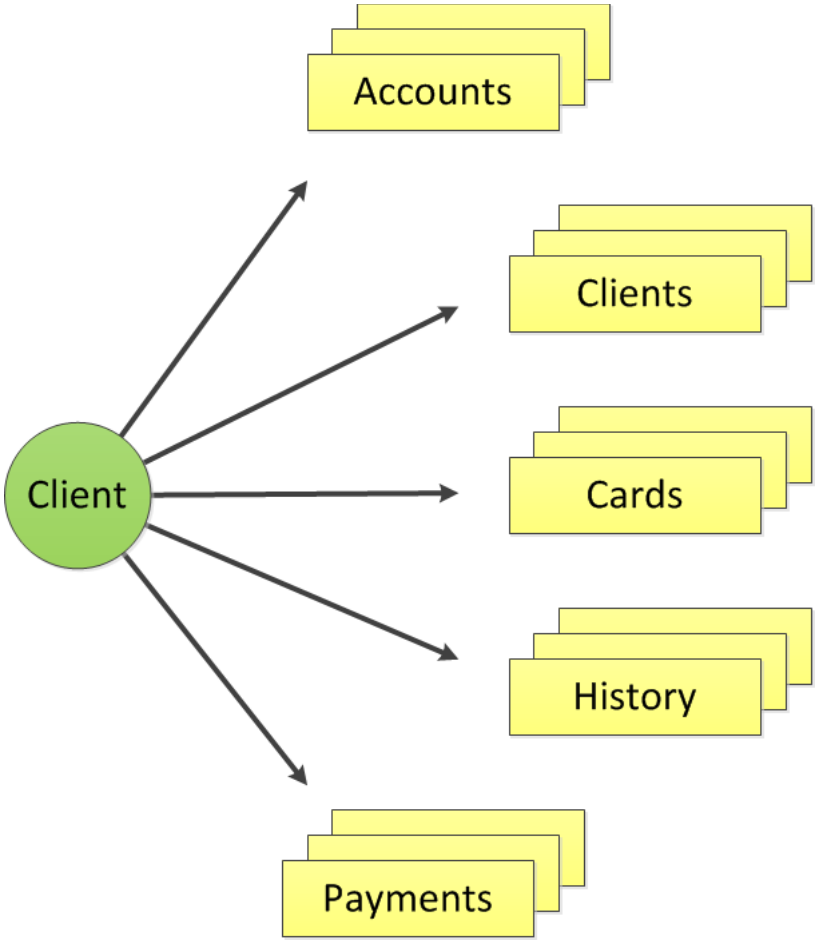

Рисунок 2.9 — Система, яка розподілена на сервіси

2.5.1 Балансування на стороні клієнта (Client Side Load Balancing)

Можемо застосувати шаблон балансування навантаження на стороні клієнта (client side load balancing). Це застосування вже знайомого балансувальника навантаження, але не на високому рівні для вхідних запитів, а на програмному рівні під час виконання запитів між власними сервісами. Коли клієнт хоче відправити запит до сервісу, він може вибрати будь-який екземпляр із списку доступних екземплярів сервісу. Це дозволяє розподілити навантаження зі сторони клієнтів по всім екземплярам сервісу рівномірно. Будь-який сервіс в мережі може бути клієнтом. Зазвичай клієнт робить запити та отримує відповіді, а сервіс може бути клієнтом для іншого сервісу і робити запити до нього, і це може статися каскадно (рисунок 2.10).

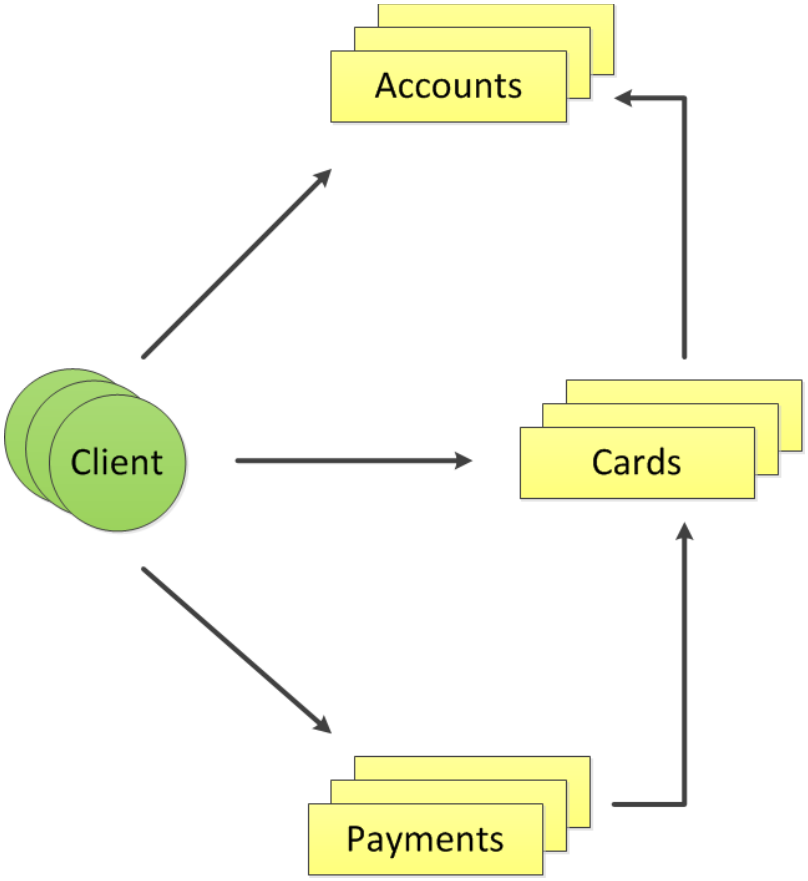

Рисунок 2.10 — Каскадні запити

2.5.2 Реєстр сервісів (Service Discovery)

Загалом, все, що обговорювалося вище, може бути найпростішим способом застосування мікросервісної архітектури. Однак у такому випадку існує проблема. Як клієнт отримає інформацію про наявні екземпляри інших сервісів? Для цього можуть бути використані файли конфігурації, але це додає ще більшу складність, оскільки потрібно оновлювати конфігурацію після кожної зміни топології. Використання шаблону пошуку сервісу (Service discovery) вирішує проблему. Цей шаблон представляє собою окремий сервіс, який відіграє роль адресної книги (реєстру) для інших сервісів (рисунок 2.11). Коли сервіс запускається, він повідомляє адресу (наприклад, хост і порт) до реєстру та надсилає повідомлення (наприклад, кожні 30 секунд), поки він є робочим. Отже, реєстр знає, які сервіси ще доступні. Більше того, цей шаблон дозволяє реалізувати гнучку масштабованість. Таким чином, можливо масштабувати сервіси під час роботи системи, динамічно додаючи або зменшуючи кількість екземплярів, коли це потрібно.

Коли клієнт хоче відправити запит до сервісу, єдине, що він повинен знати — це ім'я, яке використовується для запису сервісу у реєстрі. Таким чином, клієнт робить запит до реєстру, щоб отримати список доступних екземплярів сервісів, а потім робить прямий запит до обраного екземпляра сервісу. Екземпляр, зазвичай, обирається за допомогою логіки балансування навантаження.

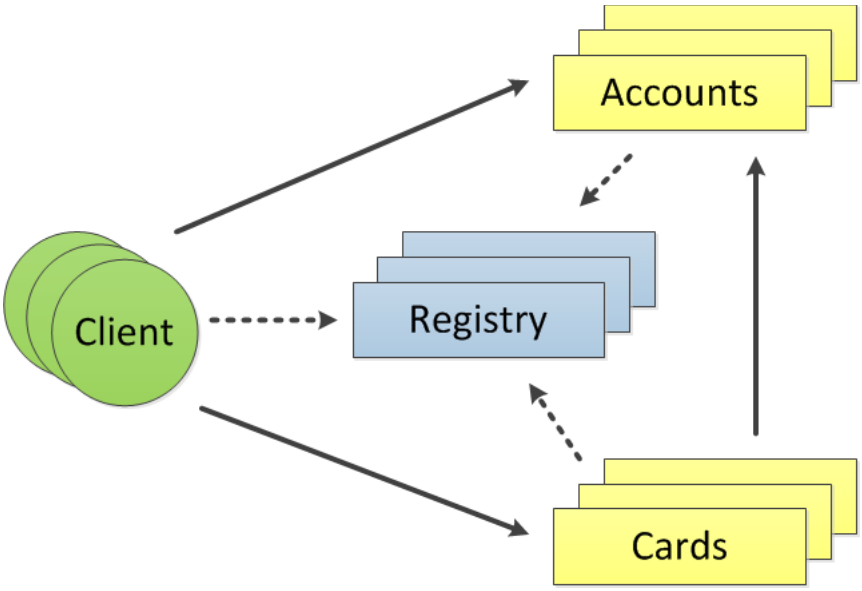

Рисунок 2.11— Робота з реєстром сервісів

2.5.3 API Gateway

Ви можете помітити, що ця архітектура вимагає від клієнтів знання про місцезнаходження реєстру та інтеграцію з ним. Це поширена практика для мікросервісних систем. Однак зовнішні клієнти не повинні нічого знати про внутрішню архітектуру системи. У мікросервісній архітектурі застосовується шаблон API шлюзу (API Gateway), який є єдиною точкою доступу до систему для зовнішніх систем (рисунок 2.12). Застосування шлюзу дозволяє приховати внутрішню складність системи для зовнішніх клієнтів.

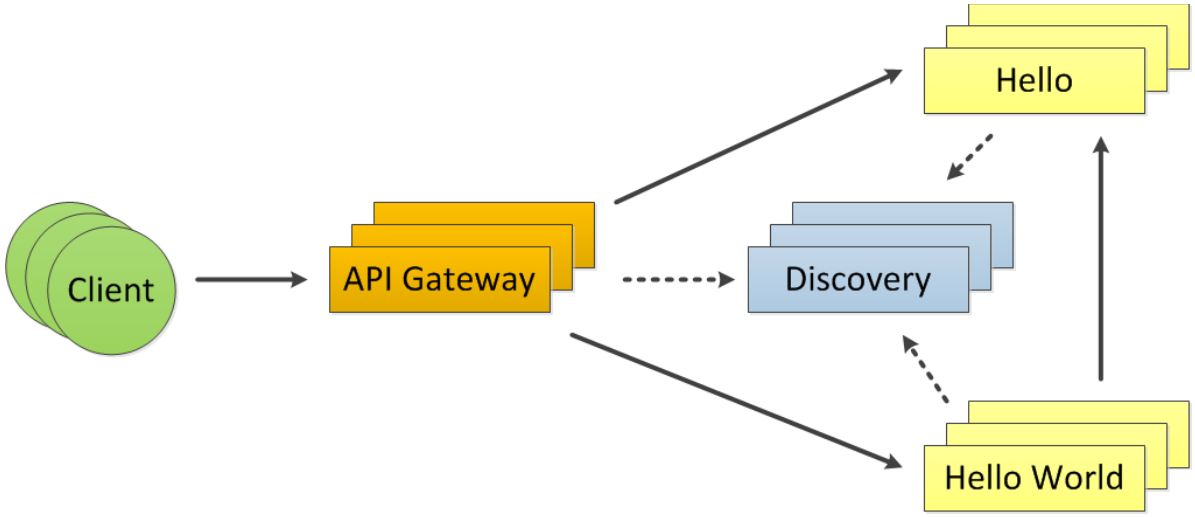

Рисунок 2.12 — API шлюз

API шлюзи надають функціональність за допомогою наявних мікросервісів. Шлюзи звертаються до реєстрів, щоб отримати список доступних сервісів та звертаються до них застосовуючи балансування навантаження. Існує два види шлюзів: проксі-сервер і логічний. Проксі-шлюзи просто передають виклик до потрібного сервісу, а логічні шлюзи можуть виконувати декілька запитів для сервісів, щоб надавати інший API своїм клієнтам.

2.5.4 Circuit Breaker

У випадку коли система складається з багатьох сервісів та один з кінцевих сервісів почав відмовляти, то можуть виникати каскадні переривання через тайм-аут (рисунок 2.13). Таким чином, клієнт може чекати декілька секунд, доки запит не перерветься через тайм-аути всіх каскадних запитів.

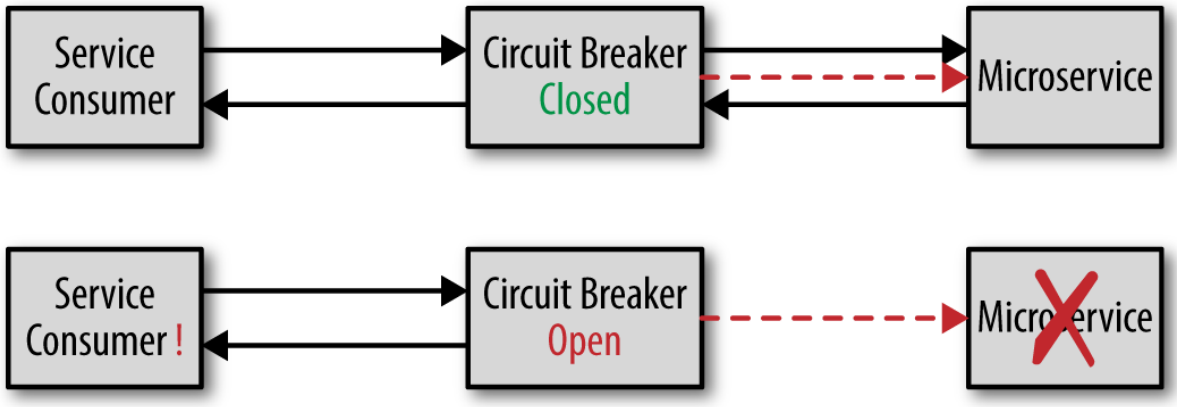

Рисунок 2.13— Шаблон проектування Circuit Breaker

Для прискорення реагування системи застосовується шаблон проектування Circuit Breaker, який навмисно завчасно перериває запити, що вірогідно перервуться через тайм-аут.

2.5.5 Комбінація шаблонів

Це основні шаблони проектування, без яких не існує жодної реальної мікросервісної системи. Комбінація цих шаблонів дозволяє будувати великі відмовостійкі системи (рисунок 2.14).

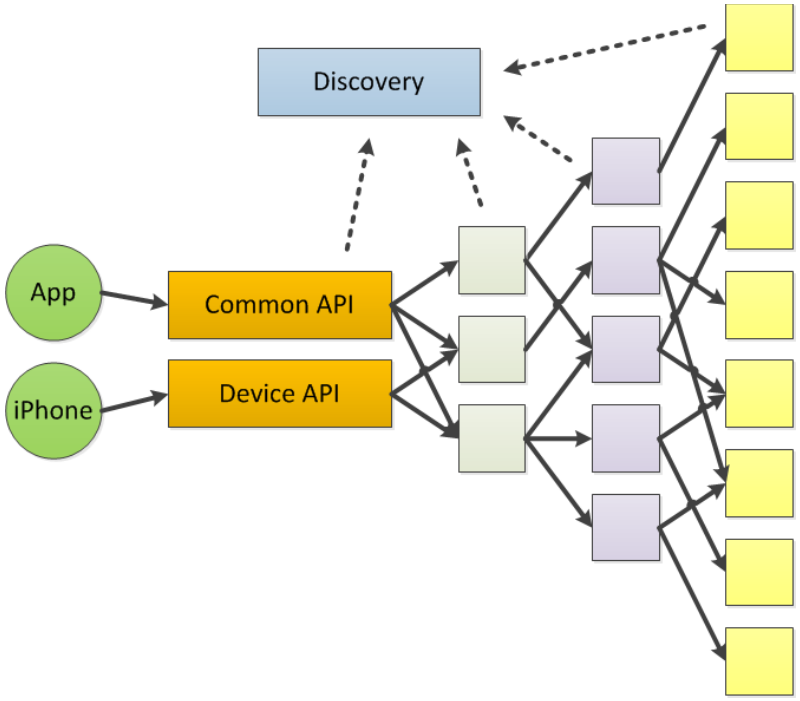

Рисунок 2.14 — Мікросервісна система

Але існує ще безліч різних шаблонів, які націлені на покращення відмовостійкості системи та полегшення її конфігурації.

2.6 Забезпечення відмовостійкості

Розглянемо як мікросервісна архітектура допомагає досягати живучості та відновлюваності всієї системи. У випадку відмови одного з екземплярів сервіса він перестає посилати запити у сервіс реєстру, котрий в свою чергу відмічає його неробочим та виводить зі списку сервісів. У цей час коли інші сервіси запитують

інформацію про цей сервіс, сервіс реєстру повертає інформацію тільки про робочі екземпляри сервісу. Таким чином при достатньому рівні надлишковості відмови деяких екземплярів сервісів ніяк не вплине та роботу системи.

Всі попередні засоби забезпечують живучість системи, але не мають прямого впливу на відновлюваність. Для відновлення інфраструктури після відмов застосовуються системи, що можуть піднімати нові екземпляри сервісів. До таких систем належать Apache Mesos, Kubernetes та інші. Зазвичай ці системи здатні відстежувати стан всіх екземплярів сервісів за швидкістю відповіді, рівнем CPU та пам'яті, та запускати нові екземпляри необхідних сервісів, або ж перезапускати ті екземпляри, які перестали віповідати на запити. Таким чином побудована інфраструктура може динамічно перелаштовуватися та адаптуватися під навантаження.

2.6.1 Apache Mesos

Оркестратор Apache Mesos — це менеджер кластеру, що забезпечує ефективну ізоляцію та поділ ресурсів між розподіленими застосунками. Mesos має відкритий код та спочатку розроблявся Каліфорнійським Університетом у Берклі. Він знаходиться на рівні між прикладним та рівнем операційної системи та дозволяє спростити розгортку та управління у широкомасштабному кластерному середовищу.

Mesos використовує функції сучасних ядер операційних систем для забезпечення ізоляції для центрального процесора, пам'яті, введення/виводу, файлової системи, місця розташування стійок тощо (рисунок 2.15). Велика ідея полягає у створенні великої колекції гетерогенних ресурсів. Mesos представляє розподілений дворівневий механізм планування, який називається ресурсними пропозиціями. Mesos вирішує, скільки ресурсів запропонувати кожному фреймворку, тоді як в фреймворк вирішує, які ресурси приймати і які обчислення на них виконувати. Це тонкий шар розподілу ресурсів, який забезпечує точний обмін ресурсами між різними кластерними обчислювальними системами, надаючи системам загальний інтерфейс для доступу до ресурсів кластера. Багато сучасних робочих застосунків і фреймворків можуть працювати на Mesos, включаючи Hadoop,

Memcached, Ruby on Rails, Storm, JBoss Data Grid, MPI, Spark та Node.js, а також різні веб-сервери, бази даних та сервери додатків.

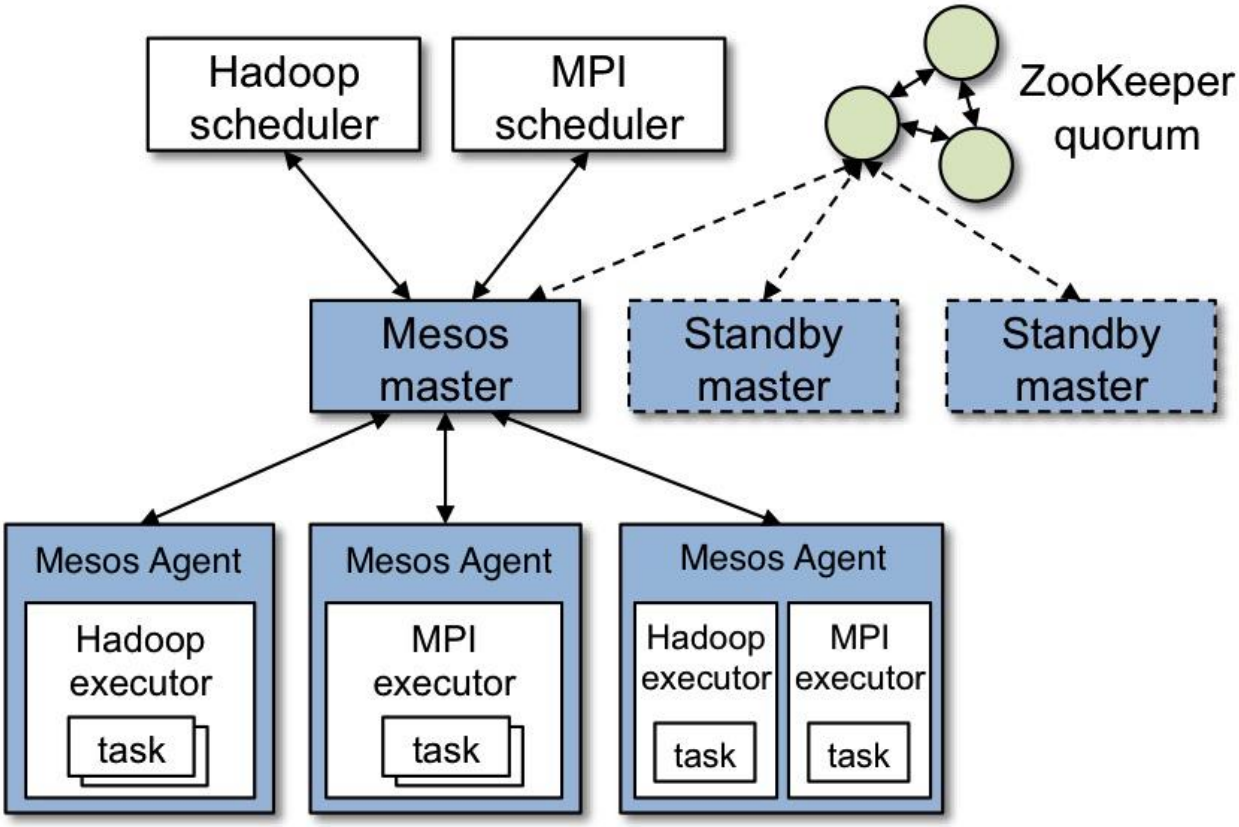

Рисунок 2.15 — Архітектура Apache Mesos

Для автоматичного масштабування Mesos може застосовувати показники використання CPU та оперативної пам'яті. Окрім того, Mesos дозволяє інтегрувати свої модулі для збору користувацьких метрик.

2.6.2 Kubernetes

Оркестратор Kubernetes — це система з відкритим вихідним кодом для автоматизації розгортання, масштабування та управління контейнеризованими додатками (рисунок 2.16). Він об'єднує контейнери, які складають додаток у логічні одиниці для легкого керування та відкриття. До основних функцій відносяться:

— розширення можливостей розробників додатків потужним інструментом для оркестування контейнерів Docker без взаємодії з базовою інфраструктурою;

— надавання стандартного інтерфейсу розгортання та примітиви для послідовного використання інтерфейсу розгортання додатків та API-інтерфейсів у хмарах;

— створення систем на основі модульного API, що дозволяє клієнтам інтегрувати системи навколо основної технології Kubernetes.

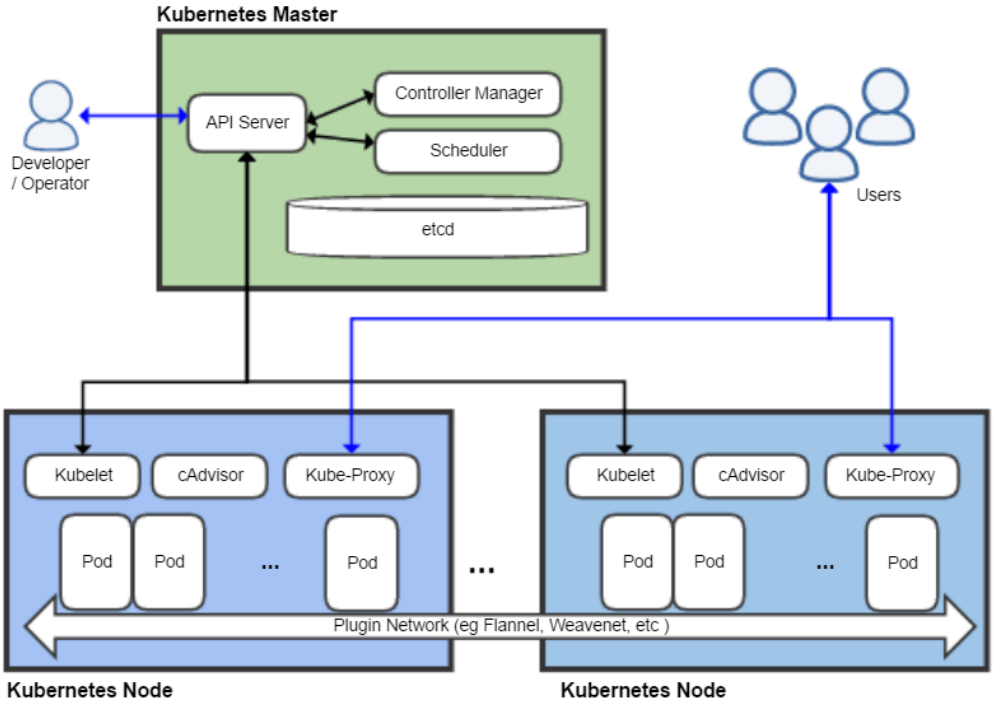

Рисунок 2.16 — Архітектура Kubernetes

Kubernetes є дуже привабливим для розробників застосунків, оскільки він зменшив їхню залежність від інфраструктурних та операційних груп. Системним адміністраторам також сподобався Kubernetes, оскільки він дає простий спосіб охоплювати рух контейнерів та забезпечити комерційне вирішення операційних завдань, пов’язаних із керуванням власним розгортанням Kubernetes, що залишається складною задачею.

Основні переваги Kubernetes — це надання розробникам додатків потужні інструменти для управління Docker контейнерами. Існує безліч ініціатив, спрямованих на розширення масштабів проекту до більшої кількості робочих

навантажень (наприклад, аналітика та державні служби передачі даних), ці ініціативи все ще перебувають на дуже ранній стадії.

2.6.3 Docker Swarm

Як платформа, Docker змінив спосіб, яким відбувалася доставка програмного забезпечення на сервера. Docker Swarm — це платформа для контейнерів із відкритим кодом та є власним оркестратором кластерів для Docker. Будь-яке програмне забезпечення, служби або інструменти, що працюють з контейнерами Docker, працюють в рівній мірі з Swarm. Крім того, Swarm використовує ту ж утиліту командного рядку, що й Docker.

Swarm перетворює набір Docker хостів у віртуальний, єдиний хост (рисунок 2.17). Swarm особливо корисний у випадку, коли простота розгортання дуже важлива.

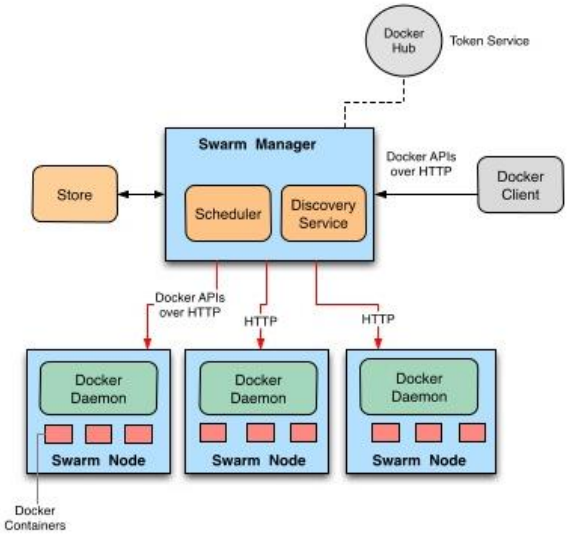

Рисунок 2.17 — Архітектура Docker Swarm

2.6.4 Порівнення оркестраторів

Порівняємо оркестратори за наступними критеріями:

— Декларація програми

— Мережа

— Масштабованість

— Балансування навантаження

— Висока доступність

Декларація програми

Apache Mesos: программа може бути розгорнута під керівництвом певного планувальника. Для простих задач застосовується планувальник Maraphon, але для більш складних задач потрібно під кожний сервіс створювати окремий планувальник.

Kubernetes: програма може бути розгорнута в Kubernetes за допомогою поєднання послуг (або мікросервісів), розгортання та компонентів.

Docker Swarm: Додатки можуть бути розгорнуті як мікро-послуги або послуги в групі кластерів в Docker Swarm. Файли YAML можуть бути використані для ідентифікації декількох контейнерів.

Мережа

Apache Mesos: не створює жодних віртуальних мереж та діє на рівні мережі між фізичними або віртуальними машинами, на яких встановлені агенти.

Kubernetes: мережева модель — це однорівнева мережа, що дозволяє всім подам взаємодіяти один з одним. Мережеві правила визначають, яким чином поди взаємодіють один з одним.

Docker Swarm: реалізує всередині віртуальну мережу, де кожен контейнер може бути під'єднаний до тієї чи іншої мережі, а самі мережі можуть бути з'єднані одне з одним. Користувачі мають можливість шифрувати трафік даних контейнера при створенні накладеної мережі самостійно.

Масштабованість

Apache Mesos: повідомляє про вільні ресурси планувальникам, які вже самі повинні реалізувати логіку масштабування.

Kubernetes: для розподілених систем, Kubernetes є системою загального призначення. Це складна система, оскільки вона забезпечує надійні гарантії щодо кластерного стану та уніфікованого набору API.

Docker Swarm: порівняно з Kubernetes, може розгорнути контейнер набагато швидше, і це дає змогу швидше реагувати на попит. Але у той самий час має менше механізмів для контролю процесу розгортки.

Висока доступність

Apache Mesos: враховуючи підтримку масштабувальника підтримує кластер у життєдіяльному стану, але полягається на планувальники.

Kubernetes: всі екземпляри сервісів розподілені між вузлами, і це забезпечує високу доступність та толерантність до помилок. Служба балансу навантаження виявляє пошкоджені екземпляри та перезапускає їх. Таким чином, це підтримує високу доступність.

Docker Swarm: оскільки послуги можуть бути дубльовані, Docker Swarm також пропонує високу доступність. Вузли менеджера Swarm несуть відповідальність за весь кластер і керують ресурсами вузлів робочого. Порівняно з Kubernetes, має меншу кількість інструментів для контролю життєдіяльності сервісів.

Балансування навантаження

Apache Mesos: не надає жодних можливостей масштабування навантаження, оскільки сервіси взаємодіють без втручання Mesos.

Kubernetes: поди виставляються через службу, яку можна використовувати для балансування навантаження в кластері. Досягається це тим, що кожна взаємодія сервісів контролюється оркестратором.

Docker Swarm: надає внутрішній DNS, який може використовуватися для розповсюдження вхідних запитів до назви сервісу. Сервіси можуть бути призначені автоматично або можуть працювати в портах, зазначених користувачем. Має менше можливостей налаштування роботи балансувальника навантаження, оскільки для балансування використовується лише DNS.

Висновки до розділу 2

Реалізація відмовостійкої хмарної системи потребує застосування багатьох інструментів та підходів. Кожен компонент системи повинен підтримувати роботу з надлишковими ресурсами що знаходяться на різних фізичних пристроях, що дозволяє продовжувати роботу навіть у випадках відмови обладнання. Система повинна підтримувати масштабованість, що до дозволяє змінювати кількість використаних ресурсів у разі необхідності.

Для оркестрації всіма сервісами застосовуються спеціалізовані програмні засоби, що надають додатковий рівень абстракції над обладнанням в кластері. Apache Mesos є дуже гнучким інструментом і дозволяє реалізувати велику кількість різних функцій, але є дуже низькорівневим і потребує написання додаткових планувальників та фреймворків. Він працює для будь яких застосунків, що можуть бути як Docker контейнерами, так і звичаними задачами, що виконуються на машинах за розкладом. Kubernetes та Docker Swarm працюють лише з контейнерами, але є біль простими в налаштуванні і використанні. Kubernetes є більш гнучким і складним у використанні, тоді як Docker Swarm пропонує просте рішення, яке швидко налаштовується.

3 МЕТОДИ РЕАЛІЗАЦІЇ СИСТЕМИ

Основним середовищем розробки було середовище IntelliJ IDEA Ultimate [15]. Для запуску програми був використаний засіб контейнеризації Docker.

Для реалізації серверної частини використовувалась мова Java 10 та Spring Framework. Веб-клієнт реалізований за допомогою Typescript, фреймворку Angular2 та засобів верстки HTML та CSS.

Реалізація аутентифікації побудована за допомогою Javascript Web Token (JWT).

3.1 Середовище розробки IntelliJ IDEA

Розробка великих програмних продуктів майже неможлива без застосування спеціальних інтегрованих середовищ розробки (IDE), що значно спрощують роботу з проектом. Окрім можливостей звичайного текстового редактору, IDE спрощують роботу з системами контролю версій, комунікацію з базою даних, інтеграцію з фреймворками та інструментами для збірки та тестування систем. Для Java існує три основні IDE: IntelliJ IDEA, NetBeans та Eclipse. NetBeans та Eclipse є безкоштовними та підтримують майже всі основні фреймворки та бібліотеки. Середовище IntelliJ IDEA є платним та, окрім базових можливостей інших IDE, є більш стабільним, має кращі механізми пошуку у проекті, широкий набір додаткових якісних плагінів та зручні методи для рефакторингу коду. Для студентів є можливість отримати IntelliJ IDEA абсолютно безкоштовно.

Саме тому весь код був написаний у інтегрованому середовище розробки IntelliJ IDEA для різних мов програмування від компанії JetBrains.

До основних можливостей IntelliJ IDEA відносяться:

— розумне автодоповнення, інструменти для аналізу якості коду, зручна навігація, розширені рефакторингом і форматування для Java, Groovy, Scala, HTML, CSS, JavaScript, CoffeeScript, ActionScript, LESS, XML і багатьох інших мов;

— підтримка всіх популярних фреймворків і платформ, включаючи Java EE, Spring Framework, Grails, Play Framework, GWT, Struts, Node.js, AngularJS, Android, Flex, AIR Mobile і багатьох інших;

— інтеграція з серверами додатків, включаючи Tomcat, TomEE, GlassFish, JBoss, WebLogic, WebSphere, Geronimo, Resin, Jetty і Virgo;

— інструменти для роботи з базами даних і SQL файлами, включаючи зручний клієнт і редактор для схеми бази даних;

— інтеграція з комерційними системами управління версіями Perforce, Team Foundation Server, ClearCase, Visual SourceSafe;

— інструменти для запуску тестів і аналізу покриття коду, включаючи підтримку всіх популярних фреймворків для тестування.

До складу IntelliJ IDEA входить модуль візуального проектування програм GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse. Під час розробки даного програмного забезпечення було використано 2018.1 версію продукту.

3.2 Інструменти для збірки проекту

Системи на базі Java можна збирати двома популярними системами: Maven та Gradle. Maven є найпоширенішим інструментом і має багато можливостей, але він для конфігурації проектів застосовує XML, що робить всі конфігурацію досить великими і складними для читання. Окрім того, для створення більш складних алгоритмів по зборці проектів потрібно писати дуже багато конфігурацій та плагінів. Саме тому для збірки проекту був використаний Gradle — інструмент для збірки з наголосом на

автоматизацію збірки та підтримку розробки на різних мовах програмування [13]. Gradle пропонує гнучку модель, яка може підтримувати весь життєвий цикл розробки від збору і упаковки коду до публікації веб-сайтів. Gradle був розроблений для забезпечення автоматизації збирання на декількох мовах і платформах, включаючи Java, Scala, Android, C / C ++ та Groovy, і тісно інтегрований з інструментами розробки і безперервних серверів інтеграції, включаючи Eclipse, IntelliJ і Jenkins.

Далі приведено список основник можливостей Gradle:

— В основі Gradle лежить багата розширювана мова предметної області (DSL), заснована на Groovy. Gradle ставить декларативну побудову проектів на наступний рівень, надаючи декларативні елементи мови, які можна застосовувати за власним вподобанням. Ці елементи також забезпечують підтримку збірки за конвенціями для Java [1], Groovy, Web і Scala проектів. Більш того, ця декларативна мова є розширюваною. Можна додавати власні елементи мови або покращувати вже існуючі, забезпечуючи короткі і зрозумілі збірки які просто підтримувати.

— Гнучкість і багатство Gradle дозволяють застосовувати загальні принципи проектування проектів. Наприклад, дуже легко скласти свою збірку з повторно використовуваних частин логіки для побудови інших проектів. З Gradle можна об’єднувати і розділяти компоненти за власним бажанням без необхідності підстраюватись до інструменту збірки.

— Підтримка Gradle для збірки кількох проектів краща серед усіх альтернатив. Залежності проекту перш за все. Завдяки Gradle є можливість налаштувати залежності як і в реальному світі, він слідує ієрархії проекту а не навпаки.

— Незважаючи на іноваційні підходи, Gradle дозволяє з легкістю мігрувати з інших систем збірки проектів таких як Maven та Ant.

— Скрипти збірки Gradle написані на Groovy, а не XML. Це дозволяє використовувати повноцінну динамічну мову для опису своїх збірок, що дає незрівнянну гнучкість та лаконічність усіх скриптів.

— Для використання Gradle не обов'язково встановлювати його собі. Gradle надає можливість використовувати спеціально побудовану обгортку, що зменшує кількість необхідних залежностей і дозволяє простіше виконувати збірки на сервісах для Continuous Integration (CI).

3.3 Засоби розгортки програмного продукту

Наразі існує два основних способи розгортання своєї системи: робити все за допомогою різноманітних скриптів, або застосувати сучасні засоби контейнеризації. На відміну від першого способу, контейнери ізолюють сервіси одне від одного і спрощують повторний процес розгортки системи. Саме тому для проекту був обраний спосіб розгортки за допомогою контейнерів.

Система Docker — провідна програмна платформа для роботи з контейнерами. Розробники використовують Docker для усунення проблем з середовищем при спільній роботі над кодом з колегами [12]. Адміністратори використовують Docker для запуску і керування програмами які разом працюють в окремих контейнерах, щоб отримати кращу сумісніс. Підприємства використовують Docker для створення гнучких практик поставки програмного забезпечення, щоб поставляти нові можливості швидше, надійніше і впевненіше, як для дистрибутивів Linux, так і для Windows Server.

З використанням контейнерів все що необхідно для запуску певної частини програми запаковане в ізольованому контейнері. На відміну від віртуальних машин, контейнери не підіймають повноцінну операційну систему, тільки бібліотеки і налаштування необхідні для забезпечення роботи програмного забезпечення (рисунок 3.1). Це дозволяє створювати ефективні, легкі, автономні системи та гарантує єдине середовище роботи незалежно від того де була розгорнута система. Docker автоматизує повторювані задачі установки та налаштування середовища розробки, так що розробники можуть зосередитися на тому, що важливо: розробка програмного забезпечення.

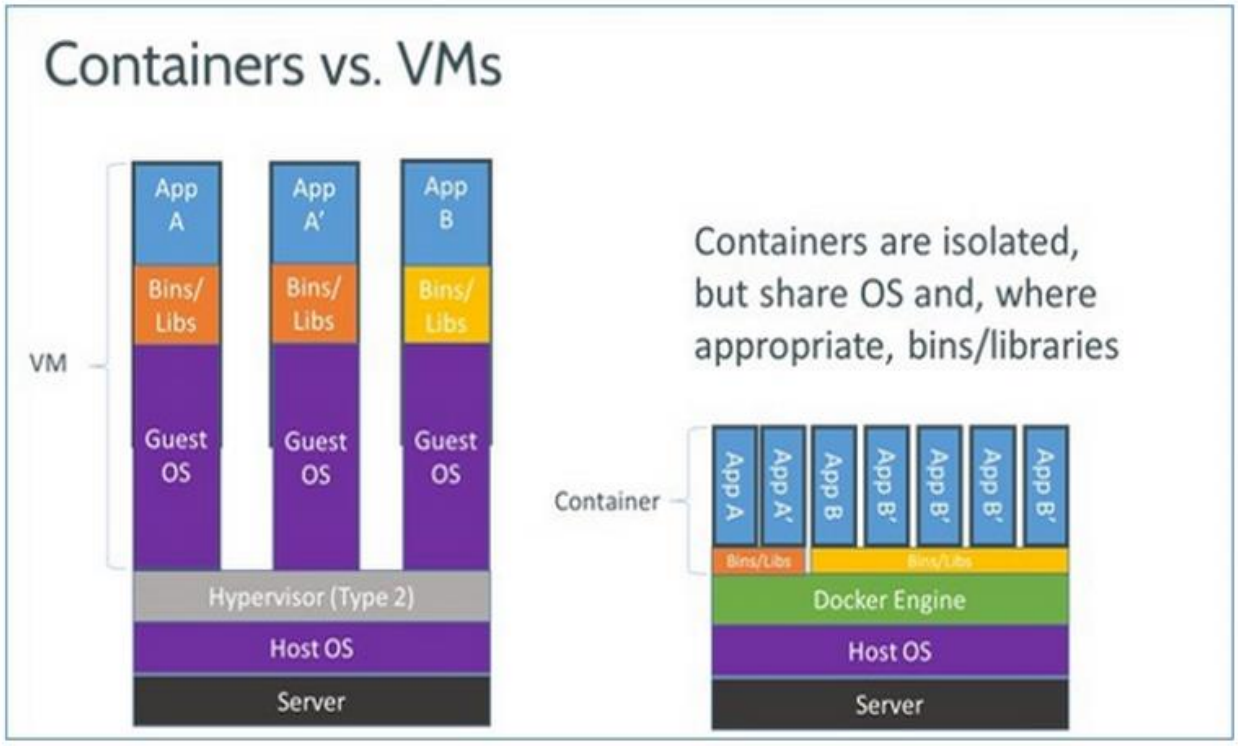

Рисунок 3.1 — Порівняння віртуальної машини та контейнера

Розробникам, що використовують Docker не потрібно встановлювати і налаштовувати складні бази даних і турбуватися про переключення між несумісними версіями середовищ виконання. Коли програма перетворена на Docker образ, складність переноситься у контейнери, які легко будуються, спільні між командою розробників і працюють в різному середовищі. Ознайомлення колеги з новою частиною програмного продукту не займає декількох годин установки програм і пояснення процедур налаштування. З кодом, який поставляється разом з Dockerfile, що містить інформацію про збірку образу, простіше працювати. Залежності завантажуються у запакованих Docker образах і будь-хто може зібрати і запустити програму у лічені хвилини.

3.4 Технології для розробки інтерфейсу

Для розробки інтерфейсу можна було скористатися одним з багаточисельних фреймворків для JavaScript: Vue, React, Angular та інші. Для даного програмного продукту був обраний саме Angular 6, оскільки він дозволяє писати веб-клієнти на

типізованій мові TypeScript та надає продуману структуру проекту, що дозволяє писати клієнти швидше, а потім з легкістю додавати нову функціональність без значних змін у інших частинах програми.

Окрім того, для створення клієнтської частини системи були використані такі технології як HTML5, CSS3, а також Bootstrap 4, які автоматизують створення інтерфейсу та спрощують доступ до різних частин сайту.

3.4.1 Фреймворк Angular 6

Оновлений Angular 6 задає каркас проекту, він йде з багатьма вбудованими блоками, які охоплюють більшість поширених сценаріїв при розробці веб-додатків [10]. Існує також чіткий поділ ролей різних елементів:

Сервіси (Services) — елементи, які використовують дані з API або розділяють стан між декількома компонентами.

Компоненти (Components) — будівельні блоки UI, які використовують сервіси. Можуть бути вкладені один в одного через структурні директивні селектори.

Директиви (Directives) — поділяються на структурні та атрибутні директиви. Структурні директиви (наприклад, *NgFor) впливають на DOM (Document Object Model), тоді як директиви атрибутів є частиною елементів і контролюють їх стиль і стан.

Пайпи (Pipes) — використовуються для форматування того, як відображаються дані у вікні перегляду.

Модулі (Modules) — експортоспроможні блоки додатку, які ізолюють компоненти, директиви, пайпи, сервіси і маршрути разом.

3.4.2 Мова TypeScript

Фреймворк написаний на TypeScript (TS) [23], це мова-надбудова над JavaScript, розроблена Microsoft, яка компілюється у чистий JavaScript (рисунок 3.2).

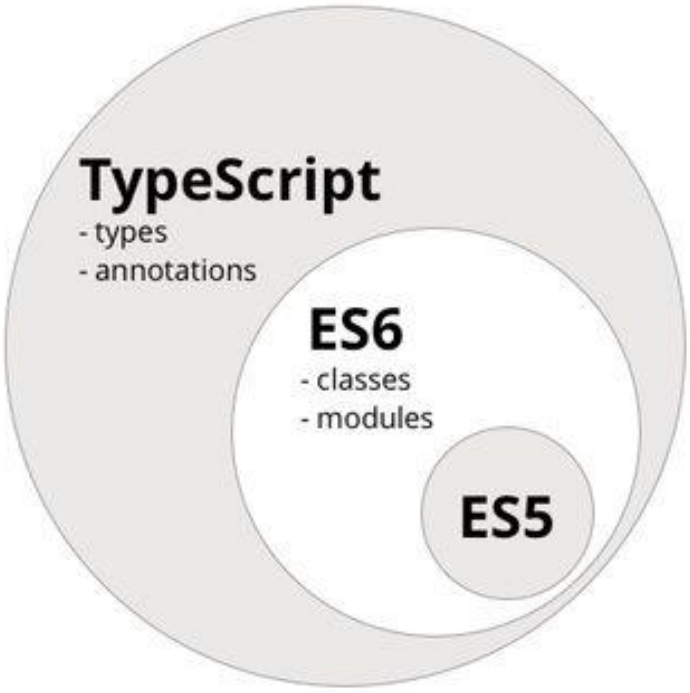

Рисунок 3.2 — Можливості TypeScript у порівнянні з JavaScript

Мова вводить нові типи, нові структури даних і більше об’єктно-орієнтованих можливостей, що робить код легшим для читання в порівнянні з чистим JavaScript. Спочатку синтаксис TypeScript був навмисно подібний C#. Проте, останнім часом TypeScript використовує стрілку (=>) та інший ES6-подібний синтаксис, що робить TS схожим більше на ES6. Недоліком використання TypeScript є додаткові складнощі при конфігурації програми.

3.4.3 Структура інтерфейсу

Для побудови структури інтерфейсу в Angular 6 використовується Hypertext Markup Language (HTML) — мова розмітки документів. Мова розмітки гіпертекстових документів HTML дозволяє визначити різні типи елементів, що забезпечують функціональність документа: текстові фрагменти із заданими параметрами форматування, списки, таблиці, зображення, гіперпосилання і т.д. Елементи HTML оголошуються за допомогою команд розмітки, що називаються тегами. Всі HTML-теги, що зустрічаються в тексті документа інтерпретуються браузером при відображенні документа.

Для стилізації розмітки використовуються каскадні таблиці стилів CSS — технологія опису зовнішнього вигляду документа, написаного мовою розмітки. CSS використовується для завдання кольорів, шрифтів, розташування і інших аспектів представлення документа. Основною метою розробки CSS було розділення вмісту написаного на HTML або іншій мові розмітки від представлення стилю документа. Це розділення може збільшити доступність документа, надати велику гнучкість і можливість управління його виглядом, а також зменшити складність і повторюваність в структурному вмісті. Крім того, CSS дозволяє представляти один і той же документ в різних стилях.

Спростити реалізацію адаптивного дизайну можна завдяки використанню Bootstrap [11]. Проект Bootstrap — це найпопулярніший набір інструментів для HTML, CSS та JS для розробки адаптивних веб-сайтів та веб-застосунків, що розробляються з метою роботи на мобільних. Цей проект спрощує розробку адаптивних веб-сайтів і веб-застосунків.

3.5 Технології для розробки серверної частини

Зазвичай для розробки систем на Java використовується або Java EE (Java Enterprise), або Spring Framework. Незважаючи на те, що Spring Framework не забезпечував яку-небудь конкретну модель програмування, він став широко поширеним в Java головним чином як альтернатива і заміна моделі Enterprise JavaBeans. Spring Framework надає велику свободу Java розробникам в проектуванні, крім того, він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків промислового масштабу.

Spring Framework — це платформа для розробці на Java, яка надає комплексну підтримку інфраструктури для розробки систем. Spring Framework управляє інфраструктурою залежностей, щоб розробник міг зосередитися на бізнес логіці системи.

Програми на Java, як правило, складаються з об'єктів, які співпрацюють одне з одним для виконання певної логіки програми. Таким чином, об’єкти в додатку мають залежності один від одного.

Хоча платформа Java забезпечує велику функціональність розробки додатків, вона не має засобів для організації основних будівельних блоків у єдине ціле, залишаючи це завдання архітекторам і розробникам. Розробник може використовувати шаблони проектування, такі як Factory, Abstract Factory, Builder, Decorator і Service Locator для складання різних класів і об’єктів, які складають програму [6]. Проте ці моделі є просто іменованими найкращими практиками розробки ПЗ, з описом того, що вони роблять, де їх застосовувати, проблеми, які вони вирішує, і так далі. Шаблони є формалізованими найкращими практиками, які потрібно застосувати до вашої програми.

Компонент Spring Framework Inversion of Control (IoC) вирішує цю проблему, надаючи формалізовані засоби складання розрізнених компонентів у повністю працюючу програму, готову до використання [6]. Spring Framework реалізує формалізовані шаблони дизайну як об’єкти першого класу, які розробники можуть інтегрувати у свої власні програми [20]. Багато організацій та установ використовують Spring Framework таким чином, щоб створювати надійні, підтримувані програми.

3.6 Взаємодія між клієнтом і сервером

Для взаємодії між клієнтом і сервером був застосований принцип взаємодії Representational State Transfer (REST). Це стиль проектування розподілених систем за допомогою обмежень. Центральною абстракцією в REST є ресурс. А головні обмеження виглядають так:

— клієнт-серверна модель;

— взаємодія без збереження стану;

— логічний інтерфейс.

Коли виконується перехід на будь-який сайт, то браузер (клієнт) посилає GET запит до сервера. Сервер приймає запит і відправляє відповідь. Браузер її приймає й відображає отриману інформацію.

В клієнт-серверній моделі сервер надає якийсь сервіс чи ресурси, котрі отримують клієнти, виконуючи запити. При чому клієнт може бути чим завгодно: Android-додатком, браузером або банкоматом (рисунок 3.3).

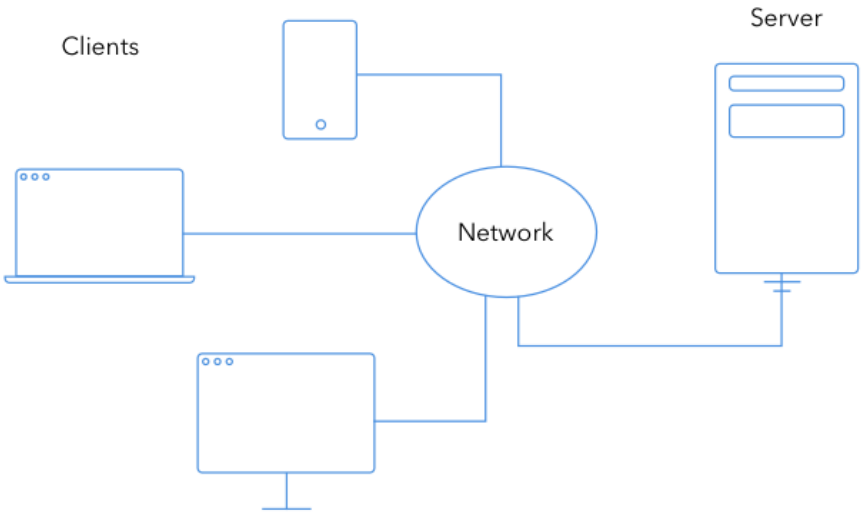

Рисунок 3.3 — Взаємодія між сервером і клієнтами

Перевага такого принципу в тому, що так підтримується розподіл інтересів. Завдяки цьому сервером може бути одна машина, а клієнти тим часом можуть бути геть різними.

В архітектурі REST сервер не повинен зберігати ніякої інформації про стан операції. Сесії повинен зберігати клієнт. Це означає, що якщо сервер отримав два різних запити від одного клієнта, вони не повинні впливати один на одного. Через це, всю інформацію, потрібну для здійснення дії, клієнт повинен відправляти відразу (рисунок 3.4).

Реалізація цього підходу дозволяє абстрагувати клієнт і сервер одне від одного. Для взаємодії використовуються HTTP методи GET, PUT, POST, PATCH, DELETE.

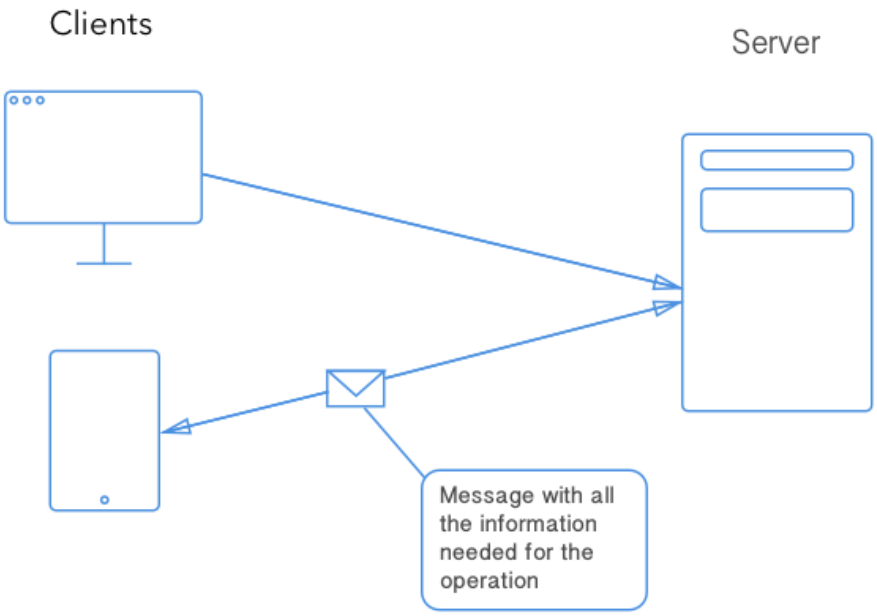

Рисунок 3.4 — Передача необхідної інформації

Такий підхід дозволяє економити купу часу та ресурсів і полегшує масштабування сервера.

Не повинно бути функції login/logout. Натомість повинна бути функція, яка за авторизаційними даними повертає користувачу токен, який він повинен додавати до кожного запиту.

Ресурс — це представлення віртуального об'єкту (такого як зображення), реального об'єкту (клієнт) чи колекції об'єктів. Взагалі, ресурс може бути чим завгодно, розробник API вирішує що в нього буде ресурсом.

У випадку написання RESTful API для сервісу для вивчення слів ресурсом можуть слугувати слова, набори слів, користувачі, режими тренувань та інше.

В REST кожен ресурс повинен бути унікальним. Тому їм присвоюються ID. Для слів /words/1 та /words/2 це два різних слова з ID 1 та 2. Також ресурси можуть бути вкладеними. Наприклад, URI слова 2 в першому наборі слів буде виглядати так /sets/1/words/2.

Метод GET — не єдиний метод HTTP. Ще є POST, PUT, PATCH та DELETE. Більшість запитів до API будуть на виконання CRUD-операцій (створення-читання-

оновлення-видалення). Можна провести паралель між SQL та HTTP операціями (таблиця 3.1).

Таблиця 3.1. CRUD операції для SQL та HTTP

Операція	SQL	HTTP
Створення	INSERT	POST
Читання	SELECT	GET
Оновлення	UPDATE	PUT/PATCH
Видалення	DELETE	DELETE

Приклади використання API:

— щоб отримати слово 2, клієнт повинен виконати GET /words/2;

— щоб створити нове слово, клієнт повинен виконати POST /words.

Підсумовуючи огляд методів реалізації, у якості середовища для розробки було використане середовище IntelliJ IDEA 2017, для збірки проекта був застосований Gradle, а для розгортання системи був використаний Docker. Клієнтська частина була написана на TypeScript за допомогою Angular 6, а інтерфейс був побудований за допомогою Bootstrap 4.

Висновки до розділу 3

Для розробки проекту були використані перевірені часом технології та фреймворки, що мають широку підтримку від розробників та надійну інтеграцію з іншими інструментами. Для розробки клієнтської частини був використаний фреймворк Angular 6, а для розробки серверної частини був використаний фреймворк Spring 5. Взаємодія між клієнтом і сервером відбувається за допомогою REST, що дозволяє виконувати запити через HTTP. Для розгортання системи був використаний Docker, що дозволило для кожного сервісу створити окремий образ та розгортати сервіс разом з усіма необхідними залежностями у будь-якому середовищі.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Основна розробка програмного продукту поділяється на 2 частини: написання серверу та веб клієнта для браузера. Сервер написаний на мові Java 10 з використанням Spring Framework. Клієнтська частина написана на мові Typescript з використанням фреймворку Angular 6 та платморми Node для запуску веб-серверу.

Веб клієнт взаємодіє з сервером за допомогою REST, що дозволяє зменшити зв'язок між обома компонентами до мінімуму.

4.1 Реалізація мікросервісних підходів

Мікросервісна архітектура складається з багатьох маленьких сервісів, кожен з яких динамічно змінювати кількість своїх екземплярів. Це обумовлює необхідність реалізації підходів, що дозволяють системі продовжувати працювати у такому динамічному середовищі. До цих підходів відносяться: реєстр сервісів, наявність єдиного серверу конфігурації, балансування навантаження, механізми повторів та переривання запитів. Для реалізації підходів були використані бібліотеки з проекту Spring Cloud: Eureka, Zuul, Hystrix, Ribbon, Config Server.

4.1.1 Реєстр сервісів

Використання Eureka дозволило створити сервіс, що зберігає реєстр всіх сервісів та їх екземплярів. Якщо з'являється новий сервіс, екземпляр сервісу, або навпаки, екземпляр сервісу перестає відправляти повідомлення про свій стан — дані в реєстрі оновлюються. Окрім того, Eureka надає інтерфейс адміністратора для перегляду інформації про всі зареєстровані сервіси (рисунок 4.1).

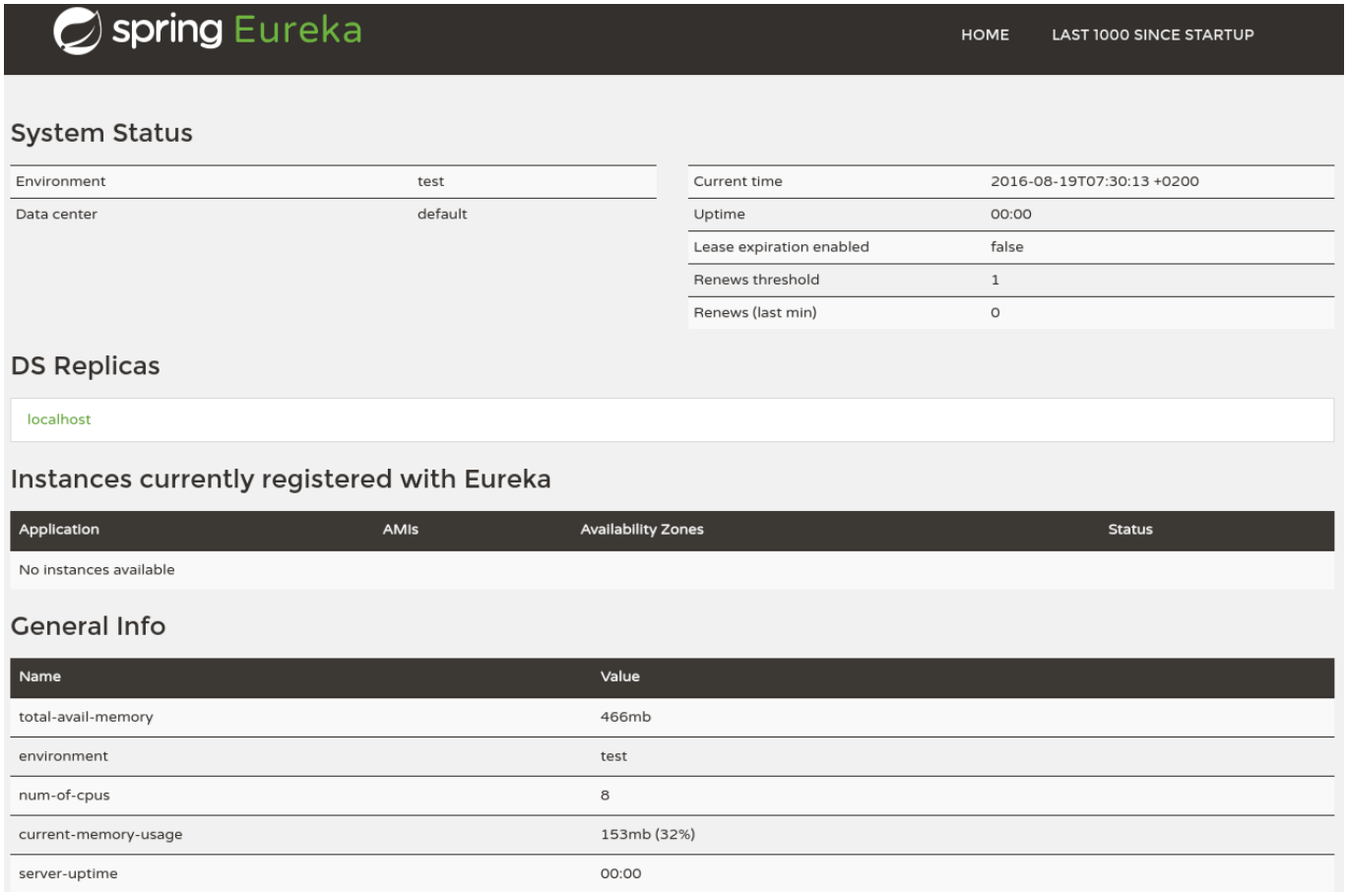

Рисунок 4.1 — Реєстр сервісів Eureka

Для взаємодії з Eureka-сервером кожен клієнт має використовувати eureka-client та сконфігурувати місцезнаходження серверу. Таким чином, кожні 30 секунд всі клієнти повідомляють інформацію про свій стан до сервера. Якщо якість екземпляри сервісів не повідомили про свій стан, то вони вважаються вимкненими.

4.1.2 Сервер конфігурації

Оскільки кожен сервіс може мати велику кількість екземплярів, та може бути ситуація, коли необхідно оновити конфігурацію у всіх екземплярах сервісу, то для вирішення цієї проблеми можна використовувати єдиний сервер конфігурації. Так, кожен екземпляр сервісу має звернутися до серверу конфігурації та отримати всі необхідні налаштування.

Для застосування серверу конфігурації потрібно застосувати бібліотеку config-server-client та сконфігурувати місцезнаходження серверу. Шлях до серверу може

бути як статичний, так і динамічний, з використанням імені сервера та реєстру сервісів.

4.1.3 Взаємодія сервісів одне з одним

При зверненні до сервісу спочатку потрібно знайти де знаходяться екземпляри цього сервісу. Для цього був використаний раніше описаний реєстр сервісів Eureka. За запитом реєстр віддає інформацію про місцезнаходження всіх екземплярів сервісу. Оскільки кожен сервіс може мати декілька екземплярів, тому дуже важливо рівномірно розподіляти навантаження між всіма екземплярами сервісу. Для цього було застосоване балансування навантаження на стороні клієнту. Окрім того, HTTP запити можуть перериватися у мережі, тому був реалізований механізм повторення невдалих запитів. Для балансування навантаження на стороні клієнта і повторення запитів була використана бібліотека Ribbon.

Може виникнути ситуація, коли один з сервісів нижнього рівня перестав відповідати. В результаті клієнт буде чекати кінця запиту кожного з сервісів на шляху до необхідного і отримувати помилки про великий час очікування. Таким чином користувач буде спостерігати довгий час виконання запиту. Для покращення взаємодії з системою був реалізований підхід для переривання невдалих запитів, про які можна здогадатися заздалегідь. Коли точка доступу повертає помилки на 90% запитів протягом однієї хвилини, то вона закривається клієнтом на одну хвилину. Протягом цієї хвилини 1% запитів пропускаються до точки доступу. Якщо цей 1% запитів продовжую отримувати помилки, то точка доступу залишається закритою. Але якщо цей 1% запитів проходить успішно, то точка доступу знову відкривається. Діаграма роботи переривача запитів зображена на рисунку 4.2. Для реалізації цього підходу була використана бібліотека Hystrix.

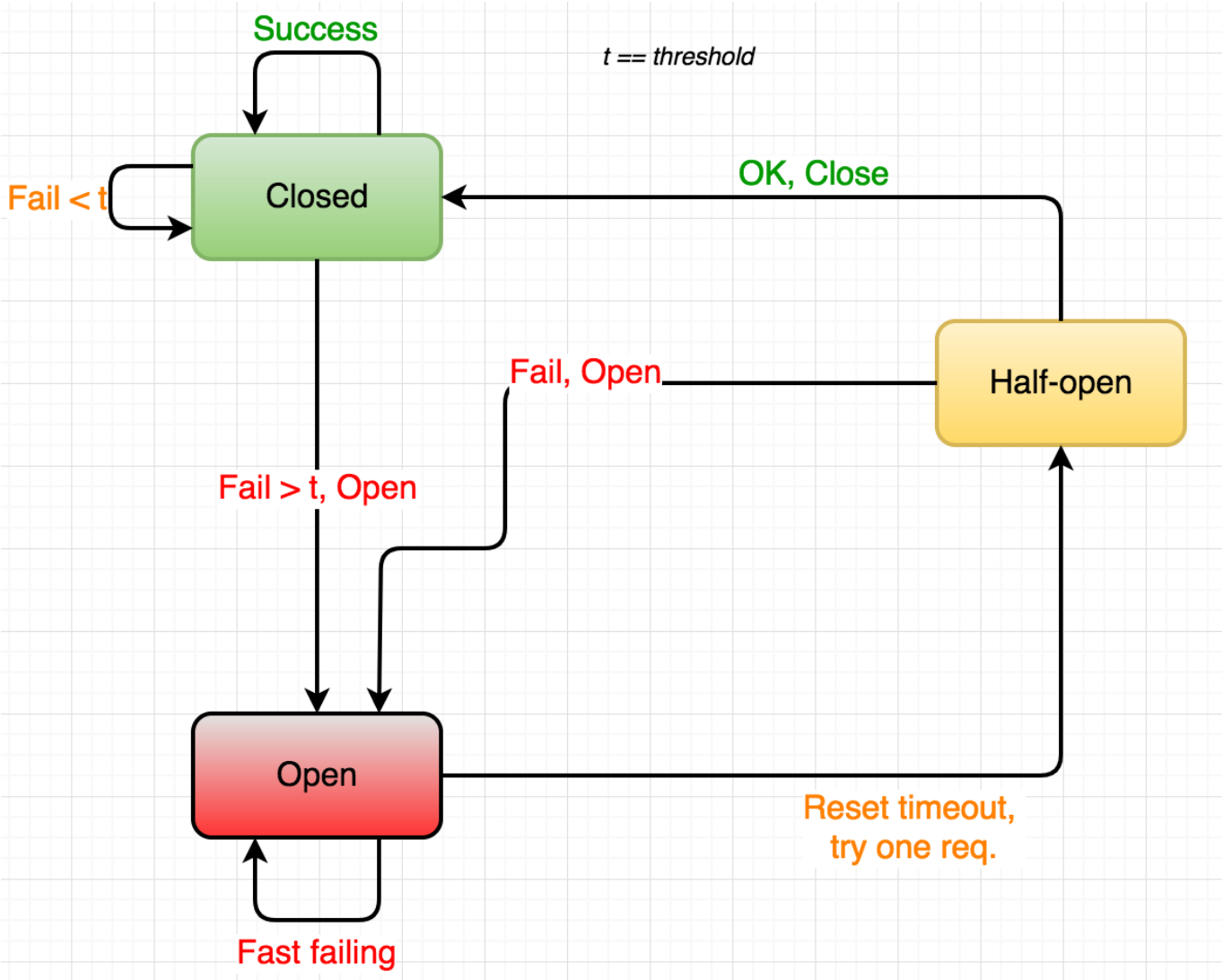

Рисунок 4.2 — Діаграма роботи Hystrix

Реєстр сервісів та необхідність балансування навантаження можуть сильно ускладнити роботу з системою для сторонніх клієнтів, тому для спрощення публічного API був використаний API шлюз. Запити всіх сторонніх клієнтів проходять через цей шлюз, а шлюз вже звертається до реєстру сервісів та балансує навантаження між екземплярами сервісів. Для реалізації API шлюзу була використана бібліотека Zuul.

4.2 Точка доступу для повернення метрик сервісу

Для реалізації точки доступу, що повертає інформацію про метрики сервісу була використана бібліотека Spring Actuator. Стандартна реалізація повертає базові

метрики, такі як загальна кількість запитів. Для того, щоб додати довільні метрики сервісів, потрібно створити Spring Bean типу Gauge або Counter, де gauge повертає поточне значення якоїсь метрики, такої як довжина черги, а counter рахує кількість певних подій та характеризує підсумкове значення за весь період роботи сервісу, таке як кількість помилок при обробці певних запитів.

Оскільки для автоматичного масштабування потрібно буде застосувати метрику, що характеризує поточне значення довжини черги повідомлень, що необхідно буде обробити, то була створена метрика типу Gauge. Для отримання поточного значення довжини черги був використаний RabbitAdmin, що дозволяє отримати мета-інформацію про певну чергу.

4.3 Робота з базою даних

Використовується реляційна база даних, що складається з 11 таблиць, які створюють єдиний інформаційний простір для зберігання та отримання доступу до даних.

4.3.1 Опис таблиць бази даних

Група таблиць “Слово”, “Зображення слів” та “Переклади слів” (рисунок 4.3) застосовуються для зберігання доданих до системи слів. У цій групі основною таблицею є “Слово”, а дві інші є допоміжними і зберігають посилання на зображення та переклади для кожного слова. Розбиття на три таблиці дозволяє реалізувати зв’язок “один-до-багатьох”, де у кожного слова може бути багато зображень чи багато перекладів. Такий підхід дозволяє зберігати загальні дані про слова під час використання сервісу. Таким чином користувачам пропонується список альтернативних перекладів та зображень.

Всі інші таблиці утворюють другу групу, що пов’язує дані про кожного користувача (рисунок 4.4). Таблиця “Користувач” містить всю необхідну інформацію про користувача, його електронний адрес, логін та пароль. Таблиця “Роль” містить назви існуючих ролей. Оскільки кожен користувач може мати багато ролей, а одну й

ту саму роль може мати багато користувачів, то для реалізації відношення “багато-до-багатьох” була створена додаткова таблиця “Ролі користувачів”, яка і підтримує цей зв’язок. На даний момент у системі існує лише одна роль: “USER”. Але розроблена архітектура дозволяє додати нову роль без порушень Open-Closed принципу. Наприклад, до системи можна додати роль “ADMIN” та дозволити користувачам, які мають цю роль, створювати публічні набори слів для вивчення, що доступні всім користувачам сервісу.

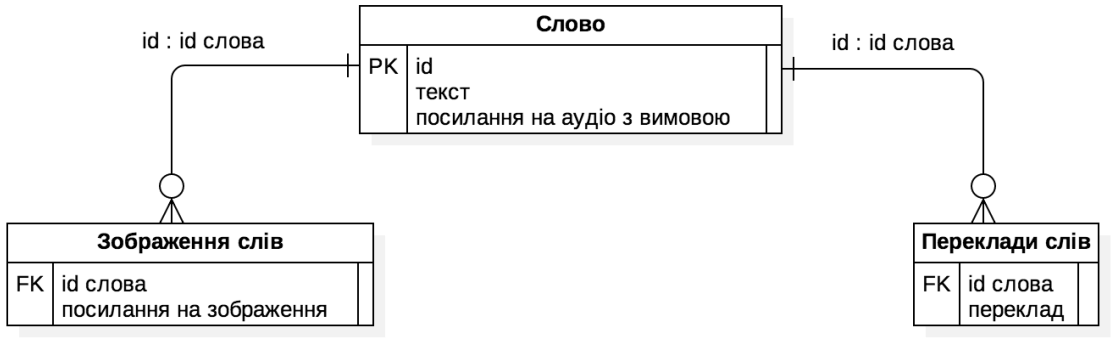

Рисунок 4.3 — Таблиці для збереження бази слів

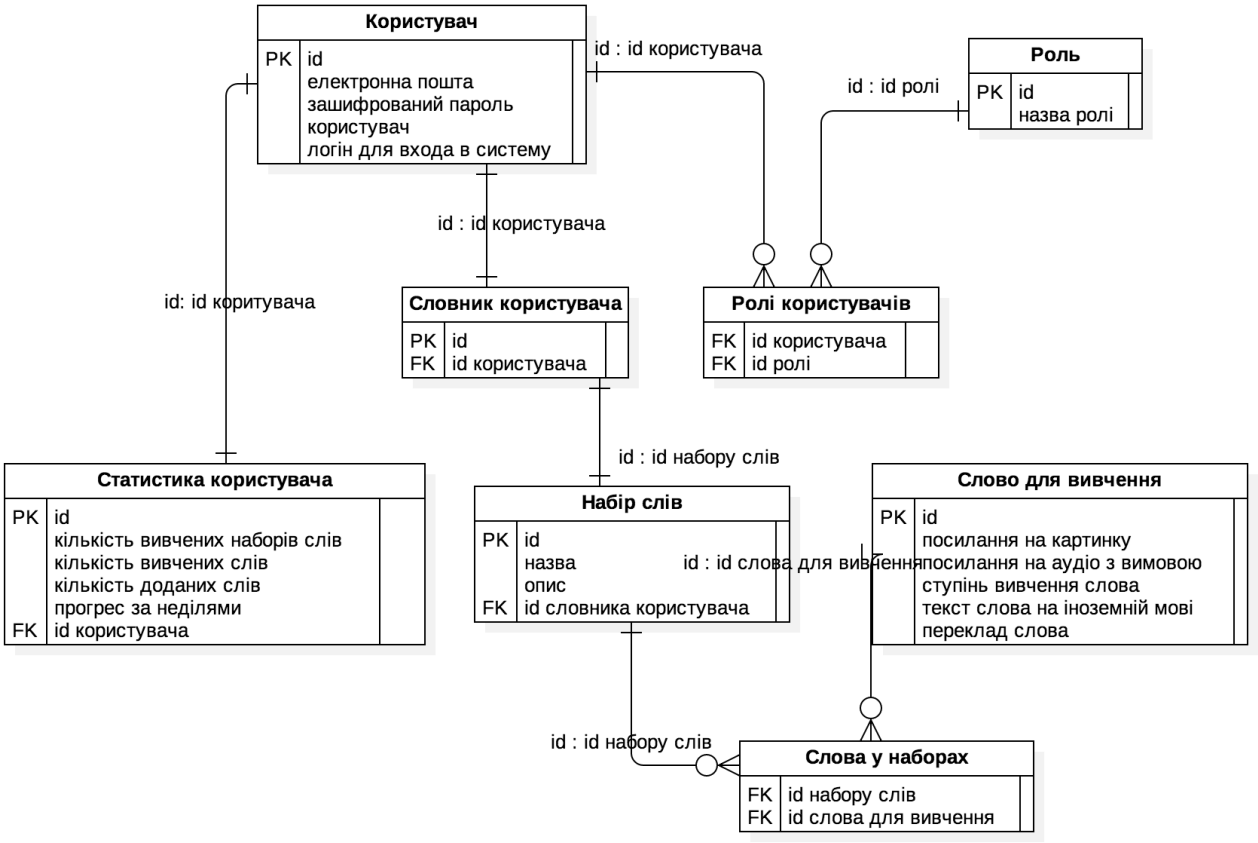

Рисунок 4.4 — Таблиці для збереження даних про користувача

Таблиця “Слово для вивчення” відображає сутність слова під час навчання. Цій таблиці присутнє конкретне значення слова, перекладу та етапу вивчення слова. Ці слова можуть знаходяться у різних наборах, що описуються таблицею “Набір слів”. Оскільки кожне слово може бути додане до багатьох наборів, а набір містить багато слів, то для нормалізації відношення “багато-до-багатьох” була застосована додаткова таблиця “Слова у наборах”.

Набори слів зв’язується з користувачем через таблицю “Словник користувача”. Це дозволяє розширювати систему та додавати до неї нові сутності.

Розглянемо більш детально структуру основних таблиць бази даних системи (таблиця 4.1 - 4.5). Для унікальних ідентифікаторів застосовано тип bigint, це дозволяє уникнути ситуації, коли об’єктів у базі даних більше ніж може поміститися у 4-байтовому int.

Таблиця 4.1. Структура таблиці “Користувач”

Ім’я поля	Тип і розмір поля	Опис поля
id	bigint	первинний ключ
email	varchar(255)	електронний адрес користувача
name	varchar (255)	ім’я користувача
password	varchar (255)	пароль користувача у зашифрованому вигляді
username	varchar (255)	логін користувача для входу в систему

Таблиця 4.2. Структура таблиці “Роль”

Ім’я поля	Тип і розмір поля	Опис поля
id	bigint	первинний ключ
role_type	varchar(255)	назва ролі користувача

Таблиця 4.3. Структура таблиці “Слово”

Ім’я поля	Тип і розмір поля	Опис поля
id	bigint	первинний ключ
sound	varchar(255)	посилання на аудіо файл з вимовою слова
text	varchar (255)	текст слова на іноземній мові

Таблиця 4.4. Структура таблиці “Слово для вивчення”

Ім’я поля	Тип і розмір поля	Опис поля
id	bigint	первинний ключ
image	varchar(255)	посилання на зображення для полегшення вивчення слова
sound	varchar(255)	посилання на аудіо файл з вимовою слова
stage	varchar(255)	рівень вивчення слова
text	varchar (255)	текст слова на іноземній мові
translation	varchar(255)	переклад слова

Таблиця 4.5. Структура таблиці “Набір слів”

Ім’я поля	Тип і розмір поля	Опис поля
id	bigint	первинний ключ
description	varchar(255)	опис набору слів
name	varchar (255)	назва набору слів
user_dictionary_id	bigint	id запису з таблиці USER_DICTIONARY

4.3.2 Об’єктно-реляційне відображення даних у системі

Для роботи з базою даних був використаний Java Persistence API (JPA), що дозволяє описувати моделі за допомогою єдиного інтерфейсу [8]. JPA це специфікація

Java EE і Java SE, що описує систему управління збереженням Java об’єктів до таблиць реляційних баз даних в зручному вигляді. Сама Java не містить реалізації JPA, однак є багато реалізацій даної специфікації від різних компаній. Такий підхід до роботи з базою даних називається Object-relational mapping (ORM), тобто технологія програмування, яка зв’язує бази даних з концепціями об’єктно-орієнтованих мов програмування. JPA не єдиний спосіб збереження об’єктів бази даних, але один з найпопулярніших в Java світі.

Застосування ORM дозволяє абстрагуватися від використаної бази даних та довіритися провайдеру JPA, який вміє працювати з різними реляційними базами даних. Це дозволяє використовувати різні бази даних без необхідності змінювати щось у коді системи. Таким чином під час розробки можна використовувати легку базу даних (наприклад H2 [14]), що буде створюватись одночасно з запуском сервісу, а для роботи з багатьма користувачами використовувати більш надійні та швидкі бази даних як PostgreSQL.

Рівень роботи з базою складається з зв’язаних моделей (entity), які відображають логічні сутності системи. Модель — це легкий об’єкт бізнес-логіки (persistent domain object) що зберігається в базі даних. Кожна модель бази повинна бути помічена анотацією @Entity, а зв’язки між моделями повинні бути організовані за допомогою допоміжних анотацій.

Для зв’язку один-до-одного потрібно застосовувати анотацію “@OneToOneMapping”. Дві анотації @ManyToOneMapping та @OneToManyMapping застосовуються коли декілька моделей відносяться до однієї моделі.

Для позначення зв’язку багато-до-багатьох застосовується анотація @ManyToManyMapping, тобто коли багато моделей відносяться до багатьох інших моделей. У такому випадку JPA створює допоміжну таблицю для нормалізації відносин між таблицями.

Під час розробки був застосований Domain Driven Design, тобто за ієрархією доменної області системи (рисунок 4.5) була створена структура бази даних.

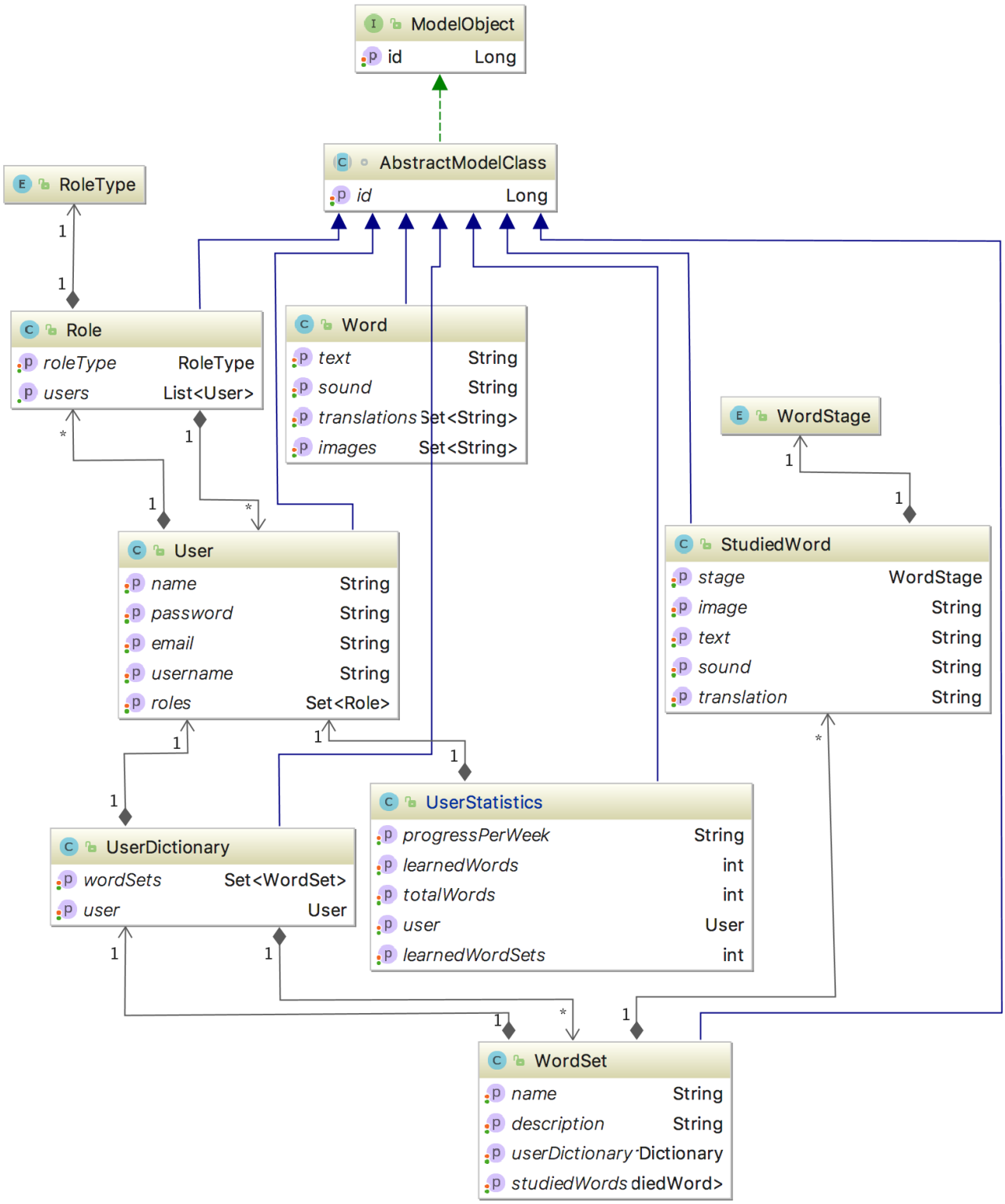

Рисунок 4.5 — Ієрархія моделей

Даний зв'язок описує лише відношення між моделями системи. Провайдер JPA піклується про структуру таблиць та відношення між ними. Абстрактний клас застосовується для того, щоб в кожній моделі не потрібно було описувати повторювані поля.

4.4 Розгортання системи

Застосування Docker дозволило значно спростити процес розгортання сервісів [3]. У файлі Dockerfile описуються кроки для побудови контейнеру.

Директива FROM дозволяє вказати базовий образ для контейнеру, де описується дистрибутив лінуксу, кроки установки додаткових утиліт та конфігурації контейнеру. Для Java серверу був використаний базовий контейнер “ openjdk10:slim” — мінімалістичний дистрибутив Alpine з встановленим Java Runtime Environment (JRE). Обраний базовий образ для веб-клієнту — node:7, дистрибутив Ubuntu з встановленим nodejs сьомої версії. Далі в кожному файлі конфігурації потрібно описати кроки для запуску сервісу (рисунок 4.6, рисунок 4.7).

```
FROM frolvlad/alpine-oraclejdk8:slim
VOLUME /tmp
ADD glossary.jar app.jar
RUN sh -c 'touch /app.jar'
ENV JAVA_OPTS=""
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
```

Рисунок 4.6 — Dockerfile для Java-серверу

```
FROM node:7
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY package.json /usr/src/app
RUN npm install
COPY . /usr/src/app
EXPOSE 4200
CMD ["npm", "start"]
```

Рисунок 4.7 — Dockerfile для веб-клієнту

Для збірки контейнеру потрібно виконати команду “docker build”, вказати назву контейнеру та шлях до вхідних файлів. Після цього Docker зберігає контейнер та додає його до локального репозиторію. Приклад команди для збірки контейнера: “docker build -t solomkinmv/glossary-webui:dev .”. Для перегляду локальних контейнерів достатньо виконати команду “docker images” (рисунок 4.8).

```
❯ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
solomkinmv/glossary-webui     dev                 39758902438e        6 days ago          886 MB
solomkinmv/glossary           0.0.2-SNAPSHOT      3f492377c943        2 weeks ago         360 MB
node                          6                   70c1274808eb        3 weeks ago         661 MB
node                          7                   2ca756a6578b        3 weeks ago         665 MB
frolvlad/alpine-oraclejdk8    slim                00d8610f052e        2 months ago        167 MB
```

Рисунок 4.8 — Перегляд списку контейнерів

Для серверу збірку контейнера можна автоматизувати, створивши окрему Gradle задачу. Для цього потрібно застосувати Docker плагін та описати задачу. Плагін застосовується за допомогою команди “apply plugin”. Для створення окремої задачі потрібно вказати ім’я програми, шлях до Dockerfile, залежність на інші задачі та кроки підготовки перед збіркою контейнера (рисунок 4.9).

```
48  apply plugin: "docker"
49
50  task buildDocker(type: Docker, dependsOn: build) {
51      push = true
52      applicationName = jar.baseName
53      dockerfile = file('src/main/docker/Dockerfile')
54      doFirst {
55          copy {
56              from jar
57              into stageDir
58          }
59      }
60  }
```

Рисунок 4.9 — Конфігурація Gradle для збірки Docker контейнера

Після налаштування Gradle можна збирати контейнер за допомогою однієї простої команди: “gradlew web:buildDocker”. Ця команда складається з назви проекту та назви задачі.

Щоб запустити будь-який контейнер застосовується команда “docker run”, вказується ім’я контейнера та порти, які потрібно відкрити. Приклад команди для запуску сервера: “docker run -p 8080:8080 -t solomkinmv/glossary”.

Щоб мати змогу запустити контейнер на іншому комп’ютері потрібно передати зібраний образ контейнера. Для спрощення передачі образу був застосований Dockerhub — спеціальне сховище образів на серверах Docker. Вигрузка образу відбувається за аналогією з Git серверами, за допомогою команди “docker push” та

зазначення імені образу. Після цього на будь-якому іншому комп'ютері можна написати "docker run <ім'я образу>" і Docker загрузить образ і запустить новий контейнер.

Оскільки для запуску всієї системи потрібно одночасно запускати декілька контейнерів та прописувати певні параметри, то для автоматизації цього процесу був застосований оркестратор Kubernetes — технологія для запуску декількох контейнерів для організації спільної роботи [2]. До переваг Kubernetes відноситься спрощення запуску багатьох контейнерів та налаштування взаємодії між ними. У yaml файлах описуються всі сервіси, які треба ввести в дію, порти для взаємодії та додаткові параметри середовища (рисунок 4.10).

```yaml
version: '2' # specify docker-compose version

# Define the services/containers to be run
services:
  webui: # name of the first service
    image: solomkinmv/glossary-webui:dev # specify the directory of the Dockerfile
    ports:
      - "4200:4200" # specify port forwarding

  server: #name of the second service
    environment:
      - SPRING_PROFILES_ACTIVE=dev,s3
    image: solomkinmv/glossary:0.0.2-SNAPSHOT # specify the directory of the Dockerfile
    ports:
      - "8080:8080" #specify ports forwarding
```

Рисунок 4.10 — Файл-конфігурації для запуску сервісів системи

Після цього запуск всієї системи зводиться до виконання "kubelet create -f .". Ця команда підіймає сконфігуровані середовища, запускає всі сервіси і відкриває порти для їх взаємодії. Розгортання системи зводиться до виклику однієї команди на сконфігурованому Kubernetes кластері, для запуску всього необхідного для повноцінної роботи. Такий підхід є передовим у індустрії і використовується провідними компаніями для запуску своїх продуктів.

4.5 Автоматичне масштабування за довільними метриками

Оркестратор Kubernetes має механізм масштабування, що дозволяє змінювати кількість екземплярів кожного з сервісів. Масштабування дозволяє виконувати всі

запити клієнтів під час пікового навантаження, але постає питання своєчасності. Неможливо постійно передбачати перепади навантаження на систему, тому завжди потрібно слідкувати за метриками системи і своєчасно діяти. Зазвичай для надійності система працює з деяким надлишком ресурсів, щоб бути в змозі витримати непередбачуване навантаження, але такий підхід використовує ресурси віртуальних машин і коштує додаткових грошей. Використання черги дозволяє автоматизувати горизонтальне масштабування і тим самим оптимізувати витрати.

Для вирішення проблеми автоматизації можна застосувати оркестратор Kubernetes та його механізм автоматичного горизонтального масштабування Horizontal Pod Autoscaler (HPA). HPA реалізується як контрольний цикл, який періодично запитує метрики ресурсу для основних показників, таких як використання процессору / оперативної пам’яті а також надає API для реалізації специфічних для системи метрик (рисунок 4.11).

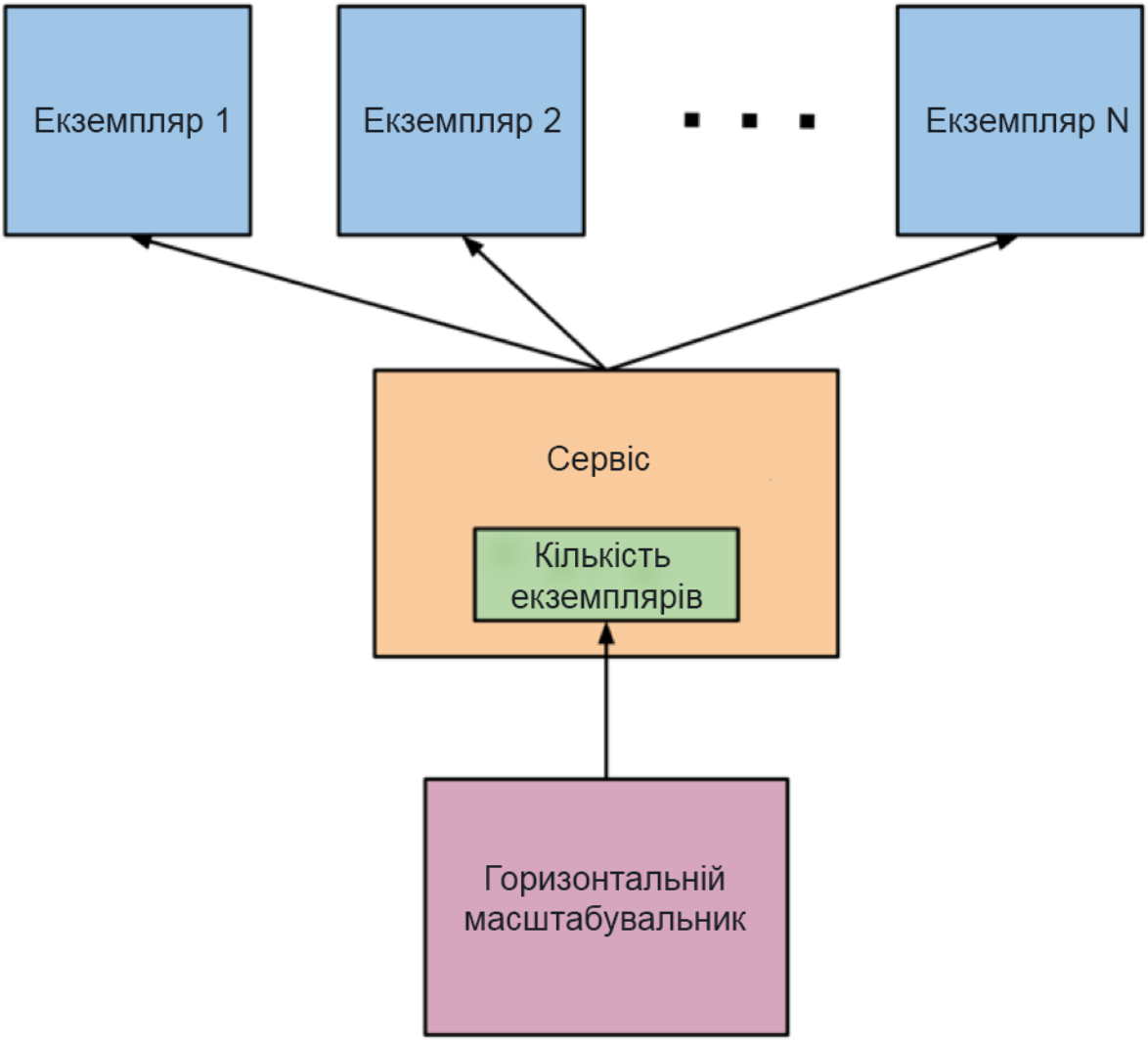

Рисунок 4.11 — Діаграма роботі горизонтального масштабувальника

Кожну ітерацію контролер запитує рівень використання ресурсів, що зазначені у описанні кожного масштабувальника (рисунок 4.12). Потім, якщо задане значення цільового рівня використання ресурсу, контролер рахує значення метрики для кожного екземпляру сервісу і порівнює його з цільовим рівнем, на основі чого і приймає рішення про необхідність масштабування вгору або вниз.

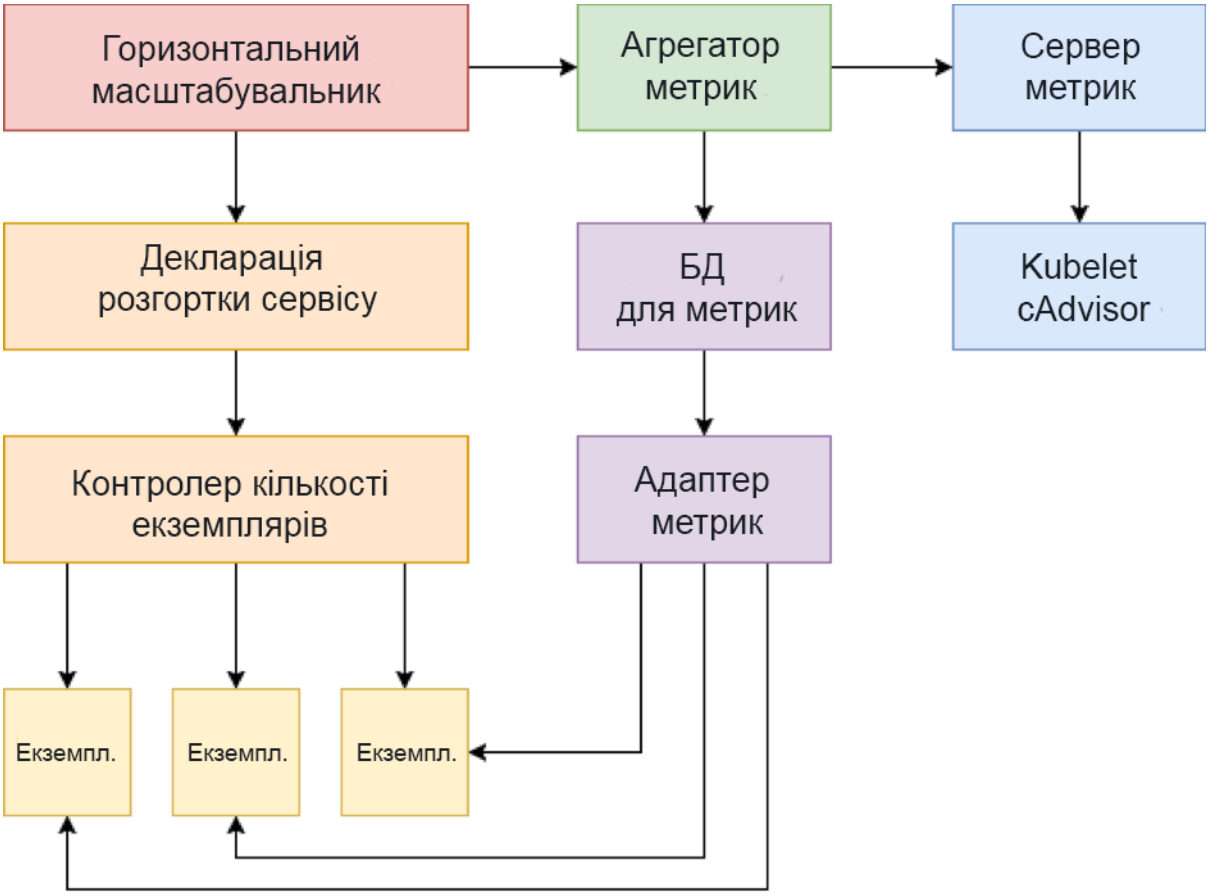

Рисунок 4.12 — Діаграма взаємодії компонентів для збору та збереження метрик

За замовчуванням Kubernetes підтримує автоматичне масштабування за рівнем використання CPU або оперативної пам'яті, але ці метрики не підходять для всіх систем. Іноді для більшої точності потрібно застосовувати інші метрики.

Для оптимізації автоматичного масштабування ідеально підійде довжина черги як індикатор потреби масштабування. Чим більше необроблених повідомлень у черзі, тим більше нових екземплярів сервісу потрібно створити. Якщо ж черга майже пуста, то кількість сервісів можна знову зменшити.

Для вирішення проблеми перш за все потрібно перевести розгортання системи у Kubernetes кластер. Для застосування горизонтального масштабування у Kubernetes потрібно для кожного сервісу, що потребує масштабування, реалізувати HTTP API для повернення необхідних метрик. Це можна зробити як вручну за допомогою будь якої мови програмування, так і за допомогою спеціалізованих бібліотек. Так, наприклад, при використання Spring Framework можна застосувати Spring Actuator ,

який надає основні метрики про систему. Деталі реалізації точки доступу, що повертає специфічні метрики користувача були описані у попередньому розділі.

Для обробки метрик сервісів був застосований сервер метрик, що є агрегатором всіх метрик з кластеру. Сервер метрик збирає дані про використання процесору та оперативної пам'яті кожного вузла, отримуючи дані за допомогою kubernetes.summary_api, що є ефективним API для передачі даних від Kubelet/ cAdvisor до сервера метрик (рисунок 4.13).

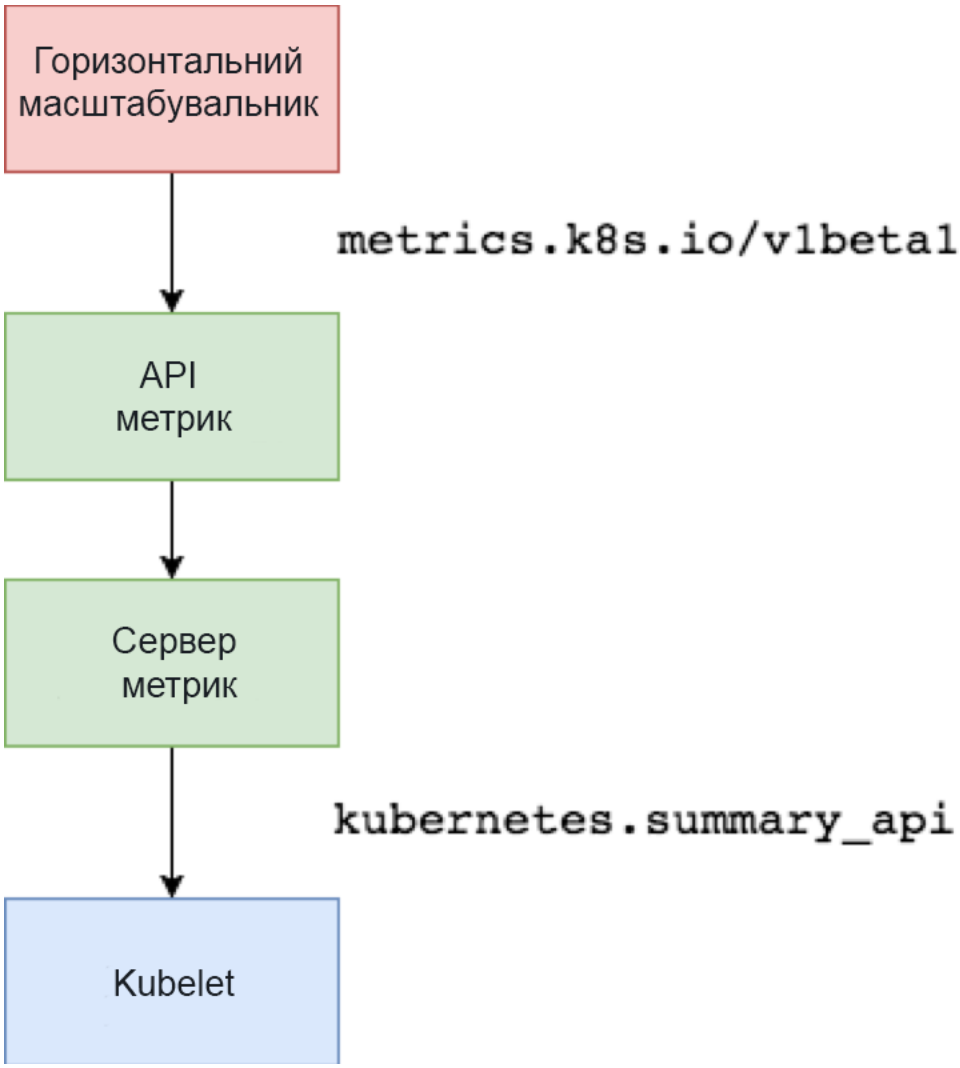

Рисунок 4.13 — Збір метрик з кожного екземпляра сервісу за допомогою Kubelet

Для запуску серверу метрик були застосовані конфігурації з офіційного репозиторію, які були запущені за допомогою командного рядка та консольної утиліти kubectl .

Для автоматичного масштабування був розгорнутий HPA у Kubernetes та вказати мінімальну та максимальну кількість екземплярів, назву метрики та цільове середнє значення (рисунок 4.14).

```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: spring-boot-hpa
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: backend
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Pods
    pods:
      metricName: messages
      targetAverageValue: 10
```

Рисунок 4.14 — Налаштування для масштабування за метрикою та цільовим значенням

У даному прикладі вказується мінімальна та максимальна кількість екземплярів сервісів, а також середнє значення метрики “messages”, залежно від якої відбувається масштабування.

Після створення HPA кількість екземплярів сервісу повинна дорівнювати мінімальній кількості, що описана у конфігураційному файлі. У даному випадку 2. Для перегляду подій що призвели до масштабування потрібно виконати наступну команду: kubectl describe hpa.

Якщо навантажити систему, то можна помітити, що кількість екземплярів сервісу виросла (рисунок 4.15). Це означає що автоматичне масштабування Kubernetes працює.

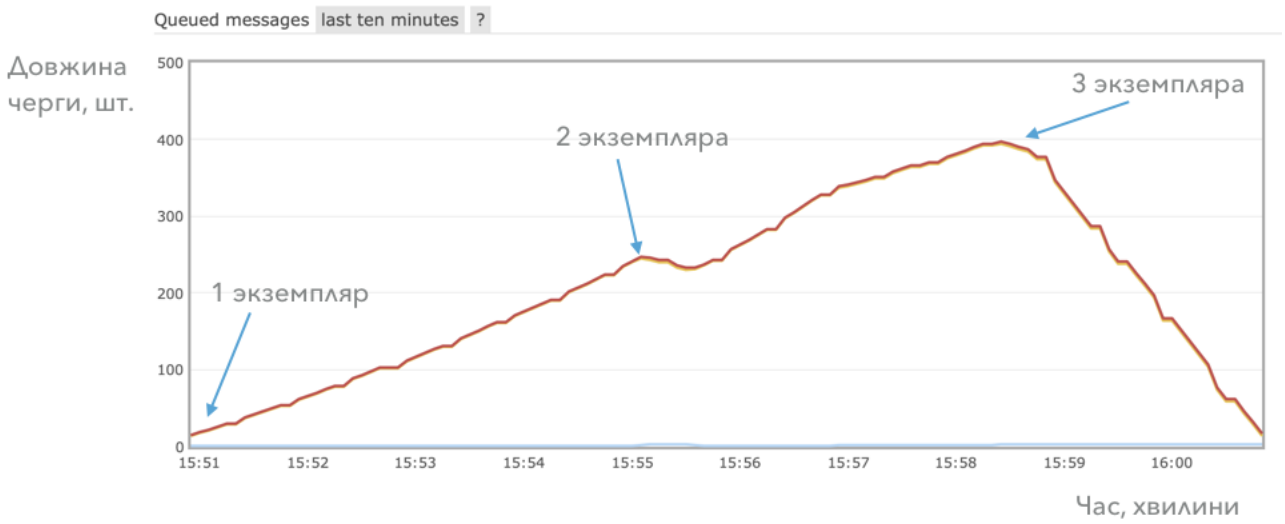

Рисунок 4.15 — Автоматичне масштабування при збільшенні черги

Застосування Horizontal Pod Autoscaler дозволяє будувати системи, що можуть динамічно підлаштовуватися під реальне поточне навантаження системи. Даний підхід не тільки підвищує стабільність систем, але й дозволяє економити ресурси на серверах, дозволяючи максимально зменшити кількість необхідних ресурсів коли рівень навантаження мінімальний. Використання користувацьких метрик покращує точність роботи автомасштабування та дає широкий спектр можливостей по налаштування розподіленої системи.

4.6 Автентифікація користувача

Оскільки у кожного користувача додані свої слова для вивчення, є певний прогрес та статистика, то неможливо обійтися без механізму автентифікації. Для вирішення цієї проблеми був застосований Spring Security [22], який надає базові методи для охорони веб-сервісу.

Для автентифікації Spring Security пропонує cookies та basic auth. Обидва ці способи швидкі у розробці, але мають ряд недоліків. Автентифікація за допомогою cookies змушує зберігати сесію на сервері та пристосована для роботи з браузерами, тоді як більшість сучасних веб-сервісів мають безліч мобільних та веб клієнтів, а basic auth змушує при кожному запиті передавати логін та пароль, що просто не допустимо з точки зору безпеки. Для вирішення цих недоліків було вирішено застосувати більш

сучасний метод автентифікації, що базується на використанні JSON Web Token, скорочено JWT [16].

Послідовність роботи з JWT виглядає наступним чином: клієнт відправляє логін та пароль на сервер, сервер створює JWT та повертає його клієнту, при наступному зверненні клієнт прикріплює до запиту токен, сервер перевіряє підпис та автентифікує клієнта на основі отриманих даних та повертає результат (рисунок 4.16).

Токен JWT у собі може містити усю необхідну інформацію для автентифікації, це дозволяє позбавитися сесії та зайвих перевірок на наявність користувача у базі даних або його ролі. Окрім того, JWT є універсальним засобом, який дозволяє працювати як з браузерами, так і з мобільними пристроями.

Оскільки Spring Security не надає реалізацію автентифікації за допомогою JWT, то потрібно було додати її, максимально ефективно інтегрувавшись з поведінкою Spring Security.

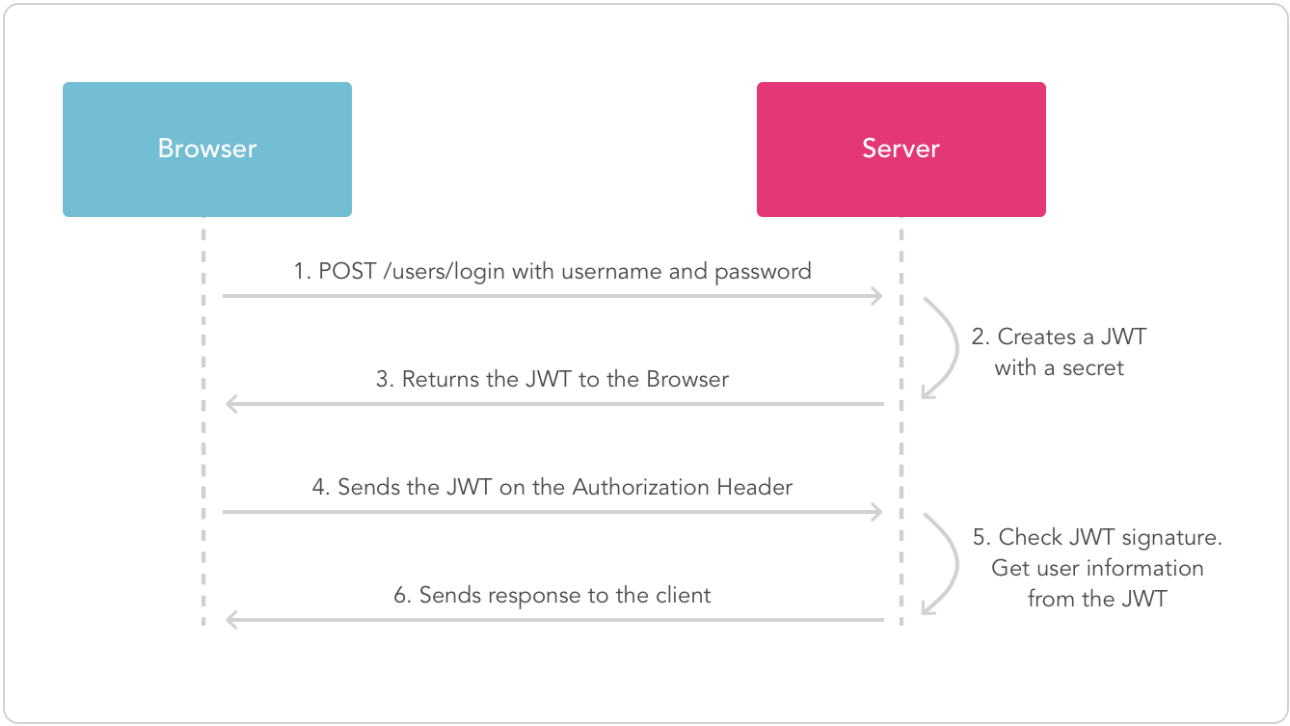

Рисунок 4.16 — Автентифікація на основі токенів

Для створення, валідації та розшифрування токенів було додано залежність на io.jsonwebtoken:jwt. Spring Security надає механізм фільтрації запитів, що дозволяє обробити, автентифікувати або відкинути запит до виклику відповідного контролера. Тому для підтримки JWT був створений новий фільтр, що обробляє запит та вилучає

з його заголовку токен, а потім викликає провайдер автентифікації та передає йому токен. Провайдер автентифікації реалізує інтерфейс AuthenticationProvider, перевіряє підпис, вилучає всю корисну інформацію з токена, таку як ім'я користувача, його роль (користувач або адміністратор) та термін дії токена. Результатом роботи провайдера є об'єкт Authentication, що містить всю необхідную для системи інформацію про користувача та його роль. Для обробки ситуацій, коли автентифікація не вдалася, був створений обробник помилок, що реалізує інтерфейс AuthenticationFailureHandler, відловлює виключні ситуації Java та генерує дружні для користувача HTTP відповіді зі статусом UNAUTHORIZED та кодом помилки.

Інтеграція Spring Security та автентифікації за допомогою фільтру запитів дозволила чітко розмежувати компоненти, що відповідають за безпеку та бізнес логіку. Будь-які зміни автентифікації не вплинуть на основну логіку роботи сервісу. Результат впровадження JWT дозволив зменшити навантаження на базу даних та побудувати REST сервіс без стану (stateless), що дозволить простіше масштабуватися при збільшенні кількості користувачів.

4.7 Робота з зображеннями

Робота з зображеннями складається з двох головних компонентів: пошук зображень для слова та загрузка користувачем власних зображень.

Для пошуку зображень був використаний відкритий API сервісу Flickr, що спеціалізується на роботі з зображеннями. За допомогою цього API можливо зробити запит, який поверне вказану кількість посилань на зображення вказаного розміру. Таким чином був реалізований сервіс, який повертає список посилань на маленькі зображення. Клієнт може з легкістю зробити запит на сервер та отримати всю необхідну інформацію по пошуковому запит. Такий рівень абстракції дозволяє клієнту не замислюватися про деталі пошукового механізму та джерело зображень.

Завантаження зображень повинна повертати посилання на збережене зображення, таким чином клієнт не має знати про місце збереження і деталі побудови посилань. Для збереження зображень реалізовано 2 сервіси, один з яких зберігає на

файлову систему, а інший на зовнішній сервіс Amazon S3 [9]. Обидва ці сервіси реалізують один і той самий інтерфейс та призначені для роботи різних профілів. Таким чином, якщо сервер буде запущений з профілем “s3”, то зображення будуть зберігатися на зовнішній сервіс. В іншому випадку всі зображення будуть зберігатися на файлову систему у папку, вказану у налаштуваннях сервісу.

Сервіс Amazon S3 — це сервіс для зберігання статичних файлів (рисунок 4.17). Він ідеально підходить для збереження таких даних як зображень та аудіо файлів. Спеціально для цього було створено дві групи для зберігання даних: glossary-images та glossary-pronunciations.

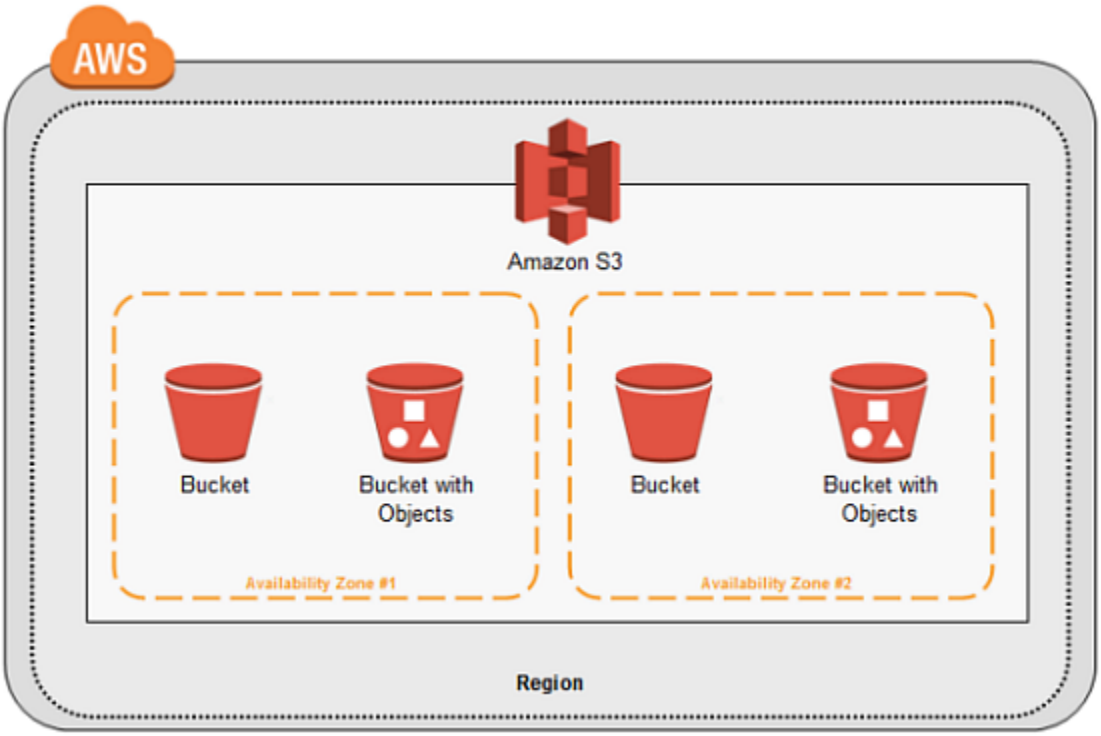

Рисунок 4.17 — Архітектура Amazon S3

Сервіс Amazon S3 використовується у більшості компаній, яким треба зберігати статичні дані.

4.8 Генерація вимови слів

Для генерації вимови слів був використаний сторонній сервіс Amazon Polly. Цей сервіс приймає текст для якого треба вимова та ряд параметрів, такі як тип

голосу, швидкість та гучність вимови і багато іншого. Результатом виклику цього API є вхідний потік даних на аудіо файл (рисунок 4.18).

Завдяки застосуванню Amazon SDK для Java вийшло позбавитися від прямої роботи з API та низькорівневої роботи з потоками вхідних даних. Завдяки бібліотеці можна покластися на розроблену абстракцію, створити об'єкт AmazonPolly за допомогою вбудованого Builder, створити запит для синтезації вимови та викликати метод "synthesizeSpeech".

Рисунок 4.18 — Взаємодія з Amazon Polly

Реалізовано два сервіси для збереження аудіо даних. Перший зберігає дані на файлову систему, а другий на Amazon S3.

4.9 Переклад слів

Для автоматичного перекладу слів був застосований Yandex Translate API, що дало змогу полегшити для користувача додавання слів. Сервіс Яндекс надає можливість виконувати безкоштовні запити з певними обмеженнями: результат повертає тільки один варіант перекладу. Для отримання перекладу потрібно відправити запит та вказати ключ з мовою для перекладу (рисунок 4.19).

```
https://translate.yandex.net/api/v1.5/tr.json/getLangs ?
key=<API key>
 & [ui=<language code>]
 & [callback=<name of the callback function>]
```

Рисунок 4.19 — Yandex Translate API

Де key — це ключ, отриманий на сайті Yandex Translate, ui — ідентифікатор з якої і на яку мову відбувається переклад. Взаємодія з стороннім сервісом виконується у відповідному класі, який додає рівень абстракції над перекладом слів.

4.10 Тестування серверної частини

Без тестів неможливо бути впевненим, що продукт працює правильно. У даному продукті упор був зроблений на інтеграційне тестування, оскільки воно може підтвердити правильність роботи всієї системи, на відміну від юніт-тестування, де кожен маленький компонент тестується окремо, що не гарантує коректної роботи всієї системи.

Інтеграційне тестування набагато складніше за своєю організацією, оскільки кожен тест має піднімати весь сервіс включаючи базу даних. Як наслідок, інтеграційні тести значно повільніші. Для тестування був застосований Spring Boot Tests, що полегшує запуск всього сервісу, запуск легкої бази що зберігає дані у пам'яті та спеціальних методів для створення запитів та перевірки відповідей сервера.

Для тестів був створений абстрактний клас MockMvcBase, що має анотацію @SpringBootTest для запуску тестів, ініціалізує тестове MockMvc середовище та оголошує метод, який створює обробник запитів для додавання заголовку автентифікації до тестів. Після цього кожен тестовий клас наслідується від MockMvcBase, описує які дані потрібно додати перед кожним тестом та імітує реальну роботи сервісу.

Так для виконання GET запиту від імені користувача потрібно виконати "mockMvc.perform(get("/api/words/{wordId}",word.getId()).with(userToken()))". Метод "perform()" приймає інструкції який саме запит потрібно виконати, а метод "with()"

приймає додаткові обробники запитів. Метод “userToken()” створений власноруч і додає дані для автентифікації. Вказаний рядок запускає всю систему і оброблює запит як і при реальному запуску. Для перевірки результати застосовується шаблон проектування Builder [5], тобто викликається метод “andExpect()” та передається очікувана умова.

Приклади умов, які можна перевірити:

— код відповіді (200, 201, 404, 500 та ін.);

— тип відповіді (text/plain, application/json та ін.);

— очікуване тіло відповіді.

4.11 Документація API

Оскільки сервер і клієнт є повністю різними програмами, то для опису взаємодії була згенерована документація API серверу, що дозволила задокументувати контракт серверу та слідувати ньому на клієнті.

Зазвичай, найбільшою проблемою, яка зв’язана з документацією є її неактуальність. Тобто часто контракт взаємодії змінюється, а документація залишається старою, тобто неактуальною. Для побудови документації був використаний Spring REST Docs, що вирішує проблему неактуальної документації, оскільки немає окремого місця для описання API [21]. Вся документація пишеться одразу в тестах. Якщо документація описує якесь відсутнє поле, результат, або навпаки, щось не описує, то система не проходить даний тест.

Описання документації відбувається у форматі шаблону Builder та пишеться одразу після “andExpect()” методів. Spring REST Docs дозволяє описати наступні елементи:

— HTTP заголовки;

— параметри HTTP запиту;

— параметри URL шляху;

— поля відповіді;

— приклад запиту;

— приклад відповіді;

Після запуску тестів генеруються файли, які описують один з аспектів для даного запиту. Для побудови єдиної сторінки документації потрібно створити index файл у форматі AsciiDoc (adoc) у директорії src/docs/. У цьому файлі можна писати текст та вставляти згенеровані фрагменти (рисунок 4.20). Окрім того, файл підтримує спеціальний синтаксис для форматування тексту та створення таблиць.

```
[[resource-practices-quizzes]]
=== Get practice quiz
A `GET` request generates quiz for the specified word set.

include::{snippets}/practice-controller-test/get-practice-quiz/response-fields.adoc[]

==== Headers

include::{snippets}/practice-controller-test/get-practice-quiz/request-headers.adoc[]

==== Request parameters

include::{snippets}/practice-controller-test/get-practice-quiz/request-parameters.adoc[]

==== Example request

include::{snippets}/practice-controller-test/get-practice-quiz/curl-request.adoc[]

==== Example response

include::{snippets}/practice-controller-test/get-practice-quiz/http-response.adoc[]

==== Links

include::{snippets}/practice-controller-test/get-practice-quiz/links.adoc[]
```

Рисунок 4.20 — Шаблон документації

Під час збірки системи фрагменти документації форматуються і вставляються в шаблон документації, це дозволяє показувати завжди актуальну документацію у форматі HTML під час роботи сервісу (рисунок 4.21).

Для серверу було сконфігуровано публічний доступ до документації за шляхом “/docs/index.html” за допомогою конфігурації Gradle (рисунок 4.22) та налаштувань Spring на роздачу статичних даних за допомогою вбудованого адаптера веб конфігурації “WebMvcConfigurerAdapter”.

```
19      jar {
20          baseName = "glossary"
21          version = null
22          dependsOn asciidoctor
23          from ("${asciidoctor.outputDir}/html5") {
24              into 'static/docs'
25          }
26      }
```

Рисунок 4.21 — Конфігурація Gradle для відображення документації

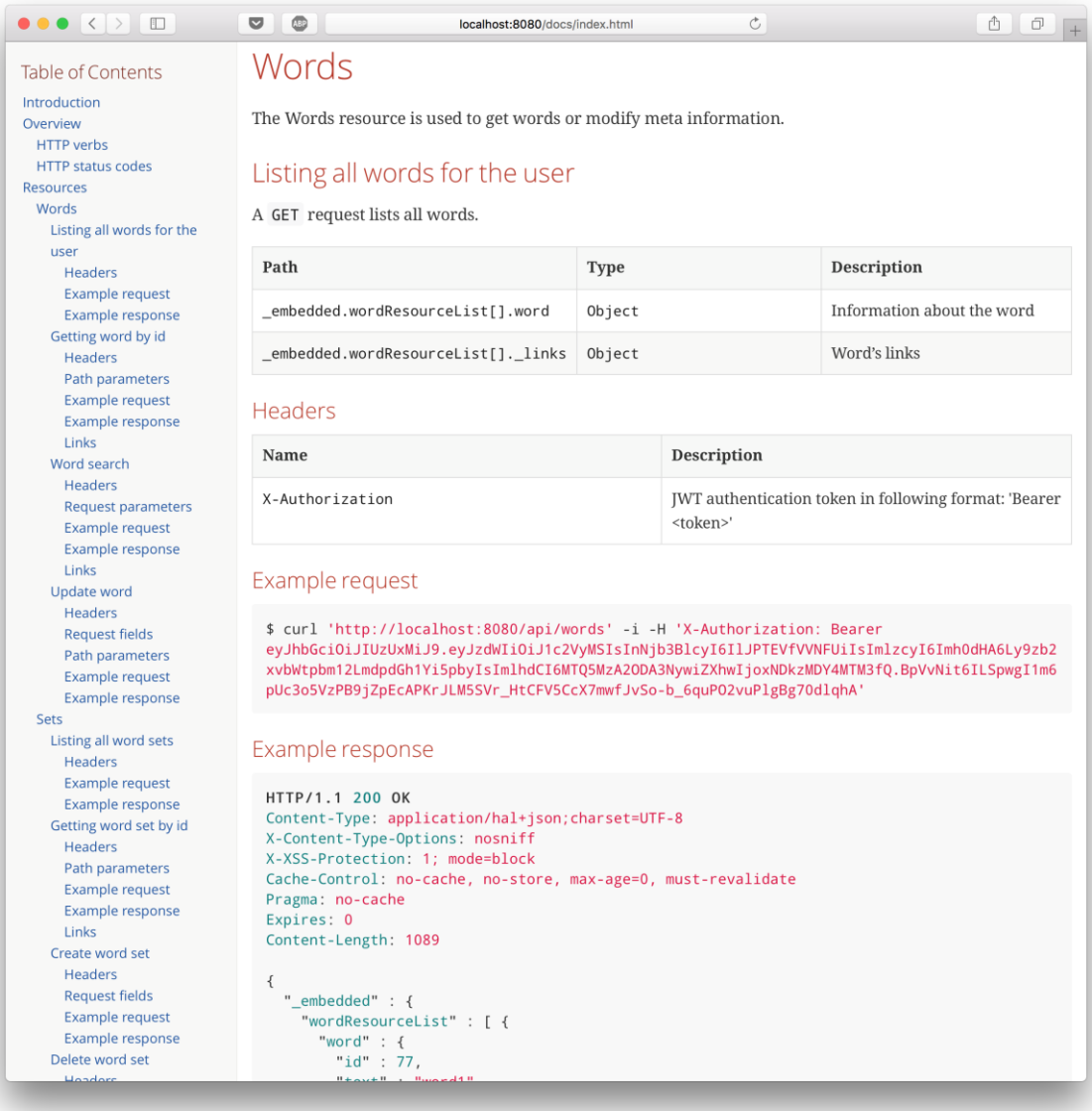

Рисунок 4.22 — Сторінка з документацією

Згенерована документація має вбудовану панель навігації, підсвітку синтаксису, автоматичне форматування JSON та підтримку мобільної версії. Окрім

того, стилі документації підтримують адаптивний дизайн, тобто відображення на пристроях з маленьким екраном. Відформатований файл з документацією може роздаватися як самим сервісом, так і бути розміщеним як звичайний HTML файл.

4.12 Генерація шаблонного коду з Lombok

У Java поширені домовленості про створення простих класів для збереження даних (JavaBeans). Всі поля класу повинні бути приватними, а для отримання значень чи їх зміни потрібно створювати методи з префіксом get чи set. Наприклад для поля “user” потрібно створити методи getUser() та setUser() [8]. Зазвичай ці методи виконують прямий доступ до поля класу, але такий підхід забезпечує необхідний рівень інкапсуляції даних. У будь який момент можна додати логіку до цих методів без необхідності переписувати код. У такого підходу є й недоліки — велика кількість шаблонного коду, який погіршує читабельність (рисунок 4.23). Для вирішення цієї проблеми була застосована бібліотека Lombok, яка генерує цей шаблонний код під час компіляції [18].

```java
public class User {
    private String name;
    private String surname;
    private String username;
    private String password;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Рисунок 4.23 — Java клас без використання Lombok

Принцип роботи Lombok досить простий. Розробник застосовує анотації, щоб позначити які елементи потрібно згенерувати. Так для створення гетерів та сетерів потрібно поставити наступні дві анотації над класом: @Getter, @Setter. Після цього замість сорока рядків коду залишається всього десять (рисунок 4.24).

```java
@Getter
@Setter
public class User {
    private String name;
    private String surname;
    private String username;
    private String password;
}
```

Рисунок 4.24 — Клас без шаблонного коду

Для збереження об’єктів у хешованих структурах даних потрібно перезаписати методи “equals()” та “hashCode()”. В більшості випадків визначення цих методів повністю шаблоне. Приблизно така ж сама ситуація і з методом “toString()”.

Достатньо лише додати анотації @EqualsAndHashCode та @ToString для генерації всіх цих шаблонних методів. Оскільки ця група анотацій дуже часто використовується, то бібліотека надає анотацію @Data, яка об'єднує всю цю групу. Загалом Lombok надає багато різних анотацій, що дозволяють генерувати найбільш поширений шаблонний код (таблиця 4.7).

Таблиця 4.7. Основні анотації Lombok

Анотація	Опис
@Data	генерація гетерів, сетерів, методів equals(), hashCode() та toString()
@Value	генерація коду для незмінюваних класів: гетери та конструктор для всіх полів
@Builder	генерація коду для створення об'єкту за шаблоном Builder
@ Slf4j	створення поля для логування

Оскільки генерація методів відбувається на етапі компіляції, то для підтримки Lombok у IntelliJ IDEA потрібно встановити спеціальний Lombok Plugin.

4.13 Веб клієнт

Веб-клієнт побудований за допомогою фреймворку Angular 6, який дозволяє розділяти класи на різні рівні абстракції: сервіси та компоненти.

Для роботи з кожним ресурсом був створений окремий сервіс, який інкапсулює у собі всі необхідні знання про частину зовнішньої системи. Так, наприклад, всі аспекти роботи зі словами були реалізовані у WordService, тоді як процедура розпізнавання мови була реалізована у окремому SpeechRecognitionService.

У Angular 6 використовується принцип Inversion Of Control, що дозволяє позбавитися від ручного створення класів і перекласти цей обов'язок на окремі файли конфігурації. Для ін'єкції залежностей потрібно виконати наступні кроки: помітити сервіс анотацією @Injectable(), додати сервіс до розділу "providers" при описі модуля та створити конструктор який приймає цей сервіс у існуючому сервісі. Після цього на

етапі запуску сервісу Angular 6 власноруч зробить ін’єкції всіх потрібних сервісів до компонентів.

Компонент відповідає за управління певної частини html шаблону. Для створення компоненту потрібно помітити будь-який клас анотацією @Component та вказати шлях до файлу-шаблону у параметрі анотації “templateUrl”. Вибір компоненту відбувається на основі поточного шляху у браузері. Налаштування шляху та необхідного компонента відбувається у окремому модулі маршрутизації (рисунок 4.25). Завдяки такому розбиттю кожен елемент сервісу має сильне внутрішнє зчеплення та низьку зв’язність з іншими компонентами. Це спрощує процес написання та підтримки існуючого коду.

```
13  const routes: Routes = [
14    {path: '', component: DictionaryComponent},
15    {path: 'set/:id', component: WordSetComponent},
16    {path: 'practice', component: PracticeComponent},
17  ...
30  ];
31
32  @NgModule({
33    imports: [
34      RouterModule.forChild(routes)
35    ], exports: [
36      RouterModule
37    ]
38  })
39  export class DictionaryRoutingModule {
40  }
```

Рисунок 4.25 — Налаштування маршрутизації та компонентів

Кожен сервіс має звертатися до відповідного ресурсу на сервері, але явне зазначення адресу сервера може призвести до великих проблем при розгортанні системи у різному середовищі. Щоб вирішити цю проблему був застосований проксі. Кожен сервіс не звертається до серверу напряму, а вказує відносний адрес до потрібного API, наприклад: “/api/images”. Таким чином запит буде направлений на той самий сервіс, звідки і був відправлений. Для того, щоб перенаправити запит на інший сервер потрібно створити файл proxy.conf.json та покласти його у корінь проекту. У цьому файлі потрібно задати умову для перенаправлення і адрес цільового серверу (рисунок 4.26). За допомогою проксі конфігурується одне місце, де вказується адрес сервера API. Це полегшує подальшу роботу з клієнтом.

```
{
  "/api": {
    "target": "http://localhost:8080",
    "secure": false,
    "changeOrigin": true,
    "logLevel": "info"
  }
}
```

Рисунок 4.26 — Налаштування проксі

Логіка веб-клієнту розбита на різні компоненти та прив'язана до різних URL шляхів. Таким чином легко розширювати функціональність веб клієнту без значних витрат часу. Окрім того, для більшої ефективності використовується лінива ініціалізація компонентів, що відносяться до вивчення слів (рисунок 4.27). Таким чином користувач що не зайшов у свій профіль не буде витрачати час на ініціалізацію недоступних для нього компонентів.

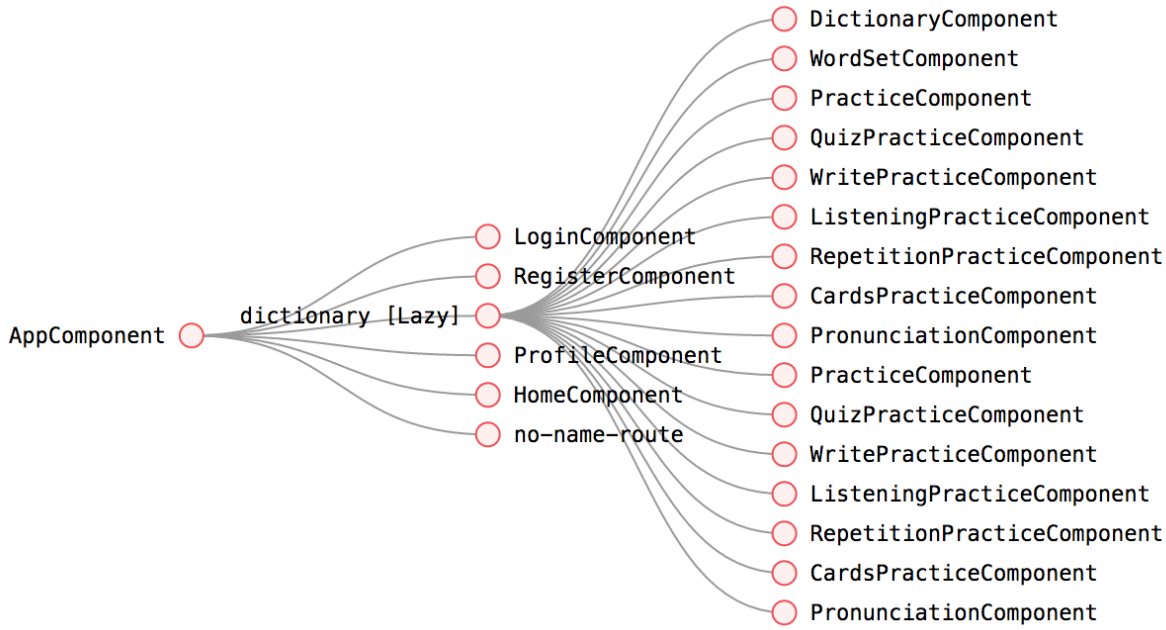

Рисунок 4.27 — Ієрархія компонентів

Такий спосіб ініціалізації компонентів можливий завдяки використанню у проекті Angular 6 та розбиттю коду на велику кількість компонентів і модулів, кожен з яких має свою частини обов'язків.

4.14 Взаємодія компонентів системи

Програмний продукт складається з різних сервісів, кожен з яких виконує окрему частину роботи (рисунок 4.28).

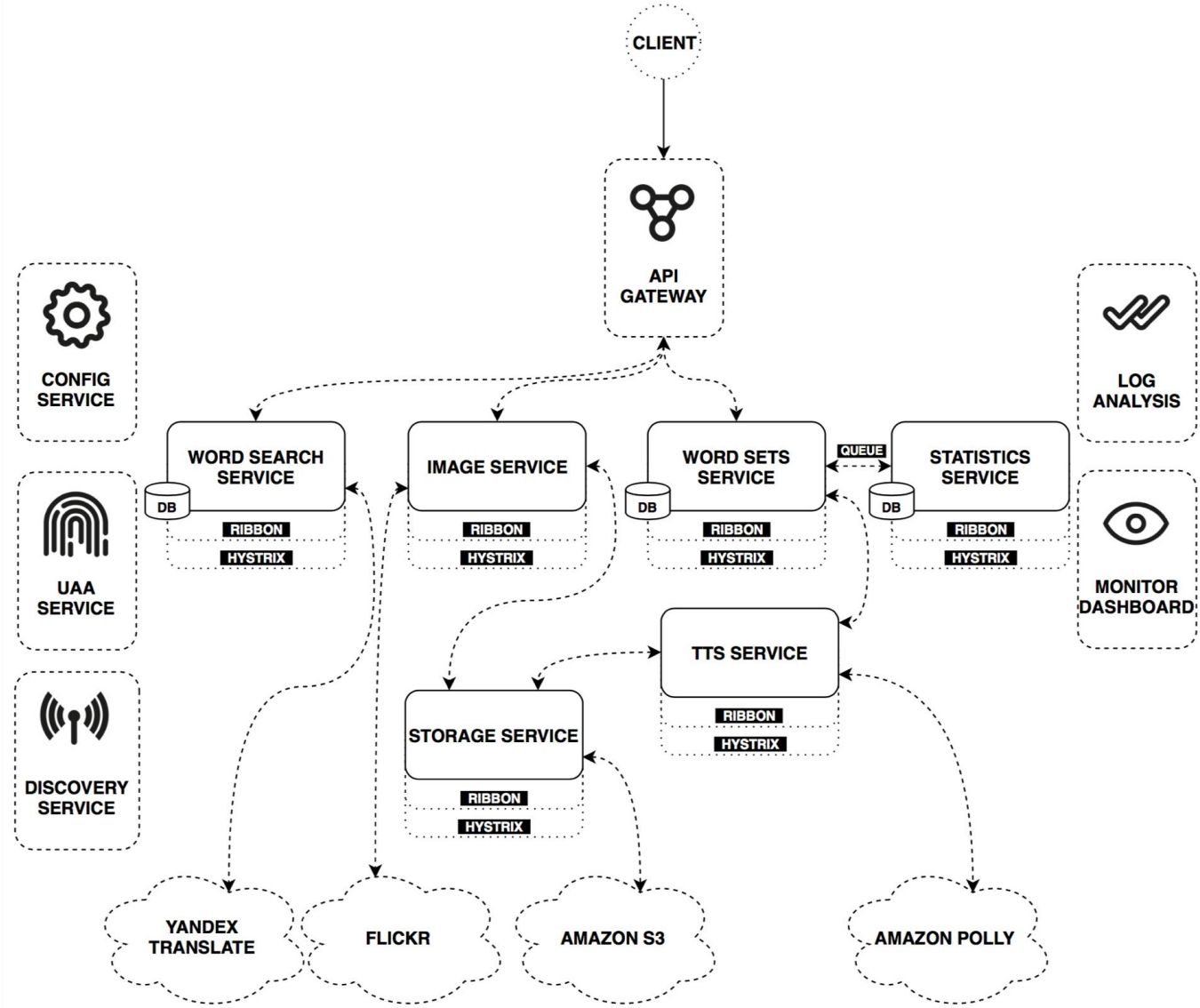

Рисунок 4.28 — Взаємодія між сервісами

Сервер взаємодіє з Amazon S3 для зберігання статичних даних. Це відбувається коли користувач загружає нове зображення або зберігає вимову нового слова. Взаємодія з YandexTranslate відбувається коли сервер вперше зберігає слово. Результат перекладу додається до списку перекладів слова. При додаванні нового слова сервер посилає запит на Amazon Polly для генерації вимови слова. Тоді як Flickr

викликається кожного разу для пошуку зображень при додаванні користувачем слова до свого набору слів.

Всі запити йдуть до сервера, тому клієнти не знають про існування зовнішніх сервісів. Якщо потрібно буде змінити сервіс перекладів чи пошуку зображень клієнту не знадобиться нічого змінювати. У випадку, коли клієнт потребує посилання на статичні ресурси, сервер може повертати прямі посилання на зовнішній ресурс, але на клієнта це ніяк не впливає.

4.15 Тренування вимови

Тренування вимови відбувається наступним чином: сервіс правильно вимовляє слово, після чого користувач натискає на кнопку для прослуховування і намагається повторити слово. Якщо слово вимовлене правильно, то сервіс переходить до нового слова. Якщо ж ні, то сервіс виводить на екран те, що розпізнав та продовжує слухати далі. У випадку, коли користувач не може правильно вимовити слово, він може натиснути на кнопку для примусового переходу до наступного слова. Сервіс розпізнає вимову користувача за допомогою Web Speech API [24], що реалізований у більшості сучасних браузерів.

4.16 Відстеження прогресу вивчення слів

Кожне слово має три етапи вивчення: не вивчене, у процесі вивчення, вивчене. Під час генерації практичних завдань сервіс пропонує для вивчення спочатку не вивчені слова, а тільки потім ті, що вже знаходяться у процесі вивчення. Після кожного практичного завдання клієнт відправляє результати відповідей на сервер. Якщо відповідь правильна, то сервер переводить етап вивчення слова на наступний рівень, у іншому випадку виставляє початковий рівень. Вивчені слова не пропонуються сервером для практичних завдань. Ці слова можна повторити лише у режимі “повторення”. Користувачу відображаються один за одним вивчені слова і користувач повинен вибрати пам’ятає він слово чи ні. Якщо він не пам’ятає, то сервер

переводить слово на початковий рівень вивчення. Такий підхід базується на системі Лейтнера [17], що розбиває інформацію на групи за рівнем вивчення. У першій групі міститься інформація що важка для вивчення або нова, тоді як у останній групі знаходиться майже вивчена інформація.

Розроблений програмний продукт реалізує всі функціональні вимоги. Для полегшення реалізації всіх вимог були застосовані наступні сторонні сервіси: Amazon S3, Amazon Polly, Yandex Translate та Flickr. Високий рівень супроводження продукту досягається завдяки застосуванню інтеграційних тестів та документуванню публічного інтерфейсу серверної частини.

Висновки до розділу 4

В системі були реалізовані підходи мікросервісної архітектури, такі як єдиний шлюз для API, повтори невдалих запитів, єдиний реєстр всіх сервісів та переривач запитів.

Взаємодія з базою даних була реалізована через Java Persistence API, що дозволяє зберігати об’єкти у реляційній базі даних. Для автентифікації та авторизації був використаний Spring Security, а сама автентифікація відбувається за допомогою передачі JWT токенів, що дозволяють перевірити справжність клієнта без додаткових звернень в базу даних.

Для розгортання системи був застосований Kubernetes, що дозволяє відстежувати кожен з екземплярів сервісів та автоматично масштабувати сервіси за потребою.

5 МЕТОДИКА РОБОТИ КОРИСТУВАЧА

У даному розділі описано системні вимоги для забезпечення стабільної та коректної роботи розробленої системи, а також детальну методику роботи користувача із програмною системою.

5.1 Системні вимоги

Система розгортається на Kubernetes-кластері.

Мінімальні вимоги сумарної кількості ресурсів для коректної роботи системи:

— 20 процесорів,

— 16 Гб оперативної пам’яті.

Для використання програмного продукту потрібна остання версія одного з сучасних браузерів: Chrome, Firefox, Opera, Safari. Користуватися сервісом можна як з повноцінних настільних операційних систем (Windows, MacOS, Linux), так і з мобільних (Android, iOS).

Мінімальні апаратні вимоги для клієнта:

— двохядерний процесор із тактовою частотою 2 ГГц,

— 2 Гб оперативної пам’яті,

— підключення до мережі Інтернет.

5.2 Взаємодія користувача с сервісом

При заході на сайт сервісу користувач бачить вікно входу у свій обліковий запис (рисунок 5.1).

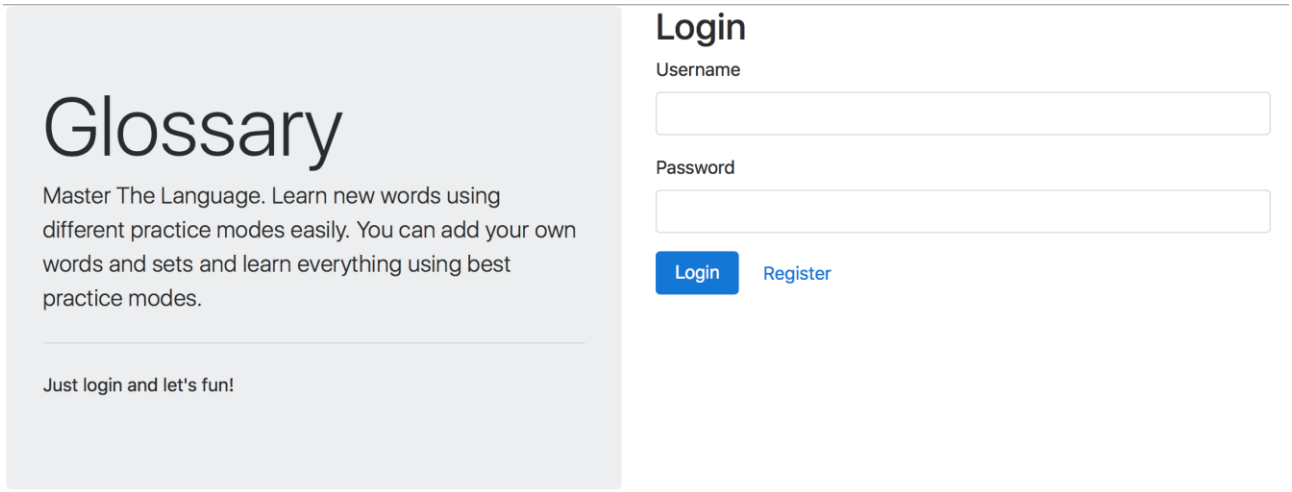

Рисунок 5.1 — Вікно входу у сервіс

Звідси користувач може перейти у вікно реєстрації (рисунок 5.2).

Рисунок 5.2 — Вікно реєстрації

При вході в систему користувач бачить коротку статистику: кількість доданих та вивчених слів (рисунок 5.3).

Користувач може переглядати список наборів слів та додавати нові (рисунок 5.4). Набори слів розташовані у вигляді сітки і динамічно переміщуються при зміні їх кількості. Також, набори можна редагувати і видаляти.

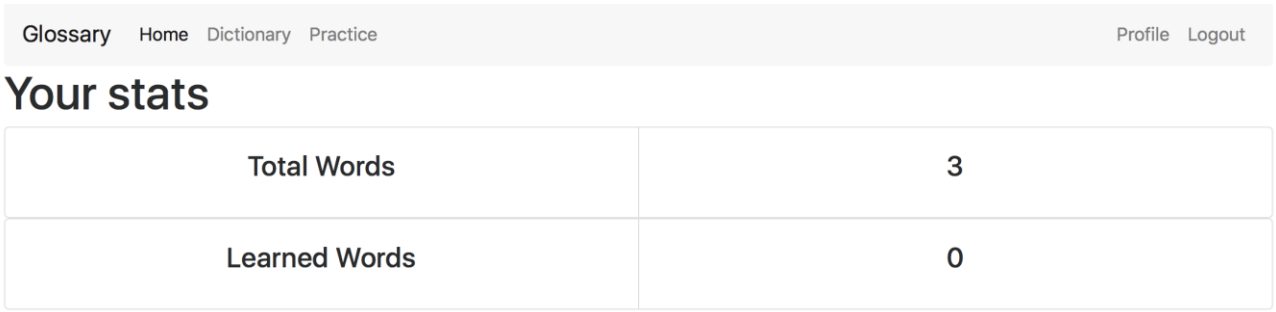

Рисунок 5.3 — Коротка статистика

У вікні перегляду слів відображаються інформація про всі слова з набору (рисунок 5.5). Поряд з кожним словом є зображення та статус вивчення. Клік по іконці звуку включить аудіо вимову слову.

У цьому ж вікні можна додавати слова. Для цього потрібно просто ввести слово на іноземній мові. Користувач може обрати один з запропонованих перекладів або ввести свій варіант вручну. Після цього користувач може обрати зображення із списку запропонованих або ж загрузити власну (рисунок 5.6).

Вибір зображень поділяється на 2 етапи: вибір з запропонованих зображень чи загрузка власних. Окрім того, є можливість додати слово без зображення. Такий спосіб погіршить процес вивчення нових слів.

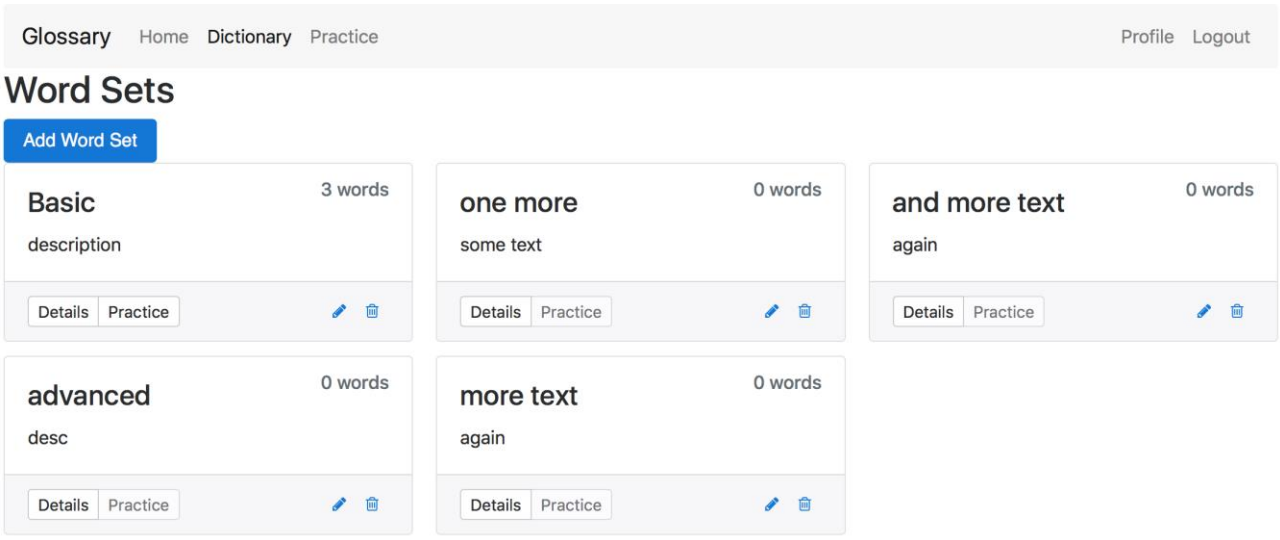

Рисунок 5.4 — Список наборів слів

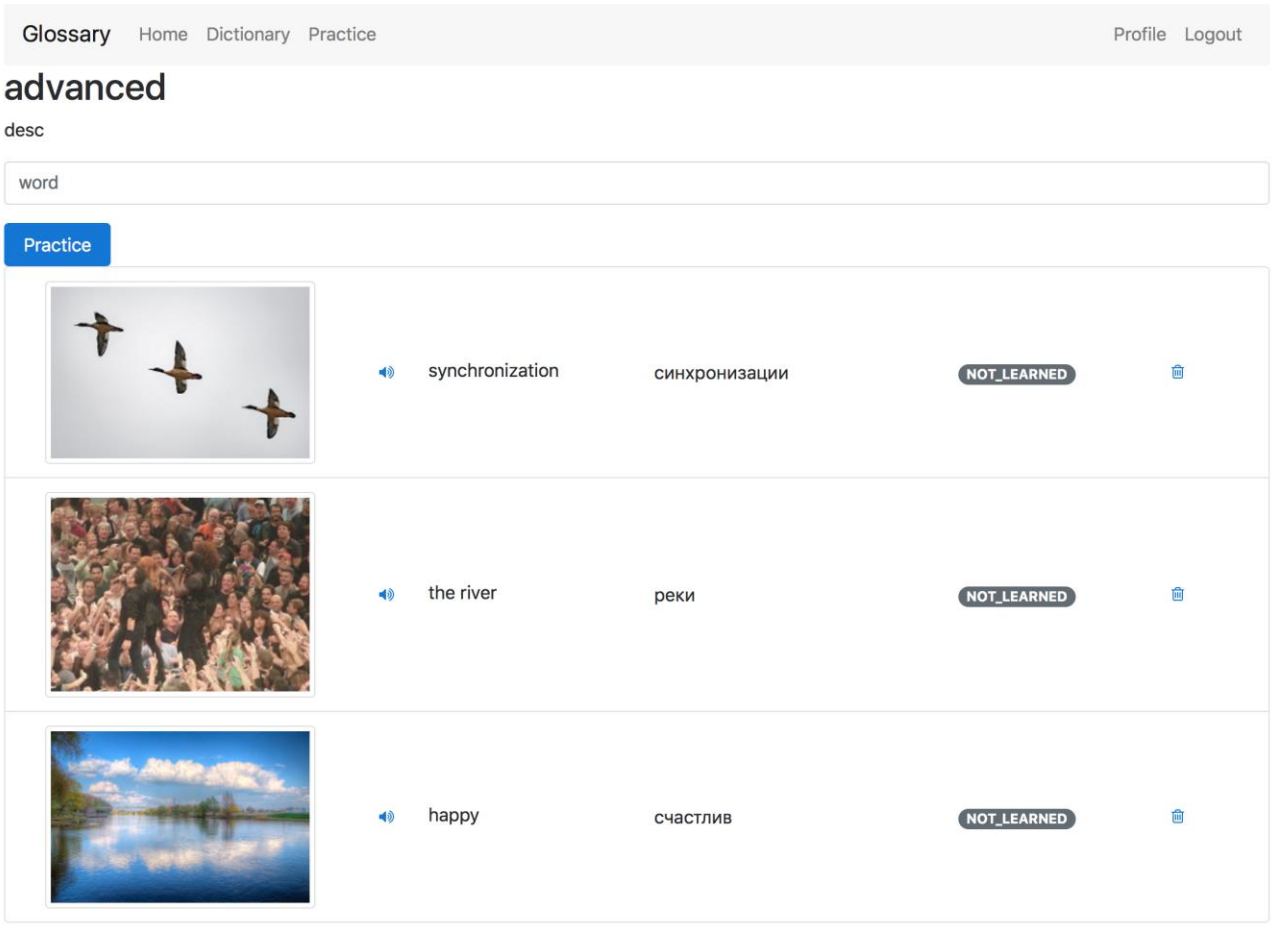

Рисунок 5.5 — Набір слів

Після додавання нових слів користувач може перейти до режимів практики (рисунок 5.7). Реалізовано 6 режимів практики:

— звичайні тести з вибором правильної відповіді;

— режим з написанням відповідей вручну;

— аудіювання;

— тренування вимови;

— картки зі словами та їх перекладом на іншій стороні;

— режим повторення, куди потраплять тільки вивчені слова.

Для деяких режимів практики є можливість вибрати напрямок тестових запитань. Тобто завдання буде вибрати правильний переклад чи навпаки, за перекладом вибрати правильне слово. Для повернення до набору слів є спеціальна кнопка. Окрім того, для на цій же сторінці можна змінити набор слів для практики.

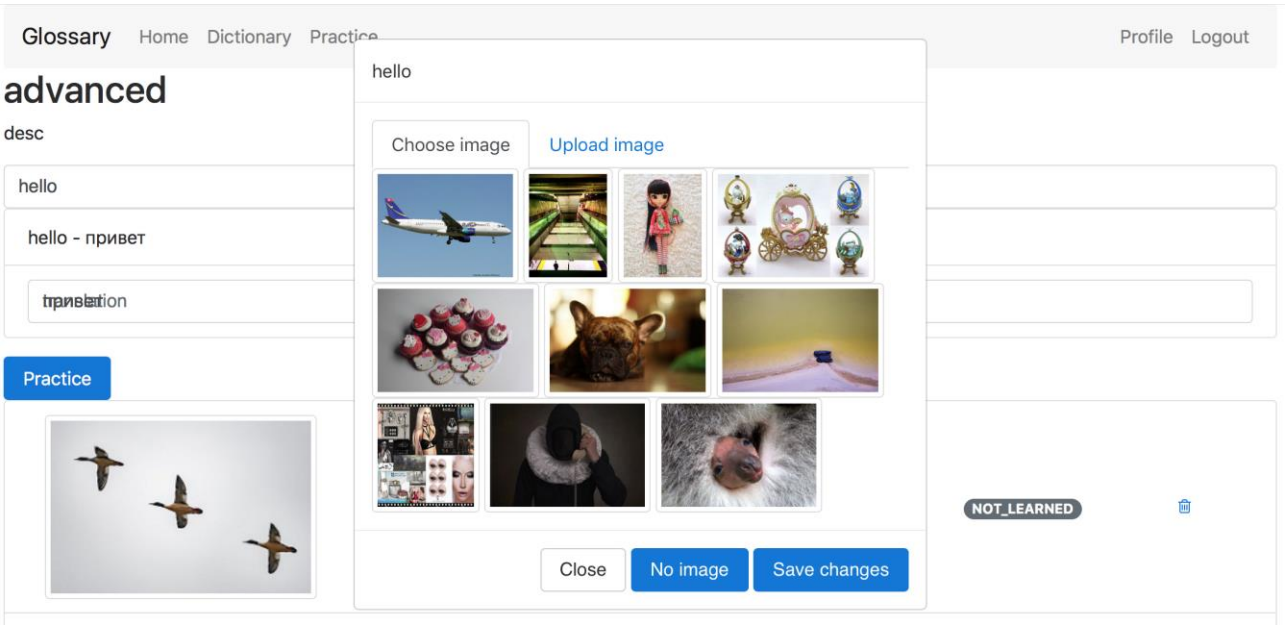

Рисунок 5.6 — Вибір зображення

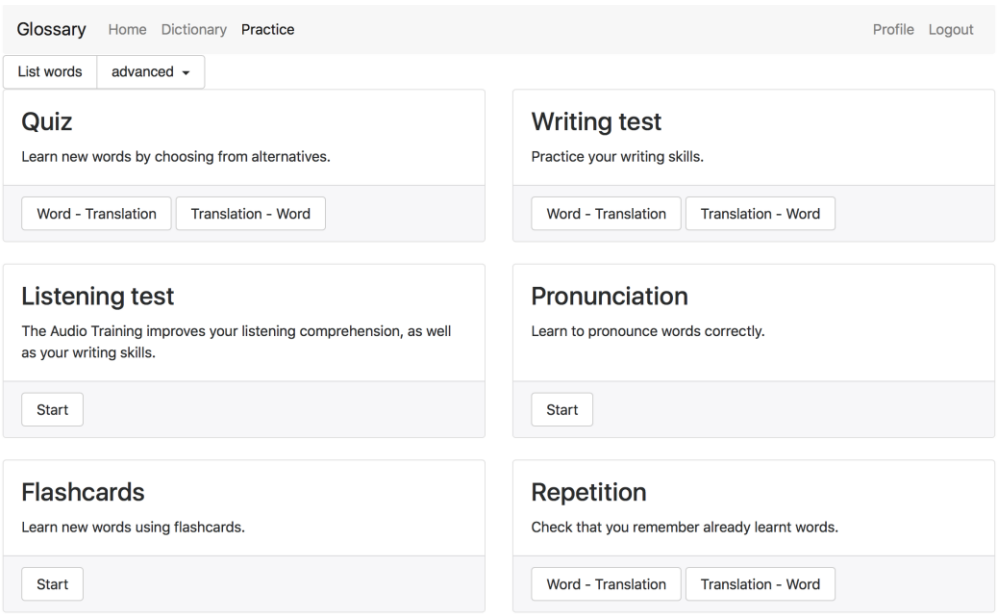

Рисунок 5.7 — Режими практики

Користувач може переглядати інформацію по своєму профілю на окремій сторінці (рисунок 5.8). Також, є окремий блок для зміни налаштувань, де можна змінити ім’я користувача, пошту та пароль (рисунок 5.9).

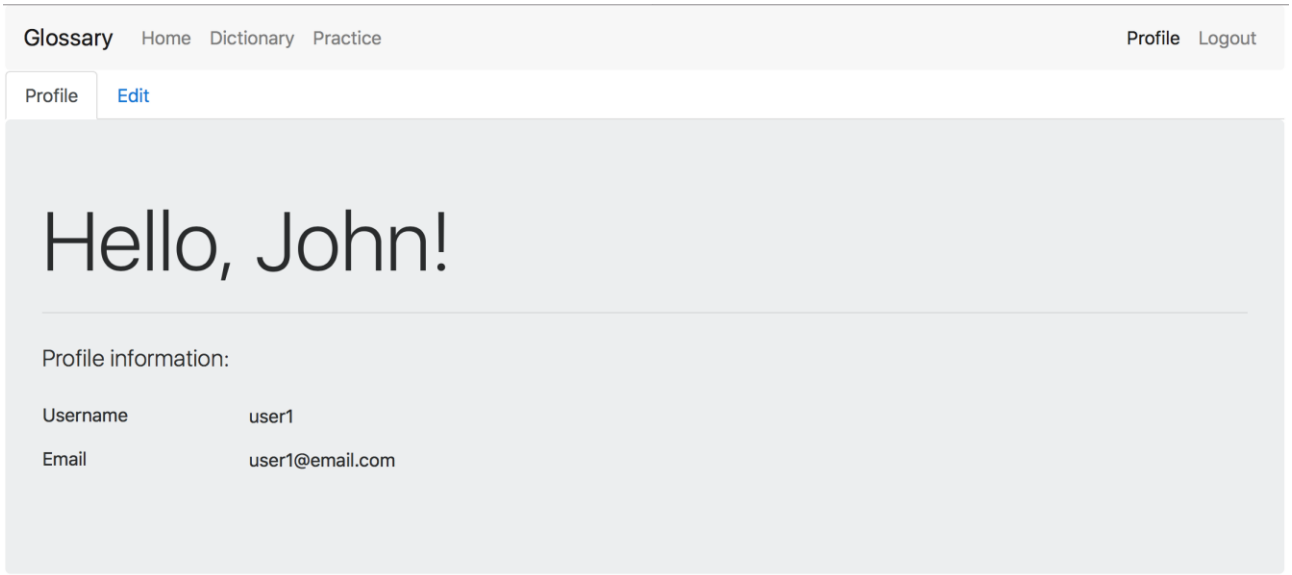

Рисунок 5.8 — Сторінка профіля

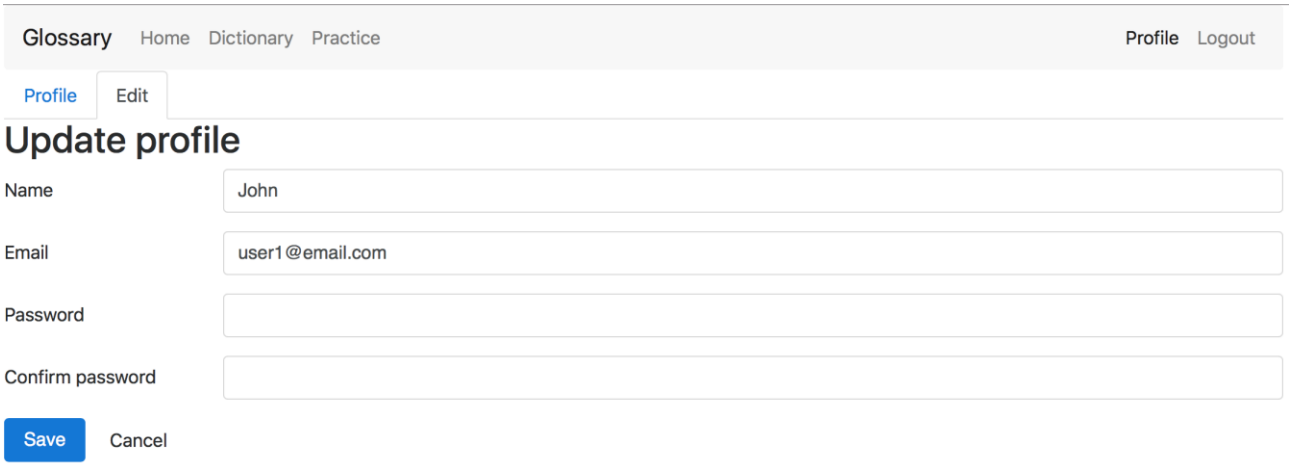

Рисунок 5.9 — Редагування профіля

Для виходу з системи потрібно натиснути на посилання “Logout” у правому верхньому кутку. Після цього вся інформація про користувача видаляється з локального сховища браузера і сервіс перенаправляє на сторінку входу в систему.

Оскільки інтерфейс веб-клієнта побудований за допомогою Bootstrap 4, то він автоматично має адаптивний дизайн для коректного відображення на мобільних пристроях.

У мобільній версії колонки розташовуються вертикально один за одним (рисунок 5.9).

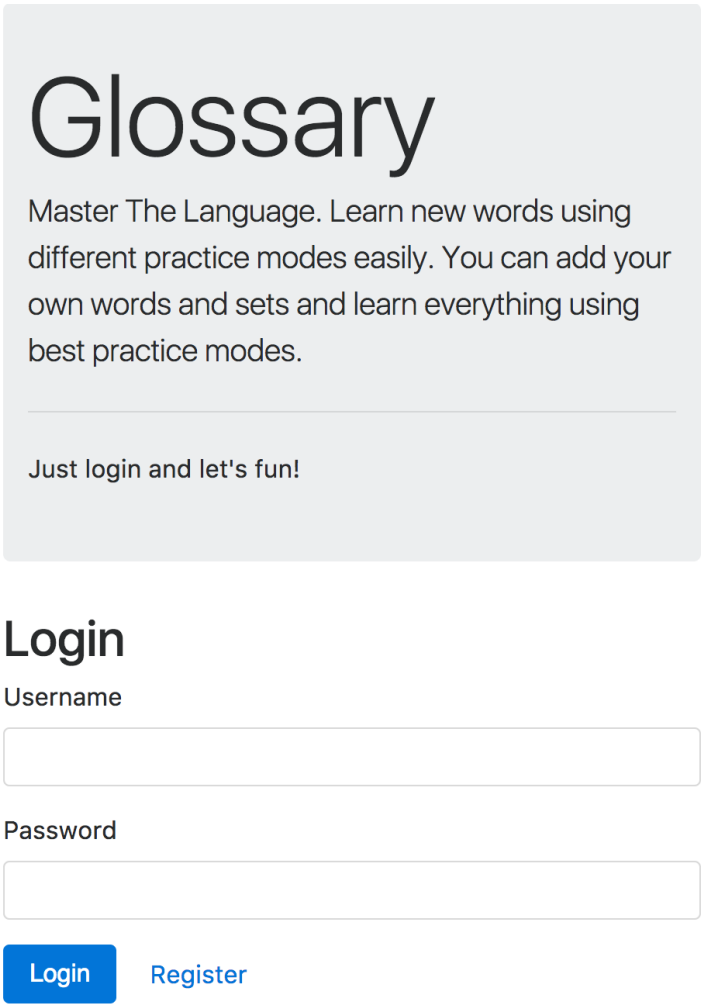

Рисунок 5.9 — Мобільна версія сторінки входу в систему

Елементи на верхній панелі навігації ховаються у випадаюче меню (рисунок 5.10, рисунок 5.11).

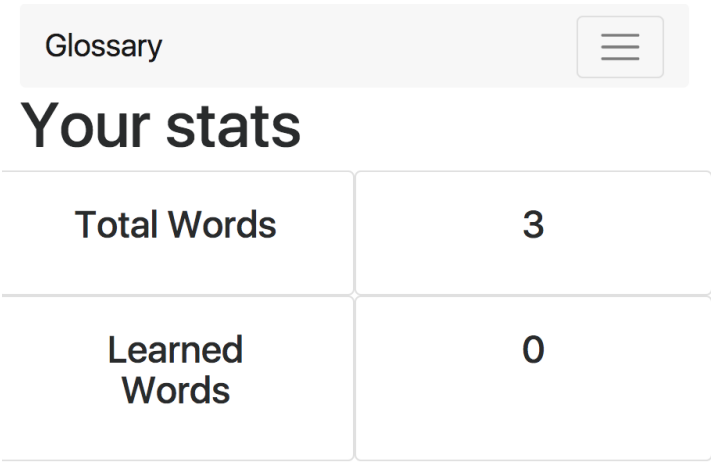

Рисунок 5.10 — Схована панель навігації

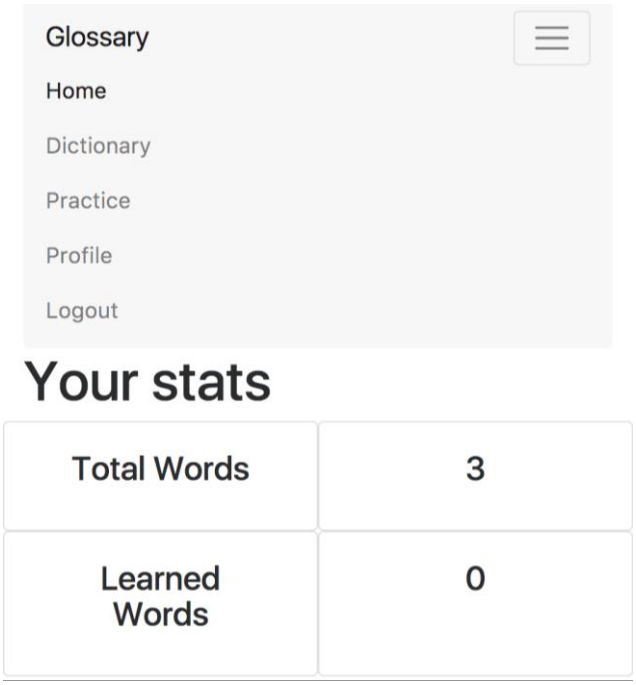

Рисунок 5.11 — Розгорнута панель навігації

У режимі перегляду наборів слів картки розташовуються один під одним (рисунок 5.12).

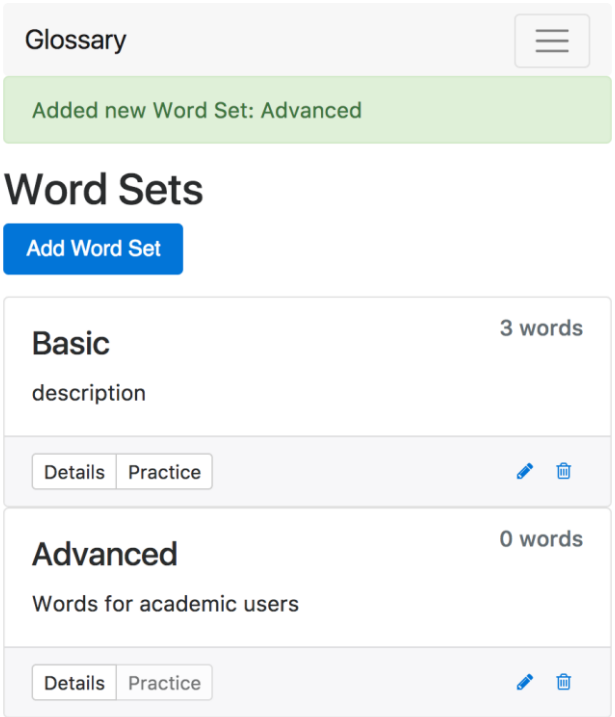

Рисунок 5.12 — Мобільна версія сторінки наборів слів

Режими практики також масштабуються для коректного відображення на мобільних пристроях (рисунок 5.13).

Рисунок 5.13 — Мобільна версія сторінки практики

Завдяки використанню HTML5 мобільна версія сайту залишається повнофункціональною і швидко працює на різних мобільних пристроях.

Таким чином користувач має зручний інтерфейс і може мати доступ як з настільних пристроїв, так і з мобільних. Використання окремих модулів дозволило реалізувати поступову загрузку компонентів за необхідністю. Це прискорило швидкість загрузки сервісу та зменшило кількість споживаного трафіку.

Висновки до розділу 5

Клієнт реалізовано у вигляді веб-застосунку, що може бути доступний з будь-якого браузеру. Застосунок має адаптивний дизайн, що дозволяє користуватися ним як з великих настільних комп’ютерів, так і з мобільних телефонів.

6 СТАРТАП ПРОЕКТ

Розділ присвячений проведенню маркетингового аналізу стартап проекту з метою визначення принципової можливості ринкового впровадження та встановлення напрямів реалізації цього впровадження. Проведення маркетингового аналізу окреслюється виконанням нижченаведених кроків.

6.1 Опис ідеї проекту

Перші три пункти подаються у вигляді таблиці (таблиця 6.1) і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки.

Таблиця 6.1. Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Сегменти споживачів	Цінність для споживачів
Система, яка дозволяє розширити словниковий запас іноземної мови	1.Підвищення словникового запасу, якості вимови. Використання сучасних технологій реалізації системи	Приватні підприємства, центри вивчення мов.	Швидкість роботи,якість результату
	2. Наявність наукового дослідження	Науково-дослідницькі лінгвістичні центри.	Можливість використаня системи у науково-дослідницькій роботі,удосконадення існуючих рішень

Аналіз техніко-економічних переваг ідеї (чим відрізняється від існуючих аналогів та замінників) порівняно із пропозиціями конкурентів передбачає:

1. Визначення переліку техніко-економічних властивостей та характеристик ідеї.

2. Визначення попереднього кола конкурентів (проектів-конкурентів) або товарів-замінників чи товарів-аналогів, що вже існують на ринку, та проводиться збір інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів відповідно до визначеного вище переліку.

3. Проводиться порівняльний аналіз показників: для власної ідеї визначаються показники, що мають:

а) гірші значення (W, слабкі);

б) аналогічні (N, нейтральні) значення;

в) кращі значення (S, сильні) (таблиця 6.2) .

Таблиця 6.2. Визначення сильних, слабких та нейтральних характеристик

№	Техніко-економічні характеристики ідеї	Продукція конкурентів		W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	LinguaLeo			
1.	Навчання по оригіналу	+	-	-	-	+
2.	Тренування: тести,вимова, аудіювання	+	-	-	-	+
3.	Самонавчання системи	+	+	-	+	-
4.	Відстеження прогресу навчання	+	-	-	+	-

6.2 Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту. Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (таблиця 6.3):

— за якою технологією буде виготовлено товар згідно ідеї проекту;

— чи існують такі технології, чи їх потрібно розробити/доробити;

— чи доступні такі технології авторам проекту;

— чи є фінансово затратним використання таких технологій;

— чи можливо замінити дані технології іншими, менш затратними;

— як вплине зміна технологій на функціональність продукту та його якість.

Таблиця 6.3. Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Використання системи Лейтнера для адаптивного вивчення слів	Мова Java	Наявна	Технологія з відкритим кодом
2	Режим тренування вимови	Web Speech API	Наявна	Технологія з відкритим кодом
3	Пошук картинок для спрощення запам’ятовування	Мова програмування Java. Flickr API	Наявна	Технологія з відкритим кодом
4	Автоматичне масштабування залежно від навантаження	Kubernetes, мова Java	Наявна	Бібліотека з відкритим кодом

Висновок: проект реалізувати можливо. Обрана технологія реалізації ідеї проекту: мова програмування Java, платформа для розгортки системи Kubernetes.

За результатами аналізу таблиці робиться висновок щодо можливості технологічної реалізації проекту: так чи ні, а також технологічного шляху, яким це доцільно зробити (з поміж названих технологій обираються такі, що доступні авторам проекту та є наявними на ринку).

6.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Спочатку проводиться аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (таблиця 6.4).

Таблиця 6.4. Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	7
2	Загальний обсяг продаж, грн/ум.од	900 грн
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	45 %

Середня норма рентабельності в галузі (або по ринку) порівнюється із банківським відсотком на вкладення. За умови, що останній є вищим, можливо, має сенс вкласти кошти в інший проект.

За результатами аналізу таблиці робиться висновок щодо того, чи є ринок привабливим для входження за попереднім оцінюванням. Цільовим ринком може бути навчальні заклади, приватні установи та окремі особи. Користувач може з легкістю вивчити нові слова завдяки декільком видам сприйняття, такі як візуальне (асоціація слова з зображенням), аудиальне (можливість послухати вимову слова), моторне (написання слова та перекладу) та вимовне (тренування правильній вимові іноземних слів).

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи.

Дані щодо всіх можливих клієнтів, їх характеристик наведені у таблиці 6.5.

Таблиця 6.5. Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Знання іноземної мови	Споживачі — організації або особам, яким необхідне вивчення іноземної мови.	Компанії заключають довготривалі договори, а стартапери віддають перевагу пробному терміну	Можливість перегляду прогресу навчання; Можлиість додати зображення до слів; Повторення вивчених слів; Можливість вивчення нових слів завдяки декільком видам сприйняття; Зручність у використанні.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (таблиці 6.6-6.7).

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку. Аналіз пропозиції необхідно виконати аналізуючи існуючі види конкуренцій.

Таблиця 6.6. Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Інтеграція під нові проекти	Потребує визначеної концептуальної моделі бази даних	Імпорт концептуальної моделі бази даних, Консультування розробників баз даних Зміна концептуальної моделі бази даних Залучення спеціаліста предметної області
2	Вразливість до несанкційованого втручання	Потребує надбудови захисту інформації	Залучення команди спеціалістів із захисту інформації Залучення команди веб-розробників
3	Домоміжний функціонал	Сервіс має визначений функціонал	Залучення команди IT-спеціалістів для додавання нового функціоналу

Таблиця 6.7 описує фактори можливостей системи, тобто наскільки розроблене програмне забезпечення є гнучким у використанні. Чи можна реалізувати можливість системи до самонавчання. Ведеться опис можливої реакції компанії на ідею розробки автоматизовану систему під різні платформи.

Таблиця 6.7. Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Незалежність від платформи	Можна використовувати різні види пристроїв	Вихід на різні ринки ІТ
2	Недоліки в існуючих альтернативах	Відсутня можливість у альтернативних сервісів тренування вимови слів.	Модифікація існуючих систем
3	Можливість до Самонавчання	Забезпечення алгоритму вивчення нових термінів	Додання нового фукнціоналу

Аналіз пропозицій зображено на таблиці 6.8.

Таблиця 6.8. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - монополія/олігополія/ монополістична/чиста	Чиста	Презентація продукту на виставках, конференціях
2. За рівнем конкурентної боротьби - локальний/національний/...	національний	Рекламування веб-сервісу міжнародних сайтах

Таблиця 6.8 (Продовження)

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
3. За галузевою ознакою - міжгалузева/ внутрішньогалузева	міжгалузева	Використовувати в різних галузях виробництва
4. Конкуренція за видами товарів - товарно-родова - товарно-видова - між бажаннями	товарно-видова	Розповідати про свої переваги перед конкурентом у цій галузі
5. За характером конкурентних переваг - цінова / нецінова	Нецінова	Надання функцій, які відсутні у конкурентів, модифікація функцій, що мають конкуренти
6. За інтенсивністю - марочна/не марочна	Марочна	Надання функцій, які відсутні у конкурентів, модифікація функцій, що мають конкуренти

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (таблиця 6.9).

На основі аналізу конкуренції, проведеного в п. 1.5 (таблиця 6.9), а також із урахуванням характеристик ідеї проекту (таблиця 6.2), вимог споживачів до товару (таблиця 6.5) та факторів маркетингового середовища (таблиця 6.6 - 6.7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлюється за таблицею 6.10.

Таблиця 6.9. Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники

		Веб-сервіси для вивчення іноземних мов	Мінімізація часу на впровадження	Оптимізація якості навчання	Лояльність клієнтів
Висновки	Визначити конкурентів, їх слабкі та сильні сторони	Можливість тренування вимови надає вагому перевагу перед конкурентами	Простота та легкість в експлуатації	Клієнти мають різний хист до навчання	Конкуренція для роботи на ринку через товари замінники

Таблиця 6.10. Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Надання можливість тренування вимови слів та можливості до самонавчання	Існуючі конкуренти або не мають можливості до самонавчання, або реалізація цього функціоналу не є конкурентоспроможною.

За визначеними факторами конкурентоспроможності (таблиця 6.10) проводиться аналіз сильних та слабких сторін стартап-проекту (таблиця 6.11)

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (таблиця 6.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (таблиця 6.11).

Таблиця 6.11. Порівняльний аналіз сильних та слабких сторін

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з Database Generator (даним продуктом)						
			-3	-2	-1	0	1	2	3
1	Надання можливість тренування вимови слів та можливості до самонавчання	20	+						

Для здійснення SWOT-аналізу на підприємстві необхідне відповідне інформаційне забезпечення, яке повинно включати: базу даних; методи та моделі, необхідні для SWOT-аналізу; набір організаційних і методичних прийомів, необхідних для підвищення надійності інформаційного забезпечення. Методика SWOT-аналізу ґрунтується на підході, який дає змогу вивчати зовнішнє і внутрішнє середовище підприємства разом. За допомогою цієї методики можна встановити взаємозв'язки між силою та слабкістю.

Таблиця 6.12. SWOT-аналіз стартап-проекту

Сильні сторони: Орієнтація на декілька мобільних платформ	**Слабкі сторони:** Відсутність нотифікацій
Можливості: Популярність мобільних платформ Потреба в урегулюванні робочих процесів на кафедрі Відсутність повноцінних альтернатив	**Загрози:** Низький рівень кастомізації Обмеженість функцій

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення.

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища (рисунок 6.13). Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення.

Таблиця 6.13. Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Орієнтація поточної моделі на ринок приватних підприємств	30 %	35 год
2	Орієнтація поточної моделі на ринок державних установ	27 %	77 год
3	Орієнтація на приватних осіб	35 %	155 год
4	Орієнтація на розробку серверної частини	65 %	115 год
5	Орієнтація на веб-розробку	45 %	97 год

Альтернатива, де отримання ресурсів є більш простим та ймовірним — №4 "Переорієнтація на розробку серверної частини", що становить 65 відсотків. Це значення перевищує інші альтернативи.

Альтернатива, де строки реалізації є більш стислими — №2 “Орієнтація поточної моделі на ринок приватних підприємств”. Терміни реалізації в цьому разі становлять 35 годин.

6.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (таблиця 6.14).

Таблиця 6.14. Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Приватні підприємствава	Готові	Високий	Висока	Просто
2	Державні установи	Потребують недовгих переговорів	Середній	Середня	Складно
3	Стартапери	Потребують довгих переговорів	Низький	Низька	Дуже складно

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку (таблиця 6.15). За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку.

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Таблиця 6.15. Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення Ринку	Ключові конкурентос-проможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
Орієнтація поточної моделі на ринок приватних підприємств	Стратегія концентрованого маркетингу	Приватні підприємтва потребують якості роботи, яку надає підтримка декількох платформ даним продуктом	Стратегія спеціалізації (спирається на диференціацію)

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту.

Наступним кроком є вибір стратегії конкурентної поведінки (таблиця 6.16).

Таблиця 6.16. Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів	Чи буде компанія копіювати основні характеристики конкурента	Стратегія конкурентної поведінки
Ні	Так	Ні	Стратегія заняття конкурентної ніші

6.5 Аналіз ринкових можливостей запуску стартап-проекту

Для цього у таблиці 6.17 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Після формування маркетингової моделі товару слід особливо відмітити — чим саме проект буде захищено від копіювання.

Захист може бути організовано за рахунок захисту ідеї товару (захист інтелектуальної власності), або ноу-хау, чи комплексне поєднання властивостей і характеристик, закладене на другому та третьому рівнях товару.

Для цього можна скористатись правом на оформлення патенту чи авторського права. Будь-яке ноу-хау, тобто те інноваційне, що розроблено, не потрібно розкривати у заявці на отримання патенту, але загальні особливості потрібно описати.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар. Проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту). Також необхідно визначити ринок та інтереси користувачів відносно кожного сезону, відповідно вікових категорій та різних регіонів країни. Проаналізувати ринок та цінову політику конкурентів, можливі витрати на підтримку рекламної кампанії для кращого попиту на послуги, шо надає програмний продукт.

Таблиця 6.17. Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
Швидкість роботи та оптимізація алгоритму	Можливість перегляду прогресу навчання	Конкуренти або не мають орієнтованості на аналіз навчання, або здійснюють неповний аналіз

Надалі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (таблиця 6.18).

М/Нм — монотонні або немонотонні;

Вр/Тх/Тл/Е/Ор — вартісні, технічні, технологічні, ергономічні або органолептичні (останній — для продуктів харчування)

Після формування маркетингової моделі товару слід особливо відмітити — чим саме проект буде захищено від копіювання.

Захист може бути організовано за рахунок захисту ідеї товару (захист інтелектуальної власності), або ноу-хау, чи комплексне поєднання властивостей і характеристик, закладене на другому та третьому рівнях товару.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (таблиця 6.18).

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (таблиця 6.19):

— проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту);

— вибір та обґрунтування оптимальної глибини каналу збуту;

— вибір та обґрунтування виду посередників;

— вибір та обгрунтування потреб користувачів в даному регіоні;

— вибір та обгрунтування цінової політики.

Таблиця 6.18. Визначення меж встановлення ціни

№	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	25…215 грн	100…700 грн	27000…52000 грн	33…108 грн

Таблиця 6.19. Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Система повинна надаватися в режимах пробної версії та повної сплатити після закінчення випробувального строку	Проста та легкість в експлуатації	Веб-сайт	Збут за допомогою посередника

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 6.21).

Маркетингова програма, як правило, спрямована на вирішення окремих комплексних проблем, наприклад на організацію виробництва нового продукту, на завоювання нового сегмента або ринку в цілому. Слід мати на увазі, що специфіка маркетингової стратегії зумовлює і специфіку програми. Розробляючи програму, необхідно насамперед враховувати ключові фактори комерційного успіху, чітко розрізняючи об'єктивні зовнішні обмеження. Перед початком нового проекту потрібно проаналізувати ринок даної області, свідомо прийняти рішення чи дійсно це необхідно.

Таблиця 6.20. Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Продаж програми через авторизовану мережу	Веб-сайти	Вивчення нових слів завдяки декільком видам сприйняття.	Довести, що програмний продукт оптимально аналізує процес навчання	Залучення рекламних компаній, робити упор на самонавчання

Висновки до розділу 6

Програмне забезпечення має переваги над існуючими конкурентами та є конкурентноздатним на ринку. Система має можливість самонавчання. Програмне забезпечення призначене для використання користувачами, що прагнуть збільшити свій словниковий запас англійської мови. Мова програмування Java надає швидкий, універсальний і зручний інтерфейс користувача. Основною аудиторією можуть бути навчальні заклади, приватні установи та окремі особи. Доволі високу її собівартість можна виправдати перевагами, які надає система у порівнянні із схожими рішеннями. Завдяки проведеній дослідно-конструкторській роботі виконано аналіз існуючих рішень, дослідження предметної області, проектування та програмування системи та запровадження нової технології: створення самонавчальної системи для вивчення слів, включаючи тренування вимови.

ВИСНОВКИ

Була описана архітектура хмарних інфраструктур, основні категорії стійкості систем та їх види метрик. Були проаналізовані основні підходи до побудови стійкої хмарної інфраструктури.

В результаті дипломної роботи була побудована відмовостійка хмарна система, що надійно працює в умовах хмарної інфраструктури та гарантує толерантність до помилок різного рівня. Розроблена система має можливості автоматичного масштабування як за метриками ресурсів, так і за метриками користувача. Мікросервісна архітектура дозволила точно налаштувати продуктивність системи залежно від навантаження на кожен з сервісів системи. Застосування мікросервісної архітектури дало простір для масштабування, гнучкості та відмовостійкості, але привнесло додаткові навантаження на сервери, ускладнило розробку та тестування.

Для оркестрації та розгортки системи був обраний Kubernetes, що дозволяє оперувати групами Docker контейнерів та надає API для реалізації поведінки автоматичного масштабування. Використання Kubernetes дозволило відстежувати кожен з екземплярів сервісів та автоматично масштабувати сервіси за потребою. Реалізація автоматичного масштабування дозволила збудувати систему, що підлаштовуються під поточне навантаження. Цей підхід не тільки підвищив стабільність системи, але й дозволив зберегти ресурси, що дозволило мінімізувати обсяг необхідних ресурсів, коли рівень навантаження мінімальний. Використання користувацьких показників покращило точність автоматичного масштабування та надало широкий спектр можливостей налаштування розподіленої системи.

Клієнт реалізовано у вигляді веб-застосунку, що може бути доступний з будь-якого браузеру. Застосунок має адаптивний дизайн, що дозволяє користуватися ним як з великих настільних комп’ютерів, так і з мобільних телефонів.

За результатами виконання інтеграційних та ручних тестів підтверджена коректність отриманих результатів, отже система відповідає поставленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Блох Д. Java. Эффективное программирование / Джошуа Блох — М. Лори, 2014. — 310 с.
2. Джиджи С. Полное руководство Kubernetes, 2е издание / Джиджи Сэйфан. — М: Пакт, 2018. — 879 с.
3. Моуэт Э. Использование Docker / Эдриен Моуэт — М. ДМК-Пресс, 2017. — 354 с.
4. Сиерра К., Бейтс Б. Изучаем Java / Кэти Сиерра, Берт Бейтс — М: Эксмо, 2016. — 720 с.
5. Хорстманн К. Java SE8. Вводный курс / Кей Хорстманн. — М: Вильямс, 2014. — 208 с.
6. Шефер К., Хо К., Харроп Р. Spring 4 для профессионалов / Крис Шефер, Кларенс Хо, Роб Харроп — М. Вильямс, 2016. — 752 с.
7. Шилдт Г. SWING: руководство для начинающих / Герберт Шилдт. — М: Вильямс, 2007. — 704 с.
8. Эккель Б. Философия Java. Полное издание: Java / Брюс Эккель. — СПб: Питер, 2003. — 976 с.
9. Amazon Web Services [Електронний ресурс] — Режим доступу: https://aws.amazon.com
10. Angular 6 [Електронний ресурс] — Режим доступу: https://angular.io
11. Bootstrap 4 [Електронний ресурс] — Режим доступу: https://getbootstrap.com
12. Docker [Електронний ресурс] — Режим доступу: https://docker.com
13. Gradle Build Tool [Електронний ресурс] — Режим доступу: https://gradle.org
14. H2 Database Engine [Електронний ресурс] — Режим доступу: http://www.h2database.com/html/main.html
15. IntelliJ IDEA the Java IDE [Електронний ресурс] — Режим доступу: https://www.jetbrains.com/idea/
16. JWT [Електронний ресурс] — Режим доступу: https://jwt.io

17. Leitner System [Електронний ресурс] — Режим доступу: http://leitnerportal.com/LearnMore.aspx
18. Project Lombok [Електронний ресурс] — Режим доступу: https://projectlombok.org
19. REST API [Електронний ресурс] — Режим доступу: http://www.restapitutorial.com
20. Spring Framework [Електронний ресурс] — Режим доступу: https://spring.io
21. Spring REST Docs [Електронний ресурс] — Режим доступу: https://projects.spring.io/spring-restdocs
22. Spring Security Architecture [Електронний ресурс] — Режим доступу: https://spring.io/guides/topicals/spring-security-architecture/
23. TypeScript: JavaScript that scales [Електронний ресурс] — Режим доступу: https://www.typescriptlang.org
24. Web Speech API [Електронний ресурс] — Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

ДОДАТОК А

Застосування мікросервісної архітектури для побудови відмовостійкої хмарної системи

Апробації

УКР.НТУУ “КПІ”.ТВ3267_18М

Аркушів 11

2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVI Міжнародної науково-практичної конференції аспірантів, магістрантів і студентів м. Київ, 24-27 квітня 2018 року,

ТОМ 2

Київ- 2018

УДК 004.422.833

Магістрант 5 курсу, гр. ТВ-71мп Соломкін М.В.
Доц., к.т.н. Смаковський Д.С.

ШАБЛОНИ ПРОЕКТУВАННЯ ДЛЯ ПОБУДОВИ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Деякі сучасні програмні системи досягли такої складності, що жодна людина у світі не може повністю зрозуміти усі аспекти їх роботи. Тому, розробникам довелося застосовувати нові методології розробки систем, щоб полегшити розробку, тестування та розгортку систем на серверах. Мікросервісна архітектура – новий тип архітектури для побудови великих систем, який полягає в тому, що система будується з декількох маленьких сервісів, кожен з яких добре виконує свою частину роботи [1]. Розподілені сервіси забезпечують можливість масштабування незалежно одне від одного. Система може бути розгорнута зі збільшенням кількості екземплярів тих сервісів, що потребують більшої обчислювальної потужності. Це називається горизонтальним масштабуванням.

З урахуванням можливості горизонтального масштабування можемо ввести шаблон балансування навантаження на стороні клієнта (Client Side Load Balancing). Коли клієнт робить запити до якогось сервісу, він може вибрати будь-який екземпляр із списку доступних екземплярів сервісу. Це дозволяє розподілити навантаження зі сторони клієнтів по всім екземплярам сервісу рівномірно. Однак у такому випадку існує проблема отримання інформації про наявні екземпляри інших сервісів. Для цього можуть бути використані файли конфігурації, але це додає ще більшу складність, оскільки потрібно оновлювати конфігурацію після кожної зміни топології. Використання шаблону пошуку сервісів (Service Discovery) вирішує проблему. Цей шаблон представляє собою окремий сервіс, який відіграє роль реєстру для інших сервісів. Коли сервіс запускається, він повідомляє свою адресу (наприклад, хост і порт) до реєстру та надсилає повідомлення про свій статус з певною періодичністю поки він є робочим. Отже, реєстр знає, які сервіси знаходяться у робочому стані. Таким чином, можливо масштабувати сервіси під час роботи системи, динамічно додаючи або зменшуючи кількість екземплярів. Коли клієнт хоче відправити запит до сервісу, єдине, що він повинен знати – це ім'я, яке використовується для запису сервісу у реєстрі. Таким чином, клієнт робить запит до реєстру, щоб отримати список доступних екземплярів сервісів, а потім робить прямий запит до обраного екземпляра сервісу. Ця архітектура вимагає від клієнтів знання про місцезнаходження реєстру та інтеграцію з ним. Однак зовнішні клієнти не повинні нічого знати про внутрішню архітектуру системи. У мікросервісній архітектурі застосовується шаблон API шлюзу (API Gateway), який є єдиною точкою доступу до систему для зовнішніх систем. Застосування шлюзу дозволяє приховати внутрішню складність системи для зовнішніх клієнтів. Зовнішні клієнти виконують всі запити до шлюзів, а шлюзи в свою чергу звертаються до реєстрів, щоб отримати список доступних сервісів, а потім вже звертаються потрібних сервісів застосовуючи балансування навантаження.

Це основні шаблони проектування, без яких не існує жодної реальної мікросервісної системи. Але існує ще безліч різних шаблонів, які націлені на покращення відмовостійкості системи та полегшення її конфігурації. Застосування мікросервісної архітектури дає великий простір для масштабування та гнучкості, але привносить додаткові навантаження на сервери та мережу, ускладнює розробку та тестування. Тому перед застосуванням мікросервісної архітектери потрібно уважно оцінити всі плюси і мінуси для конкретно взятою системи.

Перелік посилань:

1. Microservices definition [Електронний ресурс]. – Режим доступу: martinfowler.com/articles/microservices.html – Microservices – (Дата звернення – 03.03.2018).

забезпечення на наявність деструктивних властивостей. 91
ПРИЖКОВ А.О., магістрант гр. ТР-71мп
Керівник - доц., к.т.н. Смаковський Д.С.
Безпека комп'ютерних мереж з динамічною адресацію за протоколом IP. 92
СОЛОМКІН М.В., магістрант гр. ТВ-71мп
Керівник - доц., к.т.н. Смаковський Д.С.
Комп'ютерна безпека в комплексі моделювання гідроакустичних процесів. 93
ТОБІЛКО А.О., магістрант гр. ТВ-71мп
Керівник - доц., к.т.н. Кублій Л.І.
Імітаційне моделювання геометричних об'єктів. 94
ШПИКУЛЯК О.О., магістрант гр. ТР-71мп
Керівник - доц., к.т.н. Сидоренко Ю.В.
Система оцінювання електромагнітного навантаження на довкілля з боку кабельних ліній електропередачі. 95
Ярута О. О., магістрант гр. ТІ-71мп
Керівник - доц., к.е.н.Левченко Л.О.
Створення на основі ГІС карти рухомих наземних об'єктів що ідентифікуються за допомогою акустичних сенсорів. 96
АНДРОЩУК С.А., студент гр. ТІ-41
Керівник - асист. Швайко В.Г.
Розробка порталу обміну інформацією кафедри на базі ASP.NET та ANGULAR. 97
ВОЙТОВИЧ С.В., студент гр. ТР-41
Керівник - ст.викл. Гурін А.Л.
Моделювання дійсної мінімальної поверхні на основі ізотропної кривої Без'є та квазіконформної заміни параметру. 98
ВРАДІЙ Д.В., студент гр. ТР-42
Керівник - ст.викл. Гурін А.Л.
Оцінка ризику для населення та інфрастуктурних об'єктів від паводкових ситуацій методами ГІС аналізу. 99
ГАЛУШКО А.В., студент гр. ТВ-42
Керівник - асист. Швайко В.Г.
Удосконалення методу комбінаторно-морфологічного аналізу та синтезу раціональних систем. 100
ГРИКУН П.І., студент гр. ТВ-42
Керівник - проф., д.т.н. Адасовський Б.І.
Геометричне моделювання кривих і плоских сіток на основі ізотропних параметрів. 101
ДОРОЩУК Д.В., студент гр. ТР-42
Керівник - проф., д.т.н. Аушева Н.М.
Моделювання складних мозаїчних розміщень. 102
КОВЕЦЬКИЙ М.М., студент гр. ТР-42
Керівник - проф., д.т.н. Аушева Н.М.
Система автоматизації та управління будівлею з допомогою мікроконтролерів ATmega AVR. 103
КОНКІНА Н.С., студентка гр ЗПІ-ЗП-53
Керівник - ст.викл., к.т.н. Шалденко О.В.
Технології взаємодії застосунків MATLAB і C#. 104
КОСТЕНКО О.П., студент гр. ТВ-41
Керівник - доц., к.т.н. Кублій Л.І.

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»
Теплоенергетичний факультет

Київська державна академія водного транспорту
імені П.Конашевича-Сагайдачного

Інститут кібернетики ім.В.М.Глушкова НАН

V науково-практична дистанційна конференція
молодих вчених і фахівців
з розробки програмного забезпечення

«СУЧАСНІ АСПЕКТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

15 травня 2018

м. Київ

Сегеда І.В., Зіліцький В.Є. Застосування QR-кодування в Україні..... 145

Сич М. В. Контроль енергоефективності систем вентилювання: аналіз, проблеми та альтернативні рішення..... 151

Смаковський Д.С., Соломкін М.В. Підходи до побудови відмовостійких розподілених систем..... 162

Тарнавський Ю.А., Горб І. Роль аналітичних систем в організації оперативного енергетичного менеджменту..... 168

Тарнавський Ю.А., Крижанівська Ю.В. Моделювання режимів роботи інтегрованих систем енергозабезпечення..... 171

Тарнавський Ю.А., Смоліженко Д.П, Михайлюк А.В. Захист веб-систем на основі комп'ютерних тестів..... 173

Титенко С. В., Астахов А.Г. Онтологічно-орієнтована навчальна система для вивчення дисциплін історичного спрямування..... 180

Титенко С.В., Білоус А.О. Інтерактивний доступ до професійно-навчальної інформації..... 195

Шалденко О.В., Байда Д.В. Нейронні мережі як засіб для семантичної сегментації зображення..... 201

Шаповалова С.І., Бараніченко О.М. Встановлення зв'язків між об'єктами логічної задачі..... 204

Шаповалова С.І., Колот С.С. Системи моделювання Convolutional Neural Networks..... 209

Шаповалова С.І., Терпіль Д.О. Вдосконалення графічної симуляції навколишнього середовища..... 214

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КПІ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ЛІНГВІСТИКИ

Наука XXI століття: виклики, пріоритети, перспективи досліджень

Матеріали II Всеукраїнської студентської науково-практичної конференції з міжнародною участю

ТОМ II

22 березня 2018 р.

Київ 2018

simulations of the immune system. One can not but mention the virtual liver, which reproduces the physiology of the human liver, morphology, and its functions. Moreover, computer simulation allowed to create models of an infectious disease. That is why it is now possible to simulate the progress of most infectious diseases, to mathematically identify the likely outcomes of the epidemic or to help manage them through vaccination.

As a conclusion, we can say that the field of computer modeling has a wide impact on the development of modern biotechnology, humanity is seizing new opportunities. Precise diagnosis, microsurgery and the treatment of various diseases without surgery, artificial organs that function as genuine nanorobots — all these technologies provide the basis and foundation for further research and possibly the invention of new vaccines, biomedes and possibly even medicines for rare diseases. The next evolutionary stage predicts the emergence of trained models of physiological processes.

LITERATURE

1. Е.Я. Швець, О.О. Кісарін. Комп'ютерне моделювання фізіологічних систем людини. Навчально-методичний посібник. – Запоріжжя, 2009. – С. 5–6.

Максим Владиславович СОЛОМКІН
магістр (гр. ТВ-71мп)
теплоенергетичного факультету,
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

SERVICE FOR EXPANDING ENGLISH VOCABULARY
СЕРВІС ДЛЯ РОЗШИРЕННЯ СЛОВНИКОВОГО ЗАПАСУ АНГЛІЙСЬКОЇ МОВИ

The modern world is experiencing a very active internationalization. Therefore, there is a need of learning foreign languages to be able to communicate with others around the world. Now, most people need to learn English only in order to be understood in most countries, but the problem of learning a foreign language remains. For knowledge of grammar, you only need to learn the rules, but the study of new words takes large periods of time and requires the constant repetition of these words.

89	ПРОКОПОВА Ганна Дмитрівна	COMPARATIVE ANALYSIS AND INVESTIGATION OF DYPLOID GENETIC ALGORITHMS	Чіжова Н.В.	Чіжова Н.В.	219
90	ПРОХНІЦЬКИЙ Владислав Андрійович, ДАШКОВ Тарас Андрійович	VR TECHNOLOGIES	Антоненко І.І.	Антоненко І.І.	222
91	РІЗНИК Вадим Олександрович	PROTECTION TECHNOLOGIES IN SYSTEMS OF INTERNET BANKING	Галицька І.Є.	Медкова О.В.	224
92	РОМАНЧУК Сергій Анатолійович	MATERIAL OF THE FUTURE: GRAPHENE	Бондаренко О.І.	Бондаренко О.І.	227
93	РОМАНЮК Катерина Русланівна	COMPARATIVE CHARACTERISTICS OF CLOUD SERVICES	Полягушко Л.Г.	Мойсеєнко С.М.	229
94	РУДЕНКО Катерина Василівна	ALTERNATIVE ENERGY SOURCES IN THE 21ST SCENTURY	Кравченко Т.В.	Кравченко Т.В.	213
95	САЗОНОВА Катерина Максимівна	THE APPLICATION OF DISTRIBUTED COMPUTING IN VARIOUS FIELDS OF SCIENCE USING THE EXAMPLE OF THE BERKELEY OPEN INFRASTRUCTURE	Волкова С.Г.	Волкова С.Г.	223
96	СЕРВАТМАНД Марьям	MAIN PROBLEMS OF SCIENCE DEVELOPMENT	Чіжова Н.В.	Чіжова Н.В.	235
97	СКАЛЕЦЬКА Лілія Олегівна	MODERN METHODS OF MATHEMATICAL ANALYSIS AND COMPUTER MODULATIONS IN BIOMEDICAL ENGINEERING	Чіжова Н.В.	Чіжова Н.В.	238
98	СОЛОМКІН Максим Владиславович	SERVICE FOR EXPANDING ENGLISH VOCABULARY	Смаковський Д.С.	Мойсеєнко С.М.	240
99	СОЛЯРИК Антон Сергійович, БАБЕНКО Віталій Олегович	3D PRINTERS	Носовець О.К.	Борковська І.П.	243
100	СОСЄДСЬКА Марина Андріївна	TRENDS IN THE DEVELOPMENT OF QUANTUM CRYPTOGRAPHY	Сторожук А.Ю.	Жицька С.А.	245
101	СТАРОВОЙТЕНКО Олексій Володимирович	THE ROLE OF REDIS IN SOFTWARE DEVELOPMENT	Крилов Є.В.	Соколова Л.Ф.	248
102	СТЕЦЕНКО Наталія Ярославівна	RNA LÄßT SICH DURCH CRISPR-CAS9 ZERSCHNEIDEN	Дабагян І.М.	Дабагян І.М.	249
103	СТРАЙГОРОДСЬКА Ліна Олегівна	SINGLE ATOM CATALYSTS: STRUCTURE AND PROPERTIES	Бутова К.Д.	Моренцова А.В.	251
104	СУЧЕК Дмитро Сергійович	UKRAINE IS A COSMIC STATE	Козьміна Н.А.	Козьміна Н.А.	254
105	ТАТАРІН Валерій Вячеславович	ETHICAL PROBLEMS OF TECHNICAL SCIENCES OF THE XXI CENTURY	Бондаренко О.І.	Бондаренко О.І.	256
106	ТЕРЕЩЕНКО Катерина Миколаївна	IOTA IS FINANCIAL INSTRUMENT FOR FUTURE	Дробязко Ю.І.	Дробязко Ю.І.	259
107	ТЕСЛЯ Сергій Юрійович	FORMING OF THE STRUCTURE WEAR RESISTANT COATINGS: SELF-FLUX ALLOYS – CARBIDES OF TRANSITION METALS	Степанчук А.М.	Нікітіна Н.С.	262
108	ТИМЧЕНКО Анастасія Володимирівна	MATHEMATISCHE MODELLIERUNG DER OBERFLÄCHENEFFEKTE VON METALLOXIDEN	Донцова Т.А.	Гуменюк З.В.	264
109	ТІХОНИЧЕВ Роман Миколайович	METHODS OF PROTECTION OF INFORMATION RESOURCES	Стьопочкіна І.М.	Медкова О.В.	266
110	ТКАЧЕНКО Марія Олексіївна	ECOTOXICOLOGICAL SUBSTANTIATION OF THE USE OF A NEW TYPE OF FERTILIZER SULFATE-HUMATE-AMMONIUM IN CONDITIONS OF THE PODZOLIC SOILS OF POLISSYA REGION	Риженко Н.О.	Кравченко Т.В.	268
111	ТРІЩ Віталій Русланович	COMPUTER MODELING OF WATER OZONIZATION PROCESS	Безносик Ю.О.	Шніп Ю.В.	272
112	ТРУХАНОВА Анастасія Андріївна	BENEFITS AND PECULIARITIES OF RENEWABLE ENERGY USE	Головко В.М.	Ахмад І.М.	273

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Національний університет “Києво-Могилянська академія”
Вища економіко-гуманітарна школа (Польща)

СТАЛИЙ РОЗВИТОК — XXI СТОЛІТТЯ: УПРАВЛІННЯ, ТЕХНОЛОГІЇ, МОДЕЛІ

Дискусії 2018

Колективна монографія

Київ, Україна
2018

года (*Махнитко А.Е., Баркан В.И., Варфоломеева Р.В., Веремийчук Ю.А.*)
3.3. Sustainable Development of the Ukrainian Energy Critical Infrastructure: Overview of Risk Assessment Techniques (*Karaieva N.V.*) — Сталий розвиток критичної інфраструктури енергетики України: огляд методів оцінки ризику (*Караєва Н.В.*) 290
3.4. Формування факторного простору впливу на режим електроспоживання промислових підприємств (*Розен В.П., Великий С.С., Сторожилова Г.І.*) 296
3.5. Управління енергозабезпеченням підприємства в системі економічної безпеки з метою підтримання сталого розвитку (*Дергачова В.В., Колешня Я.О.*) 305
3.6. Стратегічні пріоритети та економічні стимули розвитку “зеленої енергетики” в країнах світу (*Біла С.О.*) 312
3.7. Стратегічні напрямки розвитку відновлюваної енергетики в Україні (*Рязанова Н.О.*) 323
3.8. Інформаційні технології в енергетиці на сучасному етапі сталого розвитку суспільства (*Антонов В.М.*) 331
3.9. Енергетична безпека — визначення, або стара пісня про головне (*Бараннік В.О.*) 339
3.10. Транскордонний перенос забруднень атмосферного повітря енергетичних об’єктів у контексті сталого розвитку суміжних з Україною країн (*Варламов Г.Б., Коваленко Г.Д., Вітько В.І.*) 346
3.11. Устойчивое развитие в гармонии с природой: аксиомы и принципы новой энерго-экологической парадигмы (*Варламов Г.Б., Ши Цзе*) 361

Розділ 4. ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В СИСТЕМІ ЗАБЕЗПЕЧЕННЯ СТАЛОГО РОЗВИТКУ **369**
4.1. Інформаційні системи та технології забезпечення стійкого розвитку міст і агломерацій (*Аверкина М.Ф.*) 369
4.2. Інноваційні технології в системі сталого розвитку: гендерно-статевий та екологічно-етичний аспекти (*Антонов В.М., Новіцька І.В.*) 373
4.3. Інформаційні технології при визначенні збитків здоров’ю від забруднення навколишнього природного середовища (*Шевченко І.В.*) 379
4.4. Попередня оцінка екологічної шкоди від звалищ побутових відходів в Україні з використанням ГІС-аналізу (*Рогожин О.Г.*) 384
4.5. Розробка програмного забезпечення для взаємодії PDM- і CAD-систем (*Орел Д.С.*) 390
4.6. Варіанти модифікації політочкових перетворень (*Сидоренко Ю.В.*) 394
4.7. Система побудови діаграми “краватка-метелик” — інструмент оцінки ризиків сталого розвитку енергетики (*Караєва Н.В., Кондратенко І.Л.*) 398
4.8. Засоби ідентифікації морських об’єктів на основі гідроакустичних портретів (*Пуха С.П., Гайдаржи В.І., Бичков І.О.*) 409
4.9. Горизонтальне автоматичне масштабування сервісів на основі довжини черги у Kubernetes кластері (*Соломкін М.В., Смаковський Д.С.*) 414

ДОДАТОК Б

Застосування мікросервісної архітектури для побудови відмовостійкої хмарної системи

Акт впровадження

УКР.НТУУ “КПІ”.ТВ3267_18М

Аркушів 2

2018