

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Приладобудівний факультет

Кафедра комп'ютерно-інтегрованих оптичних та навігаційних систем

«На правах рукопису»
УДК 004.67

До захисту допущено:
Завідувач кафедри
_____ Надія БУРАУ
«__» _____ 20__ р.

Магістерська дисертація

на здобуття ступеня магістра

**за освітньо-професійною програмою «Комп'ютерно - інтегровані технології
та системи навігації та керування»**

зі спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»

**на тему: «Розробка бази даних для автоматизованої системи навчально-
методичних матеріалів кафедри»**

Виконав:

студент 2 курсу, групи ПГ-01мп
Бондаренко Михайло Володимирович _____

Науковий керівник:

Доцент каф. КІОНС, к.т.н., доц.,
Мураховський Сергій Анатолійович _____

Консультант з розроблення стартап-проекту:

Професор кафедри менеджменту, д.е.н., проф.,
Бояринова Катерина Олександрівна _____

Рецензент:

Доцент кафедри ВП, к.т.н., доц.,
Філіппова Марина В'ячеславівна _____

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Приладобудівний факультет

Кафедра комп'ютерно-інтегрованих оптичних та навігаційних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 151 «Автоматизація та комп'ютерно-інтегровані технології» («Комп'ютерно-інтегровані технології та системи навігації та керування»)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Надія БУРАУ

« ____ » _____ 20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Бондаренку Михайлу Володимировичу

1. Тема дисертації «Розробка бази даних для автоматизованої системи навчально-методичних матеріалів кафедри», науковий керівник дисертації Мураховський Сергій Анатолійович, к.т.н., затверджені наказом по університету від « ____ » _____ 20__ р. № _____
2. Термін подання студентом дисертації 15.12.2021
3. Об'єкт дослідження База даних, що містить інформацію про навчально-методичні матеріали кафедри
4. Вихідні дані. В базі даних має бути реалізовано: доступ до бази через веб-інтерфейс, пошук за автором, назвою методичного матеріалу, назвою дисципліни; функціонал додання та редагування методичних матеріалів, можливість завантаження документів в форматах .docx(doc), .xlsx(xls), .pdf, .djvu, .jpg, .png; функціонал додавання та редагування даних користувачів бази, логування дій користувачів бази при роботі
5. Перелік завдань, які потрібно розробити. Вступ 1. Огляд відомих систем. 2. Аналіз та вибір апаратного та програмного забезпечення. 3. Розробка

архітектури бази даних. 4. Розробка функціональних модулів. 5. завантаження тестових даних та моделювання роботи. Висновки

6. Орієнтовний перелік графічного (ілюстративного) матеріалу. Рисунки та схеми в пояснювальній записці. Презентація 15-20 слайдів.

7. Орієнтовний перелік публікацій: 1 стаття.

8. Консультанти розділів дисертації

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|-----------------------------|---|----------------|------------------|
| | | завдання видав | завдання прийняв |
| Розроблення стартап-проекту | Доцент кафедри менеджменту, д.е.н., доц. Бояринова К.О. | | |

9. Дата видачі завдання _____

Календарний план

| № з/п | Назва етапів виконання магістерської дисертації | Термін виконання етапів магістерської дисертації | Примітка |
|-------|---|--|----------|
| | Провести огляд стану проблеми | 03.10.2020 | |
| | Аналіз існуючих рішень та технологій для вирішення поставленої задачі | 17.10.2020 | |
| | Спроекувати архітектуру проекту | 26.10.2020 | |
| | Реалізація серверної частини проекту | 03.11.2020 | |
| | Реалізація клієнтської(візуальної) частини | 13.11.2020 | |
| | Розгортання проекту в хмарі | 23.11.2020 | |
| | Оформлення рукопису дисертації | 15.12.2021 | |

Студент

Михайло БОНДАРЕНКО

Науковий керівник дисертації

Сергій МУРАХОВСЬКИЙ

Реферат

Магістерська дисертація складається з 116 сторінок, в ній міститься 36 рисунків, 22 таблиці, використано 53 джерела.

Актуальність. Враховуючи сучасні виклики, в тому числі спричинені пандемією COVID-19, у світі набувають широкого розповсюдження в освіті методи дистанційного навчання. Існує багато відомих платформ дистанційного навчання та онлайн-бібліотек, що виконують подібні функції, проте мають складний функціонал. В даній роботі запропоновано спрощену автоматизовану систему, яка побудована на основі нереляційної бази даних.

Мета роботи - створення бази даних для автоматизованої системи навчально-методичних матеріалів кафедри

Для досягнення поставленої мети роботи вирішуються наступні задачі:

1. Аналіз сучасних засобів побудови баз даних.
2. Вибір мови розробки програмного забезпечення та Cloud-технології.
3. Програмна реалізація серверної та клієнтської частини.
4. Розгортання проєкту в хмарному середовищі.
5. Розробка стартап проєкту.

Об'єкт дослідження - база даних автоматизованої системи навчально-методичних матеріалів кафедри

Предмет дослідження - програмне забезпечення для доступу до бази даних автоматизованої системи навчально-методичних матеріалів кафедри.

Наукова новизна магістерської дисертації полягає у створенні автоматизованої системи архітектура якої покладається на використання хмарних технологій, що дозволяє суттєво покращити характеристики продуктивності розроблюваної системи, а також надає можливість суттєвого знизити затрати на утримання серверу за рахунок використання архітектури безсерверних обчислень.

Апробація результатів: Наукові результати обговорювались на Міжнародній студентській науковій конференції “ Модернізація Та Сучасні Українські Та Світові Наукові Дослідження”, 29 травня 2021 року.

Практична цінність: спроектована автоматизована система надає можливість отримати централізоване сховище наукових матеріалів з можливістю повнотекстового пошуку, що дозволить поліпшити доступ до наукових матеріалів з метою їх дослідження та використання у нових наукових роботах.

Публікації

53. Бондаренко М.В. NoSQL Бази даних та ORM для роботи з ними./ Бондаренко М.В. // Молодіжна наукова ліга, Модернізація та сучасні українські та світові наукові дослідження. – 2020. С 7-10.

Ключові слова: база даних, хмарні обчислення, нативне зображення, безсерверні обчислення.

Abstract

The master's dissertation consists of 116 pages, it contains 36 drawings, 22 tables, used 53 sources.

Topicality. Given the current challenges, including those caused by the COVID-19 pandemic, distance learning methods are becoming widespread in education around the world. There are many well-known distance learning platforms and online libraries that perform similar functions but have complex functionality. This paper proposes a simplified automated system based on a non-relational database.

The purpose of the work is to create a database for the automated system of educational and methodical materials of the department

To achieve this goal the following tasks are solved:

1. Analysis of modern means of building databases.
2. Choice of software development language and Cloud technology.
3. Software implementation of server and client part.
4. Deployment of the project in a cloud environment.
5. Development of a startup project.

The subject of research is software for access to the database of the automated system of educational and methodical materials of the department.

The scientific novelty of the master's dissertation is to create an automated system whose architecture relies on the use of cloud technologies, which can significantly improve the performance of the developed system, as well as significantly reduce server maintenance costs by using server-free computing architecture.

Practical value: the designed automated system provides an opportunity to obtain a centralized repository of scientific materials with the possibility of full-text search, which will improve access to scientific materials for research and use in new scientific papers.

Approbation of results: Scientific results were discussed at the International Student Scientific Conference "Modernization and Modern Ukrainian and World Scientific Research", May 29, 2021.

Publications

Bondarenko M.V. NoSQL Databases and ORM to work with them./ Bondarenko MV // Youth Scientific League, Modernization and modern Ukrainian and world scientific research. - 2020. C 7-10.

Keywords: database, cloud computing, native image, serverless computing.

Зміст

| | |
|--|----|
| ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ | 10 |
| ВСТУП | 12 |
| РОЗДІЛ 1. АНАЛІЗ ЗАСОБІВ ДЛЯ ПОБУДОВИ ПРОЕКТУ | 14 |
| 1.1. Вибір СУБД | 15 |
| 1.1.1. СУБД MySQL | 15 |
| 1.1.2. СУБД Oracle DB. | 18 |
| 1.1.3. СУБД PostgreSQL | 20 |
| 1.1.4. Документо-орієнтована СУБД MongoDB | 22 |
| 1.1.5. Документо-орієнтована хмарна СУБД DynamoDB | 25 |
| 1.1.6. Резидентна СУБД Redis | 27 |
| 1.1.7. Розподілена з широким стовпчиком СУБД Apache Cassandra. | 29 |
| 1.1.8. Теорема CAP..... | 31 |
| 1.2. Вибір мови розробки | 34 |
| 1.2.1. Мова програмування Python | 34 |
| 1.2.2. Мова програмування Java. | 36 |
| 1.2.3. Мова програмування Kotlin. | 39 |
| 1.3. Аналіз існуючих постачальників Cloud технологій | 42 |
| Висновки до розділу 1 | 45 |
| РОЗДІЛ 2. РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИН ПРОЕКТУ. | 46 |
| 2.1. Розробка серверної частини застосунку | 46 |
| 2.2. Розробка візуальної частини застосунку | 61 |
| Висновки до розділу 2 | 68 |
| РОЗДІЛ 3. РОЗГОРТАННЯ ЗАСТОСУНКУ В АРХІТЕКТУРІ БЕЗСЕРВЕРНИХ ОБЧИСЛЕНЬ ТА ОПТИМІЗАЦІЯ ЧАСУ ВИКОНАННЯ ЗАПИТІВ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ НАТИВНОГО ЗОБРАЖЕННЯ. | 69 |
| 3.1. Безсерверні обчислення | 69 |
| 3.2. Оптимізація часу виконання за допомогою технології Native Image | 71 |
| 3.2.1. Швидший запуск | 72 |
| 3.2.2. Зменшений обсяг використовуваної пам'яті | 73 |

| | |
|---|-----|
| 3.2.3. Якою ціною обходяться переваги нативного зображення..... | 74 |
| 3.3. Розгортання проекту у хмарі та налаштування неперервної інтеграції та розгортання..... | 77 |
| Висновки до розділу 3 | 85 |
| Розділ 4. РОЗРОБКА СТАРТАП ПРОЕКТУ «АВТОМАТИЗОВАНА СИСТЕМИ НАВЧАЛЬНО-МЕТОДИЧНИХ МАТЕРІАЛІВ КАФЕДРИ» | 86 |
| 4.1. Опис ідеї проекту | 86 |
| 4.2. Технологічний аудит ідеї проекту..... | 88 |
| 4.3. Аналіз ринкових можливостей запуску стартап-проекту | 89 |
| 4.4. Розроблення ринкової стратегії проекту..... | 100 |
| 4.5. Розроблення маркетингової програми стартап-проекту | 104 |
| Висновок до розділу 4..... | 108 |
| ВИСНОВОК | 110 |
| СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ..... | 112 |

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

- СУБД – Система управління базами даних
- SQL (Structured Query Language) - Мова Структурованих Запитів
- RDBMS (Relational Database Management System) - Система управління реляційними базами даних
- OLTP (Online Transaction Processing) – Онлайнова обробка транзакцій
- DW (Data warehouse) – Сховище даних
- NoSQL (not only SQL) – Нереляційні бази даних
- AWS (Amazon Web Services) – Веб Сервіси Амазону
- HTTP (HyperText Transfer Protocol) – Протокол передачі гіпертексту
- API (Application Programming Interface) – Прикладний програмний інтерфейс
- SDK (Software Development Kit) – Комплект для розробки програмного забезпечення
- SSD (Solid-State Drive) – Твердотілий накопичувач
- CQL (Cassandra Query Language) – Мова запитів Кассандри
- CAP (Consistency, Availability, Partition Tolerance) – Узгодженість, доступність, толерантність розділу
- JDBC (Java Database Connectivity) – Джава підключення до бази даних
- ODBC (Open DataBase Connectivity) – Відкрите підключення до бази даних
- JVM (Java Virtual Machine) – Віртуальна машина Джава
- JRE (Java Runtime Environment) – Середовище виконання Джава
- JIT (Just-in-time) – Компіляція «на льоту»
- LLVM (Low Level Virtual Machine) – Віртуальна машина низького рівня
- IaaS (Infrastructure as a Service) – Інфраструктура як послуга
- PaaS (Platform as a Service) – Платформа як послуга
- DAO (Data Access Object) – Об’єкт доступу даних

URL (Uniform Resource Locator) – Уніфікований локатор ресурсів
PDF (Portable Document Format) – Формат портативних документів
FaaS (Function as a Service) – Функціонує як послуга
JDK (Java Development Kit) – Комплект розробки Джава
GC (Garbage Collector) – Збірник сміття
JVMTI (JVM Tool Interface) – Інтерфейс інструмента JVM
JMX (Java Management Extensions) – Розширення керування Джава
JFR (Java Flight Recorder) – Реєстратор польоту Джава
CI (Continuous integration) – Постійна інтеграція
CD (Continuous delivery) – Неперервна доставка
IDE (Integrated Development Environment) – Інтегроване середовище розробки

ВСТУП

Необхідність розробки автоматизованої системи навчально методичних матеріалів кафедри пов'язана зі стрімким процесом відцифровування, котрий диктує тенденції розвитку виробництв та процесів. В останні часи все більше галузей поступово переходить до повністю цифрової форми ведення обліку документів та цінних бумаг, що в свою чергу провокує необхідність створення централізованого сховища для електронних документів.

Заклади освіти також не стоять на місці і все більше надають перевагу цифровим технологіям у цілях аудиту наукових матеріалів. Варто також зазначити, що в закладах освіти проходять багато наукових конференцій, постійно вдосконалюються та розробляються методичні матеріали, а також публікуються статті, тези, доповіді. Якщо років 30 назад важко були представити, що збірник статей конференції буде опублікований у цифровому вигляді то в нинішній час це майже необхідність, оскільки електронний вигляд даних дозволяє зберігати їх у неушкодженому стані більший час, а також це робить наукові матеріали більш доступними для загальний мас які зацікавлені у науковій діяльності.

Саме тому постає питання створення централізованої бази даних котра б утримувала у собі науково-методичні матеріали та надавала б простий та вільний доступ до них для всіх бажаючих. Вирішенням цього питання є впровадження автоматизованої системи навчально-методичних матеріалів.

Ключова особливість системи це надання користувачу якомога простіший інтерфейс для використання при цьому не втрачаючи основні аспекти функціоналу. Важливим фактором системи також є архітектурні рішення при побудові серверної та клієнтської частини, а саме те, що серверна частина побудована використовуючи безсерверні обчислення котрі надають суттєву

перевагу з точки зору цінової рентабельності системи, оскільки при такій реалізації платити необхідно лише за той час коли система у явному вигляді використовується, а не постійно, як це доводиться робити при використанні звичайних серверів.

РОЗДІЛ 1. АНАЛІЗ ЗАСОБІВ ДЛЯ ПОБУДОВИ ПРОЕКТУ

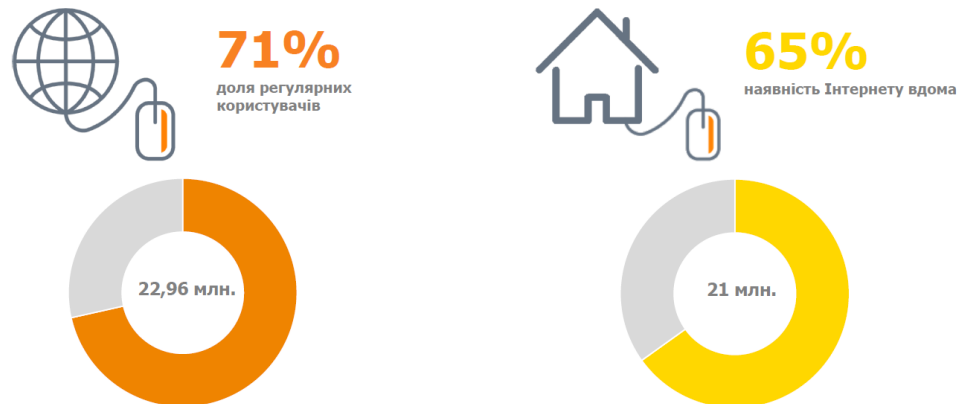
На сьогоднішній день питання оцифровування інформації належить до одних із найактуальніших в житті суспільства. Процес оцифровування – це явище, яке охопило майже всі сфери діяльності людини і пов'язане із застосуванням новітніх інформаційних технологій.

Система наявної на кафедрі літератури являється актуальним рішенням для відслідковування та збереження стану усього доступного матеріалу. Слід також зазначити, що в сучасних реаліях збереження інформації та отримання до неї доступу в електронному вигляді являється найбільш простою та оптимальною формою, оскільки більшість людей мають при собі пристрій котрий може працювати з електронними файлами.

Складність розробки моделі та структури збереження інформації про літературу пов'язана з тим, що жодне з існуючих рішень не може бути використане при розробці вказаної вище системи. На те є декілька причин, перша полягає в тому, що існуючі подібні системи мають закритий вихідний код, що робить їх використання неможливим, оскільки в такому випадку ми будемо обмежені функціоналом яка надає система, без можливості його змінити або доповнити. Друга причина лежить у тому факті, що системи за типом 1С:Бібліотека та OPAC-Global направлені більше на побудову бібліотечних систем, через що в них наявна велика кількість зайвого функціоналу.

На рисунку 1.1 наведена статистика за даними результатів дослідження Інтернет асоціації України яка показує, що на сьогодні всесвітньою мережею регулярно користуються 22,96 млн українців, або 71%, порівняно з показником 63% станом на кінець 2018 року. Загальна кількість інтернет-користувачів у 2019 році збільшилась на 8% [1].

Інтернет: ОСНОВНІ ПОКАЗНИКИ



Вересень 2019, N=2110, Вся Україна без АР Крим та окупованих територій України, вік 15+ .
До 3 кварталу 2014 року дані з АР Крим. З 3 кварталу дані без АР Крим та окупованих територій України.

FACTUM
GROUP

Рисунок 1.1 – Статистика наявності доступу до мережі Інтернет в Україні
Враховуючи представлену вище статистику задача побудови системи котра зберігає дані про літературу стає ще більш актуальною.

1.1. Вибір СУБД

Оскільки основна задача даної системи це збереження даних, то ніщо не справиться з цим краще ніж база даних. База даних (англ. database) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування. В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, окрім даних, містить їх опис та може містити засоби для їх обробки [2]. Наразі, постає питання вибору системи управління базою даних (СУБД).

Проведемо порівняльний аналіз декілька існуючих СУБД.

1.1.1. СУБД MySQL

Спершу розглянемо одну з найбільш відомих СУБД – MySQL, логотип якої представлений на рисунку 1.2.



Рисунок 1.2 – Логотип MySQL

Програмне забезпечення MySQL забезпечує дуже швидкий, багатопотоковий, багатокористувацький та надійний сервер баз даних SQL (Structured Query Language). MySQL Server призначений для критично важливих виробничих систем з великим навантаженням, а також для вбудовування в програмне забезпечення масового використання [3].

MySQL має розвинену систему доступу до баз даних. Користувачеві бази даних може бути надано доступ до всієї бази даних, окремих таблиць і окремих стовпців таблиць. Присутнє розмежування на операції, які може здійснювати користувач із записами. Для організації такої складної (на перший погляд) структури доступу використовується декілька таблиць у спеціальній базі даних. На підставі значень цих таблиць вибудовується політика надання доступу [4].

Також варто відзначити що в MySQL постачає широкий спектр додаткових утиліт, таких як MySQL Workbench, представлений на рисунку 1.3, котрий надає вичерпну інформацію про стан серверу та сам по собі являється графічним інтерфейсом для проектування бази даних, MySQL Shell котрий являється консольною оболонкою для роботи з сервером, а також, що досить важливо, Connectors для багатьох популярних мов програмування, котрі служать для встановлення зв'язку з базою даних в середовищі мови.

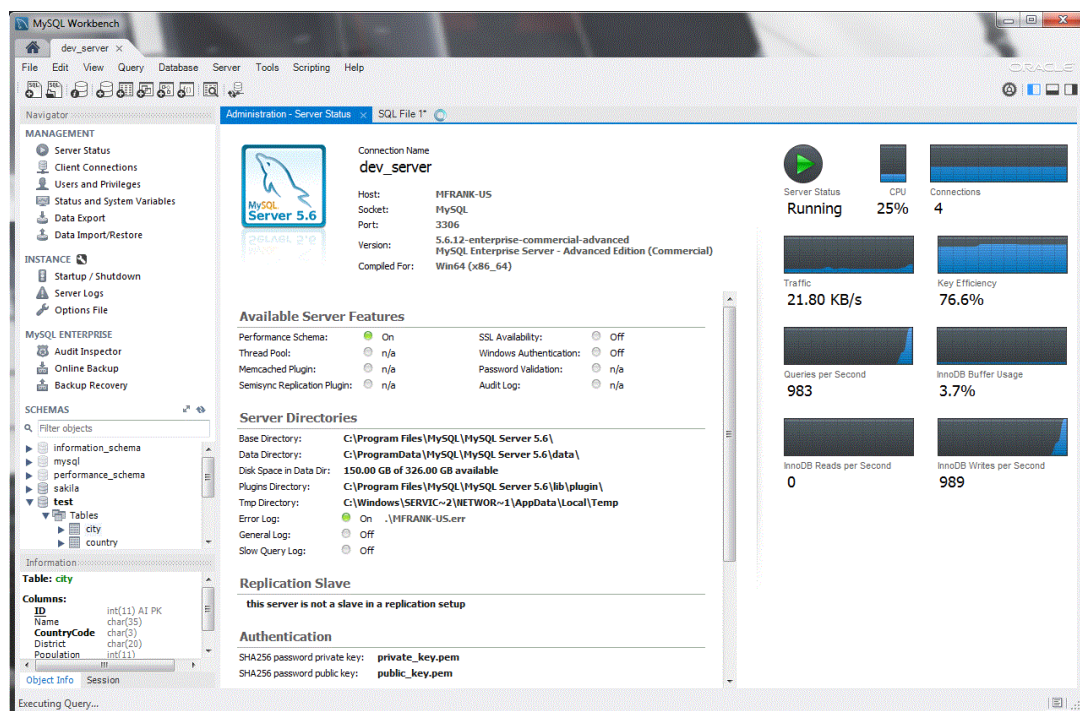


Рисунок 1.3 – Робоче середовище MySQL Workbench

Переваги:

- **Безкоштовність.** Community edition MySQL можливо безкоштовно завантажити. Маючи базовий набір інструментів для індивідуального використання, видання спільноти MySQL - хороший варіант для початку. Звичайно, є й інші, передплачені варіанти для цілей Enterprise або Cluster із більш багатим функціоналом.

- **Простий синтаксис і помірна складність.** Структура та стиль MySQL дуже прості. Розробники навіть вважають MySQL базою даних із людською мовою. Також MySQL простий у використанні. Наприклад, більшість завдань можна виконати прямо в командному рядку, зменшивши етапи розробки.

- **Сумісність з хмарними технологіями.** Бізнес-орієнтований за своєю природою і спочатку розроблений для Інтернету, MySQL підтримується найпопулярнішими хмарними провайдерами. Він доступний на таких провідних платформах, як Amazon, Microsoft та інші. Це робить MySQL ще більш привабливим і надає місце для зростання.

Недоліки:

- **Проблеми масштабування.** MySQL не був побудований з урахуванням масштабованості, що властиво його коду. Теоретично можливо масштабувати MySQL, але для цього знадобиться більше зусиль порівняно з будь-якою з баз даних NoSQL.

- **Обмежена відповідність стандартам SQL.** Структурована мова запитів має конкретні стандарти. MySQL не повністю виконує їх, тобто MySQL не підтримує деякі стандартні функції SQL. З іншого боку, MySQL має деякі розширення та чіткі функції, які не відповідають стандартам структуризованої мови запитів. Для невеликих веб-додатків це не велика справа. Проблеми можуть з'явитися, коли вам доведеться перейти на інші бази даних, що, швидше за все, станеться, коли ваш бізнес починає зростати[5].

1.1.2. СУБД Oracle DB.

Наступним потенційним кандидатом являється Oracle DB (логотип показано на рисунку 1.4). База даних Oracle (зазвичай її називають Oracle RDBMS або просто як Oracle) - це багатомодельна система управління базами даних, що виробляється та продається корпорацією Oracle. Це база даних, яка зазвичай використовується для роботи в режимі онлайн обробки транзакцій (OLTP), зберігання даних (DW) та змішаних (OLTP & DW) навантажень бази даних. Останні покоління, Oracle Database 19c, доступні в режимі on-premise, в хмарі або в гібридному хмарному середовищі.



Рисунок 1.5 – Логотип Oracle Database

Oracle Database має логічні структури та фізичні структури. Оскільки фізична та логічна структури є окремими, фізичним зберіганням даних можна керувати, не впливаючи на доступ до логічних структур зберігання. Ця структура означає, що для широкомасштабних розподілених обчислень, також відомих як сіткові обчислення, розташування даних не має значення та прозорості для користувача, що забезпечує більш модульну фізичну структуру, яку можна додавати та змінювати, не впливаючи на активність бази даних, його дані або користувачів [6].

Переваги:

- **Інновації для щоденного робочого процесу.** Завдяки Oracle 12c як гібридному хмарному програмному забезпеченню, інноваційні технології хмарних обчислень з'являються щодня. У той же час вона постійно фокусується на інформаційній безпеці. Окрім активного захисту даних, розподілу, покращення резервного копіювання та відновлення, Oracle пропонує паралельну модернізацію, щоб зменшити час простою під час оновлення бази даних.

- **Сильна технічна підтримка та документація.** Oracle гарантує гідну підтримку клієнтів та забезпечує вичерпну технічну документацію з різних ресурсів.

- **Велика ємність.** Багатомодельне рішення Oracle дозволяє розмістити та обробити велику кількість даних. Завдяки випущеному функціоналу мульти-оренди, архітектура бази даних в даний час спрощує упаковку багатьох баз даних і керувати ними гладко. У поєднанні з можливостями обробки даних в пам'яті він створює потужний механізм синхронної обробки даних.

Недоліки:

- **Висока вартість.** Хоча Oracle 12c RDBMS має безкоштовні видання, вони дуже обмежені з точки зору функціональності. Стандартне видання, яке не

включає всі доступні функції, коштує 17 500 доларів США за одиницю. Enterprise Edition становить понад 47 000 доларів за одиницю.

- **Ресурсоємність.** Базі даних Oracle потрібна потужна інфраструктура. Мало того, що установка вимагає багато дискового простору, ви також повинні враховувати постійні оновлення апаратного забезпечення при розгортанні її локально.

- **Складність навчання.** База даних Oracle - це не система, яку можна починати використовувати відразу. Для її запуску краще мати сертифікованих інженерів Oracle DB. Документація Oracle, хоча охоплює багато питань, іноді може бути непереборною і навіть заплутаною.

1.1.3. СУБД PostgreSQL

PostgreSQL – це потужна об'єктно-реляційна база даних із відкритим кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші навантаження даних. Витоки PostgreSQL сягають 1986 року в рамках проекту POSTGRES в Каліфорнійському університеті в Берклі.

Ця система управління базами даних розділяє свою популярність з MySQL. Вона спрямований на посилення стандартів дотримання та розширення. Отже, вона може обробляти будь-яке робоче навантаження, як для одномашинних продуктів, так і для складних програм [7].

Для спрощення адміністрування на сервері PostgreSQL в базовий комплект установки входить такий інструмент як pgAdmin, представлений на рисунку 1.6. Він являється графічним клієнтом для роботи з сервером, через який можливо в зручному вигляді створювати, видаляти, змінювати бази даних і керувати ними.

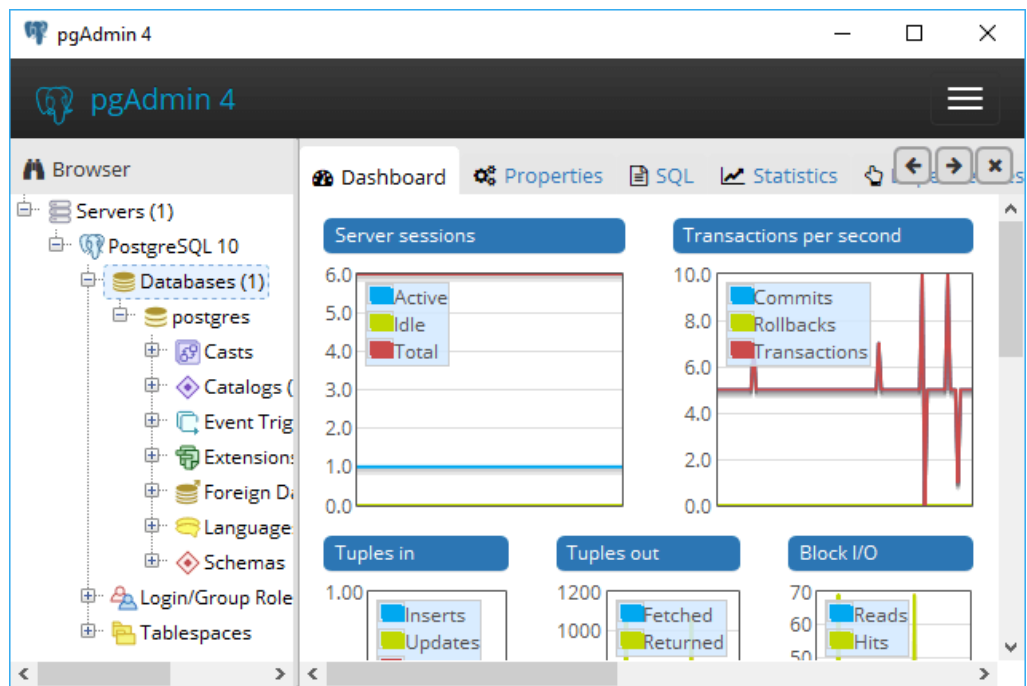


Рисунок 1.6 – робоче середовище pgAdmin

Переваги:

- **Масштабованість.** Вертикальна масштабованість є відмітною ознакою PostgreSQL, на відміну від MySQL СУБД. Враховуючи те, що практично будь-яке користувацьке програмне рішення має тенденцію до зростання, що призводить до розширення бази даних, цей конкретний варіант, безумовно, підтримує зростання та розвиток бізнесу.

- **Підтримка користувацьких типів даних.** PostgreSQL за замовчуванням підтримує велику кількість типів даних, таких як JSON, XML, H-Store та інші. PostgreSQL цим користується, будучи однією з небагатьох реляційних баз даних з сильною підтримкою функцій NoSQL. Крім того, це дозволяє користувачам визначати власні типи даних.

- **Легко інтегровані сторонні інструменти.** Система управління базами даних PostgreSQL підтримує додаткові інструменти, як безкоштовні, так і комерційні. Сфера застосування включає розширення для покращення багатьох аспектів. Наприклад, ClusterControl надає вражаючу допомогу в управлінні, моніторингу та масштабуванні баз даних SQL і NoSQL з відкритим кодом. Щоб

зробити порівняння та синхронізацію даних більш ефективними, можливе використання DB Data Diffective. Якщо виникає необхідність масштабувати дані до великих навантажень, вибір резервного копіювання та відновлення системи pgBackRest буде чудовим вибором.

- **Відкритість та керованість спільнотою.** Postgres є повністю відкритим кодом та підтримується його спільнотою, що зміцнює його як повноцінну екосистему. Крім того, розробники завжди можуть очікувати безкоштовної та швидкої допомоги тієї ж спільноти.

Недоліки:

- **Мінлива документація.** У той час як PostgreSQL має велике співтовариство і надає потужну підтримку його учасникам, у документації все ще бракує послідовності та повноти. Оскільки PostgreSQL-спільнота досить розповсюджена, документація не відповідає рівним стандартам для всіх функцій Postgre.

- **Відсутність інструментів звітності та аудиту.** Істотним недоліком PostgreSQL є відсутність інструментів перегляду, які б показували поточний стан бази даних. Доведеться постійно перевіряти, чи щось не так. Також, завжди існує ризик, що помилку помітять занадто пізно.

1.1.4. Документо-орієнтована СУБД MongoDB

Варто зазначити, що в останні часи набувають досить високої популярності так звані NoSQL бази даних – бази даних забезпечує механізм зберігання та пошуку даних, який моделюється іншими засобами, ніж табличні відносини, використовувані в реляційних базах даних [8]. Бази даних NoSQL все частіше використовуються у великих додатках даних та веб-додатках у режимі реального часу [9]. Системи NoSQL також іноді називають "не тільки SQL", щоб підкреслити, що вони можуть підтримувати SQL-подібні мови запитів або сидіти поруч із базами даних SQL в стійких поліглот-архітектурах [10].

Однією з найбільш відомих NoSQL баз даних являється MongoDB. MongoDB – крос-платформенна документ-орієнтована база даних. Модель документа MongoDB проста для розробників для вивчення та використання, але все ж надає всі можливості, необхідні для задоволення найскладніших вимог у будь-якому масштабі.

Хоча MongoDB спочатку не призначався для структурованої обробки даних, вона може використовуватися для програм, які використовують як структуровані, так і неструктуровані дані. У MongoDB бази даних підключаються до програм через драйвери бази даних. Вони широко доступні в системі управління базами даних. Кілька типів даних обробляються одночасно і для цього використовують внутрішній кеш. Також у стандартному пакеті входить Datadog, представлений на рисунку 1.8, котрий використовується для візуалізації ключових метрик системи та аналізу продуктивності інфраструктури.



Рисунок 1.7 – Моніторинг ресурсів MongoDB серверу за допомогою Datadog.

Переваги:

- **Простий доступ до даних, зберігання, введення та пошук.** Однією з переваг MongoDB, що випливає з її характеру NoSQL, є швидка та проста робота з даними. Тобто, дані можна вводити, зберігати та вилучати з бази даних швидко та без додаткового підтвердження. Як і будь-яка інша нереляційна база даних, вона робить акцент на використанні оперативної пам'яті, тому з записами можна маніпулювати дуже швидко та без жодних наслідків для цілісності даних [11].

- **Проста сумісність з іншими моделями даних.** MongoDB легко поєднується з різними системами управління базами даних, як типів SQL, так і NoSQL. Крім того, він має підключаємі API для двигуна зберігання, цей параметр дозволяє третім сторонам створювати власні механізми зберігання даних для MongoDB [12].

- **Горизонтально масштабоване рішення.** Масштабованість - там, де дані розповсюджуються через розподілену мережу керованих серверів - є основою принципу MongoDB. Це стає ще важливішим для підприємств, які працюють із програмами великих даних. Крім того, база даних може розподіляти дані через кластер машин. Як це може допомогти? Дані поширюються швидше і рівномірно, без громіздкості. Оскільки це призводить до швидшої обробки даних, продуктивність додатків також прискорюється [13].

Недоліки:

- **Велике споживання пам'яті.** Процес денормалізації, коли раніше нормалізовані дані в базі даних групуються для підвищення продуктивності, зазвичай це призводить до високого споживання пам'яті. Також ця СУБД зберігає в пам'яті всі імена ключів для кожної пари значень. Крім того, оскільки немає підтримки *joins*, у базах даних Mongo є надмірна подача даних, що призводить до великих відходів пам'яті та зниження продуктивності програми [14].

- **Ненадійність даних.** Орієнтуючись на швидку роботу з даними, MongoDB, як і будь-яка інша СУБД NoSQL, не має захисту даних. Оскільки

автентифікація користувачів не є типовою опцією Mongo, а вищий захист доступний лише для комерційного видання, не можна вважати це повністю безпечним. Крім того, існують постійні версії оновлень MongoDB, без жодних гарантій, що всі поправки чи зміни даних працюватимуть так, як це було раніше. Слід мати на увазі, що всі маніпуляції повинні формуватися навколо цих оновлень, покриваючись додатковими тестами [15].

- **Складний процес інтерпретації на інші мови запитів.** Оскільки MongoDB спочатку не був розроблений для роботи з реляційними моделями даних, продуктивність може уповільнитися в цих випадках. Крім того, переклад SQL на запити MongoDB вимагає додаткових дій для використання двигуна, що може затримати розробку та розгортання [16].

1.1.5. Документо-орієнтована хмарна СУБД DynamoDB

Конкурентною альтернативою MongoDB являється DynamoDB. Amazon DynamoDB — це повністю керована служба баз даних NoSQL, яку пропонує Amazon Web Services (AWS). DynamoDB вимагає мінімального обсягу налаштування та обслуговування з боку розробника, забезпечуючи високу продуктивність і масштабованість.

Як і інші бази даних, DynamoDB зберігає свої дані в таблицях. Кожна таблиця містить набір елементів, і кожен елемент має набір полів або атрибутів. Кожна таблиця повинна мати первинний ключ, присутній у всіх елементах таблиці, і цей первинний ключ може бути як одним атрибутом, так і комбінацією двох атрибутів: ключа розділу та ключа сортування. Ви можете посилатися на певні елементи в таблиці, використовуючи первинний ключ або створюючи власні індекси та використовуючи ключі з цих індексів.

Немає жодного сервера, на якому розміщено вашу таблицю DynamoDB. Натомість дані розподіляються між багатьма машинами — це гарантує масштабованість і високу продуктивність, але це також означає, що ви не можете підключитися до хоста бази даних і запитувати свої дані безпосередньо. Щоб

записувати та читати елементи в таблицю DynamoDB та з неї, вам потрібно використовувати DynamoDB HTTP API безпосередньо або за допомогою AWS SDK або AWS CLI. Крім того, ви можете групувати свої зчитування та записи в таблиці DynamoDB, навіть у різних таблицях одночасно. DynamoDB підтримує транзакції, автоматичне резервне копіювання та міжрегіональну реплікацію. На рисунку 1.8 представлений інтерфейс для роботи з DynamoDB.

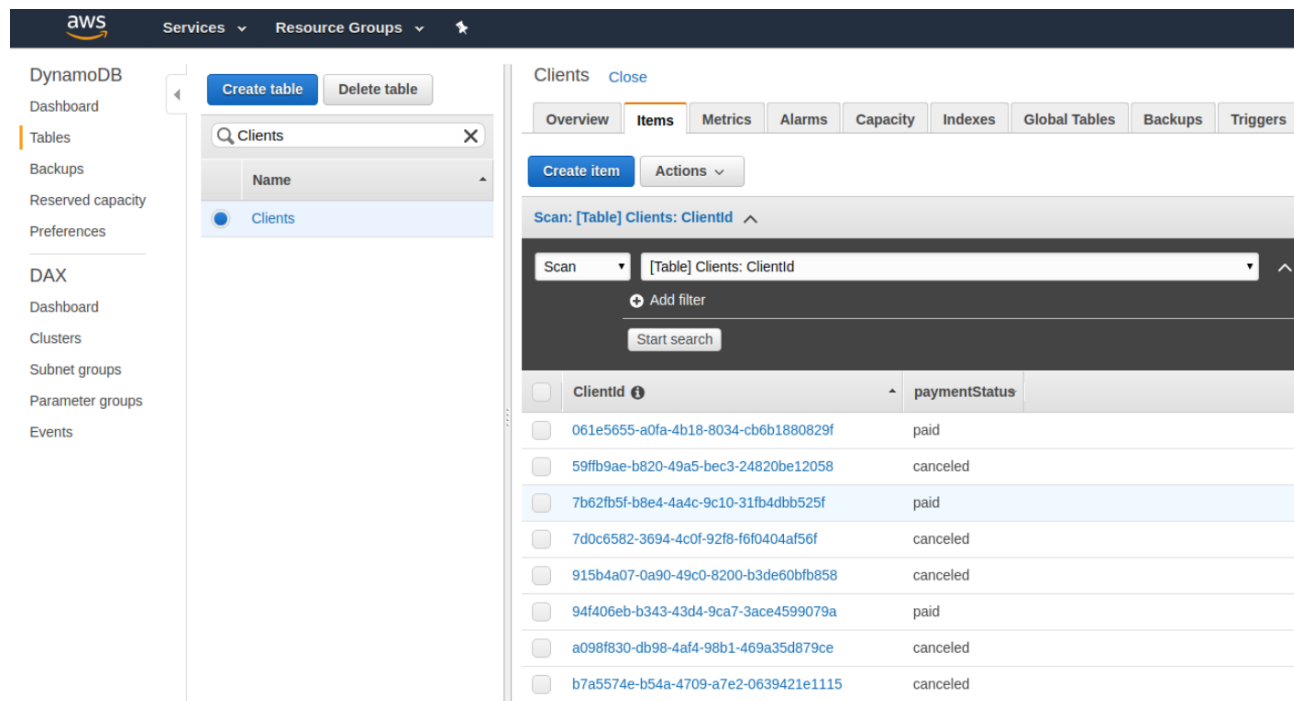


Рисунок 1.8 – Інтерфейс для роботи з DynamoDB у середовищі AWS.

Переваги:

- **Повна керованість.** DynamoDB — це повністю кероване рішення — не потрібно виконувати жодних операційних завдань для підтримки роботи бази даних. Це означає, що не потрібно оновлювати сервери, виправляти ядро, не потрібно замінювати SSD. Використання повністю керованого рішення зменшує кількість часу, який ваша команда витрачає на операції, дозволяючи замість цього зосередитися на розробці продукту[17].

- **Автомасштабування.** Також не потрібно робити нічого для масштабування продуктивності таблиць DynamoDB або їх розмірів у міру збільшення навантаження програми. DynamoDB подбає про все це автоматично.

- **Автоматична реплікація.** Глобальні таблиці DynamoDB дозволяють мати базу даних з кількома провідними, кількома регіонами з дуже невеликими налаштуваннями та без постійного обслуговування. Це може допомогти прискорити роботу вашої програми, якщо у вас є клієнти з різних регіонів світу, які використовують локально розгорнуті екземпляри вашої програми.

- **Підтримка потокової передачі.** DynamoDB дозволяє створювати потоки оновлень до таблиць даних. Потім ви можете використовувати ці потоки, щоб ініціювати іншу роботу в інших службах AWS, включаючи функції лямбда. Це дозволяє дуже легко додавати автоматизацію на основі ваших оновлень до даних DynamoDB.

Недоліки:

- **Сильна прихильність до постачальника.** DynamoDB є запатентованим рішенням і не має версії з відкритим вихідним кодом, тому якщо коли-небудь настане час відмовитися від використання DynamoDB, доведеться перенести значний обсяг роботи на інше рішення бази даних. Деякі функції, такі як DynamoDB Streams, може бути особливо важко перебудувати в іншій базі даних.

- **Немає підтримки операцій JOIN.** Дизайн DynamoDB не дозволяє об'єднувати дані з кількох таблиць. Отже, якщо вам потрібні дані з кількох таблиць для однієї операції, ви побачите, що DynamoDB є дорожчим, повільнішим і складнішим для реалізації JOIN, ніж з реляційним сховищем даних.

1.1.6. Резидентна СУБД Redis

Наступним актуальним рішенням являється Redis. Redis - це сховище структур даних в пам'яті з відкритим кодом (ліцензований BSD), використовується як база даних, кеш-пам'ять та брокер повідомлень. Він підтримує структури даних, такі як строки, хеші, списки, набори, відсортовані набори з діапазонами запитів, растрових зображень, гіперлогів, геопросторових

індексів з радіусними запитами та потоками. На рисунку 1.9 представлений варіант використання Redis у якості брокера повідомлень.

Redis популяризував ідею системи, яка може вважатися одночасно сховищем і кешем, використовуючи дизайн, коли дані завжди модифікуються і читаються з основної пам'яті комп'ютера, а також зберігаються на диску у форматі, непридатному для випадкового доступ до даних, але лише для відновлення даних назад у пам'яті після перезавантаження системи. У той же час Redis пропонує модель даних, яка є дуже незвичною порівняно з системою управління реляційними базами даних (RDBMS). Команди користувача не описують запит, який повинен виконувати двигун бази даних, а конкретні операції, які виконуються на заданих абстрактних типах даних. Отже, дані повинні зберігатися таким чином, який згодом підходить для швидкого пошуку, без допомоги системи баз даних у вигляді вторинних індексів, агрегацій або інших загальних особливостей традиційних RDBMS [18].

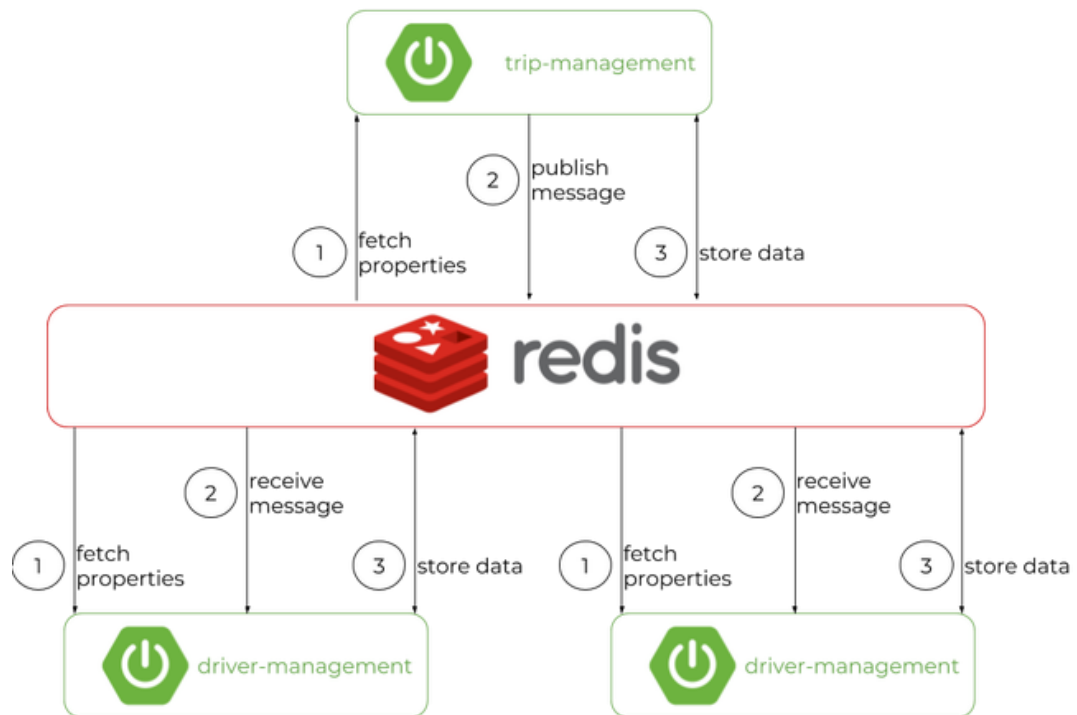


Рисунок 1.9 – Схема використання Redis у якості message broker на мікросервісній архітектурі.

Переваги:

- **Швидке рішення.** Завдяки реплікації та особливостям транзакцій Redis обробляє дані дуже швидко. Відсутність залежностей та типу сховища даних в пам'яті робить Redis гідним конкурентом навіть серед простих альтернатив SQL.

- **Масова обробка даних.** З точки зору сприйняття та очищення даних, Redis можна вважати гігантом. Він може легко завантажувати до 1 ГБ даних за один запис. Додати вбудоване кешування даних, і отримаємо машину передачі даних [19].

Недоліки:

- **Потрібні набори даних, щоб вмістилися у пам'яті.** Повна ставка та залежність від пам'яті програми - справжній недолік. Тобто, база даних вийде з ладу, якщо її розмір перевищить розмір наявної пам'яті.

- **Немає підтримки мови запитів або *join*-ів.** Що стосується сумісності з іншими типами набору даних, Redis відстає. Зважаючи на те, що певний в час може знадобитися масштабування та використання інших форматів даних, швидке введення даних як єдиного варіанту залишає цю проблему відкритою.

1.1.7. Розподілена з широким стовпчиком СУБД Apache Cassandra.

Не менш цікаве рішення можливо розглянути у якості Apache Cassandra. Cassandra - це децентралізована система, розроблена Apache. Cassandra - це безкоштовна СУБД, сила якої полягає у її багаторазовому повторенні та багаторазовому розгортанні. Ці особливості дозволяють виконувати численні копіювання запитів та розгортання їх усіх одночасно. Будучи швидко масштабованою, Cassandra дозволяє керувати великими обсягами даних шляхом реплікації в декілька вузлів [20]. Це усуває проблему збою бази даних - якщо деякі з вузлів виходять з ладу в будь-який момент, вони замінюються негайно, і система продовжує працювати до тих пір, поки щонайменше один єдиний вузол є безпечним [21].

Cassandra використовує власну мову запитів, CQL. За своїм синтаксисом вона дуже схожа на SQL, але не застосовує join-и, замінюючи їх так званими сімействами стовпців. І друга відмінність полягає в тому, що не всі стовпці таблиці зберігаються для підзапитів. Деякі з них використовуються як кластеризаційні стовпці, де сусідні дані ставлять поруч для швидкого пошуку. Чому це має значення? Вона забезпечує швидший запит від масивних наборів даних, прискорюючи обробку даних [22]. На рисунку 1.10 відображені ключові функції які надає Cassandra.

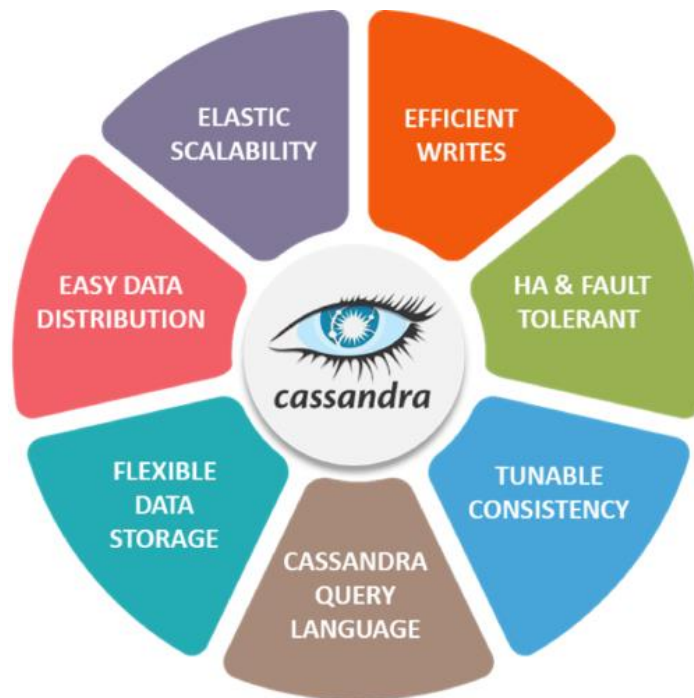


Рисунок 1.10 – Ключовий функціонал присутній в Cassandra.

Переваги:

- **Безпека даних.** Завдяки своїй функції реплікації головного вузла, Cassandra залишається стійкою до відмов. Це означає, що інженери БД можуть відчувати впевненість у безпеці даних, якщо головні вузли не виходять з ладу всі одночасно. Поки це буде малоімовірно, база даних та додаток, побудований на ній, залишатимуться надійними [23].

- **Гнучкість та поправки.** Простий синтаксис Casandra має в собі найкраще з SQL та NoSQL. Крім масштабованості, це значною мірою сприяє

гнучкості набору даних. Cassandra збирає дані на ходу, і отримання даних поділяє однакову простоту, незважаючи на розмір набору даних. Це дозволяє збільшити базу даних в повній мірі [24].

Недоліки:

- **Повільне читання.** Оскільки Cassandra спочатку була розроблена для швидкого письма, її слабкість полягає в її нездатності до швидкого читання. Однією з причин цього є те, що для надісланої інформації немає вузьких місць, тому для її обробки потрібно більше часу.

- **Вимагає додаткових ресурсів.** Оскільки Кассандра обробляє декілька шарів даних одночасно, для цього потрібна достатня потужність, що призводить до підвищеного використання JVM. Це означає додаткові інвестиції як у програмне забезпечення, так і в апаратне забезпечення.

1.1.8. Теорема CAP.

Здавалося б, що вибір СУБД не повинний викликати труднощів, ми просто обираємо ту, в якій найбільше переваг та найменше недоліків. Однак не все так просто як хотілося б, насправді в процесі вибори завжди з'являється така річ як CAP теорема. Теорема CAP - це інструмент, який дозволяє інформувати дизайнерів систем про компроміси під час проектування мережесистем спільних даних. CAP вплинув на розробку багатьох розподілених систем даних. Це дозволило дизайнерам усвідомити широкий спектр компромісів, які слід враховувати під час проектування розподілених систем даних. Протягом багатьох років теорема CAP була широко не зрозумілим інструментом, який використовується для категоризації баз даних. У теоремі зазначається, що мережеві системи спільних даних можуть гарантувати / підтримувати лише дві з наступних трьох властивостей [25]:

- **Консистентність (Consistency)** - гарантія того, що кожен вузол у розподіленому кластері повертає однакові, найсвіжіші, успішні записи. Послідовність стосується кожного клієнта, який має однаковий вигляд даних.

Існують різні типи моделей консистенції. Консистентність CAP відноситься до можливості лінеаризації чи послідовної консистентності, дуже сильної форми узгодженості.

- **Доступність (Availability)** - кожен працюючий вузол повертає відповідь на всі запити на читання та запис у розумний проміжок часу. Ключове слово тут - кожне. Щоб бути доступним, кожен вузол на (будь-якій стороні мережевого розділу) повинен мати можливість відповісти в розумний проміжок часу.

- **Толерантність до розділів** - система продовжує функціонувати та підтримує гарантії узгодженості, незважаючи на мережеві розділень. Мережеві розділення - це факт життя. Розподілені системи, що гарантують толерантність до розділів, можуть виразно відновитись із розділень, коли проблема пройде.

Теорема CAP класифікує системи на три категорії:

- **CP** (толерантний до консистенції та розділів) - На перший погляд, категорія CP є заплутаною, тобто система, яка є стійкою та толерантною до розділів, але ніколи не доступна. CP посиляється на категорію систем, де доступність жертвується лише у випадку з мережевим розділом.

- **CA** (консистентні та доступні) - системи CA є консистентними та доступними системами за відсутності мережевого розділу. Часто сервери БД одного вузла класифікуються як системи CA. Серверам БД з одним вузлом не потрібно мати справу з толерантністю до розділів і тому вважаються системами CA.

- **AP** (доступні та толерантні до розділень) - Це системи, які доступні та мають толерантність до розділів, але не можуть гарантувати послідовність.

На рисунку 1.11 представлено розподілення систем управління базами даних відповідно до CAP теореми.

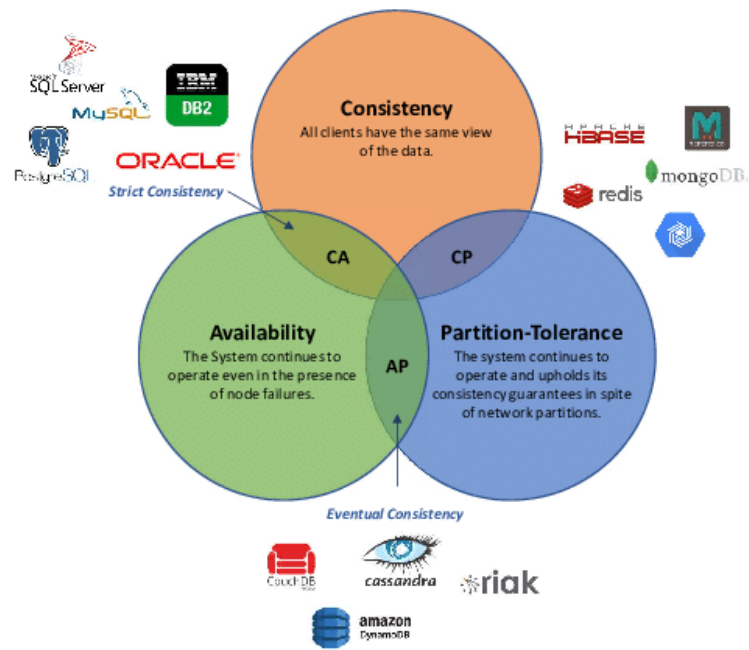


Рисунок 1.11 – Відповідність СУБД CAP теоремі.

Багато баз даних NoSQL компрометують консистентність (у сенсі теореми CAP) на користь доступності, толерантності до розділів та швидкості. Перешкодами для більш широкого прийняття NoSQL є використання мов запитів низького рівня (замість SQL, наприклад, відсутність можливості виконувати ad-hoc приєднання через таблиці), відсутність стандартизованих інтерфейсів та величезні попередні інвестиції в існуючі реляційні бази даних [26].

Натомість більшість баз даних NoSQL пропонують концепцію "кінцевої послідовності", в якій зміни бази даних поширюються на всі вузли "з часом" (як правило, протягом мілісекунд), тому запити для даних можуть не повернути оновлені дані негайно або призвести до зчитування даних, які не є точними, проблема, відома як затхле читання [27].

Враховуючи усі вище перераховані фактори було прийняте рішення обрати систему керування базами даних DynamoDB у якості фундаментальної СУБД для системи збереження літератури.

1.2. Вибір мови розробки

Оскільки конкретне рішення СУБД було обрано, час перейти до аналізу існуючих мов програмування, які можливо використовувати для вирішення поставленої задачі.

1.2.1. Мова програмування Python

Python - інтерпретована мова програмування високого рівня, загального призначення. Створена Гідо ван Россумом і вперше випущена в 1991 році, філософія дизайну Python підкреслює читабельність коду завдяки помітному використанню пробілів. Його мовні конструкції та об'єктно-орієнтований підхід мають на меті допомогти програмістам написати чіткий логічний код для малих та масштабних проектів [28].

Python має динамічну типізацію. Він підтримує декілька парадигм програмування, включаючи структуроване (зокрема, процедурне), об'єктно-орієнтоване та функціональне програмування. Python часто описується мовою "батарейки в комплекті" завдяки своїй всебічній бібліотеці стандартів [29].



Рисунок 1.12 – Логотип Python

Інтерпретатори Python доступні для багатьох операційних систем. Глобальне співтовариство програмістів розробляє та підтримує CPython, еталонну реалізацію з відкритим вихідним кодом. Некомерційна організація, Python Software Foundation, керує та спрямовує ресурси для розробки Python та CPython [30].

Мова Python має різноманітні програми в компаніях з розробки програмного забезпечення, таких як ігри, веб-фреймворки та програми, розробка мови, прототипування, додатки графічного дизайну тощо. Це надає мові більшу адаптивність, ніж інші мови програмування, що використовуються в галузі [31]. Деякі з переваг Python:

- **Поширені бібліотеки підтримки.** Він надає великі стандартні бібліотеки, які включають такі сфери, як операції з рядком, Інтернет, інструменти веб-служб, інтерфейси операційних систем та протоколи. Більшість широко використовуваних завдань програмування вже написані в ньому, що обмежує довжину кодів, які потрібно записати в Python.

- **Інтеграційна особливість.** Python інтегрується в корпоративні програми, що полегшує розробку веб-служб шляхом виклику компонентів COM або COBRA. Він має потужні можливості управління, оскільки спілкується безпосередньо з C, C ++ або Java через Jython. Python також обробляє XML та інші мови розмітки, оскільки він може працювати у всіх сучасних операційних системах за допомогою одного й того ж байт коду.

- **Продуктивність.** Завдяки потужній інтеграції процесів, фреймворків тестування блоків та розширеній можливості управління сприяють підвищенню швидкості для більшості застосувань та продуктивності програм. Це чудовий варіант для побудови масштабованих багатопроTOCOLьних мережевих додатків.

Звісно ж як і все у цьому світі Python не ідеальний і у нього є низка недоліків:

- **Складність використання інших мов.** Любителі Python настільки звикають до його особливостей та його широкої бібліотеки, тому вони стикаються з проблемою у навчанні чи роботі над іншими мовами програмування. Експерти Python можуть вважати оголошення "значень" або змінних "типів", синтаксичні вимоги додавання фігурних дужок або крапок з комою як важке завдання.

- **Слабкий у мобільних обчисленнях.** Python виявив свою присутність на багатьох настільних і серверних платформах, але це сприймається як слабка мова для мобільних обчислень. З цієї причини в ньому вбудовано мало мобільних додатків.

- **Повільна швидкість.** Python виконує за допомогою інтерпретатора замість компілятора, що призводить до уповільнення, оскільки компіляція та виконання допомагають нормально працювати.

- **Помилки під час виконання.** Мова Python динамічно типізована, тому вона має багато обмежень щодо дизайну, про які повідомляють деякі розробники Python. Спостережується навіть, що це вимагає більше часу на тестування, і помилки виявляються, коли програми остаточно запускаються.

- **Недорозвинені шари доступу до бази даних.** Порівняно з популярними технологіями, такими як JDBC та ODBC, рівень доступу до бази даних Python виявляється трохи недорозвиненим та примітивним.

1.2.2. Мова програмування Java.

Наступним на черзі проаналізуємо мову програмування Java. Java – це мова програмування загального призначення, заснована на класах, об'єктно-орієнтована і покликана мати якомога менше залежностей від реалізації. Вона призначена для того, щоб розробники додатків могли писати один раз, запускати будь-де (*WORA – write once, run anywhere*), тобто компільований код Java може працювати на всіх платформах, які підтримують Java без необхідності рекомпіляції [32].

Програми Java зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM) незалежно від основної архітектури комп'ютера. Синтаксис Java схожий на C і C ++, але в ньому є менше об'єктів низького рівня. Станом на 2019 рік, Java була однією з найпопулярніших мов програмування, яка використовується згідно з GitHub [33], особливо для веб-додатків клієнт-сервер, з повідомленнями про 9 мільйонів розробників [34].

Однією з цілей дизайну Java є портативність, а це означає, що програми, написані для платформи Java, повинні працювати аналогічно на будь-якій комбінації апаратних та операційних систем з адекватною підтримкою часу виконання. Це досягається шляхом компіляції коду мови Java до проміжного представлення під назвою Java bytecode, а не безпосередньо до специфічного для архітектури машинного коду. Інструкції щодо байт-коду Java аналогічні машинному коду, але вони призначені для виконання віртуальною машиною (VM), написаною спеціально для обладнання хоста. Кінцеві користувачі зазвичай використовують середовище виконання Java (JRE), встановлене на їх машині, для автономних додатків Java або у веб-браузері для Java-апплетів.

На рисунку 1.13 представлені потенційні напрямки використання мови Java.

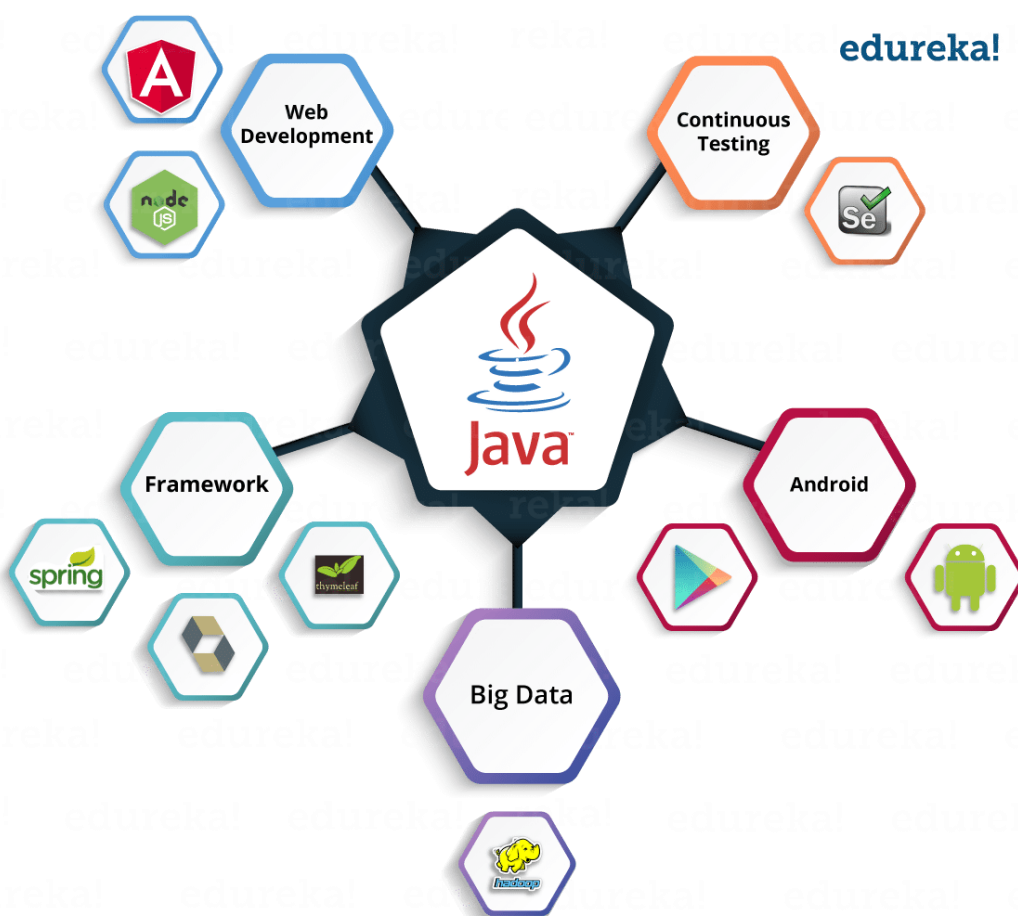


Рисунок 1.13 – Напрямки використання мови програмування Java.

Використання універсального байт-коду робить перенос простим. Однак накладні витрати на інтерпретацію байт-коду в машинні інструкції, що робляться інтерпретованими програмами, майже завжди працюють повільніше, ніж нативні виконувані файли. Just-in-time (JIT) компілятори, що компілюють байт-коди до машинного коду під час виконання, були введені з раннього етапу. Сама Java не залежить від платформи і адаптована до конкретної платформи, на якій вона працює віртуальною машиною Java, яка переводить байт-код Java на машинну мову платформи [35].

Розглянемо основні переваги які надає розробникам Java:

- **Це мова високого рівня.** Це поняття передбачає, що мова програмування більше нагадує людську мову, а не машинну. Отже, писати, читати та підтримувати його слід легко і просто.

- **Стабільність.** Кажуть, що рішення, створені за допомогою Java, є стабільними. Частково це відбувається так, оскільки регулярно випускається нова версія Java з новими функціями з розширеними функціями.

- **Об'єктна орієнтованість.** Оскільки Java належить до об'єктно-орієнтованого програмування, вона дозволяє розробнику писати типові програми та повторно використовувати код. Таким чином, можна констатувати класи, генерувати об'єкти всередині класів, працювати та підтримувати взаємодію між двома об'єктами.

- **Її обслуговування досить дешеве.** Характер роботи програми Java не покладається на якусь унікальну апаратну інфраструктуру, тому запустити сервер можна на будь-якій машині. Результат: це недороге обслуговування.

- **Безпека.** Java - перша технологія, яка надала безпеку як невід'ємну частину дизайну. JVM має спеціальний ідентифікатор, який виявляє байт-код і перевіряє перед запуском [36].

- **Багатопотоковість.** Внутрішньо програма Java може виконувати кілька завдань одночасно.

- **Надійність.** Java є найбільш надійною та потужною мовою. Її компіляторам вдається виявити кожен тип помилок у коді. Крім того, у Java є такі чудові функції, як обробка винятків та `garbage collection`, які також доводять надійність Java.

Наразі усе виглядає чудово, одна не слід забувати про потенційні недоліки які присутні в мові Java:

- **Продуктивність.** Якщо ми порівнюємо програми Java з написаними на C або C++, які є нативними, то легко помітити, що вони дещо повільніші.

- **Витрати на комерційну ліцензію.** Починаючи з 2019 року, Oracle очікує, що користувачі платитимуть за Java Standard Edition 8 при використанні для таких цілей, як бізнес, комерція та виробництво. Тож коли потрібні оновлення та виправлення помилок, доведеться врахувати нові витрати.

- **Багатослівність Java робить код досить складним.** Java натякає, що потрібно вживати багато слів, оскільки вони дуже схожі на природну мову людей. Розробники майже буквально записують свої команди та думки, тому код є надзвичайно величезним (особливо порівняно з Python).

1.2.3. Мова програмування Kotlin.

Тепер розглянемо, наступника, котрим вважається Kotlin у порівнянні із Java. Kotlin — це кросплатформна, статично типізована мова програмування загального призначення з виведенням типів. Kotlin розроблено для повної взаємодії з Java, і версія JVM стандартної бібліотеки Kotlin залежить від бібліотеки класів Java[37], але припущення типів дозволяє її синтаксису бути більш стислим. Kotlin в основному націлений на JVM, але також компілює в JavaScript (наприклад, для інтерфейсних веб-додатків, що використовують React[38]) або нативний код (через LLVM); наприклад, для нативних програм iOS, які поділяють бізнес-логіку з програмами Android.

Керівник розробки Андрій Бреслав сказав, що Kotlin розроблена як промислово потужна об'єктно-орієнтована мова і «краща мова», ніж Java, але все

ще надає повну сумісність з кодом Java, що дозволяє компаніям здійснювати поступовий перехід від Java до Kotlin[39].

Крапка з комою необов'язкова як термінатор оператора; у більшості випадків достатньо нового рядка, щоб компілятор міг зробити висновок, що оператор закінчився. Змінні в Kotlin можуть бути тільки для читання, оголошені за допомогою ключового слова `val` або змінювані, оголошені за допомогою ключового слова `var`[40]. На рисунку 1.14 представлено результат опитування проведеного Stack Overflow, котрий показує що, Kotlin посідає 4 місце у рейтингу найбільш улюблених мов.

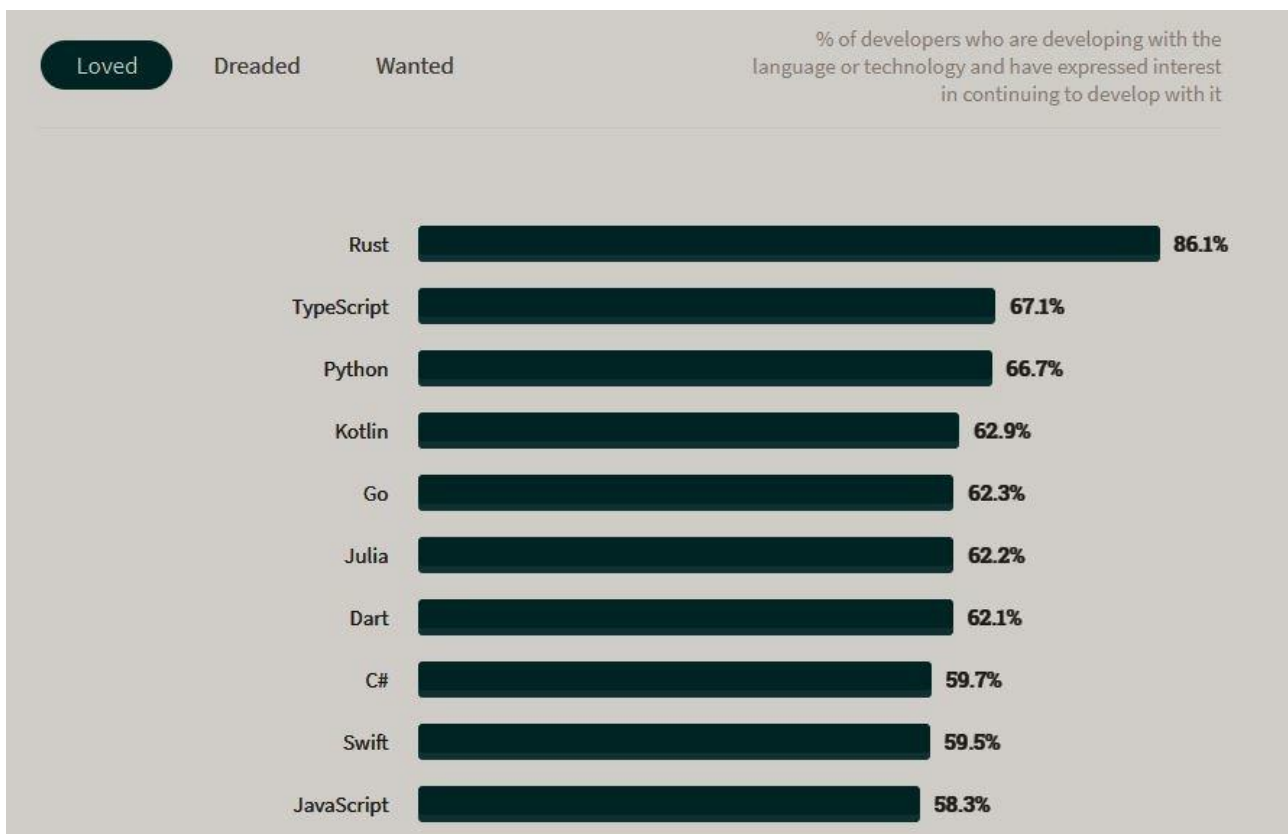


Рисунок 1.14 – Результати опитування проведеного Stack Overflow у 2020 році.

Переваги:

- **Повна сумісність з Java.** З усіма інструментами та фреймворками ви можете просто додати їх до своїх проєктів Kotlin — приємно та легко — без необхідності змінювати весь проєкт на Java.

- **Дозволяє писати менше коду.** Менше коду важливо, але є читабельність, яку також слід розглянути та бажано покращити. З Kotlin ви отримуєте їх обидва. JetBrains зробили все можливе, щоб зробити мову якомога лаконічнішою, і їм це вдалося.

- **Не вимагає накладних витрат під час виконання.** Стандартна бібліотека невелика і щільна: вона складається здебільшого з цілеспрямованих розширень стандартної бібліотеки Java. Інтенсивне використання вбудовування під час компіляції означає компіляцію функціональних конструкцій, таких як конвеєри map/filter/reduce, подібно до обов'язкової версії того самого коду.

- **Розробка Kotlin пропонує більше безпеки.** Розробники Kotlin інтегрували в семантику принципи, які запобігають різноманітним поширеним помилкам, які зазвичай трапляються під час виконання програми. Ще одна причина безпеки Kotlin на вищому рівні (порівняно з Java) полягає в його перевагах[41].

Недоліки:

- **Коливання в компіляції.** У багатьох випадках, як-от виконання інкрементальних збірок, Kotlin швидший за Java, в цьому немає жодних сумнівів. Однак Java залишається безсумнівним переможцем, коли справа доходить до створення чистих збірок для програм Android.

- **Обмежені навчальні ресурси.** Кількість розробників, які переходять на Kotlin, збільшується, але існує лише обмежена спільнота розробників, доступна для вивчення цієї мови або вирішення проблем у процесі розробки[42].

На основі усіх вище перерахованих фактів у якості мови для розробки було обрано Kotlin, обумовлено це тим, що функціонал який надає Kotlin являється доволі широким, що дозволить використовувати існуючі реалізації при побудові додатків. Також не менш важливим фактором став факт того, що Kotlin являється платформо незалежною, а тому не виникне проблем запуску додатку в будь-якому середовищі.

1.3. Аналіз існуючих постачальників Cloud технологій

Хмарні обчислення – це надання різноманітних послуг через Інтернет. Ці ресурси включають інструменти та програми, такі як зберігання даних, сервери, бази даних, мережі та програмне забезпечення.

Замість того, щоб зберігати файли на власному жорсткому диску чи локальному накопичувачі, хмарне сховище дає змогу зберігати їх у віддаленій базі даних. Поки електронний пристрій має доступ до Інтернету, він має доступ до даних і програмного забезпечення для його запуску[43].

Хмарні обчислення є популярним варіантом для людей і компаній з ряду причин, включаючи економію витрат, підвищення продуктивності, швидкість і ефективність, безпеку.

На рисунку 1.15 представлено розподіл Cloud постачальників з якого помітно, що на світовому ринку хмарових технологій домінують чотири постачальники хмарних послуг: Alibaba в Китаї та Азіатсько-Тихоокеанського регіону та AWS, Microsoft і Google в інших країнах.

Особливо домінує AWS. Згідно зі звітом Synergy Research Group за 2020 рік, «зріст Amazon продовжував тісно відображати загальне зростання ринку, тому він зберіг свою частку в 33% світового ринку. Друге місце в рейтингу Microsoft знову зростало швидше, ніж ринок, і його частка на ринку зросла майже на три відсоткові пункти за останні чотири квартали, досягнувши 18%».

- **AWS:** Завдяки величезному набору інструментів, який продовжує зростати в геометричній прогресії, можливості Amazon не мають собі рівних. Проте його структура витрат може бути заплутаною, а його зосередженість на загальнодоступній хмарі, а не на гібридній хмарі чи приватній хмарі означає, що взаємодія з вашим центром обробки даних не є головним пріоритетом AWS[44].

- **Microsoft Azure:** Близький конкурент AWS з надзвичайно потужною хмарною інфраструктурою. Якщо ви корпоративний клієнт, Azure розмовляє вашою мовою – лише деякі компанії мають корпоративне знання (і підтримку

Windows), як Microsoft. Azure знає, що ви все ще керуєте центром обробки даних, і платформа Azure наполегливо працює для взаємодії з центрами обробки даних; гібридна хмара – це справжня сила.

- **Google Cloud:** Добре фінансований аутсайдер у конкурентній боротьбі, Google пізніше вийшов на ринок хмар і не має такої міри зосередженості на підприємстві, яка допомагає залучити корпоративних клієнтів. Але його технічний досвід є глибоким, а його провідні інструменти в галузі глибокого навчання та штучного інтелекту, машинного навчання та аналітики даних є значними перевагами.



Рисунок 1.15 – Розподіл Cloud постачальників на ринку.

Розглянемо детальніше послуги які надає AWS. AWS має понад 175 хмарних сервісів для широкого кола випадків використання та галузей. Найпопулярніші послуги Amazon: Amazon EC2 (обчислювальна потужність), Amazon RDS (реляційна база даних), Amazon S3 (хмарне сховище), Amazon CloudFront (сервіс доставки вмісту) і Amazon Glacier (сервіс веб-сховища). EC2 дозволяє клієнтам Amazon використовувати кластери віртуальних комп'ютерів,

які доступні весь час. Більшість сервісів не доступні безпосередньо для кінцевих користувачів, проте вони надають функціональні можливості через API, які розробники можуть використовувати в додатках.

Комісія AWS залежить від вибраного обладнання та програмного забезпечення або мережевих параметрів. Абоненти можуть платити за один віртуальний комп'ютер AWS або комп'ютерний кластер. На умовах підписки Amazon забезпечує безпеку замовлених систем.

AWS має найбільшу частку ринку в IaaS і PaaS і є лідером у більшості хмарних продуктів, що пропонуються в усьому світі. Великі підприємства розгортають важливі для бізнесу робочі навантаження на Amazon частіше, ніж у будь-якого іншого хмарного постачальника. Компанія має потужну мережу керованих постачальників послуг із 67 провідними консультативними партнерами по всьому світу. Підприємства сприймають AWS як стратегічного постачальника хмарної інфраструктури. AWS надає комплексні рішення, починаючи від серверів і закінчуючи вбудованими операційними системами на пристроях Edge, а між ними — всеосяжну технологічну статистику[45].

AWS — це найзріліший і готовий до підприємств постачальник із впливовим послужним списком успіхів клієнтів, починаючи від малого та середнього бізнесу до великих підприємств. Підприємства, які використовують Amazon, постійно отримують вигоду, будучи ранніми користувачами нових послуг. Amazon Web Services пропонує понад 100 продуктів, які користувачі можуть протестувати безкоштовно. Компанія надає набір баз даних, інструментів і сервісів для розробників і мобільних пристроїв, які завжди безкоштовні. Крім того, AWS пропонує 1-річну пробну версію для певних продуктів і короткі безкоштовні пробні послуги для машинного навчання, аналітики, обчислень, безпеки та відповідності.

Враховуючи вище зазначені фактори, а також попередній досвід, для розробки то розгортання проекту було обрано саме AWS, з використанням

DynamoDB(для збереження ключових даних про записи, та надання можливості пошуку), Lambda(для побудови Serverless додатку), S3(для збереження файлів), CloudFront(для трансферу FrontEnd даних) та ApiGateway(для створення комунікації між FrontEnd та BackEnd).

Висновки до розділу 1

У цьому розділі ми розглянули особливості бази даних їх види і зв'язки. Відповідно до проведеного аналізу було прийнято рішення використовувати DynamoDB. Також було розглянуто мови програмування такі як Python, Java, Kotlin. На основі усіх вище перерахованих фактів у якості мови для розробки було обрано Kotlin, обумовлено це тим, що функціонал який надає Kotlin являється доволі широким, що дозволить використовувати існуючі реалізації при побудові додатків.

Ключовим архітектурним рішенням у процесі реалізації проекту являється використання хмарних технологій, що дозволить мати сервер для розгортання проекту, надасть централізоване місце для використання необхідних сервісів для побудови проекту, а також дозволить знизити затрати на підтримку серверу.

Мета роботи - створення бази даних для автоматизованої системи навчально-методичних матеріалів кафедри

Для досягнення поставленої мети роботи вирішуються наступні задачі:

1. Аналіз сучасних засобів побудови баз даних.
2. Вибір мови розробки програмного забезпечення та Cloud-технології.
3. Програмна реалізація серверної та клієнтської частини.
4. Розгортання проекту в хмарному середовищі.
5. Розробка стартап проекту.

Об'єкт дослідження - база даних автоматизованої системи навчально-методичних матеріалів кафедри

Предмет дослідження - програмне забезпечення для доступу до бази даних автоматизованої системи навчально-методичних матеріалів кафедри.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИН ПРОЕКТУ.

2.1. Розробка серверної частини застосунку.

В першу чергу розглянемо взаємодію додатку з базою даних, а саме реалізацію DAO рівня. Шаблон об'єкта доступу до даних (DAO – data access object) — це структурний шаблон, який дозволяє нам ізолювати рівень програми/бізнесу від рівня збереження (зазвичай це реляційна база даних, але може бути будь-яким іншим механізмом збереження) за допомогою абстрактного API. Для взаємодії з DynamoDB використовується AWS Java SDK 2.x[46].

Розглянемо як реалізовано DAO рівень для взаємодії із основною сутністю проекту:

Лістинг 2.1 – Робота з рівнем бази даних

```
@Singleton
class BookEntryDAO(client: DynamoDbEnhancedClient) :
    SingleKeyCrudRepository<BookEntry>(client) {
        override val table: DynamoDbTable<BookEntry> =
            enhancedClient.tableFromTableName("bookEntry".toFullAwsName())
    }
```

На перший погляд може здатися, що налаштування класу досить прості, але це враження оманливе, оскільки за таким простими строками коду ховається досить суттєва логіка. Ця логіка була винесена у зовнішню бібліотеку оскільки її можливо узагальнити та використовувати скрізь у проекті. Але про все по порядку.

Спершу розглянемо клас від якого наслідується **BookEntryDAO**. Клас **SingleKeyCrudRepository** являється узагальненим класом і надає базовий функціонал(такий як вилучення та видалення елемента за його унікальним ідентифікатором) для роботи з об'єктами котрі мають лише первинний ключ.

Лістинг 2.2 – Абстрактний клас, що надає загальний функціонал для роботи з об'єктами з первинним ключем.

```
abstract class SingleKeyCrudRepository<T> : CrudRepository<T>
{
```

```

        constructor() : super()
        constructor(enhancedClient: DynamoDbEnhancedClient) :
super(enhancedClient)

```

```

        open fun getById(partitionKeyId: String): T? =
table.getByKey(partitionKeyId)

```

```

        open fun deleteById(partitionKeyId: String): T? =
table.deleteByKey(partitionKeyId)
    }

```

Клас **SingleKeyCrudRepository** в свою чергу наслідується від абстрактного класу **CrudRepository** котрий надає основні функції для роботи з об'єктами незалежно від структури унікального ключа об'єкту, а саме збереження, видалення, пошук по глобальному вторинному індексу, отримання всіх об'єктів, що є в базі даних, а також тут оголошується абстрактне поле таблиці, саме через які усі операції і проводяться у SDK 2.x.

Лістинг 2.3 – Абстрактний клас, що надає базовий функціонал для роботи з базою даних.

```

abstract class CrudRepository<T> {
    protected var enhancedClient: DynamoDbEnhancedClient

    constructor(enhancedClient: DynamoDbEnhancedClient) {
        this.enhancedClient = enhancedClient
    }

    /**
     * Should only be used for testing to create spies.
     */
    constructor() {
        this.enhancedClient = DynamoDbEnhancedClient.builder()
            .dynamoDbClient(BeanProviderForTest.getDbClient())
            .build()
    }

    /**
     * table to be initialized later in service implementation
     */
    protected abstract val table: DynamoDbTable<T>

    open fun save(item: T): Unit = table.putItem(item)

    open fun delete(item: T): T = table.deleteItem(item)

```

```

        open fun getAll(): List<T> = table.scan().flatMap {
it.items() }

    /**
     * provides a search by a secondary partition key
     */
    open fun findByIndex(indexName: String, indexValue:
String): List<T> = table.findByGSIValue(indexName, indexValue)
}

```

Варто зазначити, що у обох класах усі функції декларовані із ключовим словом *open*, це робить функції відкритими для перевизначення у дочірніх класах. Такий крок дозволить, у разі необхідності, наприклад, перевизначити логіку збереження об'єкту, якщо необхідно якісь додаткові перевірки, однак здебільшого функції являються відкритими задля спрощення процесу тестування у разі необхідності створення моку або заглушки.

Наразі розглянемо функцію *tableFromTableName*. Дана функція викликається на об'єкті класу *DynamoDbEnhancedClient*, однак в офіційній документації таку функцію не знайти адже тут використано одну із особливостей мови програмування Kotlin, а саме функції розширення, тим самим створюється враження, що це реальний метод класу, коли насправді це не зовсім так.

Лістинг 2.4 – Функція розширення для створення таблиці.

```

/**
 * [DynamoDbEnhancedClient] extension function that is used to
 * generate table for [T] with provided table name.
 *
 * Method relies entirely on [BeanIntrospection] in order to
avoid the use of reflection
 *
 * @param T type of the modelled object
 * @param tableName name of the table that will be created
 * @author Mykhailo Bondarenko
 * @since 1.0
 */
inline fun <reified T : Any>
DynamoDbEnhancedClient.tableFromTableName(tableName: String):
DynamoDbTable<T> {
    return table(tableName,
tableSchemaFromBeanIntrospection(BeanIntrospection.getIntrospection
(T::class.java)))
}

```


Даний метод це *inline* функція з *reified* узагальненим параметром, що означає, що по-перше, після компіляції такої функції не існуватиме так як її тіло буде вставлена у кожне місце її виклику, по друге, реальний тип узагальненого параметра завжди відомий і не затирається під час виконання програми, що дозволяє вилучити із нього необхідну інформацію.

Наразі розглянемо `tableSchemaFromBeanIntrospection`, котрий і створює таблицю для проведення операцій з базою даних.

Лістинг 2.5 – Логіка створення таблиці для певного класу.

```
@PublishedApi
internal inline fun <reified T : Any>
tableSchemaFromBeanIntrospection(beanIntrospection:
BeanIntrospection<T>): TableSchema<T> {
    val schemaBuilder =
TableSchema.builder(T::class.java).newItemSupplier(beanIntrospectio
n::instantiate)

        beanIntrospection.beanProperties.forEach { beanProperty ->
            val enhancedType =
enhancedTypeFromBeanProperty(beanProperty.asArgument())
            schemaBuilder.addAttribute(enhancedType) {
                it.name(beanProperty.name)
                it.getter(beanProperty::get)
                it.setter(beanProperty::set)
                when {

beanProperty.hasAnnotation(DynamoDbPartitionKey::class.java) ->

it.addTag(StaticAttributeTags.primaryPartitionKey())

beanProperty.hasAnnotation(DynamoDbSortKey::class.java) ->

it.addTag(StaticAttributeTags.primarySortKey())

beanProperty.hasAnnotation(DynamoDbSecondaryPartitionKey::class.jav
a) ->

it.addTag(StaticAttributeTags.secondaryPartitionKey(getGSINames(bea
nProperty)))
                }
                it.attributeConverter(
                    when (enhancedType ==
EnhancedType.of(Any::class.java)) {
```

```

        true -> handleAnyConverter(beanProperty,
enhancedType)
        else ->
converterForEnhancedType(enhancedType)
    }
    )
    }
    }

return schemaBuilder.build()
}

```

Представлена функція проводить усю необхідну обробку для створення таблиці, а саме, визначення первинного та вторинного(якщо наявний) ключів, визначення глобальних вторинних індексів, надання конверторів для нестандартних типів даних, обробка узагальнених типів, а також створення таблиць для вкладених типів. Вилучення усього цього функціоналу у зовнішню бібліотеку дозволяє пришвидшити та спростити процес розробки застосунка.

Наступним розглянемо завантаження файлів на сервер та завантаження файлів із серверу. Первинна реалізація даного функціоналу покладалася на пряме завантаження файлів через проксі лямбду в ApiGateway, однак обширне тестування з файлами різних розмірів показує, що подібний підхід не здатний обробляти запити з файлами розмір яких перевищує 10Мб[47].

Вирішення цього питання було досягнуто за допомогою так званих *presigned URL(попередньо підписане посилання)*. Усі об'єкти у S3 бакеті за замовчуванням є приватними. Тільки власник об'єкта має дозвіл на доступ до цих об'єктів. Однак власник об'єкта може за бажанням ділитися об'єктами з іншими, створюючи *presigned URL*, використовуючи власні облікові дані безпеки, щоб надати обмежений у часі дозвіл на завантаження об'єктів.

Слід зазначити, що *presigned URL* можливо згенерувати у будь якій лямбді, однак, якщо ця лямбда не матиме відповідного доступу до S3 бакету, то при спробі використати отримане посилання станеться помилка доступу, саме тому для запобігання такому сценарію необхідно надати лямбді наступні права:

Лістинг 2.5 – Права доступу до бакету.

```
{
  "Action": [
    "s3:GetObject*",
    "s3:GetBucket*",
    "s3:List*",
    "s3:DeleteObject*",
    "s3:PutObject",
    "s3:Abort*"
  ],
  "Resource": [
    "arn:aws:s3:::{bucket_name}",
    "arn:aws:s3:::{bucket_name}/*"
  ],
  "Effect": "Allow"
}
```

Процес створення presigned URL в коді виглядає наступним чином:

Лістинг 2.6 – Логіка генерації попередньо підписаного посилання для завантаження файлу з серверу.

```
fun getDownloadUrl(bookId: String):
PresignedGetObjectRequest {
    val bookEntry = bookEntryDAO.getById(bookId) ?: throw
BookNotFoundException("No book found with id $bookId")
    App.logger.debug("Found book entry - {}", bookEntry)
    val fileLocation = bookEntry.fileLocation
    ?: throw NoArgumentProvidedException("Book Entry
with ID '{}' does not have attached file")

    val context = if (fileLocation.endsWith(".pdf"))
"inline"
    else "attachment"
    return s3Presigner.presignGetObject {
        it.getObjectRequest { req ->
            req.bucket(App.FILE_UPLOAD_BUCKET)
            req.key(fileLocation)

req.responseContentType(fileTypeToFileInfo[fileLocation.split(".").
last()]!!.contentType)

req.responseContentDisposition("$context;filename=${fileLocation.sp
lit("/")}.last()}")
        }
        it.signatureDuration(Duration.ofMinutes(2))
    }
}
```

Першим кроком ми вилучаємо з бази даних запис для якого ми бажаємо завантажити файл. Далі проводиться перевірка чи асоціюється з цим записом файл, якщо ні, ми викидаємо виключення. Дана перевірка проводиться у якості BackEnd валідації, хоча зі сторони FrontEnd не можливо зробити запит на завантаження, якщо запис не має файлу. Як показано на рисунку 2.1, для записів без файлу відсутня кнопка яка дозволяє завантажити файл.


| | | | |
|---|---------------|------|---|
| Модальное управление и наблюдающие устройства | Кузовков Н.Т. | TAY |  |
| Year Publication Test | Me | 2019 | Year, No File |

Рисунок 2.1 – Наявність та відсутність кнопки завантаження файлу.

Останнім кроком перед створенням `presigned URL` є визначення диспозиції змісту, якщо файл має розширення PDF, це означає, що його можливо переглянути у вікні браузера перед завантаженням, тому для таких файлів диспозиція змісту буде *inline*, для всіх інших розширень диспозиції буде *attachment* для того щоб одразу виконати завантаження.

Завантаження файлів на сервер відбувається схожим шляхом, однак в цьому випадку використовується PUT запит:

Лістинг 2.7 – Логіка генерації попередньо підписаного посилання для завантаження файлу на сервер.

```
fun getUploadUrl(fileName: String): PresignedPutObjectRequest
{
    val fileExtension = fileName.split(".").last()
    val fileInfo = fileTypeToFileInfo[fileExtension] ?:
    throw UnsupportedOperationException("Unknown file type")

    return s3Presigner.presignPutObject {
        it.putObjectRequest { req ->
            req.bucket(App.FILE_UPLOAD_BUCKET)
            req.key("${fileInfo.s3Folder}/${fileName}")
            req.contentType("application/octet-stream")
        }
        it.signatureDuration(Duration.ofMinutes(10))
    }
}
```

В запит передається ім'я файлу, по якому визначається в котрій директорії він буде зберігатися, а тип вмісту завжди вказується як *application/octet-stream*, оскільки в даному випадку точне визначення типу не надає ніякої користі, тому усі завантаження розглядаються як узагальнений набір бінарних даних.

Наразі розглянемо процес обробки користувацьких запитів сервером. Обробка запитів бере свій початок у класах котрі являються контролерами, це такі класи, методи яких, середовище AWS Lambda викликає для виконання операції обробки. В більшості випадків у якості вхідного запиту отримуємо *InputStream*, тобто упорядкований потік байтів, який потім конвертується у зручний формат даних, а саме у клас *ApiGatewayInput*, з якого в подальшому витягуються дані для обробки запита.

Серверна логіку можливо розділи на 3 ключові напрямки:

- Запити що відносяться до публікацій;
- Запити що відносяться до користувачів;
- Запити що відносяться до файлів.

Враховуючи таке розподілення маємо 3 контролери котрі оброблюють відповідні запити: *BookController*, *FileUrlController*, *UserController*. Оскільки перші етапи здебільшого схожі для усіх контролерів, то вони були винесені у абстрактний клас *GenericController*:

Лістинг 2.8 – Абстрактний клас, що надає загальний функціонал для усіх контролерів.

```
sealed class GenericController<O : Any> :  
MicronautRequestHandler<InputStream, Any> {  
    constructor() : super()  
    constructor(context: ApplicationContext) : super(context)  
  
    protected val mapper: ObjectMapper = getBean()  
  
    abstract fun processRequest(input: InputStream): O  
  
    override fun execute(input: InputStream): Any {  
        return try {  
            processRequest(input).also {
```

```

        App.logger.info("Response - {}", it)
    }
} catch (ex: Exception) {
    App.logger.warn("Exception happened during request
execution - {}", ex)
    if (ex is CoreException) {
        GatewayResponse(
            ""{"message": "${ex.message}"}"",
            Headers.createHeaders(),
            ex.statusCode
        )
    } else {
        GatewayResponse(
            ""{"message": "Internal Server
Error"}"",
            Headers.createHeaders(),
            500
        )
    }
}

override fun convertInput(input: Any?): InputStream {
    return if (input is InputStream) input
    else
mapper.writeValueAsString(input).byteInputStream()
}

protected inline fun <reified T> getBean(): T =
this.applicationContext.getBean(T::class.java)
}

```

В абстрактному класі окрім конвертації вхідних даних, також відбувається обробка винятків, тобто, якщо в ході виконання запиту виникла помилка, то ця помилка буде спіймана, а користувачу повернеться повідомлення котре проінформує його про проблему. В цілому помилки які виникають в ході обробки запита можливо поділити на 2 типи: очікувані та неочікувані.

Виникнення очікуваних помилок регламентовано в коді програми у явному вигляді і здебільшого вони виникають через дані котрі надсилає користувач, наприклад якщо користувач намагається здійснити дію при цьому не маючи відповідних прав доступу, або коли дані які відправляє користувач не проходять валідацію. В таких випадках в коді викидається виняток із відповідним

повідомленням та статус кодом(400-ті статус коди котрі відносяться до типу клієнтських помилок) які буду повернені користувачу, в подальшому користувач може відкоригувати дані запиту і повторити спробу.

Неочікувані помилки у явному вигляді в коді програми не виникають і здебільшого зв'язані з перебоями у взаємодії програми із зовнішніми ресурсами будь то база даних чи бакет збереження файлів. Повідомлення що супроводжує дані помилки, в більшості випадках, занадто технічне і користувачеві не обов'язково його показувати. Саме тому усі неочікувані помилки повертаються користувачу з повідомленням *Internal Server Error* та 500 статус кодом.

Окрім вище вказаних помилок можуть виникнути проблеми пов'язані із постачальником Cloud технологій, як то трапилося у кінці 2020 року з відключенням сервісів AWS. Подібні ситуації нажаль неможливо передбачити і підготуватися заздалегідь також не вийде, оскільки в такому випадку запити на сервер навіть не надходять. Залишається лише сподіватися, що проблеми зі сторони постачальник будуть вирішені швидко.

Розглянемо більш детально процес обробки користувацьких запитів на прикладі запиту для отримання останніх доданих публікацій. Як вже зазначалося вище, процес починається в контролері, в даному випадку в класі *BookController*:

Лістинг 2.9 – Логіко контролеру для роботи з записами у базі даних.

```
@Introspected
class BookController : GenericController<GatewayResponse> {
    constructor()
    constructor(context: ApplicationContext) : super(context)

    private val bookBean: BookBean = getBean()
    private val userBean: UserBean = getBean()

    override fun processRequest(input: InputStream):
GatewayResponse {
        val apiInput =
mapper.readAndLogInput<ApiGatewayInput>(input)

        return when (apiInput.resource) {
            "/books" -> getAll(apiInput)
            "/books/latest" -> getLatest(apiInput)
        }
    }
}
```

```

        "/books/page" -> getBooksPage(apiInput)
        "/books/{id}" -> {
            when(apiInput.httpMethod) {
                "GET" -> bookById(apiInput)
                "DELETE" -> deleteBookById(apiInput)
                else -> throw
                    RuntimeException("Unsupported method for this path")
            }
        }
        "/createEntry" ->
createOrUpdateBookEntry(apiInput)
        "/updateEntry" ->
createOrUpdateBookEntry(apiInput)
        "/search" -> performSearch(apiInput)
        else -> throw RuntimeException("Unknown request
path")
    }
}

private fun getAll(input: ApiGatewayInput):
GatewayResponse {
    val books = bookBean.getAllBooks()

    return response200(books, input)
}

private fun getLatest(input: ApiGatewayInput):
GatewayResponse {
    val books = bookBean.getLatest()

    return response200(books, input)
}

private fun getBooksPage(input: ApiGatewayInput):
GatewayResponse {
    val page =
input.queryStringParameters?.get("page")?.toInt() ?:
noArgError("Page is not specified")
    val pageDTO = bookBean.getBooksPage(page)

    return response200(pageDTO, input)
}

private fun bookById(input: ApiGatewayInput):
GatewayResponse {
    val bookId = input.pathParameters?.get("id") ?:
noArgError("No id provided")

    val book = bookBean.getById(bookId)

```



```

        return response200(book, input)
    }

    private fun deleteBookById(input: ApiGatewayInput):
GatewayResponse {
        val bookId = input.pathParameters?.get("id") ?:
noArgError("No id provided")

        val deletedBook = bookBean.deleteById(bookId)
        return response200(deletedBook, input)
    }

    private fun createOrUpdateBookEntry(input:
ApiGatewayInput): GatewayResponse {
        (input.getTokenFromCookies("token") ?:
forbiddenError("Authentication token is not present in
request")).apply {
            userBean.verifyUserRoles(this, UserRole.ADMIN)
        }

        val fileName =
mapper.readTree(input.body!!)["fileName"]?.textValue()
        val bookEntry =
mapper.readValue<BookEntry>(input.body!!)

        val book = bookBean.createOrUpdateBookEntry(bookEntry,
fileName)
        return response200(book, input)
    }

    private fun performSearch(input: ApiGatewayInput):
GatewayResponse {
        val searchMode =
input.queryStringParameters?.get("searchMode") ?:
noArgError("Search mode is not specified")

        val result =
bookBean.performSearch(SearchMode.valueOf(searchMode),
input.queryStringParameters!!)
        return response200(result, input)
    }

    private fun response200(result: Any?, input:
ApiGatewayInput) = GatewayResponse(
        mapper.writeValueAsString(result),
        Headers.createHeaders(input),
        200
    )
}

```

Першим кроком при виконанні запита виконується визначення ресурсу, тобто шляху, котрий запитує користувач, в даному випадку нас цікавить шлях */books/latest*, після чого логіка обробки переходить у приватний метод, де по необхідності дістаються дані з запиту. Далі процес обробки переходить на рівень сервісу де в подальшому проводяться необхідні валідації і вже потім витягуються дані з бази даних.

Варто зазначити, що кожен контролер має бути анотований анотацією *@Introspected*, використання якої дозволяє фреймворку Micronaut згенерувати метадані про клас, що в свою чергу дозволяє уникнути використання рефлексії в процесі виконання програми. Відмова від використання рефлексії, в свою чергу, суттєво поліпшує процес налаштування створення Native Image.

Наступник ключовим елементом серверної частини являється функціонал пошуку існуючих записів у базі даних. Пошук публікацій реалізований за допомогою сканування з фільтруючими виразами у DynamoDB. Якщо потрібно додатково звузити результати сканування, існує можливість додатково надати вираз фільтра. Вираз фільтра визначає, які елементи в результатах сканування мають бути повернуті. Усі інші результати відхиляються. Запит для пошуку за ім'ям реалізований наступним чином:

Лістинг 2.10 – Приклад здійснення запиту пошуку записів у базі даних за назвою запису.

```
fun searchByName(name: String): List<BookEntry> {
    App.logger.debug("Looking for entries with name - {}",
name)
    val res = table.scan(
        ScanEnhancedRequest.builder()
            .filterExpression(
                Expression.builder()
                    .expression("contains(#name, :name)")
                    .putExpressionName("#name", "name")
                    .putExpressionValue(":name",
AttributeValue.builder().s(name).build())
                    .build()
            )
            .build()
    )
}
```

```
)
    return res.flatMap { it.items() }
}
```

За цим запитом ми проводим пошук в якому перевіряється, що заданий користувачем набір символів присутній у назві запису у базі даних, перевірка проводиться за допомогою вбудованого виразу `contains`. Зазначимо, що слово *name*, при проведенні сканування являється зарезервованим, саме тому його необхідно екранувати знаком решітки. Пошук елементів за ім'ям автора відбувається схожим чином: `expression("contains(author, :name) or contains(coAuthors, :name)")`, відмінність полягає тільки у тому, що у випадку з автором пошук відбувається по двом полям запису – автор та спів автори.

Один із останніх, але не найменш важливих, функціоналів реалізованих в проєкт це користувачі. В загальному випадку логіка проєкту побудована з розрахунком на те, що користувачу не доведеться реєструватися задля отримання доступу до якого функціоналу(хоча при необхідності це можливо зробити). В той же час виникає необхідність додавати нові записи до бази даних, оскільки без контенту тут нічого робити. Саме тому були створені користувачі адміністратори, головна роль яких полягає у наповненні додатку контентом, тобто створення нових записів у базі даних, видалення та оновлення існуючих записів за необхідністю.

Розглянемо процес створення користувача. Перший крок при створенні користувача, це переконатися, що користувач створюється з унікальною поштою та ім'ям користувача. Для досягнення цієї мети для ім'я та пошту користувача створюються глобальні вторинні індекси всередині таблиці користувачів у базі даних. Використання індексів дозволить за логарифмічний час перевірити чи є в системі користувач з таким ім'ям чи поштою. Якщо перевірка пройшла успішно і введені користувачем дані являються унікальним, то процес переходить наступного кроку перед збереженням користувача, а саме хешування пароллю.

Хешування паролю захищає від можливості того, що хтось, хто отримує несанкціонований доступ до бази даних, зможе отримати паролі кожного користувача в системі. Хешування виконує одностороннє перетворення пароля, перетворюючи пароль в інший рядок, який називається хешованим паролем. «Односторонній» означає, зворотній процес, перетворення хешованого паролю назад на оригінальний пароль практично неможливо.

У якості функції хешування використовується `bcrypt`, котрий являється стандартним алгоритмом хешування в `OpenBSD`. Фактор вартості при хешуванні становить 12, тобто виконується 4096 раундів шифрування, після чого користувач зберігається до бази даних і його можливо використовувати у системі.

При спробі увійти до системи користувач вводить у форму свою пошту, або ім'я користувача, та пароль і запит відправляється на сервер, де спершу проводиться пошук збігу пошту, після чого використовуючи `bcrypt` верифікатор проводиться порівняння паролів. Якщо дані надіслані користувачем валідні, у відповідь сервера буде додано `cookie` у якому знаходиться `jwt` токен користувача який дозволить йому здійснювати операції корті потребують авторизації, або наявності певної ролі у користувача.

Варто зазначити, що токен повертається користувача з флагом `HttpOnly`, що робить токен недоступним для JavaScript коду у клієнтському браузері, як показано на рисунку 2.2. Така міра дозволяє уникнути ризику XSS атак на сервер, що відповідно сприяє захищеності застосунка.

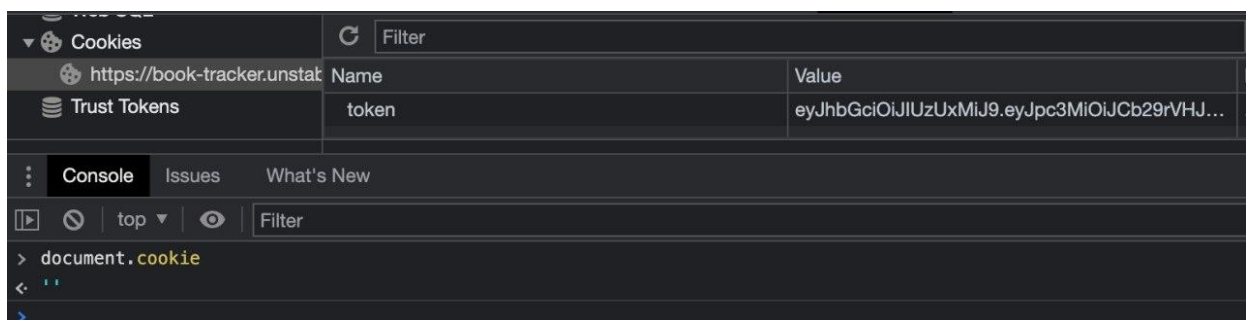


Рисунок 2.2 – Неможливість отримати токен у JS коді.

2.2. Розробка візуальної частини застосунку.

Розробка візуальної частини проекту, тобто FrontEnd-у, проводиться з використанням бібліотеки React та мови TypeScript. Корінним елементом в даному проекті являється компонент App, в якому створюються ключові елементи для взаємодії з застосунком:

Лістинг 2.11 – Основний компонент клієнтської частини додатку.

```
const App: React.FC = () => {
  const [loading, setLoading] = useState(true);
  const [roles, setRoles] = useState<UserRole[]>([]);
  const {i18n} = useTranslation()
  const fetchUserRoles = async () => {
    return await SERVER.get("roles");
  };

  const initLanguage = useCallback(() => {
    const lang = window.localStorage.getItem("language") as
LanguageChoice ?? "ua";
    i18n.changeLanguage(lang);
  }, [i18n]);

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

  useEffect(() => {
    fetchUserRoles().then(res => {
      setRoles(res.data.roles);
      setLoading(false);
    });
  }, []);

  return (
    <BrowserRouter>
      {loading ? <div/> :
        <UserRoleContext.Provider value={{roles, setRoles}}>
          <Header/>
          <Sidebar/>
          <Root sx={{mt: "1rem"}}>
            <Routes>
              <Route path="/" element={<BooksTable/>}/>
              <Route path="/search" element={<SearchPage/>}/>
              <Route path="/createEntry"
element={<BookForm/>}/>
            </Routes>
          </Root>
        }
      </BrowserRouter>
  )
}
```

```

        </UserRoleContext.Provider>
    </BrowserRouter>
);
};

```

В даному компоненті створюється контекст ролей користувача, на основі котрих створюється умовна логіка відображення компонентів. Всередині контексту одразу створюються хедер, бічна панель, та прописуються доступні шляхи навігації. При відкритті посилання користувача зустрічає сторінка зображена на рисунку 2.3.

На головній сторінці користувачу відображається 10 найбільше релевантних, за часом додання, записів із бази даних. Окрім цього з бічної панелі користувач має можливість обрати мову, або перейти на сторінку пошуку.

На бічній панелі також присутнє спадне меню вибори мови відображення, де користувач може обрати українську чи англійську мову. На рисунку 2.4 представлено вибір мови.

Book Tracker

Home

Login

Q

Search

+

Submit Request

English

| Name | Authors | Published on | Tags |
|---|---|--------------|-----------------------------------|
| Металічні корисні копалини України | Грінченко О.В., Курило М.В., Михайлов В.А., Михайлова Л.С. та ін. | 2000 | Месторождения полезных ископаемых |
| Железисто-кремнистые формации Украинского щита. Т. 1. | Семененко Н.П. под ред. | 1978 | Месторождения полезных ископаемых |
| Модальное управление и наблюдающие устройства | Кузовков Н.Т. | | TAV |
| Year Publication Test | Me | 2019 | Year, No File |
| Docx upload | Me | 2021-10-23 | Doc, Not Pdf, test |
| Update Test | Me, Another Me, Someone Else | 2021-10-22 | Doc, Not Pdf, test |
| Another Actual FE Test | Me | 2021-10-22 | Test, FE, S3 |
| Actual FE Updated | Me | 2021-10-20 | FE, Test, PDF |
| New Approach | Me | 2021-10-19 | Test, PDF |
| Generic Invoice | Me | 2021-10-01 | Invoice, PDF |

Рисунок 2.3 – Головна сторінка застосунку.

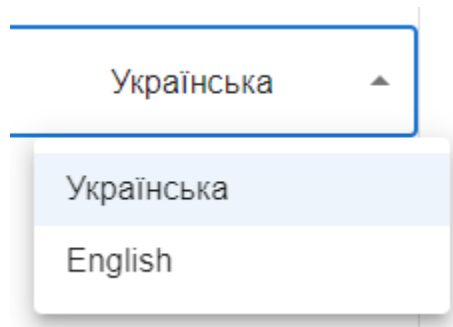


Рисунок 2.4 – Спадає меню вибори мови

Вихідний код компонента бічної панелі виглядає наступним чином:

Лістинг 2.12 – Логіка компоненту бічної панелі.

```
const Sidebar: React.FC = () => {
  const {roles} = useContext(UserRoleContext)
  const [language, setLanguage] =
    useState<LanguageChoice>("ua");
  const {t, i18n} = useTranslation();

  const initLanguage = useCallback(() => {
    const lang = window.localStorage.getItem("language") as
    LanguageChoice ?? "ua";
    setLanguage(lang);
  }, []);

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

  useEffect(() => {
    window.localStorage.setItem("language", language);
  }, [language]);

  const navigate = useNavigate();

  const navigateToAddEntry = () => navigate("/createEntry");
  const navigateToSearch = () => navigate("/search");

  return (
    <Drawer
      anchor="left"
      variant="permanent"
    >
      <Toolbar/>
      <Box sx={{overflow: "auto", width: "13rem"}}>
        <List>
          <ListItem button onClick={navigateToSearch}>
            <ListItemIcon>
```

```

        <SearchIcon/>
    </ListItemIcon>
    <ListItemText primary={t("Search")}/>
</ListItem>
</List>
<Divider/>
{roles.includes("ADMIN") ?
    <List>
        <ListItem button onClick={navigateToAddEntry}>
            <ListItemIcon>
                <AddIcon/>
            </ListItemIcon>
            <ListItemText primary={t("Add Entry")}/>
        </ListItem>
    </List> :
    <List>
        <ListItem button onClick={() => alert("Work in
progress")}>
            <ListItemIcon>
                <AddIcon/>
            </ListItemIcon>
            <ListItemText primary={"Submit Request"}/>
        </ListItem>
    </List>
}
<Select
    sx={{
        mt: "auto",
        textAlign: "center"
    }}
    fullWidth
    value={language}
    onChange={(e) => {
        setLanguage(e.target.value as LanguageChoice);
        i18n.changeLanguage(e.target.value);
    }}
>
    <MenuItem value={"ua"}>Українська</MenuItem>
    <MenuItem value={"en"}>English</MenuItem>
</Select>
</Box>
</Drawer>
);
};

```

У компоненті бічної панелі присутня логіка яка дозволяє зберегти обрану користувачем мову між сесіями використання застосунку. За замовчуванням обирається українська мова, однак при виборі користувачем іншої мови, обране

значення зберігається у *localStorage* і при відкритті сторінки знову перше, що відбувається це перевірка наявності ключа мови і якщо він є, то інтерфейс застосунку буде відображено з обраною мовою. Зміна мови представлена на рисунку 2.5.

Окрім цього в бічній панелі також присутня логіка відображення доступних розділів для звичайних користувачів та адміністраторів. Наприклад, якщо користувач являється адміністратором, то у пунктах бічної панелі окрім секції пошуку буде присутня секція додання нового запису до бази даних. Відповідно для звичайних користувачів відсутня можливість додання записів, оскільки надання такої можливості звичайним користувачам може призвести до потраплення до бази даних записів які не мають жодного відношення до наукових матеріалів. Бічна панель адміністратора представлена на рисунку 2.6.

Book Tracker

Головна

Увійти

Пошук

Подати запит

Українська

| Назва | Автори | Дата публікації | Теги | |
|---|---|-----------------|-----------------------------------|--|
| Металічні корисні копалини України | Грінченко О.В., Курило М.В., Михайлов В.А., Михайлова Л.С. та ін. | 2000 | Месторождения полезных ископаемых | |
| Железисто-кремнистые формации Украинского щита. Т. 1. | Семененко Н.П. под ред. | 1978 | Месторождения полезных ископаемых | |
| Модальное управление и наблюдающие устройства | Кузовков Н.Т. | | TAV | |
| Year Publication Test | Me | 2019 | Year, No File | |
| Docx upload | Me | 2021-10-23 | Doc, Not Pdf, test | |
| Update Test | Me, Another Me, Someone Else | 2021-10-22 | Doc, Not Pdf, test | |
| Another Actual FE Test | Me | 2021-10-22 | Test, FE, S3 | |
| Actual FE Updated | Me | 2021-10-20 | FE, Test, PDF | |
| New Approach | Me | 2021-10-19 | Test, PDF | |
| Generic Invoice | Me | 2021-10-01 | Invoice, PDF | |

Рисунок 2.5 – Зміна тексту в компонентах в залежності від обраної мови.

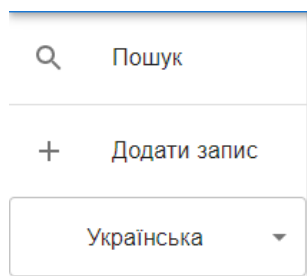


Рисунок 2.6 – Бічне меню адміністратора

Наразі розглянемо процес авторизації користувачів. Користувач може увійти натиснувши кнопку *Login*, котра знаходиться у правому верхньому кутку користувацького інтерфейсу, після чого відкривається форма авторизації, представлена на рисунку 2.8. Поля логіну та паролю являється обов’язковими, тому зробити запит з пустими полями не вдасться. У випадку коли користувач уводить невірні дані авторизації він отримає повідомлення про те, що логін чи пароль були введені не вірно, як показано на рисунку 2.9.

Слід зазначити, що незалежно від того, що саме користувач увів невірно, повідомлення завжди повідомить про невірність двох полів, оскільки явним чином повідомляти про те, що користувач невірно увів саме пароль може бути вразливістю з точки зору безпеки, оскільки таким чином можливо виявити логін користувача і намагатися підібрати пароль методикою грубої сили.

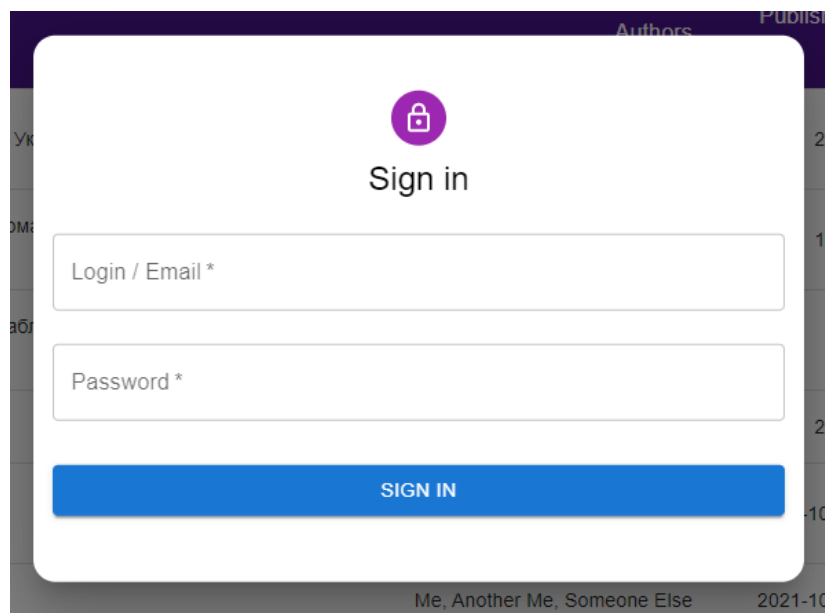


Рисунок 2.8 – Форма авторизації користувача

Login or password is incorrect



Рисунок 2.9 – Користувач увів невірні дані авторизації внаслідок чого отримує повідомлення про некоректні дані.

Якщо користувач увів правильні дані, після натиснення кнопки Sign In, форма авторизації зникне, а у правому нижньому з’явиться спливаюче вікно з повідомленням про успішну авторизацію, показано на рисунку 2.10.

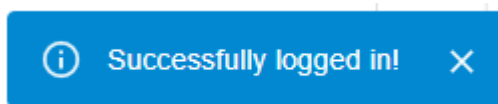


Рисунок 2.10 – Повідомлення про успішну авторизацію.

Адміністратори мають можливість додання нових записів. Як показувалося на рисунку 2.6, у бічній панелі для адміністратора доступна, окрім пошуку, ще одна кнопка на додання запису. При натисканні на неї адміністратор переходить до форми, представлена на рисунку 2.11, в котрі необхідно заповнити дані, а саме необхідно вказати: назву матеріалу, основного автора, спів авторів(якщо наявні), теги для спрощення пошуку, вказати дату чи рік видання та, власне, обрати файл для завантаження. Після заповнення форми необхідно натиснути кнопку *Submit*, після чого почнеться процес завантаження файлу на сервер одразу після якого буде створено новий запис у базі даних.

Одразу після створення нового запису відбувається перенаправлення на головну сторінку, де новий запис буде стояти першим у таблиці нещодавно доданих елементів.

Book Tracker | Home | Logout

Search

+ Add Entry

English

Name

Author

Co Authors

Tags

Publication date

2021

DATE YEAR

UPLOAD FILE

File

SUBMIT

Рисунок 2.11 – Сторінка додавання нового запису до бази даних.

Висновки до розділу 2.

В даному розділі наведено опис процесу реалізації серверної та клієнтської частини проекту.

Було продемонстровано архітектуру серверної частини, а саме:

- Логіка DAO рівня для взаємодії з базою даних;
- Процес завантаження файлів на сервер та з серверу;
- Функціонал обробки користувацьких запитів на рівні контролеру;
- Пошук записів у базі даних;
- Алгоритм обробки користувачів всередині системи.

Також було представлено основний функціонал клієнтської частини додатку: розмежування можливостей звичайних користувачів та адміністраторів, можливість завантаження файлів, редагування та видалення існуючих записів, створення нових записів, а також показано функціонал пошуку записів по базі даних.

У результаті виконаної роботи маємо працездатний додаток з реалізованою серверною та клієнтською частинами.

РОЗДІЛ 3. РОЗГОРТАННЯ ЗАСТОСУНКУ В АРХІТЕКТУРІ БЕЗСЕРВЕРНИХ ОБЧИСЛЕНЬ ТА ОПТИМІЗАЦІЯ ЧАСУ ВИКОНАННЯ ЗАПИТІВ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЇ НАТИВНОГО ЗОБРАЖЕННЯ.

3.1. Безсерверні обчислення.

Безсерверні обчислення — це модель виконання хмарних обчислень, у якій хмарний постачальник розподіляє ресурси машини на вимогу, піклуючись про сервери від імені своїх клієнтів. «Безсерверний» — це неправильна назва в тому сенсі, що сервери все ще використовуються постачальниками хмарних послуг для виконання коду для розробників. Однак розробники безсерверних програм не займаються плануванням потужності, конфігурацією, керуванням, обслуговуванням, відмовостійкістю або масштабуванням контейнерів, віртуальних машин або фізичних серверів.

Безсерверні обчислення не утримують ресурси в енергонезалежній пам'яті; обчислення скоріше виконуються короткими серіями, а результати зберігаються в пам'яті. Коли програма не використовується, для неї не виділяються обчислювальні ресурси. Ціна заснована на фактичній кількості ресурсів, спожитих програмою.

Безсерверні обчислення можуть спростити процес розгортання коду у виробництві. Безсерверний код можна використовувати в поєднанні з кодом, розгорнутим у традиційних стилях, таких як мікросервіси або моноліти. Крім того, програми можуть бути написані як суто безсерверні й взагалі не використовувати надані сервери. Це не слід плутати з моделями обчислень або мереж, які не потребують реального сервера для функціонування, наприклад, одноранговий (P2P).

Функція як послуга (FaaS) — це категорія послуг хмарних обчислень, яка надає платформу, що дозволяє клієнтам розробляти, запускати та керувати

функціональними можливостями програми без складнощів побудови та підтримки інфраструктури, яка зазвичай пов'язана з розробкою та запуском програми.

Основні характеристики безсерверних обчислень:

1) Анотація. Ви не керуєте сервером, на якому запускається програма. Ви взагалі нічого про нього не знаєте, всі нюанси операційної системи, оновлень, налаштувань мережі та іншого заховані від вас. Це зроблено для того, щоб ви могли зосередитись на розробці корисної функціональності, а не на адмініструванні серверів.

2) Еластичність. Провайдер безсерверних послуги автоматично надає вам більше або менше обчислювальних ресурсів, залежно від того, наскільки велике навантаження припадає на вашу програму.

3) Ефективна ціна. Якщо ваша програма простоює - ви нічого не платите, так як воно на цей момент не використовує обчислювальних ресурсів. Оплата ж відбувається лише за час, який ваш додаток реально працює.

4) Обмежений життєвий цикл. Ваша програма запускається в контейнері, і через короткий час, від десятка хвилин до декількох годин, сервіс автоматично його зупиняє. Звичайно, якщо програма знову повинна бути викликана - новий контейнер буде запущено.

Розробка додатків, як правило, поділяється на дві сфери: фронтенд і бекенд. Інтерфейс – це частина програми, яку бачать і з якою взаємодіють користувачі, наприклад візуальний макет. Бекенд — це частина, яку користувач не бачить; це включає сервер, де зберігаються файли програми, і базу даних, де зберігаються дані користувача та бізнес-логіка.

Зазвичай, коли конкретна безсерверна функція не була викликана деякий час, постачальник вимикає функцію, щоб заощадити енергію та уникнути надмірного надання. Наступного разу, коли користувач запустить програму, яка викликає цю функцію, безсерверному провайдеру доведеться запустити її заново

і знову розпочати розміщення цієї функції. Цей час запуску додає значну затримку, яка відома як «холодний старт».

Після того, як функція буде запущена та запущена, вона буде обслуговуватися набагато швидше на наступні запити (теплі запуски), але якщо функція не запитується знову деякий час, функція знову перейде в бездіяльність. Це означає, що наступний користувач, який попросить цю функцію, відчує холодний запуск. Ще зовсім недавно холодний запуск вважався необхідним компромісом використання функцій без сервера.

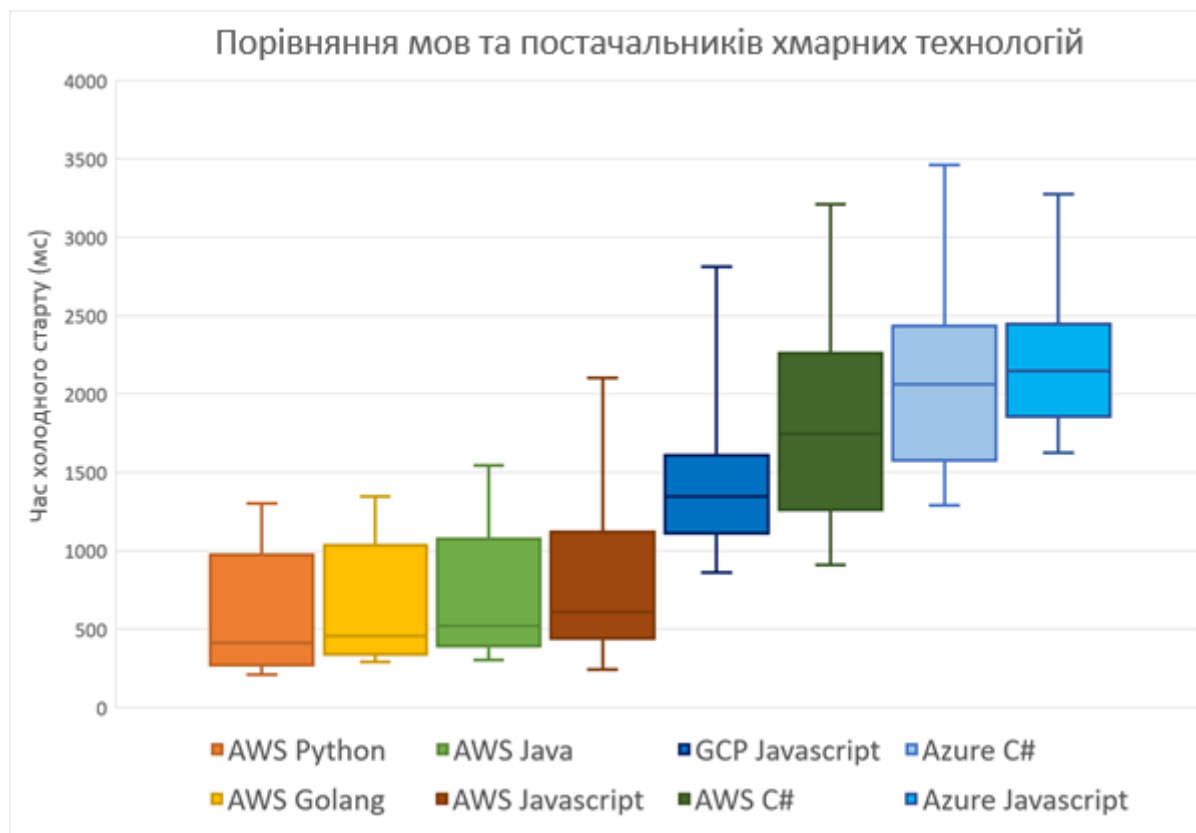


Рисунок 3.1 – Аналіз холодного старту безсерверних функцій.

3.2. Оптимізація часу виконання за допомогою технології Native Image.

GraalVM — це високопродуктивний дистрибутив JDK, розроблений для прискорення виконання програм, написаних на Java та інших мовах JVM, а також підтримкою JavaScript, Ruby, Python та низки інших популярних мов. Поліглотні

можливості GraalVM дозволяють змішувати кілька мов програмування в одній програмі, усуваючи витрати на виклики інших мов.

Native Image — це інноваційна технологія, яка компілює код Java в окремий двійковий виконуваний файл або нативну спільну бібліотеку. Байт-код Java, який обробляється під час збірки нативного образу, включає всі класи додатків, залежності, залежні бібліотеки сторонніх розробників і будь-які необхідні класи JDK. Згенерований автономний нативний виконуваний файл є специфічним для кожної окремої операційної системи та архітектури машини, яка не потребує JVM.

Розглянемо деякі важливі фактори при роботі з Native Image:

3.2.1. Швидкий запуск.

Це може бути дуже цікаво в основному для двох типів програм: програми командного рядка та безсерверні функції. Обидва типи насправді мають багато спільного. Зазвичай вони менші, і їх можна запускати кілька разів за дуже короткий період часу, а термін їх служби досить короткий.

Це причина, чому нам потрібен дуже швидкий запуск, а не чекати надання всіх функцій JVM, які можуть не використовуватися (наприклад, нам потрібно розіграти код, щоб запустити компілятор JIT). Розглянемо ключові фактори, що прискорюють швидкість старту:

- Відсутність завантаження класів. Усі класи вже завантажені, пов'язані та навіть частково ініційовані. Однак це означає, що лише класи та методи, які були відстежені під час процесу створення образу, включені в двійковий файл і можуть використовуватися під час виконання.

- Відсутність інтерпретованого коду. Згенерований нативний код не повинен бути повністю ефективним, оскільки ми не використовуємо оптимізації, керовані профілем, які є частиною компілятора C2 (GraalVM Enterprise містить функцію, яка збирає профіль з попереднього запуску котрий може бути

включений у наступні покоління нативних зображення). Однак нам все одно не потрібно ініціалізувати інтерпретатор і інтерпретувати наш байт-код.

- Немає даремно витрачених циклів ЦП для профілювання та JIT оптимізацій, дуже простий GC для запуску (SerialGC). Нам не потрібно запускати компілятор JIT і покладатися на JIT, щоб зробити код продуктивним.

- Створення купи зображень під час збірки нативного зображення. Раніше зазначалося, що нативна програма частково ініційована. Це означає, що ми можемо запустити процес ініціалізації для деяких конкретних класів під час складання (виконати статичні блоки класу), щоб підготувати деяку частину купи та прискорити запуск.

3.2.2. Зменшений обсяг використовуваної пам'яті.

Якщо ми скомпілюємо наш код у нативного зображення, ми зможемо викинути багато речей із нашого виконуваного файлу. Ці функції JVM не використовуються, тому що вони не повинні бути там, щоб зробити наш код більш ефективним. Однак слід пам'ятати, що RSS-пам'ять процесу нативного зображення зменшена, але лише щодо матеріалів, пов'язаних із JVM, і метаданих класів, пам'ять Heap залишається абсолютно незмінною.

Через природу GC, який зараз використовується в нативних зображеннях, може стати незручно використовувати більші купи, паузи GC можуть вплинути на нашу затримку. Але якщо ми залишимо додаток невеликим, ми зможемо отримати вигоду від ефективності GC нативного зображення (наразі реалізованого як очищувач поколінь). Розглянемо фактори котрі дозволяють зменшити обсяг пам'яті:

- Немає метаданих для завантажених класів. Нам все ще потрібно мати скомпільований код у нашій пам'яті без купи. Це набагато ефективніше, ніж зберігати всі метадані для динамічно завантажуваних класів у Metaspace (не потрібен процес відновлення пам'яті в Metaspace)

- Немає даних профілювання для оптимізації JIT, немає коду інтерпретатора, немає структур JIT. JVM збирає дані профілювання нашої програми, щоб визначити, які види оптимізації можна застосувати. Це не потрібно, тому що наш байт-код вже є в рідному коді. Тому ми можемо викинути весь кеш коду сегмента, який містить дані профілювання та інтерпретатори. Оптимізовані методи зберігаються у двійковому файлі іншим способом.

3.2.3. Якою ціною обходяться переваги нативного зображення.

Native-image — це фантастичний інструмент для невеликих програм, який може допомогти нам із запуском і обсягом пам'яті. Однак ми повинні заплатити за це ціну та налаштувати наші програми, щоб вони відповідали вимогам для AOT-компіляції.

Основні фактори на які необхідно звернути уваги при роботі з нативним зображенням:

- Немає підтримки JVMTI, агентів Java, JMX, JFR. Це, ймовірно, найбільший розрив між додатками, що працюють на JVM, і рідним зображенням. Native-image не підтримує жодні добре відомі профайлери, які підключаються через JVMTI або JMX, і нам потрібно розраховувати на функції ядра, щоб зрозуміти, що відбувається в нашій програмі.

- Ефективний лише для меншої купи. Через SerialGC ми не повинні використовувати більші купи, використання яких може погано вплинути на затримку в процесі роботи програми.

- Згенерований нативний код не є повністю ефективним. JIT завжди буде ефективнішим, оскільки він має доступ до профілю часу виконання програми і може спекулювати та просто вставляти пастки для деоптимізації. Ми можемо зменшити різницю між AOT та JIT за допомогою функції оптимізації, керованої профілем, яка доступна в GraalVM Enterprise Edition.

- Невизначеність з функціоналом відображення. Багато фреймворків засновані на використанні відображення, і вони повинні бути налаштовані або

необхідно використовувати додаткові файли конфігурації, які визначають, які класи і методи повинні опинитися в двійковому файлі.

На рисунках 3.2-3.5 представлені порівняльні характеристики використання двох версій GraalVM.

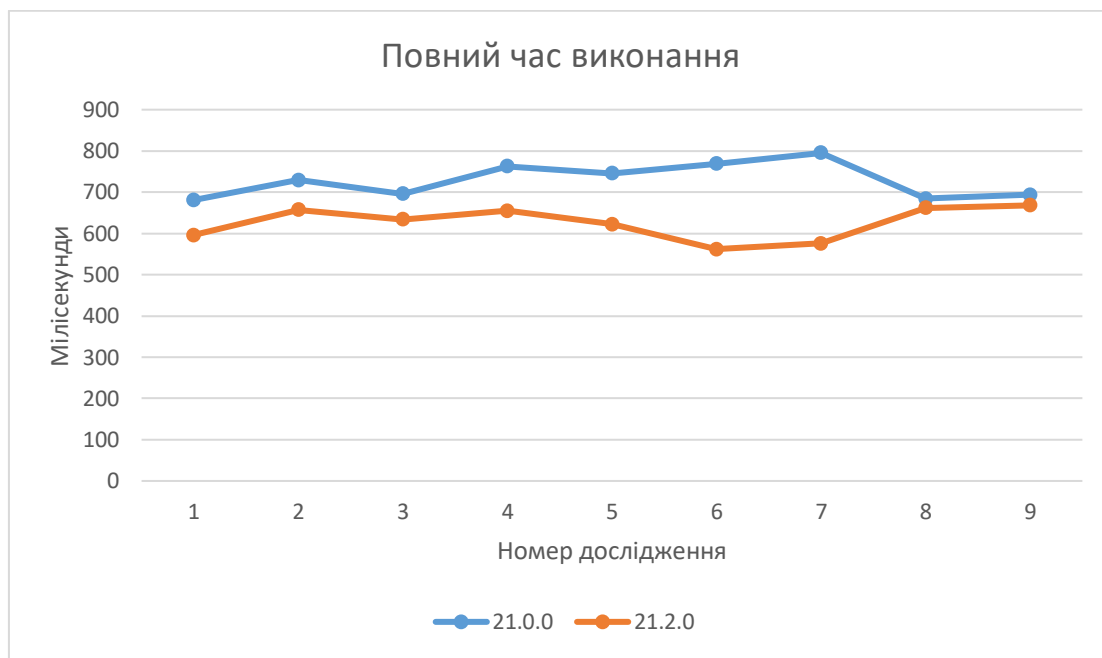


Рисунок 3.2 – Порівняння загального часу роботи лямбди

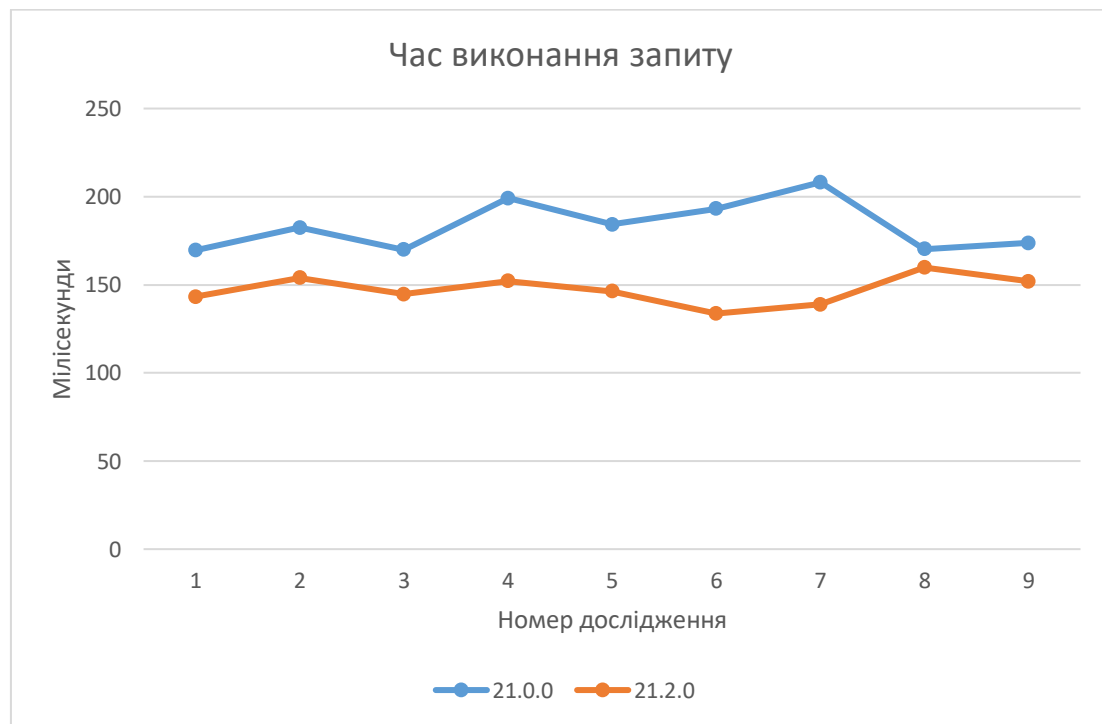


Рисунок 3.3 – Порівняння часу обробки запиту користувача

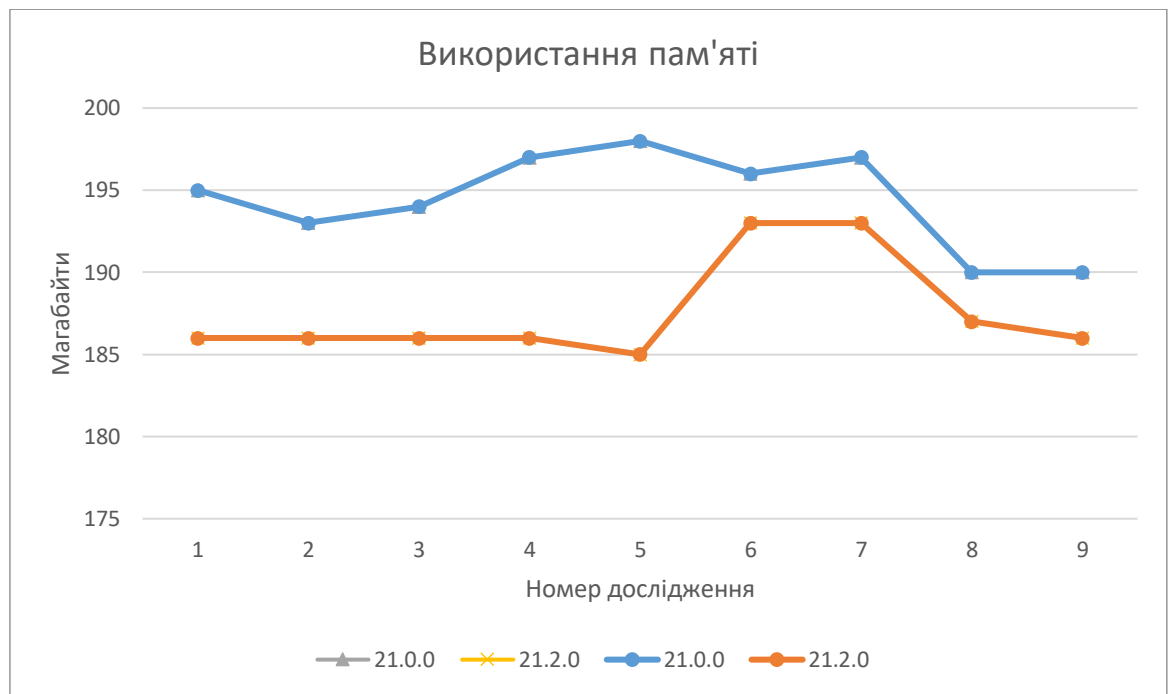


Рисунок 3.4 – Порівняння використання пам'яті на виконання запиту

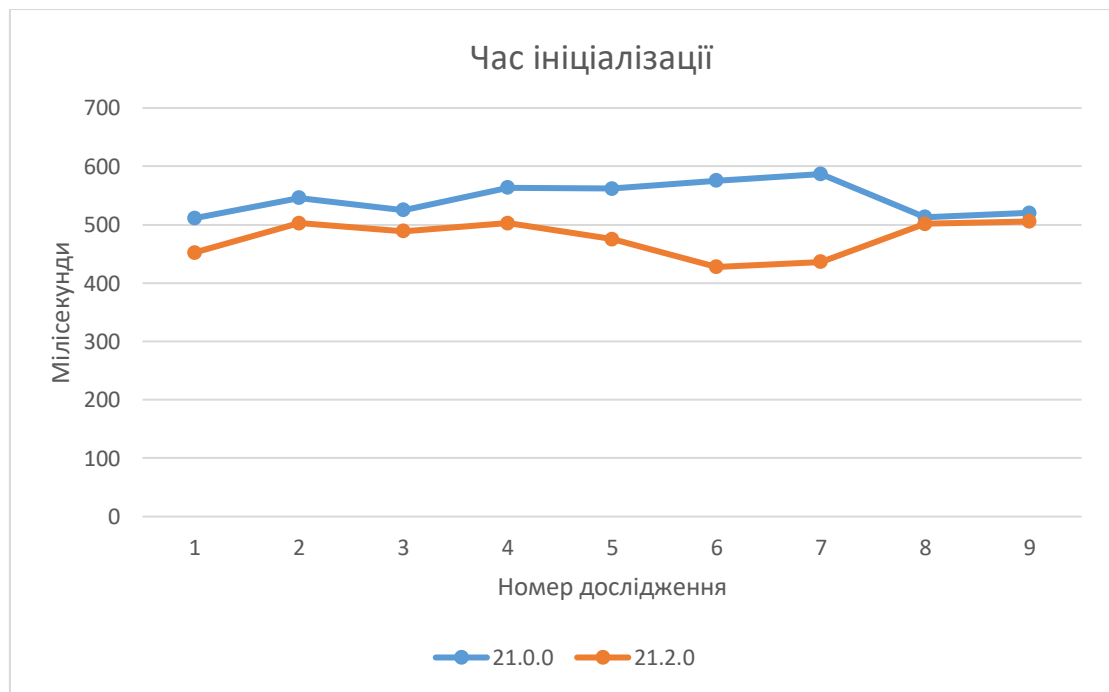


Рисунок 3.5 – Порівняння часу ініціалізації

Аналізуючи результати представлені на рисунках 3.2 – 3.5 можливо прийти до висновку, що лише використання більш нової версії GraalVM(21.2.0) дозволяє суттєво покращити показники продуктивності додатку при цьому використовуючи менше пам'яті.

3.3. Розгортання проекту у хмарі та налаштування неперервної інтеграції та розгортання.

AWS CodePipeline — це повністю керована служба безперервної доставки, яка допомагає автоматизувати конвеєри випуску для швидкого та надійного оновлення додатків та інфраструктури. CodePipeline автоматизує етапи створення, тестування та розгортання процесу випуску щоразу, коли відбувається зміна коду, на основі визначеної моделі випуску. Це дозволяє швидко та надійно надавати функції та оновлення. CodePipeline легко інтегрується зі сторонніми службами, такими як GitHub.

AWS CodePipeline автоматично створює, тестує та запускає програму щоразу, коли код змінюється; розробник використовує графічний інтерфейс користувача для моделювання конфігурацій робочого процесу для процесу випуску в рамках конвеєра. Команда розробників може вказувати та виконувати дії або групу дій, яка називається етапом. Наприклад, розробник може вказати, які тести буде виконувати CodePipeline і в яких середовищах попереднього виробництва він має розгортатися. Потім служба може виконувати ці дії в процесі паралельного виконання, в якому кілька процесорів одночасно обробляють обчислювальні завдання для прискорення робочих процесів.

Використання CodePipeline дозволяє:

- Автоматизувати процеси випуску. CodePipeline повністю автоматизує процес випуску від кінця до кінця, починаючи з вашого вихідного репозиторію до збирання, тестування та розгортання. Ви можете запобігти переміщенню змін через конвеєр, включивши дію затвердження вручну на будь-якому етапі, крім етапу Source. Ви можете випускати, коли забажаєте, у потрібний спосіб, на системах за вашим вибором, в одному або кількох екземплярах.
- Встановити послідовний процес випуску. Визначте послідовний набір кроків для кожної зміни коду. CodePipeline запускає кожен етап релізу відповідно до заданих критеріїв.

- Прискорення процесу доставки, покращення якості. Надається можливість автоматизувати процес випуску, щоб дозволити розробникам поступово тестувати та випускати код і прискорити випуск нових функцій.

- Перегляд прогресу з одного погляду. Це дозволяє нам переглядати стан конвеєрів у режимі реального часу, перевіряти деталі будь-яких сповіщень, повторювати невдалі дії, переглядати відомості про вихідні версії, які використовуються в останньому виконанні конвеєра на кожному етапі, і вручну повторно запускати будь-який конвеєр.

- Перегляд подробиць історії виконання конвеєра. Надає переглянути деталі про виконання конвеєра, включаючи час початку та закінчення, тривалість виконання.

Лістинг 3.1 – Вихідний код для розгортання FrontEnd проекту у середовищі AWS

```
import 'source-map-support/register';
import {BuildTool, FrontEndCdkApp, FrontEndStacks} from
"@insurancemenu/cdk-utils";

const cdkApp = new FrontEndCdkApp({
  appName: "BookTrackerFE",
  repoName: "book-tracker-fe",
  serviceName: "book-tracker",
  aliases: [],
})

new FrontEndStacks(cdkApp, {
  buildTool: BuildTool.NPM,
  envVars: new Map<string, string>([
    ["REACT_APP_BOOK_API",
`https://api.${cdkApp.urlDomain}/book/Prod/`],
    ["REACT_APP_BOOK_BINARY_API",
`https://api.${cdkApp.urlDomain}/bookBinary/Prod/`],
  ])
})
```

З лістингу 3.1 можливо помітити, що вихідний код структури для розгортання клієнтської частини додатка буквально 20 строчок коду. Досягається це за рахунок використання власної бібліотеки котра суттєво спрощує процес написання структури розгортання клієнтської частини.

Варто зазначити, що насправді структура клієнтської частини займає понад 400 строк коду котрі створюють різні типи ресурсів у середовищі AWS котрі необхідні для коректного функціонування додатку. Ключовий фактор який дозволяє так суттєво зменшити кількість строк коду полягає у тому, що клієнтська частина будь якого додатку розгортається за схожим підходом, тому узагальнену логіку можливо винести у бібліотеку та повторно використовувати у інших схожих проектах.

Лістинг 3.2 – Вихідний код для розгортання серверної частини додатку у середовищі AWS

```
import "source-map-support/register";
import {ServiceCdkStack} from "../lib/service-cdk-stack";
import {BackEndCdkApp, BackEndPipeline, BuildTool,
GraalVersion} from "@insurancemenu/cdk-utils";
import {Duration} from "@aws-cdk/core";
import {Runtime, Tracing} from "@aws-cdk/aws-lambda";

const cdkApp = new BackEndCdkApp({
  appName: "BookTracker",
  repoName: "BookTracker",
  basePath: "book",
  serviceName: "book-tracker",
  buildTool: BuildTool.GRADLE,
  additionalServiceUrls: [],
})

const serviceStack = new ServiceCdkStack(cdkApp.app,
`${cdkApp.appName}-Stack-${cdkApp.stageName}`, {
  retainResources: false,
  appName: cdkApp.appName,
  basePath: cdkApp.basePath,
  codeAsset: cdkApp.codeAsset,
  corsOrigins: cdkApp.stageParameters.corsOrigin,
  domainName: cdkApp.stageParameters.domainName,
  lambdaRuntime: cdkApp.isNativeImage ? Runtime.PROVIDED_AL2 :
Runtime.JAVA_11,
  lambdaMemorySize: 3008,
  lambdaTracing: Tracing.ACTIVE,
  lambdaTimeout: Duration.seconds(45),
  stageName: cdkApp.stageName,
  lambdaGlobalEnv: {
    SERVICE_NAME: cdkApp.appName,
    STAGE_NAME: cdkApp.stageName,
```

```

        LOG_LEVEL: cdkApp.stageParameters.logLevel,
        DOMAIN: cdkApp.urlDomain
    },
    pointInTimeRecovery:
cdkApp.stageParameters.pointInTimeRecovery,
    isManualDeploy: cdkApp.isManualDeploy
});

    if (!cdkApp.isManualDeploy) {
        new      BackEndPipeline(cdkApp.app,      `${cdkApp.appName}-
PipelineStack-${cdkApp.stageName}`, {
            ...cdkApp.stageParameters,
            ...cdkApp,
            buildTool: BuildTool.GRADLE,
            graalVersion: GraalVersion.V_21_3_0,
            includeStackInName: true,
            lambdaCode: serviceStack.lambdaCode,
            serviceStackName: serviceStack.stackName,
            repoOwner: "Mykhailo-IM"
        })
    }
}

```

Порівнюючи код для розгортання клієнтської частини та серверної частини помітно, що для серверної частини необхідно надати дещо більший опис, стосовно розгортання. Викликано це тим, що серверні частини застосунків мають суттєву варіацію у списку ресурсів котрі їм необхідні для функціонування. Це можуть бути лямбда функції, таблиці баз даних, бакети для збереження файлі та інше. Однак не зважаючи на суттєву різницю в ресурсах все ж таки можливо винести одну річ у зовнішню бібліотеку, а саме, можливо виокремити конвеєр для розгортання проекту.

Наразі розглянемо, що собою представляє CI/CD конвеєр.

CI/CD – це метод частої доставки програм клієнтам шляхом впровадження автоматизації на етапах розробки додатків. Основні концепції, пов’язані з CI/CD, це безперервна інтеграція, безперервна доставка та безперервне розгортання. CI/CD — це рішення проблем, які може спричинити інтеграція нового коду для команд розробників та операцій (відоме як «інтеграційне пекло»).

Зокрема, CI/CD впроваджує постійну автоматизацію та безперервний моніторинг протягом усього життєвого циклу програм, від етапів інтеграції та

тестування до доставки та розгортання. У сукупності ці пов'язані практики часто називають «конвеєром CI/CD» і підтримуються командами розробників та операцій, які працюють разом у гнучкий спосіб із підходом DevOps або підходу до надійності сайту (SRE). На рисунку 3.6 представлений процес CI/CD.



Рисунок 3.6 – Етапи неперервного інтегрування, доставки та розгортання

Акронім CI/CD має кілька різних значень. "CI" в CI/CD завжди відноситься до безперервної інтеграції, яка є процесом автоматизації для розробників. Успішний CI означає, що нові зміни коду програми регулярно створюються, тестуються та об'єднуються в спільне сховище. Це рішення проблеми одночасної розробки занадто великої кількості гілок програми, які можуть конфліктувати один з одним.

«CD» у CI/CD відноситься до безперервної доставки та/або безперервного розгортання, які є пов'язаними поняттями, які іноді використовуються як взаємозамінні. Обидва стосуються автоматизації подальших етапів конвеєра, але іноді вони використовуються окремо, щоб проілюструвати, наскільки автоматизація відбувається.

Для CI/CD можна вказати лише підключені методи безперервної інтеграції та безперервної доставки, або це також може означати всі 3 підключені практики безперервної інтеграції, безперервної доставки та безперервного розгортання. Щоб зробити це більш складним, іноді «безперервна доставка» використовується таким чином, що охоплює також процеси безперервного розгортання.

У сучасній розробці додатків мета полягає в тому, щоб кілька розробників одночасно працювали над різними функціями однієї програми. Однак, якщо організація налаштована на об'єднання всього розгалуженого вихідного коду

разом за один день (відомий як «день об'єднання»), результат роботи може бути виснажливим, ручним і трудомістким. Це тому, що коли розробник, що працює ізольовано, вносить зміни в програму, є ймовірність, що це буде конфліктувати з різними змінами, які одночасно вносяться іншими розробниками. Ця проблема може ще більше посилитися, якщо кожен розробник налаштував своє власне локальне інтегроване середовище розробки (IDE), а не команда погодила на одну хмарну IDE.

Безперервна інтеграція (CI) допомагає розробникам об'єднувати зміни в коді назад у спільну гілку або «магістраль» частіше — іноді навіть щодня. Після об'єднання внесених розробником змін до програми ці зміни перевіряються шляхом автоматичного створення програми та виконання різних рівнів автоматизованого тестування, як правило, модульних та інтеграційних тестів, щоб переконатися, що зміни не порушили програму. Це означає тестування всього, від класів і функцій до різних модулів, які складають весь додаток. Якщо автоматичне тестування виявляє конфлікт між новим і наявним кодом, CI полегшує швидко і часто виправляти ці помилки.

Після автоматизації збірок, модульного та інтеграційного тестування в CI, безперервна доставка автоматизує випуск цього перевіреного коду в репозиторій. Отже, для ефективного безперервного процесу доставки важливо, щоб CI вже був вбудований у ваш конвеєр розробки. Мета безперервної доставки — мати кодову базу, яка завжди готова до розгортання у виробничому середовищі.

Під час безперервної доставки кожен етап — від злиття змін коду до доставки готових збірок — включає автоматизацію тестування та автоматизацію випуску коду. Наприкінці цього процесу оперативна група може швидко та легко розгорнути програму в робочій станції.

Останнім етапом зрілого конвеєра CI/CD є безперервне розгортання. Як розширення безперервної доставки, що автоматизує випуск готової до виробництва збірки в репозиторій коду, безперервне розгортання автоматизує

випуск програми в робочу станцію. Оскільки на етапі конвеєра до виробництва немає ручного шлюза, безперервне розгортання в значній мірі залежить від добре розробленої автоматизації тестування.

На практиці безперервне розгортання означає, що зміна розробника до хмарного додатка може бути запущена протягом декількох хвилин після його написання (за умови, що воно пройде автоматичне тестування). Це значно полегшує постійне отримання та включення відгуків користувачів. У сукупності всі ці пов'язані методи CI/CD роблять розгортання програми менш ризикованим, завдяки чому легше випускати зміни до програм невеликими частинами, а не всі відразу. Однак є також великі початкові інвестиції, оскільки автоматизовані тести потрібно буде написати, щоб вмістити різноманітні етапи тестування та випуску в конвеєрі CI/CD.

На рисунку 3.7 представлено перший етап конвеєру CI/CD.

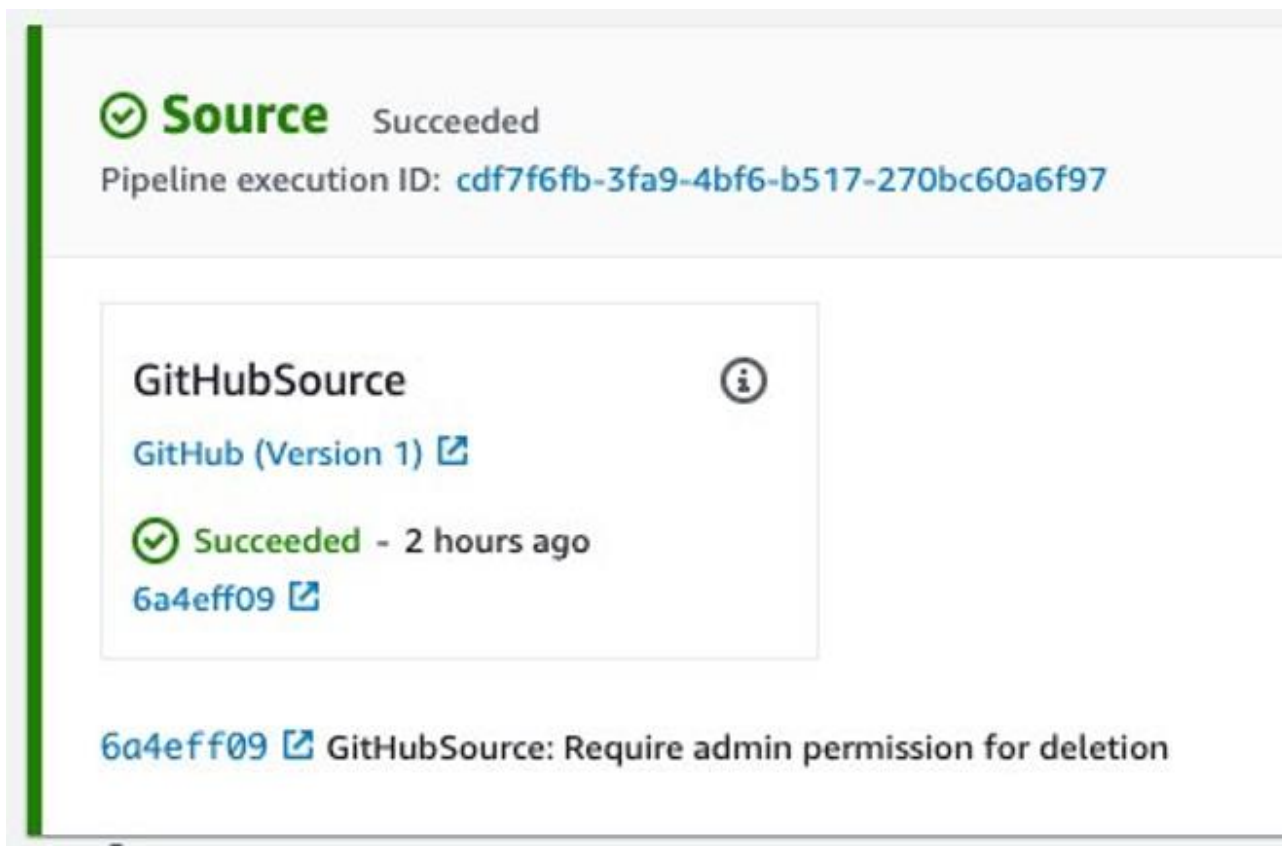


Рисунок 3.7 – Перший етап конвеєру – джерело вихідного коду.

Другий етап конвеєра складається з двох фаз, одна з них запускає тести та збирає вихідний код проекту для розгортання, інша створює структуру ресурсів котрі необхідно створити.

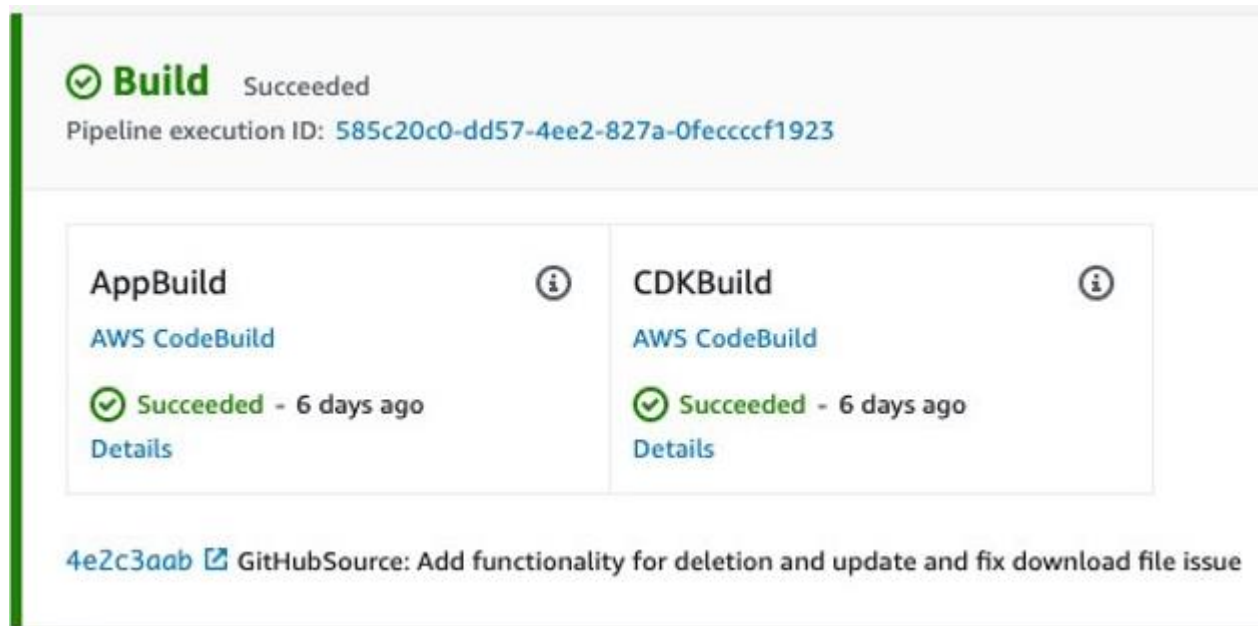


Рисунок 3.8 – Другий етап конвеєру

Останній етап це саме розгортання. На рисунку 3.9 представлений фази для клієнтської частини додатку, котра складається з двох фаз, створення ресурсів у CloudFormation та завантаження коду до S3 бакету.

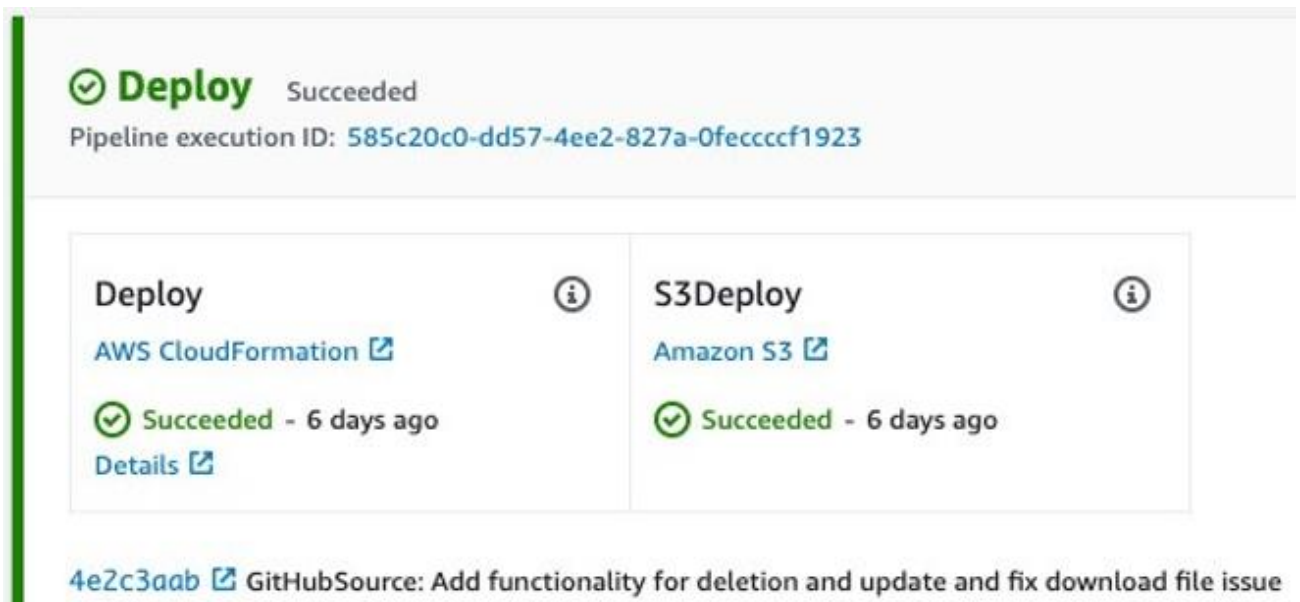


Рисунок 3.9 – Крок розгортання клієнтської частини додатку.

На рисунку 3.10 представлено останній крок розгортання серверної частини додатку. Як можливо помітити тут маємо лише одну фазу котра створює ресурсу з використанням CloudFormation.

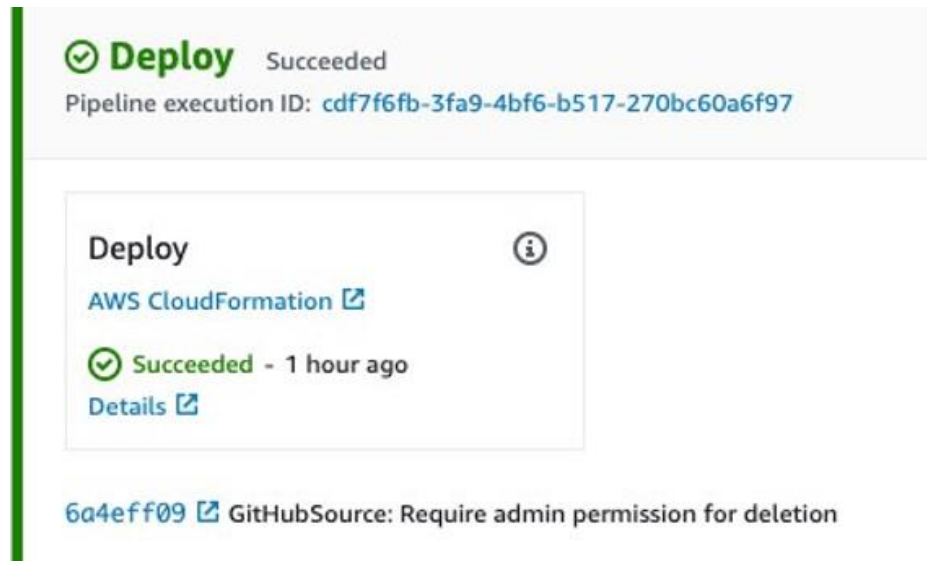


Рисунок 3.10 – Крок розгортання серверної частини додатку.

Висновки до розділу 3.

В цьому розділі проведено детальний аналіз архітектурних рішень котрі були прийняті під час побудови проекту.

Першим важливим рішенням являється використання безсерверних методик обчислення, що дозволяє суттєво знизити кошти затрачені на сервери, за рахунок того що в даному випадку кошти витрачаються тільки на корисне використання ресурсів.

Другий не менш важливий аспект, це використання нативного зображення, що надає суттєвий вииграш у плані продуктивності серверної частини, а також дозволяє позбутися проблеми холодного старту при першому запиті.

В результаті маємо повний працездатний додаток, розгорнутий в інфраструктурі AWS, котрий має власне посилання та оптимізований на швидкодію, відмово стійкість, масштабованість та мінімізацію затрат на утримання та обслуговування.

Розділ 4. РОЗРОБКА СТАРТАП ПРОЕКТУ «АВТОМАТИЗОВАНА СИСТЕМИ НАВЧАЛЬНО-МЕТОДИЧНИХ МАТЕРІАЛІВ КАФЕДРИ»

Стартап – це нещодавно створена компанія (можливо ще не зареєстрована офіційно), що формує свій бізнес на основі інновацій або інноваційних технологій, володіє обмеженою кількістю ресурсів (як людських так і фінансових) і планує виходити на ринок [48].

В цьому розділі розроблена пропозиція для проекту, який був створений в рамках магістерської дисертації. Проект представляє собою автоматизовану систему навчально-методичних матеріалів кафедри.

4.1. Опис ідеї проекту.

Ідея автоматизовану системи навчально-методичних матеріалів кафедри полягає у створенні системи котра надасть можливість збереження наукових матеріалів кафедри у вигляді електронних ресурсів, з можливістю подальшого завантаження, редагування та видалення записів, за необхідністю. Система передбачує два типи користувачів: адміністратори, та звичайні користувачі. Адміністратори відповідальні за надання нових записів до бази даних та редагування існуючих, звичайні користувачі мають можливість завантажувати доступні файли та шукати необхідну їм інформацію, за допомогою вбудованого функціоналу пошуку.

Таблиця 4.1 Опис ідеї стартап-проекту

| Зміст ідеї | Напрямки застосування | Вигоди для користувача |
|---|-----------------------|--|
| Автоматизована системи навчально-методичних | Освітні заклади | Система дозволяє зберігати наукові матеріали у електронному вигляді в одному місці, а також надає можливо швидкого пошуку по існуючим записам. |

| | | |
|--------------------|-------------------------------|--|
| матеріалів кафедри | Архіви електронних документів | Застосунок дозволяє зберігати широку кількість даних при цьому тримаючи вартість зберігання мінімальною. |
|--------------------|-------------------------------|--|

Отже, пропонується готовий програмний продукт, котрий дозволить зберігати наукові матеріали у централізованому місці, при цьому надаючи можливість проводити пошук по всім існуючим записам, стабільно обробляти запити користувачів навіть при високу навантаженні та високу швидкодію.

Проведемо аналіз техніко-економічних характеристик ідеї у порівнянні з існуючими конкурентами на ринку, що дозволить оцінити важливі складові проекту котрі будуть впливати на успішність проекту. Отримані результати представлені в таблиці 4.2.

Таблиця 4.2 Визначення сильних, слабких та нейтральних характеристик

| № п/п | Техніко-економічні характеристики ідеї | (потенційні) товари/концепції конкурентів | | | | W (слабка сторона) | N (нейтральна сторона) | S (сильна сторона) |
|-------|--|---|-------------------|-------------------|-----------------------|--------------------|------------------------|--------------------|
| | | Мій проект | Конкурент 1 | Конкурент 2 | Конкурент 3 | | | |
| 1 | Функціональність | Висока | Висока | Середня | Висока | | | + |
| 2 | Система підтвердження особистості | Присутня | Відсутня | Присутня | Присутня | | | + |
| 3 | Підтримка різних платформ | Платформонезалежність | Одна платформа | Одна платформа | Платформонезалежність | | + | |
| 4 | Метод використання | Сторінка у браузері | Системний додаток | Системний додаток | Системний додаток | | + | |

Аналізуючи отримані дані характеристик ідеї можливо прийти до висновку, що власний проект має низку переваг порівняно із існуючими конкурентами. Новий проект являється незалежним від платформи, тому він буде доступний для більшого спектру користувачів, окрім цього архітектурний підхід для реалізації проекту дозволить отримати суттєві переваги з питання швидкодії.

4.2. Технологічний аудит ідеї проекту.

В межах даного підрозділу проводимо аудит технології за допомогою якої можна реалізувати ідею створення проекту.

Визначення технологічної здійсненності ідеї проекту передбачає аналіз складових які вказані в таблиці 4.3.

Таблиця 4.3 Технологічна здійсненність ідеї проекту

| <i>№ n/n</i> | <i>Ідея проекту</i> | <i>Технології її реалізації</i> | <i>Наявність технологій</i> | <i>Доступність технологій</i> |
|------------------|--|-------------------------------------|---------------------------------|-----------------------------------|
| 1 | Підтвердження особистості користувача | Використання створеної технології | Наявна | Доступна |
| 2 | Реалізація пошуку по базі даних. | Використання створеної технології | Наявна | Доступна |
| 3 | Розширення наявної інформації в одиниці запису в базі даних | Розширення серверної моделі даних | Необхідно розробити | Доступна |
| 4 | Розділення користувацького та адміністративного функціоналів | Використання створеної технології | Наявна | Доступна |
| 5 | Оптимізація серверної частини з метою пришвидшення обробки запитів | Використання створеної технології | Наявна | Доступна |

Продовження таблиці 4.3

Обрана технологія реалізації ідеї проекту: Створення веб додатку для збереження наукових робіт з використанням технологій хмарових постачальників.

Аналізуючи отримані дані, можливо прийти до висновку, що з можливість технічної реалізації проекту достатньо висока. Технології для реалізації ключового функціоналу наявні, а архітектура побудови проекту дозволить ефективно його реалізувати з точки зору цінкових витрат.

4.3. Аналіз ринкових можливостей запуску стартап-проекту.

В межах даного підрозділу проводимо аудит технології за допомогою якої можна реалізувати ідею створення проекту.

Рентабельність — поняття, що характеризує економічну ефективність виробництва, за якої підприємство за рахунок грошової виручки від реалізації продукції (робіт, послуг) повністю відшкодовує витрати на її виробництво й одержує прибуток як головне джерело розширеного відтворення [49].

Для кількісного виміру рентабельності в цілому по аграрних підприємствах використовують такі три традиційні показники: рівень рентабельності, норму прибутку і приведену до земельної площі масу прибутку. Рівень рентабельності (R) визначається за формулою [49]:

$$R = \frac{\Pi}{B_v} \cdot 100\%$$

де Π — валовий прибуток від реалізації (робіт, послуг);

B_v — виробничі витрати на реалізовану продукцію (її виробнича собівартість).

Для повнішої уяви про реальну ефективність певного виду товарної продукції доцільно цей показник обчислювати з врахуванням витрат на її збут,

зменшивши при цьому валовий прибуток на величину цих витрат і водночас збільшивши на них знаменник формули.

Наразі визначимо ринкові можливості, котрі можна застосувати під час впровадження проекту на ринок, а також розглянемо загрози котрі можуть перешкодити процесу реалізації та впровадженню проекту на ринок.

Спершу проведено аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку, результати представлені в таблиці 4.4.

Таблиця 4.4 Попередня характеристика потенційного ринку стартап-проекту

| <i>n/ n</i> | <i>Показники стану ринку (найменування)</i> | <i>Характеристика</i> |
|-----------------|--|--|
| | Кількість головних гравців, од | 3 |
| | Загальний обсяг продаж, грн/ум.од | 3000 |
| | Динаміка ринку (якісна оцінка) | Зростає |
| | Наявність обмежень для входу (вказати характер обмежень) | Конкуренти з більшим досвідом на ринку |
| | Специфічні вимоги до стандартизації та сертифікації | Відсутні |
| | Середня норма рентабельності в галузі (або по ринку), % | 40% |

Аналізуючи результати таблиці можливо прийти до висновку, що вихід на ринок є рентабельним. Оскільки на ринку невелика кількість конкурентів, та факт того, що конкуренти вже деякий час присутні на ринку, це може дозволити проекту з використанням інноваційних технологій та підходів легко увійти на ринок.

Наступним кроком, визначимо ключову аудиторію проекту, розглянемо переваги котрі надає проект, порівняно із конкурентками та сформуємо орієнтовні вимоги котрі можуть бути у користувачів до продукту.

Таблиця 4.5 Характеристика потенційних клієнтів стартап-проекту

| <i>№ п/п</i> | <i>Потреба, що формує ринок</i> | <i>Цільова аудиторія (цільові сегменти ринку)</i> | <i>Відмінності у поведінці різних потенційних цільових груп клієнтів</i> | <i>Вимоги споживачів до товару</i> |
|------------------|--|---|---|--|
| 1 | Система накопичення та збереження наукових матеріалів з можливістю подальшого вилучення та редагування записів | Освітні заклади, Архіви електронних документів | Перевага надається універсальності продукту з точки зору можливості завантажувати великі файли та працювати з різними форматами файлів. | Низькі витрати на обслуговування. Швидкодія обробки користувацьких запитів |

Аналіз характеристик потенційних клієнтів показує, що цільовою аудиторією є освітні заклади, котрі мають властивість створювати велику кількість наукових матеріалів, а також компанії котрим необхідне місце для збереження електронних документів. Відмінності у поведінці цільових груп клієнтів здебільшого припадають на те з якими саме файлами працюють клієнти, однак з точки зору системи це не є проблемою оскільки вона здатна опрацьовувати файли, майже, будь яких форматів та розмірів.

Надалі проводимо аналіз ринкового середовища: складаємо таблиці факторів що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають, результати наведені в таблицях 4.6 та 4.7.

Сильні сторони підприємства - те, в чому підприємство досягло успіху або якась особливість, що надає додаткові можливості. Сила може полягати в наявному досвіді, доступі до унікальних ресурсів, наявності передової технології і сучасного устаткування, високої кваліфікації персоналу, високій якості продукції, що випускається, популярності торгової марки і т. п [48].

Слабкі сторони підприємства - це відсутність чогось важливого для функціонування підприємства або щось, що поки не вдається в порівнянні з іншими компаніями і ставить підприємство в несприятливе положення. Як приклад слабких сторін можна привести дуже вузький асортимент товарів, що випускається, погану репутацію компанії на ринку, недолік фінансування, низький рівень сервісу і т.п [48].

Ринкові загрози - події, настання яких може несприятливо вплинути на підприємство. Приклади ринкових загроз: вихід на ринок нових конкурентів, зростання податків, зміна смаків покупців, зниження народжуваності й т. п [48].

Таблиця 4.6 Фактори загрози

| <i>№ n/n</i> | <i>Фактор</i> | <i>Зміст загрози</i> | <i>Можлива реакція компанії</i> |
|------------------|-----------------------------|---------------------------------------|--|
| 1 | Поява конкурентів на ринку. | Збільшення конкуренції на ринку | Надання функціоналу недоступного у конкурентів |
| 2 | Випуск нових технологій | Технологічна складова стає застарілою | Підтримання бази коду релевантною |

Продовження таблиці 4.6

| | | | |
|---|-------------|---|--|
| 3 | Кібер атаки | Система буде витрачати ресурси(а відповідно і кошти) на обробку недоцільних запитів | Обмеження кількості запитів за одиницю часу для одного користувача та блокування доступу у разі перевищення ліміту |
|---|-------------|---|--|

Для отримання переваг на ринку, необхідно створити або використовувати сприятливі обставини ринку, що носять назву ринкові можливості. До таких можливостей відносять наступні: погіршення становища конкурентів на ринку, різке зростання попиту, зростання рівня доходів населення, поява нових технологій виробництва продукції та ін.

Ринкові можливості - це сприятливі обставини, які підприємство може використовувати для отримання переваг. Як приклад ринкових можливостей

можна привести погіршення позицій конкурентів, різке зростання попиту, появу нових технологій виробництва продукції, зростання рівня доходів населення і т. п. Слід зазначити, що можливостями з погляду SWOT-аналізу є не всі можливості, які існують на ринку, а тільки ті, які можна використовувати [48].

Таблиця 4.7. Фактори можливостей

| <i>№ n/n</i> | <i>Фактор</i> | <i>Зміст можливості</i> | <i>Можлива реакція компанії</i> |
|------------------|---|--|--|
| 1 | Розширення функціоналу системи | Задоволення потреб більшого спектру користувачів | Підвищення вартості використання продукції, або надання частини функціоналу за окрему ціну |
| 2 | Вихід на іноземний ринок | Зростання кількості зацікавлених клієнтів | Оперативне впровадження застосунку з метою максимізації клієнтської бази |
| 3 | Впровадження нових технологій | Покращення існуючого функціоналу | Зменшення витрат на обслуговування власної інфраструктури |
| 4 | Зворотній зв'язок з кінцевим користувачем | Аналіз потреб користувачі, а не замовників | Надання можливості користувача залишати побажання покращення функціоналу. |

Оцінюючи наведені фактори можливостей, можна помітити, що з урахування поступового оцифровування інформації стає все більш рентабельною розробка централізованої системи збереження наукових матеріалів з подальшою можливістю пошуку по наявним записам[50].

У таблиці 4.8 приведено аналіз пропозиції за визначенням загальних рис ринкової конкуренції.

Таблиця 4.8. Ступеневий аналіз конкуренції на ринку

| | | |
|---|--|---|
| <i>Особливості конкурентного середовища</i> | <i>В чому проявляється дана характеристика</i> | <i>Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)</i> |
|---|--|---|

Продовження таблиці 4.8

| | | |
|---|---|--|
| 1. Вказати тип конкуренції Досконала конкуренція | Окремі конкуренти не мають прямого впливу один на одного | Концентрація на якості та функціональності продукту |
| 2. За рівнем конкурентної боротьби Міжнародний | Продукт не являється державно залежним і може бути легко адаптований під потреби іноземних користувачів | Адаптація під потреби іноземного ринку |
| 3. За галузевою ознакою Міжгалузева | Використання у різних галузях | Підвищення рівня пізнаваності продукту та надання унікального функціоналу. |
| 4. Конкуренція за видами товарів: Товарно-видова | Конкуренція з продуктами, котрі надають подібний функціонал | Покращення характеристик продукту |
| 5. За характером конкурентних переваг Нецінова | Залежить від функціоналу та якості продукції | Розширення функціоналу та оптимізація вже існуючого. |
| 6. За інтенсивністю Марочна | Конкурентні компанії пропоную подібний продукт | Створення позитивної репутації серед користувачів |

Провівши аналіз даних в таблиці 4.8 можливо зробити висновок, що умови ринку дозволяють мати попит на продукт базуючись на його характеристиках якості та можливості застосування у суміжних сферах.

Після аналізу конкуренції проведемо більш детальний аналіз умов конкуренції в галузі. Аналіз представлено в таблиці 4.9.

Аналіз п'яти сил Портера - це корисний інструмент стратегічного планування як для бізнес-планування, так і для ринкового планування, особливо якщо мова йде про розуміння рівня конкурентоспроможності бізнесу в певній галузі [49].

Аналіз п'яти сил Портера оцінює рівень рентабельності, можливостей та ризику на основі 5 ключових факторів у галузі [49]:

- Постачальники.
- Покупці.
- Перешкоди для входу / виходу.
- Замінники.
- Суперництво.

Цілі для яких використовується даний інструмент:

- Проаналізувати привабливість та прибутковість галузі.
- Спостерігати за силою ринкової позиції власного бізнесу.

Методика використання інструменту:

Ця модель допомагає визначити конкурентні сили, які існують у галузі. У свою чергу ці сили допомагають визначити привабливість та прибутковість галузі[50].

Сила 1. Потужність постачальника (ця сила аналізує владу, яку має постачальник над бізнесом)

- Подивіться, скільки постачальників є на ринку.
- Скільки у вас постачальників.
- Чи тримають ваші постачальники владу над типом вашого бізнесу.
- Що буде вартувати вам і їм, якщо ви вирішили змінити постачальників.
- Чи багато постачальників, які контролюють ціни на ринку.

Сила 2. Потужність покупця (ця сила аналізує владу, яку покупець має над вашим бізнесом)

- Скільки всього покупців.
- Скільки у вас покупців.
- Наскільки чутливими до ціни є ваші покупці.
- Яку інформацію ви маєте про них.

Сила 3. Загроза нових учасників (ця сила аналізує, наскільки легко чи складно для нових конкурентів вийти на ринок).

- Наскільки легко відкрити бізнес на вашому ринку.
- Яких правил та положень потрібно дотримуватися.
- Скільки грошей повинен витратити новий стартап, щоб зайти на ринок.
- Чи є бар'єри, при здоланні яких, ви б отримали більший вплив.

Сила 4 Загроза заміни товарів / послуг (ця сила аналізує, як легко клієнту перейти з продукту одного бізнесу на продукт конкурента).

- Скільки замінників вашого товару існує.
- Наскільки легко вашому покупцеві перейти на інший товар.
- Чи покупець “сплачує” за перемикання на інший товар (чи це щось йому вартує).

Сила 5 Конкурентне суперництво (ця сила вивчає інтенсивність конкуренції на поточному ринку).

- Який рівень конкуренції у вашому ринковому секторі.
- Хто є вашими основними конкурентами.
- Приблизно скільки у вас конкурентів.
- Яка ваша конкурентна стратегія.

Таблиця 4.9. Аналіз конкуренції в галузі за М. Портером

| | <i>Прямі конкуренти в галузі</i> | <i>Потенційні конкуренти</i> | <i>Постачальники</i> | <i>Клієнти</i> | <i>Товари-замінники</i> |
|-------------------------|-----------------------------------|------------------------------|--|--|-------------------------------------|
| <i>Складові аналізу</i> | “1С:Бібліотека”, “OPAC-Global” | “АБИС Либра” | Постачальники хмарних технологій та серверів | Освітні заклади, Архіви електронних документів | Продукти, що надаються конкурентами |

Продовження таблиці 4.9.

| | | | | | |
|-----------|---|--|--|-----------------------------------|---|
| Висновки: | Порівняно невелика та слабка конкуренція. | Вихід на ринок відносно простий. Наявні потенційні конкуренти. | На ринку досить багато постачальників, через що вони не диктують умови роботи ринку. | Клієнти не можуть диктувати умови | Наявність конкурентів в слабо впливає на можливість входження в ринок |
|-----------|---|--|--|-----------------------------------|---|

З огляду на отримані дані можливо прийти до висновку, що незважаючи на наявність явної конкуренції на ринку, сам процес виходу не повинен викликати труднощів, оскільки проект надає можливості котрі не надають конкуренти.

Надалі визначимо та обґрунтуємо фактори конкурентоспроможності в реаліях існуючого ринку(таблиця 4.10).

Таблиця 4.10 Обґрунтування факторів конкурентоспроможності

| <i>№ n/n</i> | <i>Фактор конкурентоспроможності</i> | <i>Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)</i> |
|------------------|--|--|
| 1 | Обслуговування | Наявність підтримки на час використання продукту |
| 2 | Простота використання | Користувач буде більш зацікавлений у використанні продукту якщо інтерфейс використання буде простий |
| 3 | Ціна | Вартість надання послуг може стати ключовим фактором при виборі подібних систем. |
| 4 | Розширюваність | Можливість продукту адаптуватися до потреб користувачів |

Відповідно до результатів наведених в таблиці 4.10 можливо прийти до висновку, що проект має низку переваг у порівнянні із конкурентами, а саме – простий інтерфейс дозволить легко оперувати додатком, вибір архітектури

побудови дозволить суттєво знизити вартість необхідну на підтримку серверів, що дозволить зменшити ціну самого продукту, тим самим приваблюючи клієнтів, вибір технологій дозволяє розширювати функціонал без великих затрат у плані ресурсів та часу, що дозволить задовільнити потреби нових клієнтів у короткі терміни.

За отриманими факторами конкурентоспроможності проведемо аналіз сильних та слабких сторін стартап-проекту(таблиця 4.11).

Таблиця 4.11. Порівняльний аналіз сильних та слабких сторін

| № п/ п | Фактор конкурентоспромож ності | ББа ли 1- 20 | Рейтинг товарів-конкурентів у порівнянні з E-ArchiveMenu | | | | | | |
|--------------|--------------------------------------|-----------------------|--|----|----|---|----|----|----|
| | | | -3 | -2 | -1 | 0 | +1 | +2 | +3 |
| 1 | Обслуговування | 13 | | | | | | + | |
| 2 | Простота використання | 18 | | + | | | | | |
| 3 | Ціна | 17 | | | + | | | | |
| 4 | Розширюваність | 14 | | | + | | | | |

Враховуючи результати представлені в таблиці 4.11, можливо поміти, що проект володіє достатньою кількістю переваг, щоб бути конкурентоспроможним на ринку, основною перевагою можливо вважати простий інтерфейс використання продукту, що дозволить користувач швидко опанувати процес роботи із продуктом.

Завершальним пунктом ринкового аналізу можливостей впровадження проекту є складання SWOT- аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities), що приведені у таблиці 4.12.

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. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

Таблиця 4.12 SWOT-аналіз стартап-проекту

| | |
|--|--|
| Сильні сторони: розширюваність функціоналу, простий у використанні інтерфейс, обслуговування, адаптація від необхідності користувачів. | Слабкі сторони: відсутність імені на ринку, невеликий початковий функціонал. |
| Можливості: вихід на міжнародний ринок, розширення функціоналу системи, збільшення попиту. | Загрози: Поява конкурентів на ринку, Випуск нових технологій, Кібер атаки. |

На основі SWOT-аналізу розробимо альтернативи ринкової поведінки для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок.

Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів (табл. 4.13).

Таблиця 4.13 Альтернативи ринкового впровадження стартап-проекту

| <i>№ n/n</i> | <i>Альтернатива (орієнтовний комплекс заходів) ринкової поведінки</i> | <i>Ймовірність отримання ресурсів</i> | <i>Строки реалізації</i> |
|------------------|--|---|--------------------------|
| 1 | Встановлення конкурентоспроможної вартості продукту з ціллю заохочення користувачів. | Ймовірність висока, так як у продукті наявні переваги котрі можуть стати до вподоби користувача при цьому за привабливою ціною. | 3 місяці |

Продовження таблиці 4.13.

| | | | |
|---|--|---|-----------|
| 2 | Розширення існуючого функціоналу задля задоволення ширшого спектру користувачів | Ймовірність середня, оскільки у такому випадку отримаємо більшу аудиторію користувачів. | 6 місяців |
| 3 | Використання MVP задля залучення перших клієнтів та збору побажань по розширенню функціональності продукту | Ймовірність висока, тому що візуальна демонстрація функціоналу дозволить користувачу в повній мірі оцінити продукт. | 1 місяць |

З таблиці 4.13 можливо прийти до висновку, що найбільш перспективним являється варіант використання MVP задля залучення перших клієнтів та збору побажань по розширенню функціональності продукту. Наявність мінімально працездатного вигляду продукту, дозволить проводити демонстрації ключовим клієнтам, що дозволить їм в повній мірі оцінити усі можливості продукту.

4.4. Розроблення ринкової стратегії проекту.

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів, котрі наведені в таблиці 4.14.

Таблиця 4.14. Вибір цільових груп потенційних споживачів

| № n/n | 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 | Освітні заклади | Висока | Високий | Середня | Висока |
| 2 | Архіви електронних документів | Середня | Високий | Середня | Середня |

Продовження таблиці 4.14.

Які цільові групи обрано: Беручи до уваги основний функціонал проекту у якості цільової групи було обрано освітні заклади, оскільки функціонал який надає проект дозволить закладам освіти мати централізоване місце для збереження наукових матеріалів, що в свою чергу позитивно вплине на доступність цих матеріалів у подальших дослідженнях.

Аналізуючи отримані результати, приходимо до висновку, що найбільш релевантною цільовою групою стануть освітні заклади, оскільки це надасть їм можливість отримати централізоване сховище наукових матеріалів, з функціоналом повнотекстового пошуку.

Надалі необхідно сформувані базову стратегію розвитку для роботи в обраному сегменті ринку, результат представлений у таблиці 4.15.

Існують три базові стратегії розвитку, що відрізняються за ступенем охоплення цільового ринку та типом конкурентної переваги, що має бути реалізована на ринку.

1) Стратегія лідерства по витратах. Передбачає, що компанія за рахунок чинників внутрішнього і/або зовнішнього середовища може забезпечити більшу, ніж у конкурентів маржу між собівартістю товару і середньоринковою ціною (або ж ціною головного конкурента).

2) Стратегія диференціації. Передбачає надання товару важливих з точки зору споживача відмітних властивостей, які роблять товар відмінним від товарів конкурентів.

3) Стратегія спеціалізації. Передбачає концентрацію на потребах одного цільового сегменту, без прагнення охопити увесь ринок.

Таблиця 4.15. Визначення базової стратегії розвитку

| <i>№ п/ п</i> | <i>Обрана альтернатива розвитку проекту</i> | <i>Стратегія охоплення ринку</i> | <i>Ключові конкурентоспро можні позиції відповідно до обраної альтернативи</i> | <i>Базова стратегія розвитку</i> |
|-----------------------|---|--|--|--|
| 1 | Ідивідуалізм | Стратегія диференціально го маркетингу | Адаптація до потреб користувачів. Надання унікального функціоналу. | Стратегія диференціації |

У результаті була обрана стратегія диференціації через існування на ринку досвідчених конкурентів. Мета стратегії полягає в наданні клієнтам продукту котрий буде відрізнятися від продукту конкурентів.

Наступний етап, це вибір стратегії конкурентної поведінки(таблиця 4.16).

Стратегія лідера. Залежно від міри сформованості товарного(галузевого) ринку, характеру конкурентної боротьби компанії-лідери обирають одну з трьох стратегій: розширення первинного попиту, оборонну або наступальну стратегію або ж застосувати демаркетинг або диверсифікацію [50].

Стратегія виклику лідера. Стратегію виклику лідерів найчастіше вибирають компанії, які є другими, третіми на ринку, але бажають стати лідером ринку. Теоретично, ці компанії можуть прийняти два стратегічні рішення: атакувати лідера у боротьбі за частку ринку або ж йти за лідером.

Стратегія наслідування лідеру. Компанії, що приймають слідування за лідером – це підприємства з невеликою часткою ринку, які вибирають адаптивну лінію поведінки на ринку, усвідомлюють своє місце на ній і йдуть у фарватері фірм-лідерів. Головна перевага такої стратегії – економія фінансових ресурсів, пов'язаних з необхідністю розширення товарного(галузевого) ринку, постійними інноваціями, витратами на утримання домінуючого положення [50].

Стратегія заняття конкурентної ніші. При прийнятті стратегії зайняття конкурентної ніші (інші назви – стратегія фахівця або нішера) компанія в якості цільового ринку вибирає один або декілька ринкових сегментів. Головна особливість – малий розмір сегментів/сегменту. Ця конкурентна стратегія являється похідною від такої базової стратегії компанії, як концентрація.

Таблиця 4.16 Визначення базової стратегії конкурентної поведінки

| <i>№ n/n</i> | <i>Чи є проект «першопрохідцем» на ринку?</i> | <i>Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?</i> | <i>Чи буде компанія копіювати основні характеристики товару конкурента, і які?</i> | <i>Стратегія конкурентної поведінки</i> |
|------------------|---|---|---|---|
| | Ні | Початкова стратегія це приваблення нових користувачів, а в ході розширення і приваблення користувачів конкурентів | Наразі не має плані повністю повторювати функціонал конкурентів(хоча в ході розширення таке може випадково трапитися) | Стратегія наслідування лідера |

В результаті була обрана стратегія наслідування лідера. Вибір саме цієї стратегії, дозволить на перших етапах проекту мати чітку ціль та потреби користувачів, котрі будуть задоволені в більшій мірі ніж використовуючи продукт конкурента. В подальшому розширення існуючого функціоналу, дозволить залучити нових користувачів та планомірно розширюватися на ринку.

На основі вимог споживачів з обраних сегментів до постачальника та до продукту, а також в залежності від обраної базової стратегії розвитку (табл. 4.15) та стратегії конкурентної поведінки (табл. 4.16) розробляється стратегія позиціонування (табл. 4.17). що полягає у формуванні ринкової позиції за якою споживачі мають ідентифікувати проект.

Таблиця 4.17. Визначення стратегії позиціонування

| <i>№ n/ n</i> | <i>Вимоги до товару цільової аудиторії</i> | <i>Базова стратегія розвитку</i> | <i>Ключові конкурентоспромо жні позиції власного стартап- проекту</i> | <i>Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)</i> |
|-----------------------|--|--|---|---|
| 1 | Простота у використанні | Стратегія спеціалізації | Простота та зрозумілість інтерфейсу продукту | Простий, інтуїтивний інтерфейс |
| 2 | Відмовостійкість | Стратегія спеціалізації | Застосунок продовжує стабільно працювати при високому навантаженні | Стабільність під навантаженням |
| 3 | Розширення та вдосконалення функціоналу | Стратегія спеціалізації | Постійний процес вдосконалення продукту дозволить приваблювати користувачів новими можливостями продукту. | Розширюваність за вимогою |

З результатів представлених в таблиці 4.17 отримуємо, що проект має асоціюватись у клієнта з такими позиціями, як простота використання, відмовостійкість, тобто можливість адекватно обробляти запити користувачів під великим навантаженням, можливість розширення та вдосконалення функціоналу.

4.5. Розроблення маркетингової програми стартап-проекту.

Першим кроком є формування маркетингової концепції продукту, який отримає споживач. Для цього у табл. 4.18 підсумовані результати попереднього аналізу конкурентоспроможності товару[50].

Таблиця 4.18. Визначення ключових переваг концепції потенційного товару

| <i>№ n/n</i> | <i>Потреба</i> | <i>Вигода, яку пропонує товар</i> | <i>Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)</i> |
|------------------|---------------------------------|---|---|
| | База даних наукових матеріалів. | Простий інтерфейс, швидкодія, відмово стійкість, декілька типів пошуку записів. | Суттєва масштабованість, відмово стійкість, повний текстовий пошук. |

Аналізуючи результати представлені в таблиці 4.18 отримуємо, що ми маємо низку переваг у порівнянні з конкурентами, що надає можливість сфокусуватися саме на них під час етапу залучення клієнтів.

Наступним етапом являється розробка трирівневої маркетингової моделі товару: уточнюються ідея продукту, його фізичні складові, особливості процесу його надання[51].

1-й рівень:

При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і / або проблеми буде даний товар, яка його основна вигода. Дане питання безпосередньо пов'язаний з формуванням технічного завдання в процесі розробки конструкторської документації на виріб [51].

2-й рівень:

Цей рівень являє рішення того, як буде реалізований товар в реальному/ включає в себе якість, властивості, дизайн, упаковку, ціну [51].

3-й рівень:

Товар з підкріпленням (супроводом) - додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості , доставка, умови оплати та ін.) [51].

Таблиця 4.19. Опис трьох рівнів моделі товару

| <i>Рівні товару</i> | <i>Сутність та складові</i> | | |
|---|---|------|-------------------|
| I. Товар за задумом | База даних наукових матеріалів представляє собою місце для збереження та подальшого вилучення(за допомогою повнотекстового пошуку) наукових матеріалів(статей, публікацій, книжок тощо) | | |
| II. Товар у реальному виконанні | Властивості/характеристики | М/Нм | Вр/Тх /Тл/Е/Ор |
| | - Відмово стійкість | Нм | Тх |
| | - Простота використання | Нм | Тх |
| | - Пошук по записам у базі | Нм | Тх |
| | - Ціна використання | М | Е |
| | Якість: продукт розробляється з урахуванням потреб користувачів | | |
| III. Товар із підкріпленням | До продажу: Доступ до продукту | | |
| | Після продажу: Обслуговування користувачів з технічних питань. | | |
| За рахунок чого потенційний товар буде захищено від копіювання: закритий вихідний код та ліцензія(або патентування) | | | |

Наступним кроком є визначення цінових меж, якими необхідно керуватися при встановленні ціни на потенційний товар, це передбачає аналіз цін товарів конкурентів, та доходів споживачів продукту (табл. 4.20) [51].

Таблиця 4.20. Визначення меж встановлення ціни

| <i>№ n/n</i> | <i>Рівень цін на товари- замінники</i> | <i>Рівень цін на товари- аналоги</i> | <i>Рівень доходів цільової групи споживачів</i> | <i>Верхня та нижня межі встановлення ціни на товар/послугу</i> |
|------------------|--|--|---|--|
| | 0 - 10000 | 0 - 10000 | 100000 | 1000-2000 |

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 4.21):

Таблиця 4.21. Формування системи збуту

| <i>№ n/n</i> | <i>Специфіка закупівельної</i> | <i>Функції збуту, які має виконувати</i> | <i>Глибина каналу збуту</i> | <i>Оптимальна система збуту</i> |
|------------------|------------------------------------|--|-------------------------------------|---|
|------------------|------------------------------------|--|-------------------------------------|---|

| | | | | |
|--|------------------------------------|----------------------------|-----------------------|--------------|
| | <i>поведінки цільових клієнтів</i> | <i>постачальник товару</i> | | |
| | Помісячна оплата за надання послуг | Повна підтримка продукту | Канал нульового рівня | Безпосередня |

Основним каналом збуту продукту являється помісячна оплата за надання послуг, Така модель оплати, дозволить мати стабільний прибуток проекту від існуючих клієнтів, а це в свою чергу дозволить, з часом, відійти від покладання на інвестиційні кошти.

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.22) [51].

Таблиця 4.22. Концепція маркетингових комунікацій

| <i>№ п/ п</i> | <i>Специфіка поведінки цільових клієнтів</i> | <i>Канали комунікацій, якими користуються цільові клієнти</i> | <i>Ключові позиції, обрані для позиціонування</i> | <i>Завдання рекламного повідомлення</i> | <i>Концепція рекламного звернення</i> |
|-----------------------|---|---|--|--|---------------------------------------|
| | Зовнішні обставини спонукають клієнтів до пошуку систем збереження даних. | Соціальні мережі, пряма взаємодія з потенційним клієнтом, сайт виробника. | Відмовостійкість. Інтуїтивний інтерфейс. Повний текстовий пошук. | Розповсюдження інформації про продукт, демонстрація переваг. | Демонстрація функціоналу продукту. |

У результаті отримуємо маркетингову програму, що включає в себе концепції товару, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку ринкового середовища, в межах якого буде впроваджено проект, та відповідну обрану альтернативу ринкової поведінки.

Висновок до розділу 4.

Даний розділ присвячений розробці першого етапу стартап-проекту. Суттєвим фактором у проведенні будь-якої наукової роботи є можливість отримання прибутку у майбутньому, тобто комерціалізація, а також можливість впровадження власного продукту у промисловість. Проект, що розглядається в розділі, представляє собою автоматизовану базу даних для збереження наукових матеріалів, а реалізація проекту використовує одні з передових технологій які надають постачальники хмарних обчислень.

Основною цільовою аудиторією представленого проекту являється заклади освіти, котрі генерую велику кількість наукових матеріалів щорічно та окрім цього мають широкий спектр матеріалів які зберігаються у різних місцях. Представлений проект призначений вирішити цю проблему і надати можливість централізованого місця збереження інформації з подальшою можливістю повнотекстового пошуку задля спрощення процесу пошуку необхідної інформації.

В ході аналізу ринку та виявлення конкурентів було встановлено, що усі конкуренти надають свій продукт у якості настільного додатку, що суттєво впливає на використання продукту так як продукт являється залежним від платформи. Окрім цього продукти конкурентів являються в деякій мірі застарілими, оскільки вони були створені доволі давно і тому робота користувача з інтерфейсом не завжди являється продуктивною.

Проект котрий розглядається в даному розділі бере планку вище і намагається надати користувача якомога простіший інтерфейс користування при цьому не жертвуючи функціоналом, використовуючи найрелевантніші технології існуючі на даний час. Важливим фактором проекту також є архітектурні рішення при побудові серверної та клієнтської частини, що дозволяє суттєво знизити витрати на обслуговування тим самим роблячи ціну продукту більш привабливою для кінцевого користувача.

У результаті детального аналізу ринку, можливо сказати, що подальша реалізація продукту та вихід на ринок являються повністю можливими. Однак, слід зазначити, що слід створити якісний продукт, котрий би продемонстрував користувача його переваги та сильні сторони, що буде спонукати потенційних клієнтів до обрання даного продукту.

ВИСНОВОК

В даній роботі було проведено огляд сучасного стану автоматизованих систем збереження науково-методичних матеріалів.

Огляд сфери показує, що в даний час все більше галузей покладається на повністю цифровий обіг інформації саме тому система наявної на кафедрі літератури являється актуальним рішенням для відслідковування та збереження стану усього доступного матеріалу. В сучасних реаліях збереження інформації та отримання до неї доступу в електронному вигляді являється найбільш простою та оптимальною формою, оскільки більшість людей мають при собі пристрій котрий може працювати з електронними файлами.

Першим етапом реалізації проекту став огляд існуючих аналогів та технологій для розробки власної системи. В результаті огляду було прийнято рішення у якості основної мови розробки обрати Kotlin, а архітектуру проекту побудувати з упором на використанням засобів постачальників хмарних технологій, що дозволяє мати сервер для розгортання проекту, надає централізоване місце для використання необхідних сервісів для побудови проекту, а також дозволяє знизити затрати на підтримку серверу.

Наступний крок присвячений саме розробці програмного забезпечення де було представлено основний функціонал клієнтської частини додатку: розмежування можливостей звичайних користувачі та адміністраторів, можливість завантаження файлі, редагування та видалення існуючих записів, створення нових записів, а також показано функціонал пошуку записів по базі даних.

Фінальним кроком в розробці проекту стало розгортання інфраструктури у хмарному середовищі AWS з використанням безсерверних обчислень, нативного зображення та конвеєру постійної інтеграції. Усі прийняті рішення при побудові проекту у результаті надають повністю працездатний додаток оптимізований на швидкодію, відмово стійкість, масштабованість. При цьому проект також має

CI/CD конвеєр, що робить процес реалізації та розгортання нового функціоналу неймовірно простим і майже не потребує втручання розробника.

Завершальним етапом став процес розробка стартап проекту, в результаті чого було визначено, що основною цільовою аудиторією представленого проекту являється заклади освіти, окрім цього проект намагається надати користувача якомога простіший інтерфейс користування при цьому не жертвуючи функціоналом, використовуючи новітні технології існуючі на даний час.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Проникнення інтернету в Україні [Електронний ресурс]. – Режим доступу: https://inau.ua/sites/default/files/file/1910/dani_ustanovchyh_doslidzhen_iii_kvartal_2019_roku.pdf
2. ISO/IEC 2382:2015, Information technology — Vocabulary — Part 1: Terms and definitions [Електронний ресурс]. – Режим доступу: <https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:ed-1:v1:en>
3. MySQL 8.0 Reference Manual / General Information [Електронний ресурс]. – Режим доступу: <https://dev.mysql.com/doc/refman/8.0/en/introduction.html>
4. MySQL 8.0 Reference Manual / Security / Access Control and Account Management [Електронний ресурс]. – Режим доступу: <https://dev.mysql.com/doc/refman/8.0/en/access-control.html>
5. MySQL 8.0 Reference Manual / General Information / MySQL Standards Compliance [Електронний ресурс]. – Режим доступу: <https://dev.mysql.com/doc/refman/8.0/en/compatibility.html>
6. Multimodel Database with Oracle Database [Електронний ресурс]. – Режим доступу: https://download.oracle.com/otndocs/products/spatial/pdf/12c/Multimodel_Database_with_Oracle_Database_12c_Release_2.pdf
7. PostgreSQL 12.9 Documentation [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/files/documentation/pdf/12/postgresql-12-A4.pdf>
8. NOSQL DEFINITION [Електронний ресурс]. – Режим доступу: <https://hostingdata.co.uk/nosql-database/>
9. Paul Andlinger (2013). RDBMS dominate the database market, but NoSQL systems are catching up [Електронний ресурс]. – Режим доступу: https://db-engines.com/en/blog_post/23
10. Craig S. Mullins (2012). NoSQL (Not Only SQL database) [Електронний ресурс]. – Режим доступу: <https://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL>

11. MongoDB / CRUD Operations [Электронный ресурс]. – Режим доступа: <https://docs.mongodb.com/manual/crud/>
12. MongoDB / Data Modeling Introduction [Электронный ресурс]. – Режим доступа: <https://docs.mongodb.com/manual/core/data-modeling-introduction/>
13. MongoDB / Sharding [Электронный ресурс]. – Режим доступа: <https://docs.mongodb.com/manual/sharding/>
14. MongoDB / Lookup (aggregation) [Электронный ресурс]. – Режим доступа: <https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/>
15. MongoDB / Authentication [Электронный ресурс]. – Режим доступа: <https://docs.mongodb.com/manual/core/authentication/>
16. SQL to MongoDB Mapping Chart [Электронный ресурс]. – Режим доступа: <https://docs.mongodb.com/manual/reference/sql-comparison/>
17. Amazon DynamoDB and Serverless [Электронный ресурс]. – Режим доступа: <https://www.serverless.com/guides/dynamodb>
18. Redis / Using Redis as an LRU cache [Электронный ресурс]. – Режим доступа: <https://redis.io/topics/lru-cache>
19. Redis Mass Insertion [Электронный ресурс]. – Режим доступа: <https://redis.io/topics/mass-insert>
20. Joaquin Casares (2012). Multi-datacenter Replication in Cassandra [Электронный ресурс]. – Режим доступа: <https://www.datastax.com/blog/2012/11/multi-datacenter-replication-cassandra>
21. Cassandra / Architecture / Overview [Электронный ресурс]. – Режим доступа: <https://cassandra.apache.org/doc/latest/architecture/overview.html>
22. Cassandra / Data modeling / Introduction [Электронный ресурс]. – Режим доступа: https://cassandra.apache.org/doc/latest/data_modeling/intro.html
23. Cassandra / Cassandra Query Language (CQL) / Security [Электронный ресурс]. – Режим доступа: <https://cassandra.apache.org/doc/latest/cql/security.html>

24. Cassandra / Architecture / Dynamo [Электронный ресурс]. – Режим доступа: <https://cassandra.apache.org/doc/latest/architecture/dynamo.html>
25. Akhil Mehra (2019). Understanding the CAP Theorem [Электронный ресурс]. – Режим доступа: <https://dzone.com/articles/understanding-the-cap-theorem>
26. Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, Miriam AM Capretz (2013). Data management in cloud environments: NoSQL and NewSQL data stores [Электронный ресурс]. – Режим доступа: <https://journalofcloudcomputing.springeropen.com/track/pdf/10.1186/2192-113X-2-22>
27. Jepsen (2015). MongoDB stale reads [Электронный ресурс]. – Режим доступа: <https://aphyr.com/posts/322-call-me-maybe-mongodb-stale-reads>
28. Dave Kuhlman. A Python Book: Beginning Python, Advanced Python, and Python Exercises [Текст]. Dave Kuhlman – Platypus Global Media, 2011. – 202 с.
29. Python / About [Электронный ресурс]. – Режим доступа: <https://www.python.org/about/>
30. Python / History and License [Электронный ресурс]. – Режим доступа: <https://docs.python.org/3/license.html>
31. Mokhtar Ebrahim (2017). Python web scraping tutorial [Электронный ресурс]. – Режим доступа: <https://likegeeks.com/python-web-scraping/>
32. Write once, run anywhere? (2002) [Электронный ресурс]. – Режим доступа: <https://www.computerweekly.com/feature/Write-once-run-anywhere>
33. Rosalie Chan (2019) The 10 most popular programming languages, according to the, Facebook for programmers [Электронный ресурс]. – Режим доступа: <https://www.businessinsider.de/international/the-10-most-popular-programming-languages-according-to-github-2018-10/?op=1>
34. Arslan ud Din Shafiq (2016). JavaOne 2013 Review: Java Takes on the Internet of Things [Электронный ресурс]. – Режим доступа: <https://www.imarslan.com/javaone-2013-review-java-takes-on-the-internet-of-things>

35. Is the JVM (Java Virtual Machine) platform dependent or platform independent? What is the advantage of using the JVM, and having Java be a translated language? [Электронный ресурс]. – Режим доступа:

<https://www.programmerinterview.com/java-questions/jvm-platform-dependent/>

36. Java Security Overview [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/en/java/javase/14/security/java-security-overview1.html#GUID-2EF91196-D468-4D0F-8FDC-DA2BEA165D10>

37. Kotlin Standard Library [Электронный ресурс]. – Режим доступа: <https://kotlinlang.org/api/latest/jvm/stdlib/>

38. Kotlin overview / Kotlin for JavaScript [Электронный ресурс]. – Режим доступа: <https://kotlinlang.org/docs/js-overview.html>

39. JVM Languages Report: Extended Interview With Kotlin Creator Andrey Breslav [Электронный ресурс]. – Режим доступа: <https://www.jrebel.com/blog/interview-with-kotlin-creator-andrey-breslav>

40. Kotlin / Basic syntax [Электронный ресурс]. – Режим доступа: <https://kotlinlang.org/docs/basic-syntax.html#defining-variables>

41. Sagara Technology Idea Lab (2020). What are The Benefits of Kotlin [Электронный ресурс]. – Режим доступа: <https://sagaratechnology.medium.com/what-are-the-benefits-of-kotlin-d7fdcd1cfc0>

42. Sandeep Agarwal (2019). Pros and Cons of Kotlin for Android App Development [Электронный ресурс]. – Режим доступа: <https://medium.com/quick-code/pros-and-cons-of-kotlin-for-android-app-development-c4b0f95c1324>

43. Jake Frankenfield (2020). Cloud Computing [Электронный ресурс]. – Режим доступа: <https://www.investopedia.com/terms/c/cloud-computing.asp>

44. Cynthia Harvey (2021). AWS vs. Azure vs. Google Cloud: 2021 Cloud Platform Comparison [Электронный ресурс]. – Режим доступа: <https://www.datamation.com/cloud/aws-vs-azure-vs-google-cloud/>

45. Andriy Stashko (2021). Top Cloud Service Providers: A Quick Comparison [Електронний ресурс]. – Режим доступу: <https://www.avenga.com/magazine/top-cloud-service-providers/>

46. Developer guide - AWS SDK for Java 2.x [Електронний ресурс]. – Режим доступу: <https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/home.html>

47. Amazon API Gateway quotas and important notes [Електронний ресурс]. – Режим доступу: <https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html>

48. Розробка стартап-проектів: Конспект лекцій [Електронний ресурс] : навч. посіб. для студ. спеціальностей 151 – «Автоматизація та комп'ютерно-інтегровані технології» та 152 – «Метрологія та інформаційно-вимірювальна техніка» / О. А. Гавриш, К. О. Бояринова, К. О. Копішинська; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл: X,XX Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2019. – 188 с.

49. Рентабельність виробництва і методика визначення її показників. [Електронний ресурс]. – Режим доступу: <http://buklib.net/books/29473/>

50. 5 сил Портера [Електронний ресурс]. – Режим доступу: <https://business.diia.gov.ua/handbook/marketing/5-sil-portera>

51. Разработка продуктовой стратегии компании [Електронний ресурс]. – Режим доступу: https://pidru4niki.com/19670511/marketing/razrabotka_produktovoy_strategii_kompanii#891

52. Маркетингова діяльність [Електронний ресурс]. – Режим доступу: <http://referat-ok.com.ua/marketing/marketingova-diyalnist-2>

53. Бондаренко М.В. NoSQL Бази даних та ORM для роботи з ними./ Бондаренко М.В. // Молодіжна наукова ліга, Модернізація та сучасні українські та світові наукові дослідження. – 2020. С 7-10.