**Київський національний університет імені Тараса Шевченка**

Факультет інформаційних технологій

Кафедра програмних систем і технологій

**УДК 004.942**  *На правах рукопису*

**ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА**

**Тема:** “Розроблення системи керування базами даних з використанням SQLite3”

**Спеціальність** **–** 121 “Інженерія програмного забезпечення”

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

**ВКБР.ІПЗ - 22.00.00.000 ІПЗ**

Рішенням Екзаменаційної комісії

**Консультант**

**з питань нормоконтролю**

**к. ф.-м. н., доц.\_\_\_\_\_\_\_\_\_\_\_/О.М.Супрун/**

**Студент**

**ІПЗ-43\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Д.В.Поданенко /**

**Науковий керівник**

**к. ф.-м. н., доц.\_\_\_\_\_\_\_\_\_\_\_/О.М.Супрун/**

Допускається до захисту

**В.о. завідувача кафедри**

**д.т.н.,с.н.с. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Г.В. Порєв/**

Київ - 2020

випускна кваліфікаційна робота студента

Поданенка Дениса Володимировича

захищена з оцінкою

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Голова Екзаменаційної комісії

професор, доктор техн. наук Онищенко В.В

**Київський національний університет імені Тараса Шевченка**

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО

Зав. кафедри програмних систем і технологій

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (О.С.Бичков)

**ЗАВДАННЯ**

**НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ**

Поданенко Денису Володимировичу

1. **Тема випускної кваліфікаційної бакалаврської роботи**

**“**Розроблення системи керування базами даних з використанням SQLite3**”**

керівник проекту (роботи) Супрун Ольга Миколаївна, к.ф.-м.н., доцент

затверджені наказом вищого навчального закладу від „\_\_”\_\_\_\_\_20\_\_р. №\_\_\_\_\_\_

**2. Строк здачі студентом закінченої роботи** „\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2020 р.

**3. Вихідні дані до роботи**: підручники, навчальні посібники, статті, Інтернет-ресурси

**4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)**

Аналітична частина:

* обґрунтувати актуальність розробки системи керування базами даних;
* дослідити існуючі види баз даних;
* визначити переваги та недоліки існуючих баз даних та СУБД;
* визначити концепції СУБД яка буде створена;

Практична частина

* визначити особливості архітектурного рішення СУБД, що пропонується;
* спроектувати структуру СУБД;
* розробити СУБД;
* порівняти та проаналізувати результати розробленої СУБД із існуючими СУБД.

**5. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються**

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Консультант | Підпис, дата | |
| Завдання видав | Завдання прийняв |
| Розділ 1 Огляд існуючих рішень | Супрун О.М. |  |  |
| Розділ 2 Проектування та розробка СУБД | Супрун О.М. |  |  |
| Розділ 3 Результат роботи програми | Супрун О.М. |  |  |
|  |  |  |  |

**6. Дата видачі завдання** 29 листопада 2019

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /О.М.Супрун/

Завдання прийняв до виконання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ /

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Назви етапів бакалаврської роботи | Термін виконання  етапів роботи | Відмітка про виконання |
| 1. | Уточнення постановки задачі | 29.11.2019 - 5.12.2019 | Виконано |
| 2. | Аналіз літератури | 5.12.2019 – 10.01.2020 | Виконано |
| 3. | Аналіз існуючих баз даних | 10.01.2020 – 16.01.2020 | Виконано |
| 4. | Аналіз існуючих СУБД | 16.01.2020 – 24.01.2020 | Виконано |
| 5. | Обґрунтування вибору рішення | 24.01.2020 – 26.01.2020 | Виконано |
| 6. | Опис структурної моделі СУБД | 26.01.2020 – 5.02.2020 | Виконано |
| 7. | Побудова структурної моделі СУБД | 5.02.2020 – 10.02.2020 | Виконано |
| 8. | Розроблення програмного забезпечення | 10.02.2020 – 20.04.2020 | Виконано |
| 9. | Тестування розробленого програмного забезпечення | 20.04.2020 – 1.05.2020 | Виконано |
| 10. | Оформлення і друк пояснювальної записки | 1.05.2020 – 24.05.2020 | Виконано |
| 11. | Оформлення презентації | 24.05.2020 – 25.05.2020 | Виконано |
| 12. | Отримання рецензії | 26.05.2020 | Виконано |
| 13. | Затвердження пояснювальної записки роботи завідувачем кафедри | 27.05.2020 – 10.06.2020 | Виконано |
| 14. | Захист дипломної роботи | 26.06.2020 | Виконано |

Студент – бакалавр \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Д.В. Поданенко /

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /О.М.Супрун/

**АНОТАЦІЯ**

**Випускна кваліфікаційна бакалаврська робота:** 47 c., 6 рис., 1 додаток, 9 джерел.

**Тема:** Розроблення системи керування базами даних з використанням SQLite3

**Об’єкт дослідження:** технологія хмарних баз даних.

**Предмет дослідження:** додаток для керування базами даних на основі SQLite3.

**Мета роботи:** оптимізація роботи клієнта у віддаленому доступі, з базами даних на основі SQLite3.

**Результати дослідження:** Оцінено можливість використання баз даних на основі SQLite3 у онлайн додатку. Досліджено фраймворки Angular для Front-end частини та .Net Core3 для Back-end частини додатку на достатність їх вбудованого функціоналу для реалізації онлайн системи керування базами даних з використанням SQLite3.

**Висновок:** Розроблено веб-сервіс для керування базами даних з використанням SQLite3, описано економічне обґрунтування проекту. Розроблений веб-додаток представляє з себе сукупність:

* клієнтська частина веб-додатку для взаємодії з системою на Angular;
* серверна частина для надання інформації на .Net Core3;
* взаємодія серверної частини і баз даних за допомогою Entity Framework.

**АННОТАЦИЯ**

**Выпускная квалификационная бакалаврская работа:** 47 с., 6 рис., 1 приложение, 9 источников.

**Тема:** Разработка системы управления базами данных с использованием SQLite3

**Объект исследования:** технология облачных баз данных.

**Предмет исследования:** приложение для управления базами данных на основе SQLite3.

**Цель работы:** оптимизация работы клиента в удалённом доступе, с базами данных на основе SQLite3.

**Результаты исследования:** Оценена возможность использования баз данных на основе SQLite3 в онлайн приложении. Исследованы фреймворки Angular, для Front-end части, и .Net Core3, для Back-end части приложения, достаточно ли в них функциональности для реализации онлайн системы управления базами данных с использованием SQLite3.

**Вывод:** Разработан веб-сервис для управления базами данных с использованием SQLite3, описано экономическое обоснование проекта. Разработанное веб-приложение представляет из себя совокупность:

* Клиентская часть веб-приложения для взаимодействия с системой на Angular;
* Серверная часть для предоставления информации на .Net Core3;
* Взаимодействие серверной части и баз данных с помощью Entity Framework.

**ANNOTATION**

**Graduation qualification bachelor's work:** 47 p, 6 fig., 1 appendix, 9 sources.

**Topic:** Development of database management system using SQLite3

**Object of research:** technology of cloud databases.

**Subject of research:** optimization of client work, with remote access, with databases based on SQLite3.

**Purpose:** develop a DBMS with the possibility of convenient remote access.

**Results:** Evaluated the possibility of using SQLite3-based databases in an online application . Investigated the Angular frameworks for the Front-end part, and .Net Core3, for the Back-end part of the application, whether they have enough functionality to implement an online database management system using SQLite3.

**Conclusion:** A web service was developed for managing databases using SQLite3, and the business case for the project was described. The developed web application is a collection of:

• The client part of the web application for interacting with the system on Angular;

• The server part to provide information on .Net Core3;

• Interaction of the server side and databases using the Entity Framework.

**Зміст**

[**ВСТУП 10**](#_Toc42960456)

[**1. ТЕХНОЛОГІЯ SQL 12**](#_Toc42960457)

[**1.1. Огляд технології SQL 12**](#_Toc42960458)

[**1.2. Економічна вигода, від користування SQL 13**](#_Toc42960459)

[**1.3. Перспективи у майбутньому 13**](#_Toc42960460)

[**1.4. Огляд СУБД 14**](#_Toc42960461)

[**1.4.1. Microsoft SQL Server 14**](#_Toc42960462)

[**1.4.2. MySQL 16**](#_Toc42960463)

[**1.4.3. SQLite3 17**](#_Toc42960464)

[**Висновок 19**](#_Toc42960465)

[**2. РОЗРОБКА СУБД 21**](#_Toc42960466)

[**2.1. Тип СУБД, що розроблюється 21**](#_Toc42960469)

[**2.2. Back-end Part 23**](#_Toc42960470)

[**2.3. Front-end Part 27**](#_Toc42960475)

[**2.4. Архітектура додатку 30**](#_Toc42960481)

[**Висновок 33**](#_Toc42960482)

[**3. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ПРОЕКТУ 34**](#_Toc42960483)

[**3.1. Опис сервісу керування БД 34**](#_Toc42960484)

[**3.2. Аналіз ринку збуту додатку 34**](#_Toc42960485)

[**3.3. Визначення витрат на запуск проекту, та його підтримку 35**](#_Toc42960486)

[**ВИСНОВКИ 37**](#_Toc42960487)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 38**](#_Toc42960488)

[**ДОДАТОК А 39**](#_Toc42960489)

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ**

БД – База даних

СУБД – Система управління базами даних

SQL – Structured query language

PAAS – Platform as a service

IAAS – Infrastructure as a service

SAAS – Software as a service

# ВСТУП

Наразі триває інформаційна епоха. Це означає широкі можливості для всіх осіб вільно передавати та володіти інформацією. Не залишились осторонь і компанії. Кількість інформації і швидкість її обробки замітно зросли за останнє десятиліття.

Для зберігання інформації, зараз, найчастіше використовуються бази даних. Існують різні види баз даних:

* Ієрархічні
* Мережеві
* Реляційні
* Об’єктно-орієнтовані

Для взаємодії із базами даних використовуються системи управління базами даних. На сьогодні існує дуже багато різновидів СУБД, для різного спектру задач. Використовують СУБД, найчастіше різні діалекти SQL. Кожен діалект має спільну основу та якийсь свій функціонал, або особливості.

Промислова архітектура зазвичай однакова. Компанія володіє серверами, які частіше всього зберігає у серверних свого офісу. Працівники компанії мають доступ до серверів, лише через свої робочі комп’ютери, або корпоративну локальну мережу.

Такий підхід має багато переваг і добре себе зарекомендував. Але існують і мінуси. Перш за все це витрати на апаратуру, приміщення, підтримання обладнання. Другий мінус, це надмірна складність користування БД у віддаленому доступі.

Для користування БД у віддаленому доступі, для компанії необхідно створити хмарне середовище. Працівники повинні користуватись програмами, для шифрування даних, що сповільнить комп’ютер або мережу. Треба розробити, або купити програми аутентифікації, як McAfee або Cisco Gateway. Скоріше за все, для підтримки такого механізму, буде потрібна підтримка, котра буде допомагати працівникам у разі помилок, або отримання доступу. На все це потрібен час і гроші.

У таких ситуаціях рішенням, може стати, перехід на спеціалізовані хмарні сервіси від інших компаній. Такі почали набирати обороти у 90-х. А зараз відбувається бум хмарних технологій. Усі гіганти переводять свій продукт на використання хмарних технологій.

Хмарні технології – це безпечно, зручно, швидко і відносно не дорого.

БД у хмарі майже завжди представляються, або на рівні PAAS, або на рівні IAAS. БД на такому рівні можуть користуватись лише розробники додатків. А UI у кращому випадку буде в консолі. Це не зручно і не зовсім підходить, наприклад для аналітика магазину, або менеджера страхової компанії.

Потрібна СУБД у хмарі на рівні SAAS. Такі додатки не мають великої цінності, якщо не працювати віддалено. Але при роботі віддалено, вони мають вагому цінність для компаній.

Також такі додатки можуть стати гарним середовищем для вивчення баз даних. Коли студент і викладач мають доступ до однієї БД. Студент може бачити які зміни робить викладач і результат цих змін одразу.

Метою випускної кваліфікаційної роботи є розроблення системи керування базами даних з використанням SQLite3.

Відповідно до поставленої мети в роботі вирішувалися наступні завдання:

* Аналіз та огляд існуючих діалектів SQL, які краще за все підходять для поставленої задачі.
* Аналіз та огляд існуючих інструментів для створення СУБД у хмарі.
* Розробка структурної схеми сервісу.
* Розробка сервісу.
* Тестування і підведення підсумків.

# 1. ТЕХНОЛОГІЯ SQL

Структурована мова запитів (SQL) - це спеціалізована мова програмування, яка займається визначенням даних та маніпулювання ними, системами управління базами даних, а також є стандартною мовою програмування, що стосується реляційних даних.

Термін SQL використовується для опису як мови програмування, так і сервера баз даних. Наприклад, коли люди говорять про наявність системи SQL, те, що вони насправді мають на увазі, у них є додаток, підкріплений реляційною базою даних, як правило, Microsoft SQL Server (дуже багато людей називають це SQL або "Sequel").

Як найбільш поширена мова бази даних, SQL настільки поширений, що ви, мабуть, не знаєте, наскільки ця технологія використовується у звичаному повсякденному житті. Наприклад, коли людина купуєте товар у магазині, деталі кожної одиниці покупки будуть зберігатися у безлічі баз даних, що забезпечують контроль запасів, бізнес-аналітику та дані для маркетингових цілей.

Якщо треба отримати технічні, реляційні системи управління базами даних (RDBMS), то в інших програмах, також використовують SQL.

## **Огляд технології SQL**

База даних SQL - це високоефективний інструмент для швидкої доставки одержаних даних. Оскільки це структурована мова запитів, вона працює на структурованих даних, таких як тип продукту та місцезнаходження клієнта і знає взаємозв'язок між ними в системі. Це означає, що вона може повідомити вам цінну інформацію, наприклад, чи перукарі в Житомирі частіше купують страховку для автомобілів, ніж будівельники в Бучі.

Добре розроблена система баз даних SQL дозволить розширити знання бізнесу про кожен аспект його існування завдяки фактам, чистим та простим. Ці знання дозволяють ідентифікувати та реагувати на нові тенденції у бізнесі та на ринку. Продукти та послуги, які приносять найбільший дохід, можна виділити командою програми, а співвідношення між типами та місцями клієнтів можна встановити, щоб показати, де лежить прибутковість (тим самим мінімізуючи ринковий ризик).

## **Економічна вигода, від користування SQL**

Два слова: Business intelligence. Все більше підприємств покладаються на програмні додатки BI для аналізу даних з різних джерел, таких як бази даних SQL. ВІ заходи включають обробку даних, онлайнову аналітичну обробку (OLAP), статистичний аналіз, прогнозування та звітність. Коротше кажучи, BI має сенс у великій кількості інформації, що стосується бізнесу, від деталей його клієнтів до річних тенденцій продажів, щоб керувати майбутнім розвитком та потоком доходу.

Бази даних SQL підходять від найменших підприємств до для великих міжнародних корпорацій. Один із прикладів Microsoft SQL Server 2014 має на меті забезпечити управління даними бізнес-класу для різноманітних додатків баз даних з метою зменшення витрат, підвищеної безпеки (включаючи відновлення після аварій) та надійного часу роботи на додаток до звичайних програмних функцій, орієнтованих на BI.

Як приклад величезного обсягу даних, які можуть зберігатися в цих системах, версія Enterprise SQL-сервера може керувати базами даних, розміром яких 524 петабайта, або 524 квадратних байтів - приголомшливий обсяг інформації. Кількість ідей, зібраних із подібного, майже необмежена, від чого можна покращити ланцюжок поставок та мати більш повне уявлення про потреби клієнта, до збільшення ринкової частки компанії за допомогою більш сильних маркетингових кампаній.

## **Перспективи у майбутньому**

Гібридні хмарні можливості означають, що більшість людей в бізнесі можуть отримати доступ до серверів SQL за зниженою ціною - благо для підприємств з великими планами, але обмеженими ІТ-бюджетами. Але як із хмарними, так і локальними SQL-серверами, управління даними може зростати поряд із бізнесом. Деякі компанії не хочуть раптом перейняти такий спосіб роботи, але можна бути впевненим, що багато підприємств, які тільки починаються або ще досить малі, щоб зробити стрибок, захочуть, те що може запропонувати база даних SQL – велика кількість даних, чітко організована та готова до наступного кроку в професійному розвитку.

## **Огляд СУБД**

### **Microsoft SQL Server**

Microsoft SQL Server - це система управління реляційними базами даних, розроблена Microsoft. Як сервер бази даних, це програмний продукт з основною функцією зберігання та отримання даних, як цього вимагають інші програмні програми - який може працювати або на тому ж комп'ютері, або на іншому комп'ютері в мережі (включаючи Інтернет). Майкрософт продає щонайменше десяток різних видань Microsoft SQL Server, спрямованих на різні аудиторії та на робочі навантаження, починаючи від невеликих одно машинних додатків до великих Інтернет-додатків із багатьма одночасними користувачами.

**Переваги Microsoft SQL Server**:

* **Програмне забезпечення для управління підприємством** - Microsoft SQL Server включає професійне програмне забезпечення для управління базами даних на рівні підприємств. Кілька конкурентів, наприклад MySQL, розробили подібне програмне забезпечення за останні роки, але Microsoft SQL Server простіший у використанні та має більше функцій. Наприклад, повна підтримка тригерів підтримується у продукті Microsoft. MySQL нещодавно представив тригери, але вони не підтримуються повністю. Програмне забезпечення, пропоноване Microsoft, також пропонує тісну інтеграцію з .NET рамкою, що не стосується конкуруючих продуктів.
* **Відмінна підтримка відновлення даних** - Пошкоджені дані завжди викликають занепокоєння при втраті електроенергії або неправильному відключенні. Microsoft SQL Server має ряд функцій, які сприяють відновленню та відновленню даних. Хоча окремі таблиці неможливо створити резервну копію або відновити, доступні повні варіанти відновлення бази даних. Завдяки використанню файлів журналів, кешування та резервного копіювання продукт Microsoft дозволяє впевнено почувати, що варіантів відновлення після аварій є багато.

**Недоліки Microsoft SQL Server**:

* **Ціна** - Microsoft SQL Server створений для підтримки мільйонів записів на підприємстві. Через свою надійну технологію SQL Server також поставляється з високою ціною. Корпоративні видання SQL Server можуть коштувати тисячі доларів, залежно від варіантів та обладнання, необхідного для розміщення програми бази даних. Для більшості малих підприємств ця ціна зависока і не відповідає бюджету. З цієї причини багато малих компаній використовують MySQL для back-end бази даних. MySQL безкоштовний для приватних осіб, і широко відомий веб-та настільним програмістам.
* **Зручність і простота** - Microsoft SQL Server використовує основну мову, яка відрізняється від інших програм бази даних, таких як MySQL та Oracle. Компаніям, яким потрібні послуги програмування чи аналізу, може бути складніше знайти ресурс, який розміщує ці послуги за розумною ціною. Також програмісти повинні вивчити іншу мову та налаштування для середовища Microsoft SQL Server, тому компанії, які хочуть оновити, повинні навчити нинішніх працівників як працювати з додатком.
* **Обладнання** - Старіші версії SQL Server можуть встановлюватися на більш старому апаратному обладнанні, однак новіші версії програми потребують більш досконалих технологій для підтримки ресурсів, необхідних базі даних. SQL Server також вимагає встановленої на машині рамки .NET, яка є окремим компонентом, який використовують розробники. Якщо компанія планує мати дуже велику базу даних, жорсткий диск також потребує відповідної кількості місця на додаток до гігабайт простору, необхідного лише для двигуна бази даних.

### **MySQL**

MySQL - це система управління реляційними базами даних з відкритим кодом (RDBMS). MySQL - це вільне програмне забезпечення з відкритим кодом за умовами Загальної публічної ліцензії GNU, а також доступне під різноманітними власними ліцензіями. MySQL належить та спонсорується шведською компанією MySQL AB, яку придбала компанія Sun Microsystems (тепер корпорація Oracle). У 2010 році, коли Oracle придбав Sun, Widenius відправив проект MySQL з відкритим кодом для створення MariaDB. MySQL є компонентом стеку програмного забезпечення для веб-додатків LAMP (та інших), що є абревіатурою для Linux, Apache, MySQL, Perl / PHP / Python. MySQL використовується у багатьох веб-додатках, керованих базами даних, включаючи Drupal, Joomla, phpBB та WordPress. MySQL також використовується на багатьох популярних веб-сайтах, включаючи Facebook, Flickr, MediaWiki, Twitter, та YouTube.

**Переваги MySQL**

* **Підтримка доступна, коли це необхідно** – На протязі усієї історії підтримки користувачів Oracle, можна бачити відмінний результат. MySQL - яка почала свою діяльність як платформа з відкритим кодом - означає, що існує велика і процвітаюча спільнота розробників та ентузіастів, до яких можна звернутися за допомогою. Багато в чому це пов’язано з популярністю рішення, кінцевим результатом якого не бракує експертів.
* **Open-Source** - Купівля Oracle Sun Microsystems (і, за асоціацією, MySQL) зустріла певну суперечку з боку спільноти розробників. Загальний страх полягав у тому, що Oracle перетворить цей інструмент у закриту власну екосистему. На щастя, хоча Oracle дещо посилив свої зчеплення з MySQL, його все ще можна вважати варіантом бази даних з відкритим кодом, оскільки код все ще доступний для безкоштовного онлайн.

**Недоліки MySQL**

* **Погане масштабування -** Хоча MySQL оснащений для обробки практично необмеженого обсягу даних, технологія має тривожну тенденцію припиняти обробку, якщо MySQL змушений займатися занадто великою кількістю операцій в даний момент часу. Це порівняно низьке масштабування ефективності означає, що кожен, хто має високий рівень одночасності, мабуть, повинен шукати альтернативу.
* **Розвиток не керується громадою - отже, відстає** - Оскільки Oracle купив MySQL, то, здається, прогрес зупинився, за останні кілька років вийшов лише один головний реліз. Компанія не приймає розроблені громадою патчі, а також не намагалася пропонувати користувачам будь-яку дорожню карту для розробки MySQL. Розробникам насправді немає можливості обговорити систему управління базами даних з Oracle - і це проблема.
* **Його функціональність, як правило, сильно залежить від модифікацій**  - Його функціональність має велику залежність від модифікацій. Хоча налаштування MySQL порівняно просто, вона має меншу функціональність у звичайній формі, ніж багато інших систем баз даних на ринку. Певні функції - такі як пошук тексту та відповідність ACID - залежать не від основної системи, а від додатків та модів. Хоча це правда, що існує безліч добре створених додатків для MySQL, відстеження їх іноді може викликати біль, а деякі розробники можуть просто вибрати альтернативу, яка - хоча і не так просто при інсталювані - пропонує більш безпосередню функціональність.

### **SQLite3**

SQLite - це бібліотека на мові С, яка реалізує невеликий, швидкий, автономний, високонадійний, повнофункціональний механізм баз даних SQL.

SQLite - це найбільш використовуваний двигун бази даних у світі. SQLite вбудований у всі мобільні телефони та більшість комп’ютерів і постачається у безлічі інших програм, якими користуються люди щодня.

Формат файлів SQLite стабільний, сумісний між платформами та назад, і розробники зобов’язуються зберегти його таким чином принаймні до 2050 року. Файли баз даних SQLite зазвичай використовуються як контейнери для передачі багатого вмісту між системами і як довгостроковий архівний формат даних. У активному використанні є понад 1 трлн баз даних SQLite.

Вихідний код SQLite знаходиться у загальнодоступному доступі і може безкоштовно використовувати його для будь-яких цілей.

**Переваги SQLite3**

* **Краща продуктивність -** Читання та запис із бази даних SQLite швидше, ніж читання та запис окремих файлів з диска. На 35% швидше, ніж файлова система проти зовнішніх BLOB. Додаток має лише завантажувати потрібні йому дані, а не читати весь файл і проводити повний аналіз в пам'яті. Невеликі редагування перезаписують лише ті частини файлу, які змінюються, скорочуючи час запису та знос на SSD-накопичувачах.
* **Міграція** - Файл програми переноситься для всіх операційних систем, 32-розрядної та 64-розрядної, великих та малих endian-архітектур. Федерація програм, можливо, всі написані різними мовами програмування, може отримати доступ до одного і того ж файлу програми без жодних проблем щодо сумісності. Кілька процесів можуть приєднуватися до одного файлу програми та можуть читати та писати, не заважаючи один одному. Різноманітний вміст, який інакше може бути збережений як "файли файлів", інкапсульований в один файл диска для більш простого транспорту через scp / ftp, USB-палицю та / або вкладення електронної пошти.
* **Надійність** - Вміст можна постійно та атомно оновлювати, так що при втраті електроенергії або збої в роботі втрачається мало або нічого не відбувається. Помилки набагато рідше в SQLite, ніж у власно написаному I/O файлі коду. SQL-запитів у багато разів менше, ніж у еквівалентному процедурному коді, і оскільки кількість помилок на рядок коду є приблизно постійною, це означає менше помилок у цілому.
* **Доступність** - Вміст бази даних SQLite можна переглядати за допомогою різноманітних сторонніх інструментів. Вміст, що зберігається в базі даних SQLite, швидше за все буде відновлений десятиліттями в майбутньому, задовго після того, як будуть втрачені всі сліди оригінальної програми. Дані живуть довше, ніж код. Файли баз даних SQLite рекомендовані Бібліотекою Конгресу США як формат зберігання для довготривалого збереження цифрового контенту.

**Недоліки SQLite3**

* **Мала функціональність** – В SQLite3 оминули частину функціоналу SQL. Наприклад немає Right-join, for-each trigger, drop-column.

## **Висновок**

Були вибрані основні бази даних сьогодення. Усі вони мають багато переваг і недоліків.

Microsoft SQL Server – для виконання завдання роботи не підходить. Ця БД більше підходить для великих и складних додатків. Наприклад ERP-системи, або комп’ютерні ігри. Розроблений додаток, наразі, не позиціонує себе, в якості складної системи. Також варто відзначити велику ціну за користування їм. Так існують різні варіанти постачання Microsoft SQL Server, але усі вони платні.

MySQL – для виконання завдання роботи не підходить. Ця БД останнім часом, дуже рідко випускає нові версії. Oracle не докладає достатньо зусиль для розвитку MySQL. Необхідність додавати модифікації.

SQLite3 – задовольняє вимогам розроблюваного додатку. Ця БД все більше і більше завойовує ринок. Велика кількість девайсів і сервісів, вже користуються нею. Повність open-source і керується громадою. Безкоштовна, що є важливим для нового проекту. Особлива архітектура, у відмінності від інших БД SQLite3 не має серверів. Що добре вписується в архітектуру розробленого додатку.

# 2. РОЗРОБКА СУБД



## **Тип СУБД, що розроблюється**

**Система керування базами даних** (СУБД) - це програмний пакет, призначений для визначення, маніпулювання, отримання та управління даними в базі даних. СУБД, як правило, маніпулює самими даними, форматом даних, іменами полів, структурою запису та структурою файлів. СУБД також визначає правила перевірки та маніпулювання цими даними. СУБД позбавляє користувачів обрамлення програм для підтримки даних. Мови запитів четвертого покоління, такі як SQL, використовуються разом із пакетом СУБД для взаємодії з базою даних.

Система управління базами даних отримує інструкції від адміністратора бази даних (DBA) і відповідно доручає системі внести необхідні зміни. Ці команди можуть завантажувати, витягувати або змінювати наявні дані з системи.

СУБД завжди забезпечує незалежність даних. Будь-яка зміна механізму зберігання та форматів зберігання виконується без зміни всієї програми. Існує чотири основні типи організації баз даних:

* Реляційна база даних: Дані організовані як логічно незалежні таблиці. Взаємозв'язки між таблицями відображаються за допомогою спільних даних. Дані в одній таблиці можуть посилатися на подібні дані в інших таблицях, що підтримує цілісність посилань між ними. Ця особливість називається референтною цілісністю - важливим поняттям у системі реляційних баз даних. Операції, такі як "select" та "join", можуть виконуватися на цих таблицях. Це найбільш широко використовувана система організації баз даних.
* Плоска база даних: Дані впорядковані в єдиний вид запису з фіксованою кількістю полів. Цей тип бази даних стикається з більшою кількістю помилок через повторюваний характер даних.
* Об'єктно-орієнтована база даних: Дані впорядковані за подібністю до об'єктно-орієнтованих концепцій програмування. Об'єкт складається з даних та методів, тоді як класи групують об'єкти, що мають подібні дані та методи.
* Ієрархічна база даних: Дані організовані за ієрархічними зв’язками. Це стає складною мережею, якщо порушуються відносини один до багатьох.

У наш час все більше і більше додатків переводять, або одразу розроблюють на хмарній платформі.

Архітектура хмарних обчислень відноситься до компонентів та підкомпонентів, необхідних для хмарних обчислень. Ці компоненти, як правило, складаються з передньої платформи (товстий клієнт, тонкий клієнт, мобільний пристрій), платформи зворотного зв'язку (сервери, сховище), хмарної доставки та мережі (Інтернет, Інтранет, Інтерклод). Ці компоненти поєднуються в архітектурі хмарних обчислень.

Архітектури хмарних обчислень складаються з платформ, що називаються клієнтами або хмарними клієнтами. Ці клієнти - це сервери, товсті клієнти, тонкі клієнти, нульові клієнти, планшети та мобільні пристрої, з якими користувачі безпосередньо взаємодіють. Ці клієнтські платформи взаємодіють із зберіганням хмарних даних через додаток (проміжне програмне забезпечення), через веб-браузер або через віртуальний сеанс. Зокрема, для віртуальних сеансів потрібен захищений кадр алгоритму шифрування, який охоплює весь інтерфейс.

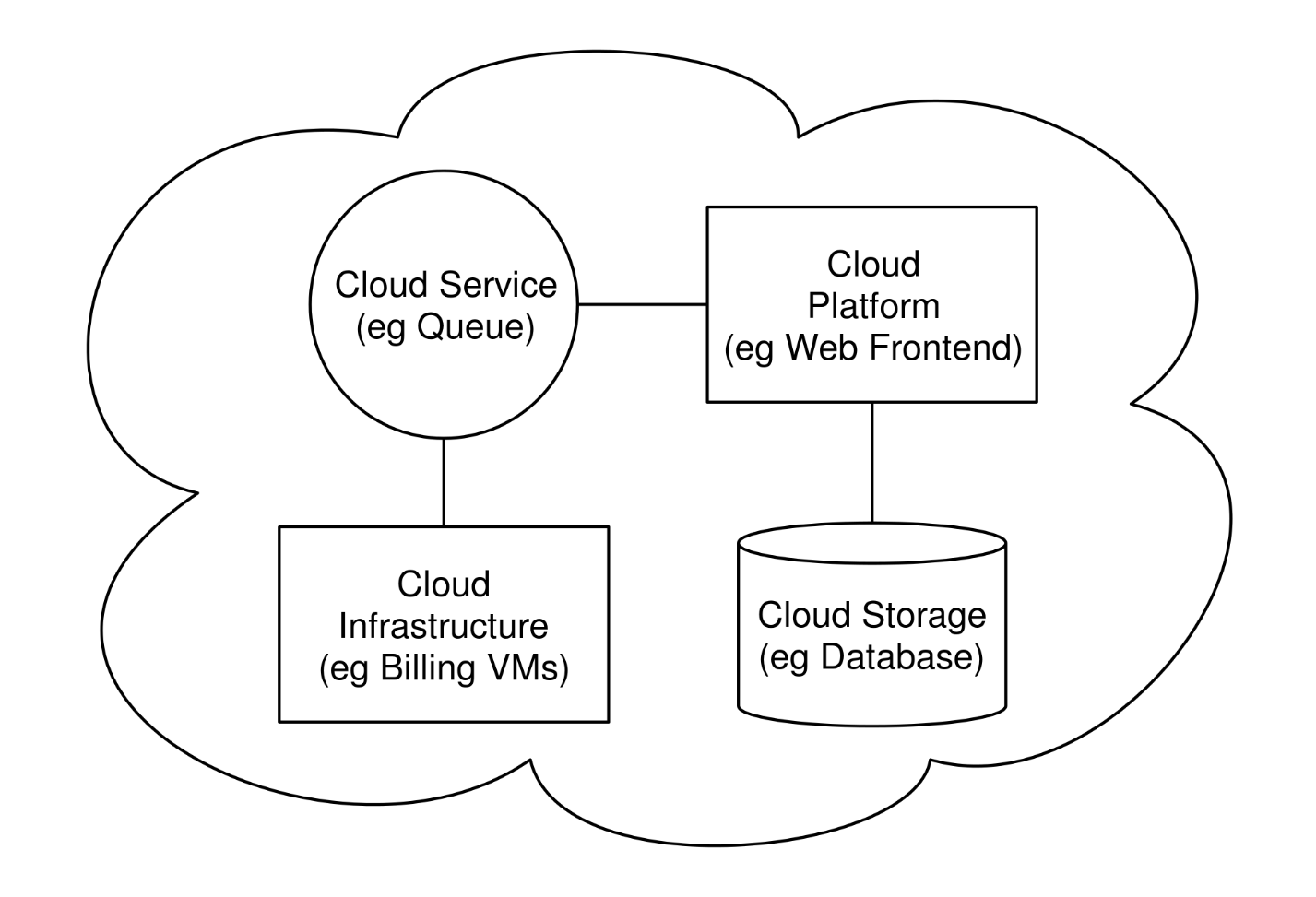


Рисунок 2.1 – Архітектура додатку на хмарній платформі

Так як більшість СУБД, це старі продукти, вони написані як звичайні desktop додатки. Досі не існує, лідера у ніші хмарних СУБД.

## **Back-end Part**

**Back-end** – це частина програмної системи, яка не є видимою або доступною для користувача цієї системи.

Back-end фокусується на:

* Мові сценаріїв, такі як JavaScript, Node.js, PHP, Python, Ruby або Perl або мови компільованого типу C #, Java або Go
* Автоматизовані рамки тестування мови, що використовується
* Доступ до даних додатків
* Застосування бізнес-логіки
* Адміністрування бази даних
* Масштабованість
* Висока доступність
* Питання безпеки, автентифікація та авторизація
* Архітектура програмного забезпечення
* Перетворення даних
* Методи та програмне забезпечення резервного копіювання

Серед безлічі фреймворків для Back-end, для виконання кваліфікаційної бакалаврської роботи .Net Core.

.**NET Core** - це фактично крос-платформний Фреймворк, з відкритим кодом для розробки, підключених до інтернету, хмарних сучасних веб-додатків. Завдяки цьому, цілком можливо створити ефективніші веб-додатки, а також сервіси, а також мобільні додатки та навіть програми IoT. Значно простіше кажучи, це значне перероблення рамки ASP.NET. Технологія постачається з великою кількістю переваг, таких як поліпшення продуктивності, більш висока безпека, менше кодування тощо. Багато компаній належним чином прийняли цю найновішу технологію для створення власних додатків. Хоча ASP.NET залишається прекрасним поєднанням різних моделей веб-розробки, що складається з усіх необхідних сервісів для створення надійних веб-додатків для різних видів бізнесу.

.Net Core має наступні переваги:

* Гарну продуктивність
* Підтримка крос-платформної розробки
* Менше програмного коду
* Легше розуміння програм
* Підтримка веб-додатків, які базуються на хмарних технологіях

Back-end розроблений на партерні MVC.

**MVC –** це модель дизайну програмного забезпечення, яка зазвичай використовується для розробки інтерфейсів користувача, яка розділяє відповідну логіку програми на три взаємопов'язані елементи. Це робиться для того, щоб відокремити внутрішнє представлення інформації від способів подання та прийняття інформації від користувача. Цей вид візерунка використовується для проектування макета сторінки. Традиційно використовується для графічних інтерфейсів користувачів настільних ПК (GUI), ця модель стала популярною для розробки веб-додатків. Популярні мови програмування, такі як JavaScript, Python, Ruby, PHP, Java, C # і Swift, мають рамки MVC, які використовуються для розробки Інтернету або мобільних додатків прямо з коробки.

Такий патерн краще всього підійде для реалізації REST API.

**Representational state transfer (REST)** – це архітектурний стиль програмного забезпечення, який визначає набір обмежень, які будуть використані для створення веб-служб. Веб-сервіси, які відповідають архітектурному стилю REST, називаються послугами RESTful Web, забезпечують сумісність між комп'ютерними системами в Інтернеті. Веб-сервіси RESTful дозволяють запитуючим системам отримувати доступ та маніпулювати текстовими поданнями веб-ресурсів за допомогою рівномірного та заздалегідь визначеного набору операцій без стану. Інші види веб-служб, такі як веб-сервіси SOAP, піддають власні довільні набори операцій.

"Веб-ресурси" вперше були визначені у Всесвітній павутині як документи або файли, визначені за їх URL-адресами. Однак сьогодні вони мають набагато більш загальне та абстрактне визначення, яке охоплює кожну річ, сутність чи дію, які можна будь-яким чином ідентифікувати, назвати, вирішити, обробляти чи виконувати будь-якими способами. У веб-службі RESTful запити, що надсилаються до URI ресурсу, викликають відповідь з корисним навантаженням, відформатованим у HTML, XML, JSON або іншому форматі. Відповідь може підтвердити, що в стан ресурсу була внесена деяка зміна, і відповідь може забезпечити гіпертекстові посилання на інші пов'язані ресурси. Якщо використовується HTTP, як це найбільш часто, доступними операціями (методами HTTP) є GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS та TRACE.

Використовуючи протокол без стану та стандартні операції, системи RESTful мають на меті швидку продуктивність, надійність та можливість зростати за рахунок повторного використання компонентів, якими можна керувати та оновлювати, не впливаючи на систему в цілому, навіть під час її роботи. Термін REST був введений і визначений у 2000 році Роєм Філдінгом у своїй докторській дисертації. Дисертація Філдінга пояснила принципи REST, які були відомі як "HTTP-об'єктна модель", починаючи з 1994 року, і використовувались при розробці стандартів HTTP 1.1 та Уніфікованих ідентифікаторів ресурсів (URI). Термін покликаний викликати образ поведінки добре спроектованої веб-програми: це мережа веб-ресурсів, де користувач просувається через додаток шляхом вибору ідентифікаторів ресурсів, таких як http: // www .example.com / articles / 21 і такі операції з ресурсами, як GET або POST (переходи стану додатків), в результаті чого подання наступного ресурсу (наступного стану програми) передається кінцевому користувачеві для їх використання.

У якості інструменти для роботи з БД на стороні Back-end, було вирішено використовувати Entity Framework.

**Entity Framework** – це структура з відкритим кодом об'єктно-реляційного відображення (ORM) для .NET Core. Технологія була частиною .NET Framework, але після версії Entity Framework 6 вона відокремлена від .NET Framework.

Entity Framework - це сукупність технологій в .NET Core, які підтримують розробку програмно-орієнтованих програм. Архітектори та розробники програм, орієнтованих на дані, як правило, борються з необхідністю досягнення двох дуже різних цілей. Вони повинні моделювати сутності, відносини та логіку бізнес-проблем, які вони вирішують, а також вони повинні працювати з двигунами даних, які використовуються для зберігання та отримання даних. Дані можуть охоплювати декілька систем зберігання даних, кожна з яких має власні протоколи; навіть додатки, які працюють із єдиною системою зберігання, повинні співвідносити вимоги системи зберігання даних з вимогами написання ефективного та бездоганного коду програми. Entity Framework дозволяє розробникам працювати з даними у вигляді об'єктів і властивостей, що стосуються домену, таких як клієнти та адреси клієнтів, без необхідності стосуватися самих базових таблиць та стовпців бази даних, де зберігаються ці дані. За допомогою Entity Framework розробники можуть працювати на більш високому рівні абстракції, коли вони обробляють дані, і можуть створювати та підтримувати орієнтовані на дані додатки з меншим кодом, ніж у традиційних програмах.

У якості інструменти для ведення зручної документації та тестування REST API, був використаний **Swagger UI**.

**Swagger UI -** дозволяє кожному - будь то ваша команда розробників або кінцеві споживачі - візуалізувати та взаємодіяти з ресурсами API, не маючи жодної логіки впровадження. Він автоматично генерується з вашої специфікації OpenAPI (раніше відомої як Swagger), а візуальна документація полегшує реалізацію зворотного кінця та споживання на стороні клієнта.

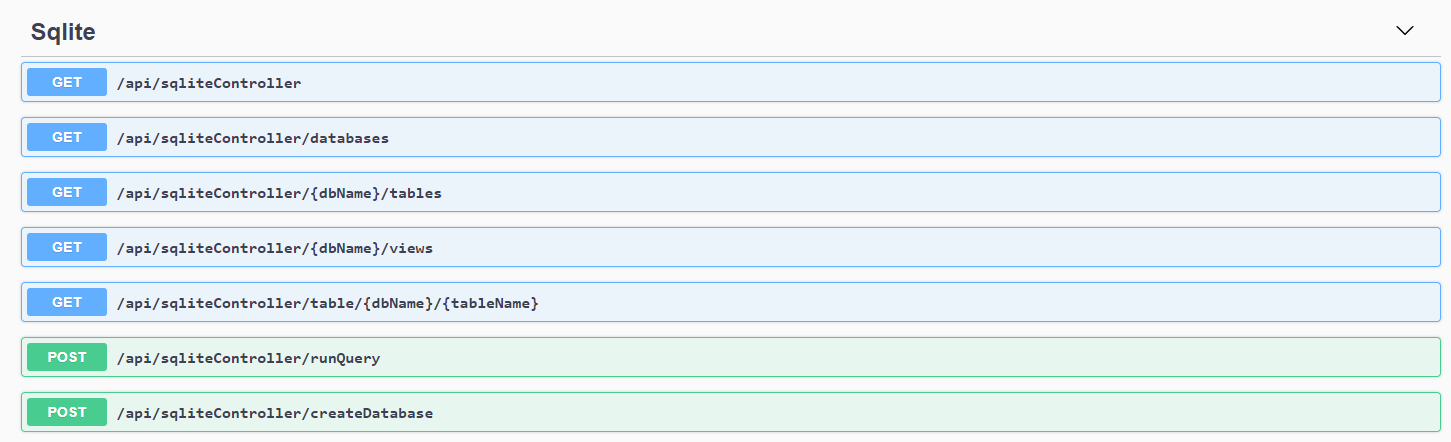
****

Рисунок 2.2 – Документація додатку, сгенерована за допомогою Swagger UI



## **Front-end Part**

**Front-end** – фронт-енд частина програмного забезпечення або веб-сайту - це все, з чим взаємодіє користувач. З точки зору користувача, фронт є синонімом інтерфейсу користувача. З точки зору розробника, це дизайн інтерфейсу та програмування, яке робить функції цього інтерфейсу. І навпаки, back-end включає функції та обробку даних, яка відбувається за кадром.

Однією з головних цілей розвитку фронтенду є створення плавного або «без тертя» UI. Іншими словами, фронт-енд програми або веб-сайту повинен бути інтуїтивно зрозумілим та простим у використанні. Хоча це звучить як проста мета, це може бути напрочуд складним, оскільки не всі користувачі чи пристрої однакові. Наприклад, додаток, розроблений для мобільного пристрою, вимагає значно іншого інтерфейсу, ніж настільний додаток. Веб-сайти повинні добре працювати на кількох пристроях та розмірах екрана, тому сучасні веб-розробки зазвичай передбачають адаптивний дизайн.

Приклади фронт-енд елементів інтерфейсу включають:

* додаток або макет сторінки
* графіка
* аудіо та відео елементи
* текстовий зміст
* елементи інтерфейсу користувача (кнопки, посилання, панелі інструментів, панелі навігації тощо)
* області введення (діалогові вікна, поля форм, текстові області тощо)
* користувальницький потік (як один інтерфейс веде до іншого)
* налаштування користувачів, теми та кастомізація

Для реалізації фронт-енд частини додатку, був обраний Angular.

**Angular** (зазвичай його називають "Angular 2+" або "Angular v2 і вище") - це структура веб-додатків з відкритим кодом на основі TypeScript, яку очолює команда Angular у Google та громада людей та корпорацій. Angular - це повне перезапис з тієї ж команди, яка побудувала AngularJS.



Рисунок 2.3 – Архітектура angular додатків.

Також варто відмітити технологію RxJS. Вона грає дуже важливу роль у Angular і моєму проекті.

**RxJS -** це бібліотека для складання асинхронних і event-based програм за допомогою спостережуваних послідовностей. Вона надає один основний тип, Observable, допоміжні типи (Observer, Schedulers, Subjects) та оператори, позаємствовані з Array#Extras (map, filter, reduce, every, etc), що дозволяє обробляти асинхронні події як колекції.

ReactiveX поєднує схему Observer із патерн Iterator та функціональне програмування з колекціями, щоб заповнити потребу в ідеальному способі управління послідовностями подій.

Основними поняттями RxJS, які вирішують управління асинхронними подіями, є:

* Observable: представляє ідею виклику колекції майбутніх значень чи подій.
* Observer: це сукупність зворотних викликів, яка вміє слухати значення, доставлені від Observable.
* Subscription: представляє виконання коду, коли приходять дані з Observable, в першу чергу корисна для скасування виконання.
* Operators: це чисті функції, які забезпечують функціональний стиль програмування колекцій з такими операціями, як map, filter, concat, Reduct тощо.
* Subject: є еквівалентом EventEmitter, і єдиним способом мультикастингу значення або події для кількох Observer.
* Schedulers: це централізовані диспетчери для контролю сумісності, що дозволяють нам координувати, коли обчислення відбуваються, наприклад, setTimeout або requestAnimationFrame або інші.



## **Архітектура додатку**

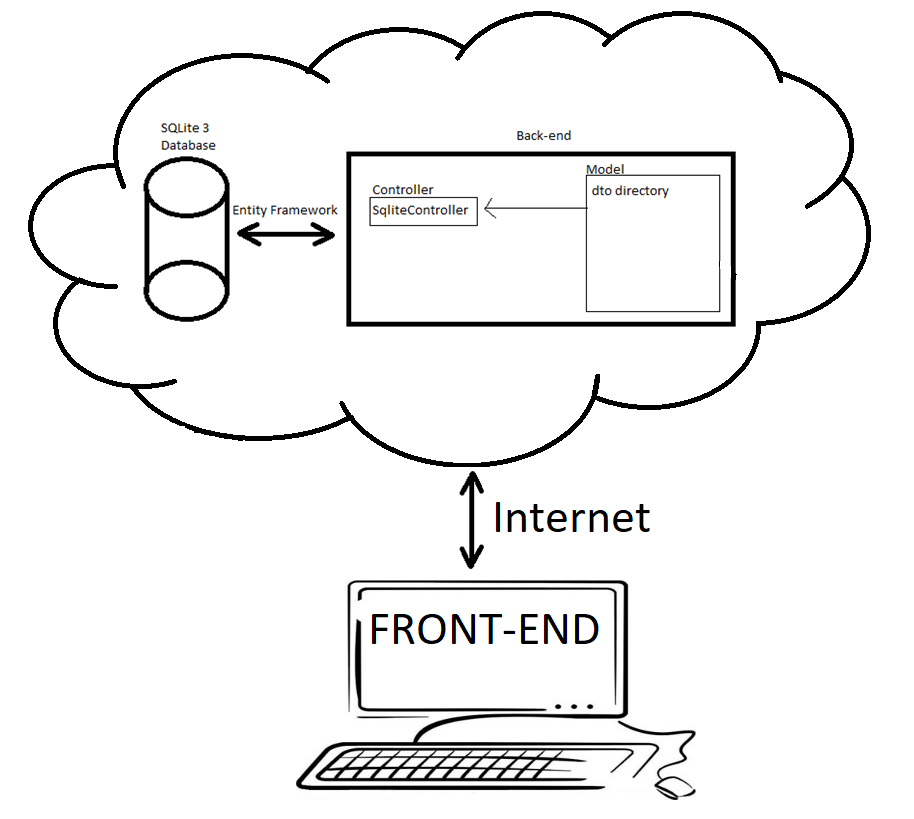


Рисунок 2.4 – Архітектура додатку

Фізично будуть знаходитись сервери. Сервери із даними, будуть поруч із сервером для Back-end. Це дозволить зберегти час при обробці запиту клієнта.

На серверах будуть директорії користувачів. В цих директоріях, будуть знаходитись бази даних, доступні для користувача.

На back-end, отримавши запит з клієнта, його почне обробляти SqliteController. У контролері SqliteConteroller знаходяться CRUD методи для роботи із БД, та методи для роботи із запитами до БД. У директорії dto зберігаються моделі, які формуються із запитів, та відправляються на front-end. В свою чергу back-end реалізує REST API для клієнта.

На front-end, реалізується View додатка. Ця частина додакта, буде знаходитись у браузері користувача. Користувач буде взаємодіяти з View та обробляти дані, отримані з back-end.

Back-end має лише один контролер SqliteConteroller. Цей контролер має наступне API:

* GET “api/sqliteController/databases” – метод для отримання всіх баз даних;
* GET “api/sqliteController/{dbName}/tables” – метод для отримання всіх таблиць, для заданої бази даних;
* GET “api/sqliteController/{dbName}/views” – метод для отримання всіх View, для заданої бази даних;
* GET “api/sqliteController/table/{dbName}/{tableName}” – метод для отримання даних таблиці;
* POST “api/sqliteController/runQuery” – метод котрий приймає запрос і виконує його, для заданої бази даних;
* POST “api/sqliteController/createDatabase” – метод для створення бази даних;
* POST “api/sqliteController/createTable” – метод для створення таблиці, в заданій базі даних;
* POST “api/sqliteController/createColumn” – метод для розширення таблиць;
* POST “api/sqliteController/insertIntoTable” – метод для додавання даних у таблиці;
* POST “api/sqliteController/updateTable” – метод для зміни даних у таблиці;

Front-end складається з багатьох компонентів.

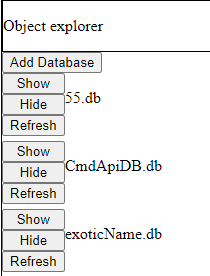


Рисунок 2.5 – Меню додатка

На рисунку можна побачити компонент db-menu-bar. Він складається з 3 кнопок, та назви бази даних. Кожен таки компонент представляє собою, якусь базу даних, та точку взаємодії із цією базою даних.

Кнопка Show – показує контент бази даних та 2 кнопок додавання таблиці, view та кнопки запиту до бази даних.

Кнопка Hide – скриває контент бази даних.

Кнопка Refresh – обновлює контент бази даних.

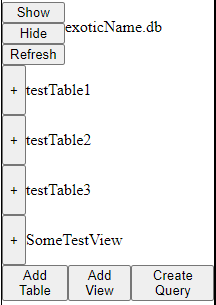


Рисунок 2.6 – Контент бази даних

На рисунку можна побачити table-menu-bar компонент. Він складається с кнопки відкриття таблиці, або view та назви таблиці, або view. Також можна побачити кнопки описані више у компоненті db-menu-bar.

## **Висновок**

Розвиненість хмарних технологій, а також їх популярність змушує розробляти новий проект саме на хмарній платформі. .Net Core платформа добре для цього підходить.

Back-end повинен бути також і крос-платформеним, що можливо на платформі .Net Core. Краще всього підійде класичне REST API, так як з браузера і мобільного додатку, можна слати однакові запити, та обробляти їх.

Front-end має бути простим у використані та не відтавати від трендів. Angular сьогодні, це топ 1 фреймворк. Кожні пів року, технологія отримує обновлення, а також має сильну і велику громаду. В Angular можна включити, велику кількість фіч розроблених іншими розробниками. Що відкриває широкий спектр можливостей, при розробці додатків.

# 3. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ПРОЕКТУ

Результати проведеного аналізу технології SQL і можливості її використання в додатках на хмарній платформі, а також аналіз кількості працівників, котрі працюють віддалено, дозволяють створити комерційну реалізацію сервісу керування БД. Для цього необхідно провести ряд організаційно-економічних заходів з метою оцінити економічну привабливість проекту. Для досягнення цієї мети були виділені наступні завдання:

* Опис сервісу керування БД
* Аналіз ринку збуту додатку
* Визначення витрат на запуск проекту, та його підтримку

## **Опис сервісу керування БД**

Розроблюваний сервіс призначений для керування БД і може бути використаний у різних областях. Головною відмінністю від існуючих аналогів, є реалізація СУБД у хмарі. Таким чином доступ до даних, які обробляються у СУБД, може бути незалежним від мережі і обладнання. Також, це дозволить компаніям сфокусуватись на даних та роботі з ними, замість організації власної структури, а так як власної структури не буде, це буде економити кошти для компанії.

## **Аналіз ринку збуту додатку**

Аналізований ринок збуту складається з різних організацій, котрі працюють з обробкою даних і їм потрібен сервіс в котрий можна віддалено зайти та працювати. А також організації, котрі хочуть перейти на хмарні технології, і шукають готові рішення.

Серед основних сегментів збуту сервісу можна виділити наступні:

* Організації, котрі працюють з даними, але не мають віддаленого доступу до них.
* Організації, котрі хочуть спробувати хмарні технології.
* Організації, котрі хочуть відмовитись від власної структури підтримки баз даних.
* Навчальні організації.

С першого сегменту можна назвати декілька видів організацій. Такі як мережі супермаркетів, страхові компанії, та інші. Частіше всього в таких організаціях не має віддаленого доступу до БД. На час пандемії, готове рішення із хмарною БД, буде дуже вигідно для них.

У другому сегменті знаходяться компанії, котрі знали про хмарні технології, але не наважувались їх використовувати. Через їх складність, або ціну. Розроблений додаток безкоштовний і має зрозумілий і дружній до користувача інтерфейс. Таке готове рішення, точно дасть знати компаніям чи потрібна їм, така технологія.

Третій сегмент, це компанії котрі хочуть скоротити штат. Скоріше за все, через фінансові причини. У розробленому додатку реалізовано весь необхідний функціонал, а також додаток - безкоштовний. Тому розроблена СУБД якнайкраще підійде компаніям.

До четвертого сегменту відносяться навчальні заклади, які мають курси по навчанню SQL. Простий і інтуїтивно-зрозумілий додаток, добре підійде для новачків. А доступ до спільних даних, дозволить в режимі онлайн проводити уроки, та показувати живі приклади.

## **Визначення витрат на запуск проекту, та його підтримку**

Точну ціну визначити неможливо, так як ринок не стоїть на місці. Але можна чітко визначити, які саме будуть витрати і звідки брати прибуток.

Серед початкових витрат, треба виділити наступні:

* Приміщення для серверних: серверні приміщення, для початку, краще орендувати. Так в місяць у дата центрі оренда стойки обійдеться 11800₴ у місяць, або оренда шкафа 19800₴;
* Сервери, та їх комплектуючі: середня ціна сервара 20000₴. Кількість користувачів буде зростати, тому для старта потрібно, біля, 10 серверів;

Серед постійних витрат будуть наступні:

* Реклама: для початку планується контекстна реклама, банерна реклама у google ads, реклама на youtube. Ціна такой реклами залежить від старту рекламної компанії, та кількості рекламних компаній. Середня ціна за кожний вид реклами 10000₴;
* Підтримка серверів (електроенергія, персонал, заміна обладнання): так як на початку життя проекту, серверні будуть орендуватись, то у витрати підтримки не буде входити плата за електроенергію. Потрібна буде команда розробників. А саме техлід(3000$), прожект менеджер(1000$), 2 мідли(1500$), девопс(1500$). А також команда сапорту, 2 людини(400$).;

Стратегії прибутку можуть бути різними:

* Першу версію безкоштовною, а наступні платними, або по підписці;
* Додати рекламу на сайт і можливо платну можливість, для того щоб її прибрати;
* Додати можливість пожертвування (як це зроблено у WinRar);

Для старту проекту необхідно приблизно 2 000 000₴. Проект можна монітезувати і це буде не самою складною задачею.

# ВИСНОВКИ

В ході виконання кваліфікаційної бакалаврської роботи, був розроблений веб-сервіс для роботи із базами даних на основі SQLite3.

Веб-сервіс включає в себе:

* Клієнтська частина веб-додатку для взаємодії з системою на Angular;
* Серверна частина для надання інформації на .Net Core3;
* Взаємодія серверної частини і баз даних за допомогою Entity Framework.

В результаті аналізу існуючих технологій SQL, була обрана найкраща, яка має менше всіх мінусів. В процесі розробки був створений клієнтський інтерфейс. Інтерфейс - простий і інтуїтивно зрозумілий. Серверна частина додатку, яка має весь запланований функціонал, для прототипу.

Розроблений сервіс володіє наступними перевагами:

* Створений на сучасних технологіях;
* Знаходиться у хмарі;
* Легко монетизується;
* Простий у використані;

У подальшому розвитку, проект буде доповнений кращою валідацією, аутентифікацією. Буде розроблений та імплементований графічний дизайн.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ttps://www.techwalla.com/articles/
2. <https://uk.wikipedia.org/>
3. <https://angular.io/>
4. <https://dotnet.microsoft.com/>
5. <https://rxjs-dev.firebaseapp.com/>
6. <https://www.sqlite.org/>
7. Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. SQL: полное руководство.
8. Jay A. Kreibich. Using SQLite.
9. Andrew Lock. ASP.NET Core in Action.

# ДОДАТОК А

namespace DiplomBackEnd.Controllers

{

[Route("api/sqliteController")]

[SwaggerTag("Sqlite3 Test Controller")]

public class SqliteController : Controller

{

private readonly IConfiguration \_configuration;

private SqliteConnectionStringBuilder \_connectionStringBuilder;

public SqliteController(IConfiguration configuration)

{

\_configuration = configuration;

\_connectionStringBuilder = new SqliteConnectionStringBuilder();

}

[HttpGet]

public ActionResult<IEnumerable<string>> test()

{

return new string[] { "test", "test" };

}

[HttpGet("databases")]

public ActionResult<IEnumerable<string>> getDatabases()

{

DirectoryInfo info = new DirectoryInfo(\_configuration["Path:SqliteDatabases"]);

FileInfo[] files = info.GetFiles();

List<string> responce = new List<string>();

foreach (FileInfo file in files) {

responce.Add(file.Name);

}

return Ok(responce);

}

[HttpGet("{dbName}/tables")]

public ActionResult<IEnumerable<string>> getTabels(string dbName)

{

\_connectionStringBuilder.DataSource = \_configuration["Path:SqliteDatabases"] + "\\" + dbName;

List<string> result = new List<string>();

using (var connection = new SqliteConnection(\_connectionStringBuilder.ConnectionString))

{

connection.Open();

var selectCommand = connection.CreateCommand();

selectCommand.CommandText = "SELECT name FROM sqlite\_master WHERE type = 'table' ORDER BY name;";

using (var reader = selectCommand.ExecuteReader())

{

while (reader.Read())

{

result.Add(reader.GetString(0));

}

}

}

return Ok(result);

}

[HttpGet("{dbName}/views")]

public ActionResult<IEnumerable<string>> getViews(string dbName)

{

\_connectionStringBuilder.DataSource = \_configuration["Path:SqliteDatabases"] + "\\" + dbName;

List<string> result = new List<string>();

using (var connection = new SqliteConnection(\_connectionStringBuilder.ConnectionString))

{

connection.Open();

var selectCommand = connection.CreateCommand();

selectCommand.CommandText = "SELECT name FROM sqlite\_master WHERE type = 'view' ORDER BY name;";

using (var reader = selectCommand.ExecuteReader())

{

while (reader.Read())

{

result.Add(reader.GetString(0));

}

}

}

return Ok(result);

}

[HttpGet("table/{dbName}/{tableName}")]

public ActionResult<string> getTable(string dbName, string tableName)

{

\_connectionStringBuilder.DataSource = \_configuration["Path:SqliteDatabases"] + "\\" + dbName;

GetTableObject result = new GetTableObject();

using (var connection = new SqliteConnection(\_connectionStringBuilder.ConnectionString))

{

connection.Open();

var getColumnNamesCmd = connection.CreateCommand();

getColumnNamesCmd.CommandText = $"PRAGMA table\_info({tableName});";

using (var reader = getColumnNamesCmd.ExecuteReader())

{

while (reader.Read())

{

result.ColumnNames.Add(reader.GetString(1));

}

}

var getTableCmd = connection.CreateCommand();

getTableCmd.CommandText = $"SELECT \* FROM {tableName};";

using (var reader = getTableCmd.ExecuteReader())

{

while (reader.Read())

{

List<string> row = new List<string>();

for (int i = 0; i < result.ColumnNames.Count; i++)

{

row.Add(reader.GetString(i));

}

result.TableContent.Add(row);

}

}

}

return Ok(result);

}

[HttpPost("runQuery")]

public ActionResult<string> runQuery([FromBody] QueryObject query)

{

\_connectionStringBuilder.DataSource = \_configuration["Path:SqliteDatabases"] + "\\" + query.DbName;

using (var connection = new SqliteConnection(\_connectionStringBuilder.ConnectionString))

{

connection.Open();

using (var transaction = connection.BeginTransaction())

{

var queryCmd = connection.CreateCommand();

queryCmd.CommandText = query.Query;

queryCmd.ExecuteNonQuery();

transaction.Commit();

}

}

return Ok(query);

}

// Create

[HttpPost("createDatabase")]

public ActionResult<CreateDbObject> createDatabase([FromBody] CreateDbObject createDb)

{

if (createDb != null && createDb.DatabaseName.Length > 0)

{

string dbFullPath = \_configuration["Path:SqliteDatabases"] + "\\" + createDb.DatabaseName + ".db";

FileInfo fi = new FileInfo(dbFullPath);

fi.Create();

return Ok(dbFullPath);

}

else

{

return BadRequest();

}

}

[HttpPost("createTable")]

public ActionResult<CreateTableObject> createTable([FromBody] CreateTableObject createTable)

{

\_connectionStringBuilder.DataSource = \_configuration["Path:SqliteDatabases"] + "\\" + createTable.DbName;

using (var connection = new SqliteConnection(\_connectionStringBuilder.ConnectionString))

{

connection.Open();

using (var transaction = connection.BeginTransaction())

{

var addTableCmd = connection.CreateCommand();

string rows = "";

foreach (Row row in createTable.Rows)

{

string tmpRow = $"{row.RowName} {row.Type}";

if (row.IsPrimaryKey)

{

tmpRow += " PRIMARY KEY";

}

if (row.IsNotNull)

{

tmpRow += " NOT NULL";

}

if (row.IsUnique)

{

tmpRow += " UNIQUE";

}

if (rows.Length == 0)

{

rows += tmpRow;

}

else

{

rows += $", {tmpRow}";

}

}

addTableCmd.CommandText = $"CREATE TABLE {createTable.TableName} ({rows});";

addTableCmd.ExecuteNonQuery();

transaction.Commit();

}

}

return Ok(createTable);

}

[HttpPost("createColumn")]

public ActionResult<CreateColumnObject> addColumn([FromBody] CreateColumnObject CreateColumnObject)

{

\_connectionStringBuilder.DataSource = \_configuration["Path:SqliteDatabases"] + "\\" + CreateColumnObject.DbName;

using (var connection = new SqliteConnection(\_connectionStringBuilder.ConnectionString))

{

connection.Open();

using (var transaction = connection.BeginTransaction())

{

var addColumnCmd = connection.CreateCommand();

addColumnCmd.CommandText = $"ALTER TABLE {CreateColumnObject.TableName} ADD COLUMN {CreateColumnObject.ColumnName} {CreateColumnObject.Type};";

addColumnCmd.ExecuteNonQuery();

transaction.Commit();

}

}

return Ok(CreateColumnObject);

}

//Insert

[HttpPost("insertIntoTable")]

public ActionResult<InsertIntoObject> insertIntoTable([FromBody] InsertIntoObject insertIntoObject)

{

\_connectionStringBuilder.DataSource = \_configuration["Path:SqliteDatabases"] + "\\" + insertIntoObject.DbName;

using (var connection = new SqliteConnection(\_connectionStringBuilder.ConnectionString))

{

connection.Open();

using (var transaction = connection.BeginTransaction())

{

var insertCmd = connection.CreateCommand();

string columns = String.Join(",", insertIntoObject.Columns);

var formatedValues = insertIntoObject.Values.Select(value => $"'{value}'");

string values = String.Join(",", formatedValues);

insertCmd.CommandText = $"INSERT INTO {insertIntoObject.TableName} ({columns}) VALUES ({values});";

insertCmd.ExecuteNonQuery();

transaction.Commit();

}

}

return Ok(insertIntoObject);

}

//Update

[HttpPost("updateTable")]

public ActionResult<UpdateTableObject> updateTable([FromBody] UpdateTableObject updateTableObject)

{

\_connectionStringBuilder.DataSource = \_configuration["Path:SqliteDatabases"] + "\\" + updateTableObject.DbName;

using (var connection = new SqliteConnection(\_connectionStringBuilder.ConnectionString))

{

connection.Open();

using (var transaction = connection.BeginTransaction())

{

var updateTableCmd = connection.CreateCommand();

string columns = "";

foreach (UpdateColumnObject updateColumn in updateTableObject.UpdatedColumns)

{

if (columns.Length == 0)

{

columns += $"{updateColumn.ColumnName} = '{updateColumn.ColumnValue}'";

}

else

{

columns += $", {updateColumn.ColumnName} = '{updateColumn.ColumnValue}'";

}

}

updateTableCmd.CommandText = $"UPDATE {updateTableObject.TableName} SET {columns};";

updateTableCmd.ExecuteNonQuery();

transaction.Commit();

}

}

return Ok(updateTableObject);

}

}

}

public class Startup

{

public IConfiguration Configuration { get; }

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddSwaggerGen

(c => c.SwaggerDoc(name: "v1", new OpenApiInfo { Title = "My Api", Version = "v1" }));

services.AddControllers();

services.AddCors();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

//app.UseCors(

// options => options.WithOrigins("http://localhost:4200", "http://localhost:4200/").AllowAnyMethod()

//);

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint(url: "/swagger/v1/swagger.json", name: "MY API V1"));

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseCors(builder =>

builder.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader()

);

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}