Київський національний торговельно-економічний університет

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка API для перетворення вмісту електронних таблиць в формат NoSQL бази даних»

Студента 4 курсу, 13 групи, спеціальності 122 «Комп'ютерні науки»	_____ *підпис студента*	Самчук Антон Валерійович
Науковий керівник кандидат фізико-математичних наук, доцент	_____ *підпис керівника*	Шклярський Сергій Михайлович
Гарант освітньої програми кандидат технічних наук, доцент	_____ *підпис керівника*	Демідов Павло Георгійович

Київ 2021

Київський національний торговельно-економічний університет

Факультет інформаційних технологій

Кафедра комп'ютерних наук та інформаційних систем

Спеціальність 122 «Комп'ютерні науки»

Затверджую

Зав. кафедри _____ Пурський О.І.

«20» грудня 2020р.

Завдання

на випускну кваліфікаційну роботу (проект) студентці

Самчук Антон Валерійович

(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи (проекту)

«Розробка API для перетворення вмісту електронних таблиць в формат NoSQL бази даних»

Затверджена наказом ректора від *«04» грудня 2020 р. № 4111*

2. Строк здачі студентом закінченої роботи *29 травня2021 року*

3. Цільова установка та вихідні дані до роботи

Мета роботи: розробка підходу та побудова API

Об'єкт дослідження: технологія перетворення процес моніторингу NoSQL

Предмет дослідження: засоби перетворення систем баз даних та їх управління

4. Перелік графічного матеріалу_____

_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Шклярський С.М.	15.12.2020 р.	15.12.2020 р.
2	Шклярський С.М.	15.12.2020 р.	15.12.2020 р.
3	Шклярський С.М.	15.12.2020 р.	15.12.2020 р.

6. Зміст випускної кваліфікаційної роботи (проекту) (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1.Обґрунтування систем управління баз даних

1.1. Введення в NoSQL. Функціонування та методи

1.2. Огляд етапів проектування системи управління баз даних

1.3. Аналіз варіантів розв'язання досліджуваної задачі.

РОЗДІЛ 2. Технологій NoSQL та API принцип їх роботи

2.1. Опис методів роботи NoSQL.

2.2. Технології та принципи роботи API

2.3. Розробка API за допомогою Flask

РОЗДІЛ 3. Опис API для завантаження таблиць у NoSQL БД

3.1. Побудова бази даних за допомогою MongoDB

3.2. Розробка API для перетворення електронної таблиці в формат бази даних NoSQL

3.3. Результат роботи розробленого API

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТОК

7. Календарний план виконання роботи

№ Пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної роботи*	*01.10.2020*	*01.10.2020*
2	*Розробка та затвердження завдання на випускну кваліфікаційну роботу*	*15.12.2020*	*15.12.2019*

3	*Вступ*	*03.02.2021*	
4	*РОЗДІЛ 1. Обґрунтування систем управління баз даних*	*26.02.2021*	
5	*РОЗДІЛ 2. Технологій NoSQL та API принцип їх роботи*	*06.04.2021*	
6	*РОЗДІЛ 3.* Опис API для завантаження таблиць у NoSQL бд	*12.05.2021*	
7	*Висновки*	*15.05.2021*	
8	*Здача випускної кваліфікаційної роботи на кафедру науковому керівнику*	*20.05.2021*	
9	*Попередній захист випускної кваліфікаційної роботи*	*26.05.2021*	
11	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*	*27.05.2021*	
12	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*	*31.05.2021*	
13	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання «15» грудня 2020 р.

9. Керівник випускної кваліфікаційної роботи (проекту)

Шклярський С.М.

(прізвище, ініціали, підпис)

10. Гарант освітньої програми Демідов П.Г.

(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент-дипломник

Самчук А.В.

(прізвище, ініціали, підпис)

12. Відгук керівника випускної кваліфікаційної роботи (проекту)

Випускна кваліфікаційна робота студента Самчука А.В. присвячена актуальній для сьогоденного інформаційного середовища темі розробки універсальних засобів перетворення різноманітних форматів передачі та зберігання даних. В якості об'єкта дослідження студентом було вибрано процес та технологію перетворення реляційних та пост-реляційних форматів даних: табличного та документного форматів.
Предметом дослідження були вибрані підходи, основані на парадигмі API, що в цілому дозволило реалізувати відповідний програмний інтерфейс та продемонструвало, що Самчук А.В. володіє відповідними фундаментальними знаннями та практичними навичками по вибраній для дослідження тематиці.
До недоліків роботи можна віднести недостатню проробку питань, що стосуються адаптації розробленого API до конкретної структури даних що веде до надмірних затрат по настроюванню системи.
В цілому робота виконана на достатньому професійному рівні, відповідає вимогам Вищої школи до ВКР освітнього рівня “бакалавр” та може бути рекомендована до захисту.

Керівник випускної кваліфікаційної роботи (проекту)

_____ 30.05.2021 р.

(підпис, дата)

13. Висновок про випускну кваліфікаційну роботу (проект)

Випускна кваліфікаційна робота (проект) студента _____

(прізвище, ініціали)

може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми_____Демідов П.Г._____

(підпис, прізвище, ініціали)

Завідувач кафедри _____Пурський О.І._____

(підпис, прізвище, ініціали)

«_____»_____2021 р.

Анотація

У випускній кваліфікаційній роботі розглянуто можливості реалізації NoSQL баз даних у розробці серверного додатку та використання внутрішнього процесу API для розширення функціоналу запису і збереження даних. Досліджено переваги у використанні перед SQL базами даних. Головні вимоги та критерії до досліджуваної предметної області: швидкість розгортання, економія ресурсів, складність розробки, швидкість виконання, складність у підтримці.

Ключові слова: NoSQL, SQL, API, серверний додаток.

Anotation

The final qualification work considers the possibilities of implementing NoSQL databases in the development of a server application and the use of the internal API process to expand the functionality of writing and storing data. The advantages in use over SQL databases are investigated. The main requirements and criteria for the researched subject area: speed of deployment, saving of resources, complexity of development, speed of execution, complexity in support.

Keywords: NoSQL, SQL, API, server application.

ЗМІСТ

ВСТУП

NoSQL була спроектована і створена після масового Впровадження реляційніх база Даних. Кожна СУБД реалізує одну з моделей баз даних для логічної структуризації використовуваних даних. Існує кілька таких моделей, серед яких найпопулярнішою є реляційна. Хоча вона і є великою потужною та гнучкою, є ситуації, рішення яких вона запропонувати не може. Тут на допомогу прийде порівняльна нова модель, що називається NoSQL. Вона набирає популярність і пропонує всі цікаві рішення та додатковий функціонал. Ці системи використовують метод структурування даних, вони пропонують більшу свободу дій при обробці інформації.

Різкий стрибок популярності NoSQL баз даних і пов'язані з ним історії використання нереляційних СУБД показали світу IT важливість реалістичної оцінки пріоритетів компанії. Деякі вендори успішно впровадили у себе NoSQL сховища і отримали помітне зниження збитків і підвищення якості додатків. Інші зазнали невдачі, пізно зрозумівши, що прийняте рішення їм не підходить. А треті просто залишилися зі своїми технологіями.

Багато NoSQL рішення спрямовані на закриття абсолютно конкретних проблем SQL сховищ - у першу чергу на посилення горизонтальної масштабованості. Багато нереляційних баз даних відмінно працюють, виконуючи мету свого створення, але при цьому вони вже не є тим універсальним продуктом, яким є SQL. У більшості компаній просто немає таких обсягів даних і інших специфічних умов роботи, в яких NoSQL рішення були б панацеєю або просто були б вигідні в якості основної бази даних. NoSQL сховища показують себе з дуже хорошого боку в симбіозі з реляційними базами даних. Наприклад, в системах, де основний обсяг інформації зберігає SQL, а за кеш відповідає NoSQL. Для захоплення більш істотних позицій усім нереляційним системам все ще не вистачає безлічі базових речей - універсальності, надійності, цілісності і передбачуваності.

Об'єктно-орієнтовані API дозволяють розробникам додатків без праці здійснювати запис і витяг структур даних. Завдяки використанню ключів секцій додатки можуть вести пошук по парам «ключ-значення», наборам стовпців або частково структурованим документів, що містять серійні об'єкти і атрибути додатків.

Усі NoSQL системи мають власний API для взаємодії або вбудовану мову запитів, часто є просто урізаною версією SQL. Це рішення має свої позитивні сторони:

- Простота роботи. Багато NoSQL рішення, в основному сховища виду "ключ-значення" мають в порівнянні з реляційними базами даних дуже сильно урізану функціональність.
- Простіший синтаксис запитів - менше помилок. Для спрощення роботи з базою даних деякими розробниками використовується ORM (Object-Relational Mapping) - технологія, що дозволяє автоматично транслювати операції з об'єктами в запити до бази даних. Найчастіше подібні рішення працюють неефективно і плодять безліч непотрібних або відверто помилкових запитів.

Недоліки у даного рішення так само присутні, притому в довгостроковій перспективі вони можуть серйозно переважити всі позитивні моменти:

- Додаток сильно прив'язується до конкретної СУБД. Мова SQL універсальна для всіх реляційних сховищ і користувачеві

не доведеться кардинально переписувати весь код у разі зміни СУБД. Навіть якщо дві NoSQL системи концептуально практично не відрізняються, вони будуть мати дуже мало загальних стандартів в API і в специфіці запитів.

- Обмежена ємність вбудованої мови запитів. SQL має дуже багату історію і безліч стандартів. Це дуже потужний і складний інструмент для операцій з даними і складання звітів. Практично всі мови запитів і методи API сховищ NoSQL були створені на основі тих чи інших функцій SQL - як наслідок, вони мають значно меншу функціональність.
- Низька цінність і вузькопрофільність знань - фахівців з хорошим знанням SQL набагато простіше знайти, в той час коли специфікою роботи API деяких NoSQL рішень на серйозному рівні мало хто захоплюється - це значить, що багато специфічні моменти оператору бази даних доведеться освоювати "на ходу".

Дипломна робота спрямована на досліджування нереляційних баз даних NoSQL, мета якої побудова надійного сховища (бази) інформації з гнучкою моделлю даних, що відрізняється від класичних реляційних СУБД. Ключову роль функціонування бази даних буде виконувати об'єктно-орієнтовані API для запису, збереження та читання необхідної інформації.

Мета і завдання дослідження полягає в створенні нереляційної БД для збереження інформації і даних за допомогою засобів та інструментів.

Об’єкт дослідження: процес перетворення вмісту електронних таблиць за допомогою технології API у формат нереляційної БД.

Предмет дослідження: комп’ютерне програмне забезпечення для побудови надійної та відмово стійкої бази даних, перетворення, зберігання, даних за допомогою API.

Методи дослідження: використання процесу API для запису та збереження даних у таблицях.

РОЗДІЛ 1.

ОГЛЯД СУЧАСНОГО СТАНУ ТЕХНОЛОГІЙ NoSQL БАЗ ДАНИХ

1.1. Введення в NoSQL. Функціонування та методи

NoSQL - це підхід до реалізації масштабування сховища (бази) інформації з гнучкою моделлю даних, що відрізняється від класичних реляційних СУБД. У нереляційних базах проблеми масштабованості (scalability) і доступності (availability), важливі для Big Data, вирішуються за рахунок атомарності (atomicity) і узгодженості даних (consistency). Спочатку слово NoSQL було акронімом з двох слів англійської мови: No («Не») і SQL (скорочення від англ. Structured Query Language - «структурована мова запитів»), що дає терміну сенс «заперечує SQL». Можливо, що перші, хто став вживати цей термін, хотіли сказати «No RDBMS» («Не реляційна СУБД») або «no relational» («Не реляційний»), але NoSQL звучало краще і в підсумку прижилося (в якості альтернативи пропонувалося також NonRel). Пізніше для NoSQL було придумано пояснення «Not Only SQL» («не тільки SQL»). NoSQL став загальним терміном для різних баз даних і сховищ, але він не позначає якусь одну конкретну технологію або продукт. Сама по собі ідея нереляційних баз даних не нова, а використання нереляційних сховищ почалося ще за часів перших комп'ютерів. Нереляційні бази даних процвітали за часів мейнфреймів, а пізніше, за часів домінування реляційних СУБД, знайшли застосування в спеціалізованих сховищах, наприклад, ієрархічних службах каталогів. Поява ж нереляційних СУБД нового покоління відбулося через необхідність створення паралельних розподілених систем для високо масштабованих інтернет-додатків, таких як пошукові системи[20].

NoSQL-бази оптимізовані для додатків, які повинні швидко, з низькою часовою затримкою (low latency) обробляти великий обсяг даних з різною структурою. Таким чином, нереляційні сховища безпосередньо орієнтовані на Big Data. Однак, ідея баз даних такого типу зародилася набагато раніше терміну «великі дані», ще в 80-і роки минулого століття, за часів перших комп'ютерів (мейнфреймів) і використовувалася для ієрархічних служб каталогів[11].

Взагалі термін NoSQL позначає «не тільки SQL» (Not Only SQL), характеризуючи відгалуження від традиційного підходу до проектування баз даних. Спочатку так називалася Open Source (опенсорсна) база даних, створена Карло Строззі, яка зберігала всі дані як ASCII-файли, а замість SQL-запитів доступу до даних використовувала скрипти командного рядка (CLI). На початку 2000-х років Google побудував свою пошукову систему і додатки (GMail, Maps, Earth і інші сервіси), вирішивши проблеми масштабованості і паралельної обробки великих обсягів даних. Так була створена розподілена файлова яка координує системи, а також має можливість створювати рядкові сховища (column family store), засноване на обчислювальної моделі MapReduce. Після того, як корпорація Google опублікувала опис цих технологій, вони стали дуже популярні у розробників відкритого програмного забезпечення. В результаті цього був створений Apache Hadoop і запущені основні пов'язані з ним проекти. Наприклад, в 2007 році інший IT-гігант, Amazon.com, опублікувавши статті про свою високо доступних бази

даних Amazon DynamoDB. Далі в цю гонку NoSQL- технологій для управління великими даними включилося безліч корпорацій: IBM, Facebook, Netflix, eBay, Hulu, Yahoo! та інші IT-компаній зі своїми пропрієтарними і відкритими рішеннями на рис. 1.1[20].

Рис. 1.1. Різновид програмного забезпечення націлене на розробку та функціонування БД NoSQL

У порівнянні з класичними SQL-базами, нереляційні СУБД мають наступні переваги[17]:

- лінійна масштабованість - додавання нових вузлів в кластер збільшує загальну продуктивність системи;
- гнучкість, що дозволяє оперувати напівструктуровані дані, реалізуючи, в. т.ч. повнотекстовий пошук по базі;
- можливість працювати з різними уявленнями інформації, в т.ч. без завдання схеми даних;
- висока доступність за рахунок реплікації даних і інших механізмів відмовостійкості, зокрема, шаринга - автоматичного розподілу даних за різними вузлів мережі, коли кожен сервер кластера відповідає тільки за певний набір інформації, обробляючи запити на його читання і запис. Це збільшує швидкість обробки даних і пропускну здатність додатки;
- продуктивність за рахунок оптимізації для конкретних видів моделей даних (документної, графовой, колонкової або «ключ-значення») і шаблонів доступу;
- широкі функціональні можливості - власні SQL-подібні мови запитів, RESTful-інтерфейси, API і складні типи даних, наприклад, map, list і struct, що дозволяють обробляти відразу безліч значень.

Зворотною стороною вищевказаних переваг є наступні недоліки:

- обмежена ємність вбудованих мов запиту. Наприклад, HBase надає всього 4 функції роботи з даними (Put, Get, Scan, Delete), в Cassandra відсутні операції Insert і Join, незважаючи на наявність SQL подібної мови запитів. Для вирішення цієї проблеми використовуються сторонні засоби трансляції класичних SQL-виразів в виконавчий код для конкретної нереляційних бази. Наприклад, Apache Phoenix для HBase або універсальний Drill.
- складності в підтримці всіх ACID-вимог до транзакцій (атомарность, консистентность, ізоляція, довговічність) через те, що NoSQL-СУБД замість CAP-моделі (узгодженість, доступність, стійкість до поділу) скоріше відповідають моделі BASE (базова доступність, гнучке стан і підсумкова узгодженість). Втім, деякі нереляційні СУБД намагаються обійти це обмеження за допомогою настроюваних рівнів узгодженості, це можливо побачити на прикладі Cassandra. Аналогічним чином Riak дозволяє налаштовувати необхідні характеристики доступності-узгодженості навіть для окремих запитів за рахунок завдання кількості вузлів, необхідних для підтвердження успішного завершення транзакції.
- сильна прив'язка додатка до конкретної СУБД через специфіку внутрішнього мови запитів і гнучкою моделі даних, орієнтованої на конкретний випадок;
- недолік фахівців з NoSQL-баз в порівнянні з реляційними аналогами.

Підводячи підсумок опису основних аспектів нереляційних СУБД, варто відзначити деяку некоректність запиту «NoSQL vs SQL» в зв'язку з різними архітектурними підходами і прикладними завданнями, на які орієнтовані ці IT-засоби. Традиційні SQL-бази відмінно справляються з обробкою строго типизированной інформації невеликої кількості. Наприклад, локальна ERP-система або хмарна CRM. Однак, в разі обробки великого обсягу напівструктурованих і неструктурованих даних, тобто Big Data, в розподіленої системі слід вибирати з безлічі NoSQL-сховищ, з огляду на специфіку самого завдання[19].

1.2. Огляд етапів проектування системи управління баз даних

Концептуальне проектування

Концептуальне (інфологічне) проектування - побудова семантичної моделі предметної області, тобто інформаційної моделі найбільш високого рівня абстракції. Така модель створюється без орієнтації на якусь конкретну СУБД і модель даних. Терміни «семантична модель», «концептуальна модель» і «інфологічна модель» є синонімами. Крім того, в цьому контексті рівноправно можуть використовуватися слова «модель бази даних» та «модель предметної області» (наприклад, «концептуальна модель бази даних» та «концептуальна модель предметної області»), оскільки така модель є як спосіб реальності, так і способом проектованої бази даних для цієї реальності.

Логічне (Даталогічна) проектування

Логічне (Даталогічна) проектування - створення схеми бази даних на основі конкретної моделі даних, наприклад, реляційної моделі даних. Для реляційної моделі даних даталогічна модель - набір схем відносин, зазвичай із зазначенням первинних ключів, а також «зв'язків» між відносинами, що представляють собою зовнішні ключі[22].

Фізичне проектування

Фізичне проектування - створення схеми бази даних для конкретної СУБД. Специфіка конкретної СУБД може включати в себе обмеження на іменування об'єктів бази даних, обмеження на підтримувані типи даних і т.п. Крім того, специфіка конкретної СУБД при фізичному проектуванні включає вибір рішень, пов'язаних з фізичним середовищем зберігання даних (вибір методів управління дисковою пам'яттю, поділ БД по файлах і пристроїв, методів доступу до даних), створення індексів і т.д.

Робота з SQL-запитами

Уже за назвою видно, що NoSQL не підтримує SQL-запити. Це означає, що у кожної такої бази своя методика роботи з даними і загального стандарту немає. Чи не вийде вивчити операції в Redis, який працює за принципом «ключ-значення» і швидко освоїти MongoDB, де все грунтується на документах.

Деякі NoSQL-бази намагаються підтримувати щось з SQL, але на практиці це працює погано[19].

Застосування

Звичайні SQL-бази відмінно підходять для типових задач, де надійність і передбачуваність важливіше швидкості. Наприклад, для записів про пацієнтів, переміщеннях товарів зі складу або шкільних оцінок.

Але якщо вам потрібна швидкість, масштаб і велика потужність - подивіться на NoSQL. Єдиний мінус - у кожної бази свої правила роботи з даними, тому швидко перейти від однієї до іншої не вийде.

1.3. Аналіз варіантів розв’язання досліджуваної задачі.

Що таке API?

API означає «інтерфейс прикладного програмування». Для розробників програмного забезпечення є безліч API-інтерфейсів, що дозволяють інтегрувати інформацію з інших додатків в власне.

API – це система інструментів і ресурсів в додатку, яка дозволяє розробникам створювати програмні продукти, які взаємодіють з іншими службами[13]. API – це те, що дозволяє двом окремим додаткам взаємодіяти один з одним. API дозволяють користувачам взаємодіяти з іншими службами або сайтами, не залишаючи ваш сайт. API важливі, тому що без них було б дуже складно зрозуміти синтаксис всіх додатків і сервісів, з якими взаємодіє ваш інструмент. API-інтерфейси пояснюють, як називається кожен об'єкт і які дії можна виконати з кожним з них.

Що означає інтерфейс прикладного програмування?

Наприклад: додаток для банкомату. Коли ви підходите до банкомату, ви очікуєте, що він дозволить вам отримати доступ до свого облікового запису і завершити транзакцію, наприклад, зняти готівку. Як і банкомат, програма надає функцію, але воно не робить це саме по собі – воно повинно спілкуватися як з користувачем, так і з «банком», до якого воно звертається. Додаток також обробляє входи і виходи. Веб-додаток, мобільне або серверний додаток схожі на комп'ютер, який вирішує конкретну проблему. Програмне забезпечення може бути клієнтським додатком, таким як сайт бронювання квитків, або серверним додатком, таким як серверне програмне забезпечення, яке направляє запити до бази даних.

Програмування: API дозволяють банкомату спілкуватися з вашим банком. Програмування – це інженерна частина програмного забезпечення додатки, яка переводить введення в висновок. Іншими словами, він переводить ваш запит на отримання готівки в базу даних банку, перевіряє, чи є на вашому рахунку достатньо грошей для зняття запитаної суми, банк видає дозвіл, потім банкомат повідомляє банку, скільки ви зняли, щоб банк міг оновити свій баланс.

Інтерфейс: призначений для користувача інтерфейс (UI), то, як ми взаємодіємо з додатком. У разі банкомату, це екран, клавіатура і слот для готівки. Ми вводимо наш пін-код, вказуємо, скільки готівки ми хотіли б зняти, а потім забираємо виплачені гроші. Інтерфейси – це те, як ми спілкуємося з машиною. З API це майже те ж саме, тільки ми замінюємо користувачів програмним забезпеченням[15].

Якщо веб-сайт або програма пропонує API, це означає, що їх розробники створили набір URL-адрес, які повертають чисті відповіді на дані.

Віддалені API

Дистанційні API призначені для взаємодії через мережу. Під «віддаленим» ми маємо на увазі, що ресурси, якими маніпулює API, знаходяться десь за межами комп'ютера, що виконує запит. Оскільки найбільш широко використовуваною мережею зв'язку є Інтернет, більшість API розроблено на основі веб-стандартів. Не всі віддалені API є веб-API, але справедливо припустити, що всі веб-API є дистанційними.

Веб-API

Веб-API зазвичай використовують HTTP для здійснення запиту і надають визначення структури відповідати на них. Ці відповідні повідомлення зазвичай приймають форму файлу XML або JSON. І XML, і JSON є кращими форматами, оскільки вони представляють дані

таким чином, щоб їх могли легко використовувати інші додатки. Веб-API діляться на дві основні категорії: віддалений виклик процедур (RPC) і Representational State Transfer (REST).

API віддаленого виклику процедур (RPC), як правило, представляють собою один URI (універсальний ідентифікатор ресурсу), який можна використовувати для виклику багатьох операцій через POST. Ці API включають передачу структурованого запиту, який включає ім'я операції, яку ви хочете викликати, і будь-які аргументи, які ви хочете передати операції. RPC в основному процедурний характер[10].

Representational State Transfer (REST) API – це не певний тип API, а архітектура, заснована на специфікації HTTP. REST використовує сильні сторони HTTP. Наприклад, REST використовує URI в якості унікальних ідентифікаторів для ресурсів, використовує робочі дієслова HTTP для ресурсів і дає клієнтам можливість зв'язуватися між ресурсами, щоб вказати, що ці ресурси пов'язані[4].

Що таке ключ API?

Ключі API використовуються для аутентифікації для користувацьких агентів, які взаємодіють або роблять запити до API. Ці ключі можуть бути відправлені в API з використанням рядка запиту, заголовка запиту або у вигляді файлу cookie.

Компанія, що управляє API, може використовувати ключі, щоб дозволити зареєстрованим користувачам тільки відправляти запити, відстежувати, хто робить запити, здійснювати контроль за використанням API і блокувати або обмежувати користувачів, які перевищують певні обмеження. Коли API побачить відправлений ключ, він підтвердить, що ви той, ким ви себе називаєте, і авторизує вас на виконання певної дії.

Що таке виклик API?

Виклик API – це коли додаток використовує API для зв'язку з сервером або іншим додатком. Це відбувається кожен раз, коли програма запускає зовнішній фрагмент коду, який не є частиною основної програми. Наприклад, якщо програма працює в Windows і їй потрібно відкрити вікно нового файлу, вона викличе API, щоб повідомити операційній системі, що їй потрібно відкрити вікно[6].

Приклади API

На початку розвитку API, типові програми повинні були використовувати тільки кілька API для забезпечення виконання усіх функцій. Тепер додаток може використовувати сотні API-інтерфейсів для доступу до даних або функцій інших служб.

API Facebook: Facebook фактично пропонує кілька API для різних цілей. Рекламний API Facebook дозволяє користувачам відстежувати і контролювати ефективність своїх рекламних кампаній. API Graph Facebook дозволяє іншим програмам отримувати доступ до функцій Facebook.

API Карт Google: якщо ви відвідуєте веб-сайт з вбудованою картою Google, це один з прикладів API Карт Google в дикій природі. Веб-сайт повинен взаємодіяти з картами Google, щоб отримувати інформацію і вставляти вміст на веб-сторінку.

API Dropbox: API Dropbox дозволяє користувачам читати / записувати файли, що зберігаються в Dropbox, дозволяє їм використовувати синхронізацію і зберігання файлів Dropbox і знаходити інші варіанти використання Dropbox [3].

Використання API для взаємодії з NoSQL

API (application programming interface, прикладний інтерфейс програмування) являє собою сукупність деяких методів взаємодії з базою даних. Як водиться, класифікацій багато, але ми пропонуємо зупинитися на двох основних – внутрішній процес та зовнішній процес API і SQL / NoSQL.

Якщо БД (частково) працює всередині клієнтського процесу, то її інтерфейс називається внутрішній процес (in-process API). У цьому випадку останній має бібліотеку функцій, які викликають методи безпосередньо в процесорі бази даних. Такий механізм знижує час очікування до мінімуму і збільшує пропускну здатність до максимуму (тобто по максимуму використовує надану пам'ять). Однак при внутрішній процес підході тільки один клієнтський додаток одноразово має доступ до БД, так що API втрачає перевагу в гнучкості. Крім того, підхід пов'язаний з додатковим ризиком: якщо клієнтську програму зависає або дає збій, то ж відбувається з усією БД - адже вони ділять один процес на двох[1].

Якщо ж БД працює в окремому процесі (в цьому випадку ми говоримо про зовнішній процес API (out-of-process API), то для доступу до неї зазвичай використовується протокол, заснований на TCP / IP. Багато СУБД (системи керування базами даних) - останнім часом і інші види БД - підтримують або ODBC (open database connectivity, відкритий механізм взаємодії з БД), або JDBC (java database connectivity, з'єднання з БД на Java). Це спрощує створення клієнтських додатків - існує більш ніж достатньо бібліотек, що використовують дві вищевказаних технології. використання мережевого протоколу дозволяє значно збільшити гнучкість бази даних (точніше, доступу до неї), але TCP програє в порівнянні з прямим доступом щодо показників використання пам'яті і часу очікування системи.

РОЗДІЛ 2.

ТЕХНОЛОГІЙ NOSQL ТА АРІ ПРИНЦИП ЇХ РОБОТИ

2.1. Опис методів роботи NoSQL

Реляційні бази даних працювали протягом багатьох років, але тепер з'явилася проблема, з якою вони більше не можуть впоратися.

Розвиток технології та мереж не стоїть на місті, з кожним роком обсяг генерування гігабайт в секунду зростає. Його зберігання і обробка являє собою серйозний інженерний виклик.

Як виявилося, реляційні бази даних погано пристосовані до роботи з дійсно великими обсягами даних. Вони спроектовані для роботи на одній машині, і якщо ви хотіли б обробляти більша кількість запитів, то єдиний варіант - купити комп'ютер з великою кількістю оперативної пам'яті і більш потужним процесором. На жаль, кількість запитів, які здатна обробити одна машина, обмежена, а для розподіленої роботи на кількох машинах нам потрібна інша технологія баз даних[12].

Існує два широко поширених методу використання декількох машин в разі реляційної бази даних: реплікація і шардінг, але цих методів може бути недостатньо, щоб впоратися з проблемою.

Реплікація читання - методика, при якій кожне оновлення бази даних поширюється на інші машини, які можуть обробляти тільки запити та читання на рис. В цьому випадку всі зміни виконуються одним сервером, званим провідним вузлом, в той час як інші сервера, звані репліками читання, лише підтримують копії даних. Користувач може читати з будь-якої з машин, але змінювати дані лише через провідний вузол. Це зручний і дуже популярний метод, але він дозволяє лише обробляти більше запитів на читання і ніяк не вирішує завдання обробки необхідних обсягів даних[11].

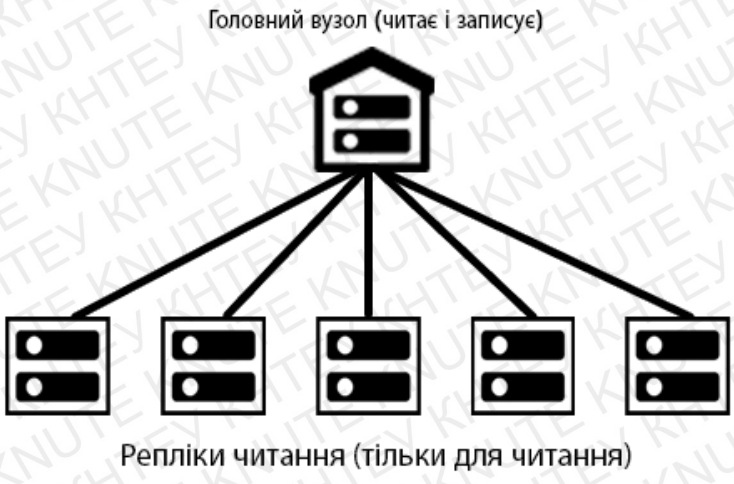

Рис. 2.1. Графічний схема методу Репліки читання

Шардінг - підхід, при якому використовується кілька примірників реляційної бази даних на рис. 2.2. Кожен з них обробляє операції запису і читання для частини даних. Якщо в базі даних зберігається, наприклад, інформація про покупців, з допомогою шардінга одна машина

може обробляти всі запити про покупців, чиї імена починаються на А, інша - зберігати всі дані про покупців, чиї імена починаються на В, і так далі.

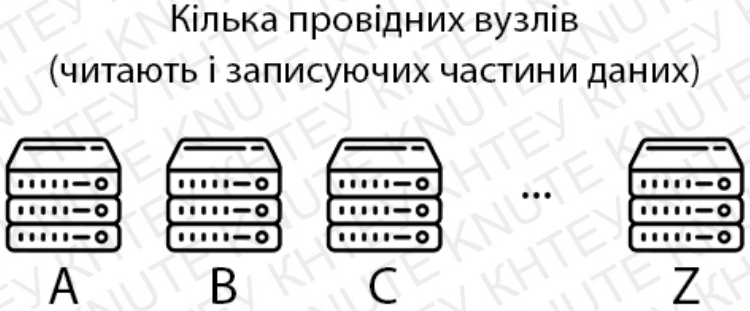

Рис. 2.2. Графічна схема методу Шардінг

Хоча шардінг дозволяє записувати більше даних, управління такою базою даних - справжній кошмар: доводиться вирівнювати дані по машинах і масштабувати кластер в обидві сторони по мірі необхідності. Хоча в теорії це виглядає просто, правильна реалізація - дуже складне завдання.

Чи можна вдосконалити реляційні бази даних?

Думаю, ми переконались, що реляційні бази даних не кращим чином пристосовані до генеруються в сучасному світі обсягами даних. Хоча, можливо, ви все ще дивуєтеся, чому ніхто досі не створив "поліпшеною" реляційної бази даних, яка могла б ефективно працювати на декількох машинах. Може здатися, що ця технологія просто ще не розроблена, і дуже скоро з'являться розподілені реляційні бази даних.

На жаль, цього не станеться. Це неможливо з точки зору математики, і нічого вдіяти з цим не можна.

Щоб зрозуміти, чому це так, необхідно звернутися до так званої теореми CAP (вона ж теорема Брюєра). Її довели в 1999 році, і вона стверджує, що у працююча на декількох машинах розподілена база даних може мати наступні трьома властивостями:

Узгодженістю (Consistency) - в результаті будь-якої операції читання повертаються результати останньої відповідної операції записи. Якщо система узгоджена, після запису нових даних, прочитати старі, вже перезаписані, неможливо.

Доступністю (Availability) - розподілена система в будь-який момент може обслужити вхідний запит і повернуться не помилковий відповідь.

Стійкістю до порушення зв'язності (Partition tolerance) - база даних продовжує відповідати на запити на читання і запис навіть в разі, коли частина її серверів тимчасово не здатні взаємодіяти один з одним. Цей тимчасовий збій називається порушенням зв'язності мережі і може викликатися безліччю фактором, починаючи від фізичних проблем з мережею через повільно працює сервера і закінчуючи фізичним ушкодженням мережевого обладнання.

Всі ці властивості, безумовно, зручні, і нам би дуже хотілося, щоб база даних поєднувала їх все. Жодна розсудлива розробник не захоче відмовитися від, скажімо, доступності, не отримавши нічого натомість. На жаль, теорема CAP також стверджує, що одночасне виконання всіх трьох властивостей – неможливо[17].

Усвідомити це може бути непросто, але можливо. По-перше, якщо нам потрібна розподілена база даних, вона повинна бути "стійкою до порушення зв'язності". Це навіть не обговорюється. Порушення зв'язності відбуваються весь час і наша база даних повинна працювати, незважаючи на це.

Тепер давайте зрозуміємо, чому ми не можемо домогтися одночасно узгодженості і доступності на рис. 2.4. Уявіть, що у нас є проста база даних, яка працює на двох машинах: A і B. Будь-який її користувач може виконувати запис на будь-яку з машин, після чого дані копіюються на іншу на рис. 2.3.

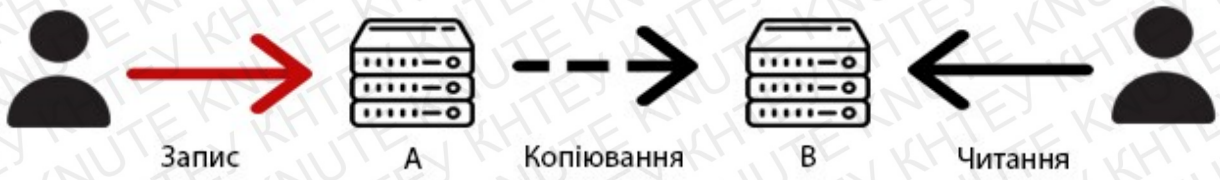

Рис. 2.3. Графічна схема відображення передачі даних від користувача до користувача

Тепер уявіть собі, що ці машини тимчасово втратили можливість обміну повідомленнями один з одним, і машина B не може відправляти дані на машину A або отримувати дані від неї. Якщо в цей проміжок часу машина B отримає запит на читання від клієнта, у неї є дві можливості:

Повернути свої локальні дані, навіть якщо вони не найсвіжіші. В цьому випадку віддається перевага доступності (повернути хоч якісь дані, навіть застарілі).

Повернути помилку. В цьому випадку віддається перевага узгодженості: клієнт не отримає застарілі дані, але і не отримає взагалі ніяких.

Рис. 2.4. Графічна схема відображення не дотримання узгодженості і доступності

Реляційні бази даних прагнуть втілити властивості "узгодженості" і "доступності" одночасно, і, отже, не можуть працювати в розподіленому середовищі. Спроба реалізувати всі можливості реляційної бази даних в розподіленої системі виявиться або нереалістичною, або просто нездійсненним.

З іншого боку, бази даних NoSQL, надають основне значення масштабованості та продуктивності. У них зазвичай відсутні такі "базові" можливості, як з'єднання і транзакції, а модель даних виявляється зовсім інший, можливо, в чомусь навіть обмежує. Все це дає можливість зберігання великих обсягів даних і обробки більшого числа запитів, ніж було можливо коли-небудь раніше[20].

Як баз даних NoSQL вдається поєднувати узгодженість і доступність?

Вам може здаватися, що вибравши NoSQL БД, ви завжди будете отримувати або якісь застарілі дані, або помилку в разі будь-якого збою. На практиці ж, доступність і узгодженість - аж ніяк не єдині можливі варіанти. Існує широкий спектр доступних для вашого вибору варіантів.

У реляційних базах даних цих параметрів немає, але NoSQL дозволяє вам керувати виконанням запитів подібним чином. Так чи інакше, вони дозволяють задавати два параметра при виконанні операцій запису або читання в NoSQL базі даних:

W - скільки машин в кластері має підтвердити збереження даних при виконанні операції запису. Чим більше число машин, куди ви запишете свої дані, тим легше буде прочитати найбільш свіжі дані при наступній операції читання, а й тим більше часу це займе.

R - з якої кількості машин ви хотіли б читати дані. У розподіленої системі, поширення даних по всім машинам кластера може зайняти деякий час, так що на деяких серверах дані будуть актуальними, а інші будуть відставати. Чим більше число машин, з яких читаються дані, тим вище шанси прочитати актуальні дані.

Розглянемо практичний приклад. Якщо у вашому кластері п'ять комп'ютерів, і ви вирішили записувати дані лише на один, а потім читати дані з одного випадково обраного комп'ютера, то з імовірністю 80% ви прочитаєте застарілі дані. З іншого боку, при цьому буде використовуватися мінімум ресурсів. Так що, якщо застарілі дані вас влаштовують, це не такий вже поганий варіант. В цьому випадку параметри W і R рівні 1.

З іншого боку, якщо ви записуєте дані на всі п'ять машин в базі даних NoSQL, то можете читати дані з будь-якої машини, і кожен раз гарантовано отримаєте актуальні дані. Виконання тієї ж операції при більшій кількості машин займе довше часу, але якщо актуальні дані для вас важливі, то ви можете використовувати цей варіант. В цьому випадку W = R = 5.

Яка мінімальна кількість операцій читання і запису, необхідне для узгодженості база даних?

Ось проста формула: R + W ≥ N + 1, де N - число машин в кластері. Це означає, що при п'яти серверах, можна вибрати або R = 2 і W = 4, або R = 3 і W = 3 або R = 4 і W = 2.

В цьому випадку не має значення, на які машини записуються дані, читання завжди буде проводитися, принаймні, з однієї машини з актуальними даними[11].

У інших баз даних, наприклад, DynamoDB, обмеження відрізняються, і вони дозволяють лише узгоджені операції записи. Кожен елемент даних зберігається на трьох серверах, і при запису будь-яких даних, він записується на дві машини з трьох. Але при читанні даних можна вибрати один з двох варіантів[20]:

1. Строго узгоджене читання, при якому дані читаються з двох машин з трьох і завжди повертаються останні записані дані.
2. Читання, узгоджене в кінцевому рахунку, при якому вибирається випадковим чином одна машина, з якої читаються дані. При цьому, однак, можуть тимчасово повертатися застарілі дані.

Спосіб організації даних

У SQL-базах все просто: є, умовно кажучи, таблиці, і є зв'язки між ними. Всі дані зберігаються в цих таблицях.

У NoSQL-базах все інакше - там може не бути таблиць, а замість них - свої моделі даних. Кожна з них підходить під свої завдання, універсальної немає[12].

Документно-орієнтовані бази даних

Ці бази даних надають можливість зберігання складних вкладених документів, в той час як більшість реляційних баз даних підтримує лише одномірні рядки. Така можливість може стати в нагоді в багатьох випадках, наприклад, при необхідності зберігання в системі інформації про користувача з декількома адресами. При використанні документно-орієнтованої бази даних, в подібному випадку можна просто зберігати складний об'єкт, що включає масив адрес, в той час як в реляційної базі даних вам довелося б створити дві таблиці: одну для інформації про користувача, а другу для адрес.

Документно-орієнтовані бази даних дозволяють скоротити розрив між об'єктною моделлю і моделлю даних. Деякі реляційні бази даних, такі як PostgreSQL, тепер теж підтримують документно-орієнтоване сховище, але в більшості реляційних баз даних ця можливість все ще відсутня.

Якщо ми зберігаємо дані в таблиці, у нас є стовпці і рядки. І якщо у нас про кого-то є дані, а про іншого немає, - десь в таблиці будуть пропуски. А якщо в таблиці немає потрібного стовпчика, а нам потрібно покласти в неї новий тип даних, нам доведеться створювати новий стовпець, і він для всіх буде порожнім:

Реляційна БД змушує нас заздалегідь придумувати, як буде працювати база даних; які там будуть поля; які допустимі типи даних. Наприклад, в таблицю вище вже не додаси інформацію про те, що Родіон носить бороду - точніше, додати щось її можна, але тоді у нас з'явиться купа порожніх клітинок. А якщо цих стовпців потрібно додавати багато? Це вкрай нераціонально.

Тепер уявіть, що є механізм, який дозволяє зберігати ці дані в більш вільному форматі. наприклад:

name: Міша

age: 35

city: Брянськ

job: Редактор

stickerpack: Доктор Хаус

<Name> Женя </ name> зараз проживає в <city> Москві </ city> і працює <job> директором </ job>, а у вільний час <hobby> сплавляється на байдарці </ hobby>

Родіон.City = Ульяновськ

Родіон.Boroda = true

Кожна з цих записів (про Мішу, Женю і Родіона) - це три окремих документа. І база даних настільки розумна, що може при необхідності розпізнати, що там де лежить. Якщо ми запитаємо у неї Boroda, то вона прошерстил всі документи і пошукає там розмітку зі словом «Борода». У перших двох документах цієї розмітки не буде, а в третьому - буде. Саме цей документ нам база даних і поверне.

Бази даних типу "ключ / значення"

Бази даних типу "ключ / значення" зазвичай реалізують найпростішу NoSQL-модель. По суті, вони надають вам розподілену хеш-таблицю, що дозволяє записувати дані по заданому ключу і читати їх назад з його допомогою[17].

У кожного запису є назва поля і його значення. наприклад:

name: 'Миша'

birthday: '9/09/1992'

city: 'Киев'

country: 'Украина'

position: 'Курьер'

Перша частина - це ключ, друга частина - значення. І можна підсипати скільки хочеш нових ключів.

Це корисно, наприклад, для словників або механізмів автозаміни: «Якщо зустрілося таке слово - заміни на ось таке».

Графові бази даних

Багато предметні області, наприклад, соціальні мережі або інформацію про фільми та акторів, можна представити у вигляді графів. Хоча граф можна представити і за допомогою реляційної бази даних, це складно і незручно. Якщо вам потрібні графові дані, краще скористатися спеціалізованою графовой базою даних, яка може зберігати інформацію про графа в розподіленому кластері і дає можливість ефективної реалізації алгоритмів на графах.

Якщо дані можна представити у вигляді графа або дерева, то вам підійде і база даних з таким же підходом до зберігання й пошуку.

Дерево - це коли дані зберігаються по системі «батько - нащадки». Є якийсь батьківський шматок даних, у нього є пов'язані з ним нащадки. У тих теж можуть бути свої нащадки і так далі. Кожна одиниця даних може бути чиїмось сином (але тільки когось одного) і мати скільки-то власних нащадків, побудовано на рис. 2.5.

В деревах зручно зберігати дані, наприклад, для пошукових алгоритмів. У «деревах» також зберігаються файли на вашому комп'ютері: є кореневий каталог, в ньому вкладені папки, в них ще папки, в них файли. Один і той же файл не може зберігатися одночасно в двох місцях.

Графи - це коли дані пов'язані взагалі як хочеш. Один шматок даних може бути пов'язаний з будь-якими іншими в будь-якій кількості і в будь-якому напрямку , побудовано на рис. 2.6. Дерево - окремий випадок графа[14].

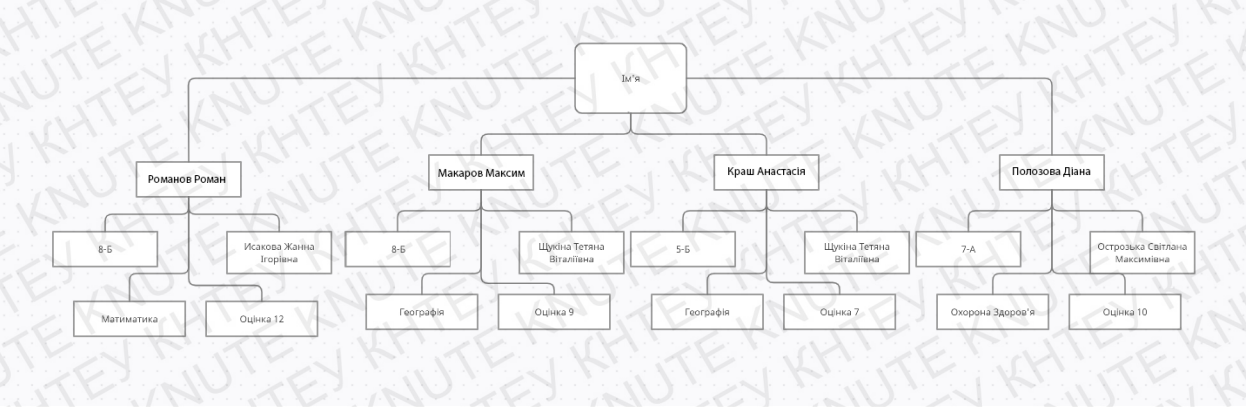

Рис. 2.5. Графічний вид даних у вигляді “Древо”

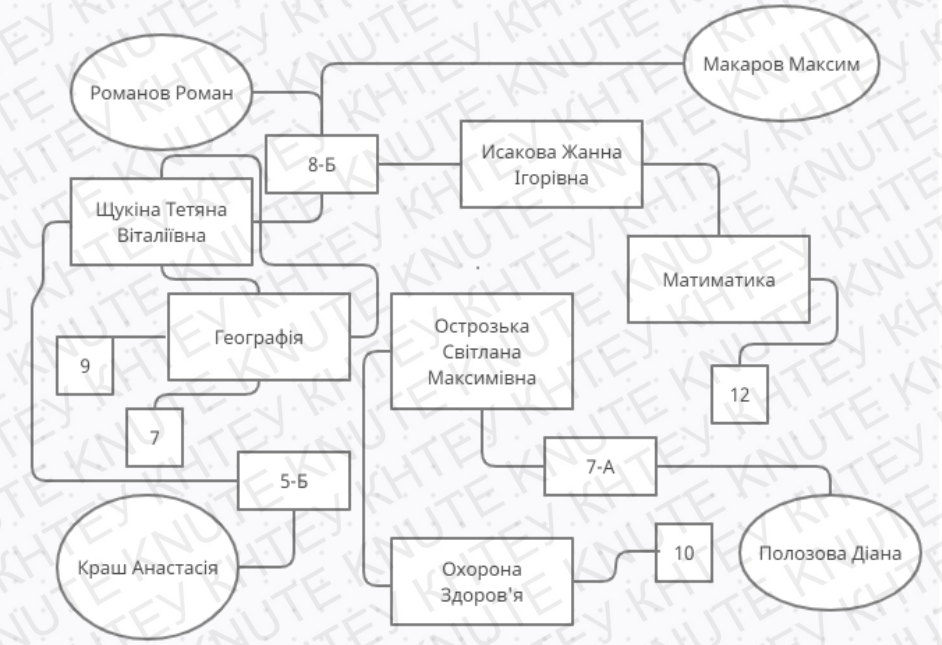

Рис. 2.6. Графічний вид даних у вигляді “Граф”

Столбцова бази даних

Основна відмінність між столбцової і іншими типами баз даних полягає в способі зберігання даних на диску. Реляційні бази даних створюють по файлу для кожної таблиці і зберігають значення для всіх рядків послідовно. Столбцові бази даних створюють по файлу для кожного стовпця ваших таблиць[19].

Така структура дозволяє агрегувати дані і виконувати певні запити ефективніше, однак необхідно переконатися, що дані відповідають обмеженням таких баз даних.

Можемо уявіти собі одну величезну таблицю, в якій зберігаються всі дані в базі. Відмінність від традиційної схеми в тому, що в SQL-базах робота йде з рядками, а тут - з колонками на табл 2.1.

Ім'я	Клас	Вчитель	Предмет	Оцінка
Романов Роман	8-Б	Исакова Жанна Ігорівна	Матиматика	12
Макаров Максим	8-Б	Щукіна Тетяна Віталіївна	Географія	9
Краш Анастасія	5-Б	Щукіна Тетяна Віталіївна	Географія	7
Полозова Діана	7-А	Острозька Світлана Максимівна	Охорона Здоров'я	10

Табл. 2.1. Приклад побудови таблиці в БД NoSQL

Яку базу даних вибрати?

Вибір бази даних - зазвичай болісна проблема, і при такій кількості можливих варіантів може здатися нездійсненним завданням. Гарна новина полягає в тому, що немає потреби вибирати тільки одну.

Замість створення єдиного монолітного додатки, що реалізує всі можливості і має доступ до всіх даних системи, можна скористатися ще одним сучасним патером під назвою "мікросервіси": розбити додаток на набір незалежних сервісів. Кожен сервіс вирішує свою вузьку задачу, і використовує тільки свою базу даних, яка максимально підходить для вирішення цього завдання.

2.2. Технології та принципи роботи API

Сучасні API часто приймають форму веб-сервісів, які надають користувачам (як людям, так і іншим веб-сервісам) якусь інформацію. Зазвичай ця процедура обміну інформацією і формат передачі даних структуровані, щоб обидві сторони знали, як взаємодіяти між собою.

Формати передачі даних у API

Існує безліч форматів даних, за допомогою яких користувачі взаємодіють з API. Найбільш відомі:

Формат передачі даних XML — жорстко ієрархічно структурований формат передачі даних. Інформація у ньому описується за допомогою простих тегів на рис. 2.7[9].

```xml
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Рис. 2.7. Візуальний вигляд даних XML

та JSON — нескладний, легкий, а значить і більш швидший формат, який виглядає таким чином на рис. 2.8:

```json
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

Рис. 2.8. Візуальний вигляд даних JSON

Параметри JSON

Наступні параметри можуть бути застосовані з format=json та format=jsonfm[16]:

- utf8: Закодовує більшість (але не всі) символи не-ASCII у UTF-8, замість заміни їх шістнадцятковими керівними послідовностями. Type: boolean.
- ascii: Закодовує всі символи не-ASCII як шістнадцяткові керівні послідовності. Type: boolean.
- formatversion: Форматування виводу. 1.25+
    - Сумісний із попередніми формат застосовує ключ * для вузлів вмісту та кодує не-ASCII символи як шістнадцяткові керівні послідовності.
    - Сучасний формат. Відповідь надається в зрозумілішому форматі, більшість не-ASCII символів закодовано в UTF-8. (рекомендовано)

- callback: Функція, до якої буде передано результат. З міркувань безпеки всі специфічні для користувача дані будуть обмежені. Ось що заборонено для підтримки безпеки:
    - Не можна отримати талони, тому дії, які змінюють стан, недоступні.
    - Клієнт розглядається як анонімний (неавторизований) користувач, навіть якщо він був авторизований через action=login. Це значить, що модулі, яким потрібні додаткові дозволи, працюватимуть лише з тими дозволами, які надано анонімному користувачу.

Методи HTTP запитів

HTTP визначає набір методів запиту, щоб вказати бажану дію, яке повинно бути виконано для даного ресурсу. Хоча вони також можуть бути іменниками, ці методи запиту іноді називають HTTP-дієсловами. Кожен з них реалізує різну семантику, але деякі загальні функції є загальними для групи з них: наприклад, метод запиту може бути безпечним, ідемпотентна або кешіруемим.

Зазвичай при зверненні до веб API використовуються запити HTTP. Тому потрібно хоча б коротко сказати про стандартні методи, які можуть міститися в HTTP запиті. Ці методи також називають HTTP дієсловами (Verbs)[9]:

- GET — мабуть, найпопулярніший тип запиту. Використовується для отримання або читання даних. Запити з використанням цього методу можуть тільки отримувати дані.
- HEAD — запрошує заголовки, ідентичні тим, що повертаються, якщо зазначений ресурс буде запитано за допомогою методу GET. Переклад заголовок тут говорить красномовно сам за себе
- PUT — зазвичай використовується для оновлення ресурсу, шляхом заміни даних запиту.
- POST — зазвичай використовується для створення нового ресурсу або використовується для відправки сутностей до певного ресурсу. Плюси методу POST очевидні: можна передавати необмежені обсяги інформації, причому, ніхто не побачить цю інформацію після того, як ви її відправили (мається на увазі, в рядку браузера). Якщо не помиляюся, таким чином можна передати пароль, реєстраційні дані з форм тощо.

- DELETE — видаляє дані.
- OPTIONS — використовується для опису параметрів з’єднання з ресурсом.
- PATCH — використовується для часткової зміни ресурсу.

Приклади:

Якщо ми хочемо отримати інформацію про ресурс, URL якого http://www.example.com/customers/12345, ми можемо послати запит:

GET http://www.example.com/customers/12345

Якщо ми хочемо оновити ресурс, ми можемо послати PUT-запит:

PUT http://www.example.com/customers/12345/orders/98765

Звичайні GET запити здатний посилати веб-браузер. Щоб мати змогу надсилати інші типи запитів можуть знадобитися скриптові мови програмування або спеціальні інструменти (про це поговоримо трохи згодом).

HTTP коди відповідей

Сервер може посилати різні коди у відповідь на запити користувачів. Це можуть бути коди помилок або просто коди, що інформують користувачів про стан сервера[10]. Наприклад:

Найбільш відомі коди — 2xx: OK Success (успішна відповідь від серверу), 4xx (проблеми на стороні клієнта) і 5xx (проблеми на стороні сервера). Про те, які коди повертати в тій чи іншій ситуації, вирішують розробники самих API.

REST API

REST API — це ідеологія побудови API, яка розшифровується як Representational State Transfer API. Вона грунтується на наступних принципах, сформульованих її творцем, Роєм Філдінгом[4]:

Клієнт-серверна архітектура

- Stateless сервер;
- Кешування;
- Багатошарова структура;
- Єдиний інтерфейс;
- Код на вимогу.

Аутентифікація по API

Зазвичай для використання API потрібен спеціальний ключ, за допомогою якого сервер дізнається про користувача. API ключ може бути відсутнім або надаватися за запитом (наприклад, після реєстрації на сайті) [5]. Наприклад, як у прикладі з погодним сайтом, ключ імпортував мені показники погоди. Ще один приклад, гра у яку ввести API ключ з Facebook здатна імпортувати список друзів.

Інструменти для роботи з API

Вище вже згадувалось: звичайні GET запити можна надсилати за допомогою браузера. Але спеціальні інструменти, які призначені для розробки і тестування API надають можливість не тільки відправляти різні типи запитів, але й зберігати запити, показувати результати в різних форматах, виступати в ролі proxy сервера. І багато багато-багато чого іншого.

Серед таких інструментів[18]:

- Postman — в тому числі існує в розширенні для Google Chrome, яке в безкоштовній версії дозволяє посилати запити, записувати їх, показувати історію. Доволі зручний і зрозумілий інструмент;
- jMeter — інструмент, який здобув популярність перш за все завдяки інструменту для Perfomance Testing (тестування навантажаження), яке можна проводити з його допомогою. Але це лише одна з безлічі його застосувань;
- Fiddler — дозволяє переглядати HTTP запити;
- SoapUI — потужний продукт для розробки і тестування веб додатків;
- Advanced REST Client — ще одне розширення для Chrome для роботи з API (конструкція запитів, їх показ теж в зручному вигляді).

Тестування API

А тепер — дуже коротко про те, як тестувати API. Звичайно, тут є своя специфіка. Тестування спрямоване на перевірку функціонування перш за все логіки додатку.

Типові помилки в API[21]:

- Збій обробки помилкових умов;
- Невикористані flag
- Відсутній або дублюється функціонал;
- Питання налаштування: труднощі при підключенні і отриманні відповіді від API;
- Проблеми з безпекою API;
- Питання по багатопоточності;
- Проблеми з продуктивністю: буває час відгуку API дуже високий;
- Помилки;

- Некоректна обробка валідних значень;
- Дані відповіді некоректно структуровані (JSON або XML).

При тестуванні API потрібно враховувати, що API створюються багато в чому для інтеграції з іншими сервісами. І працюють з ними не люди, а інші програмні системи. Тому потрібно оцінювати API з позиції зручності його використання разом з іншими продуктами, з позиції легкої інтеграції з ними. Кожен API повинен бути гнучким, також мати зрозумілу і детальну документацію.

При тестуванні API можливо використовувати загальноприйняті техніки тестування ПЗ[21]:

- Оглядове дослідницьке тестування — тести повинні виконати набір викликів, задекларованих в API, щоб перевірити загальну працездатність системи;
- Перевірка документації — перевіряється повнота описів функцій API, її зрозумілість і, в свою чергу, є фінальним результатом.
- Аналіз граничних значень — в API запитах в явному вигляді можуть передаватися значення параметрів. Це відмінний привід виділити кордону вхідних і вихідних значень і перевірити їх.
- Розбиття на класи еквівалентності — навіть у невеликого API є безліч варіантів використання і безліч комбінацій вхідних і вихідних змінних. Тому ми можемо зайвий раз використовувати наші навички виділення еквівалентних класів.
- Юзабіліті-тестування — перевіряє, чи є API функціональним і володіє зручним інтерфейсом, також перевіряється інтеграція з іншими;
- Тестування безпеки — перевіряє використовуваний тип аутентифікації і шифрування даних за допомогою HTTP;
- Автоматизоване тестування — створення скриптів, програм або настройка додатків, які зможуть тестувати API на регулярній основі.

1.3. Розробка API за допомогою Flask

Flask - це невеликий і легкий веб-фреймворк, написаний на мові Python, що пропонує корисні інструменти і функції для полегшення процесу створення веб-додатків з використанням Python. Він забезпечує гнучкість і є більш доступним фреймворком для нових розробників,

так як дозволяє створити веб-додаток швидко, використовуючи тільки один файл Python. Flask - це розширювана система, яка не зобов'язує використовувати конкретну структуру директорій і не вимагає складного шаблонного коду перед початком використання [2].

Flask використовує механізм шаблонів Jinja для динамічного створення HTML-сторінок з використанням знайомих понять в Python, таких як змінні, цикли, списки і т. д.

Попередні вимоги

Перед початком написання скрипта потрібно:

- Локальна середовище програмування Python 3. Установка і налаштування локальної середовища програмування для Python 3 для локального комп'ютера.
- Розуміння концепцій Python 3, таких як типи даних, умовні вирази, цикли for, функції та інші.

Установка Flask

На цьому етапі потрібно активувати середу Python і встановити Flask з допомогою установника пакетів pip.

Також потрібно активували середу програмування, переконайтеся, а тобто перебувати в директорії проекту, і за допомогою наступної команди активувати середу на рис. 2.9:

Рис. 2.9. Директорія зберігання API додатку

Для взаємодій з flask потрібно встановити пакети Python і ізолювати код проекту від основної системи Python. Це робиться за допомогою pip і python на рис. 2.10.

```
support tech@DESKTOP-GNC8P83 C:\Users\support tech\Desktop\API
$ pip install flask
Collecting flask
  Downloading Flask-2.0.1-py3-none-any.whl (94 kB)
     |████████████████████████████████| 94 kB 515 kB/s
Requirement already satisfied: itsdangerous>=2.0 in c:\users\support tech\appdata\local\programs\python\python39\lib
\site-packages (from flask) (2.0.0)
Requirement already satisfied: click>=7.1.2 in c:\users\support tech\appdata\local\programs\python\python39\lib\site
-packages (from flask) (8.0.0)
Requirement already satisfied: Jinja2>=3.0 in c:\users\support tech\appdata\local\programs\python\python39\lib\site-
packages (from flask) (3.0.0)
Requirement already satisfied: Werkzeug>=2.0 in c:\users\support tech\appdata\local\programs\python\python39\lib\sit
e-packages (from flask) (2.0.1)
Requirement already satisfied: colorama in c:\users\support tech\appdata\local\programs\python\python39\lib\site-pac
kages (from click>=7.1.2->flask) (0.4.4)
Requirement already satisfied: MarkupSafe>=2.0.0rc2 in c:\users\support tech\appdata\local\programs\python\python39\
lib\site-packages (from Jinja2>=3.0->flask) (2.0.0)
Installing collected packages: flask
Successfully installed flask-2.0.1
```

Рис. 2.10. Завантаження Flask до Python

Створення базового додатка

Тепер, коли налаштування прошло успішно, можна починати використовувати Flask. На цьому етапі створимо невеликий веб-додаток всередині файлу Python і запускаємо його для початку роботи сервера, який відобразить певну інформацію в браузері.

В директорії API потрібно відкрити файл з ім'ям hello.py для редагування. Для створення та редагування скрипта скористаємось текстовим редактором Sublime Text.

Sublime Text - пропріетарний текстовий редактор. Підтримує плагіни на мові програмування Python.

Файл hello.py буде служити мінімальним прикладом того, як обробляти запити HTTP. Усередині нього імпортується об'єкт Flask і створюються функція, яка повертає відповідь HTTP на рис. 2.11:

Рис. 2.11. Скрипт hello.py

У блоці коду необхідно попередньо імпортувати об'єкт Flask з пакета flask. Потім ви створюєте ваш екземпляр додатку Flask з ім'ям app. Ви передаєте спеціальну змінну __name__, яка містить ім'я поточного модуля Python. Вона вказує примірнику його розташування. Це необхідно, тому що Flask встановлює ряд шляхів за кадром.

Створивши екземпляр app, ви починаєте використовуєте його для обробки надходять веб-запитів і відправки відповідей користувачеві. @ app.route - це декоратор, який перетворює стандартну функцію Python в функцію перегляду Flask, конвертуються повертається значення функції у відповідь HTTP, який відображається клієнтом HTTP, наприклад веб-браузером. Ви передаєте значення '/' в @ app.route () для позначення того, що ця функція буде відповідати на веб-запити для URL /, який є основним URL-адресою.

Умова __name__ == "__main __" гарантує, що метод run () викликається тільки тоді, коли main .py запускається як основна програма. Метод run () не викликатися, якщо основний .py імпортується в інший модуль Python.

В процесі роботи з API можливо побачити усі виконані запити через API додаток, а тобто: застосований метод, код запиту, версія HTTP та код помилки або успішного запиту на рис. 2.12.

```
support tech@DESKTOP-GNC8P83 C:\Users\support tech\Desktop\API
$ python hello.py
 * Serving Flask app 'hello' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 508-480-820
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [28/May/2021 21:58:59] "GET /greeting HTTP/1.1" 200 -
```

Рис. 2.12. Список виконаних запитів

Функція перегляду hello () повертає рядок 'Hello, World!' як відповідь на рис. 2.13.

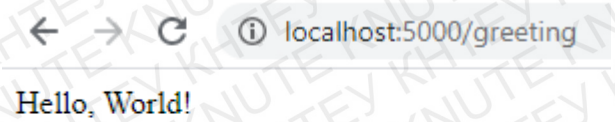

Рис. 2.13. Отримання результату запиту /greeting

РОЗДІЛ 3.

ОПИС API ДЛЯ ЗАВАНТАЖЕННЯ ТАБЛИЦЬ У NoSQL БД

3.1. Побудова бази даних за допомогою MongoDB

На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних, завдяки чому MongoDB працює швидше, має кращу масштабованість, її легше використовувати[7].

Але, навіть враховуючи всі недоліки традиційних баз даних і гідності MongoDB, важливо розуміти, що завдання бувають різні і методи їх вирішення бувають різні. В якійсь ситуації MongoDB дійсно поліпшить продуктивність програми, наприклад, якщо треба зберігати складні за структурою дані. В іншій же ситуації краще буде використовувати традиційні реляційні бази даних. Крім того, можна використовувати змішані підхід: зберігати один тип даних в MongoDB, а інший тип даних - в традиційних БД.

Вся система MongoDB може представляти не тільки одну базу даних, що знаходиться на одному фізичному сервері. Функціональність MongoDB дозволяє розташувати кілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність[8].

Налаштування та підготовка Бази Даних

Після установки треба створити на жорсткому диску каталог, в якому будуть знаходитися бази даних MongoDB на рис 3.1.

В ОС Windows за замовчуванням MongoDB зберігає бази даних по шляху C: \ data \ db.

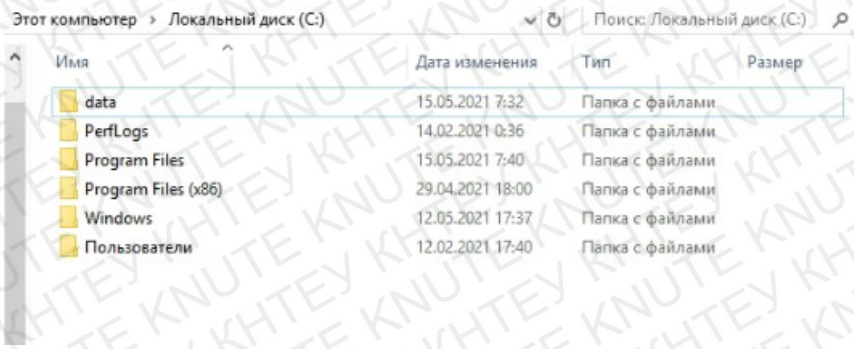

Рис. 3.1. Папка data для збереження баз даних

Якщо ж виникла необхідність використовувати якийсь інший шлях до файлів, то його можна передати при запуску MongoDB у прапорі --dbpath.

Щоб почати використовувати MongoDB, потрібно підключити mongo.exe оболонку до запущеного екземпляра MongoDB на рис. 3.2.

"C:\Program Files\MongoDB\Server\4.4\bin\mongod.exe" --dbpath="c:\data\db

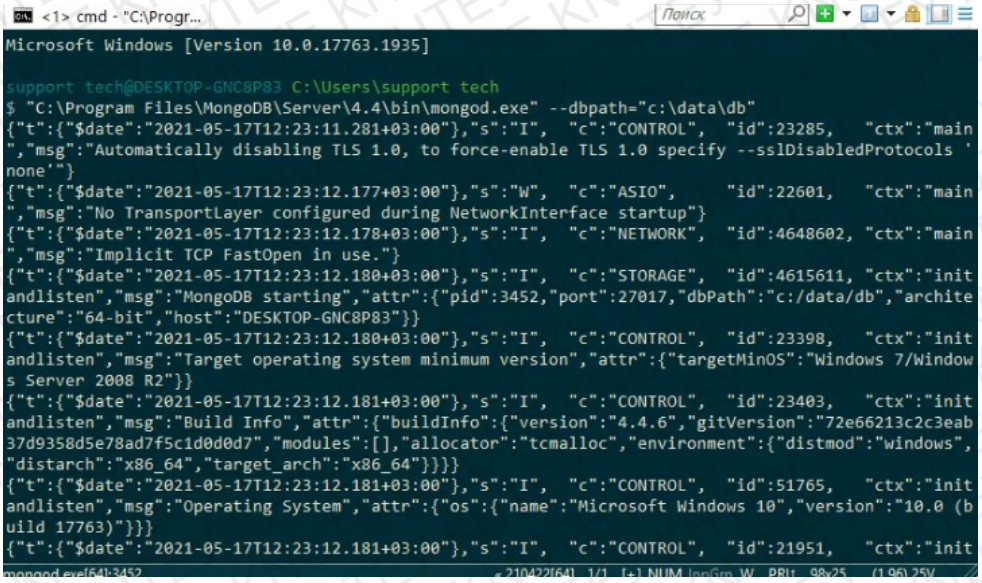

Рис. 3.2. Підключення папки data до MongoDB

Для запуску MongoDB потрібен exe на рис 3.3.

"C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe"

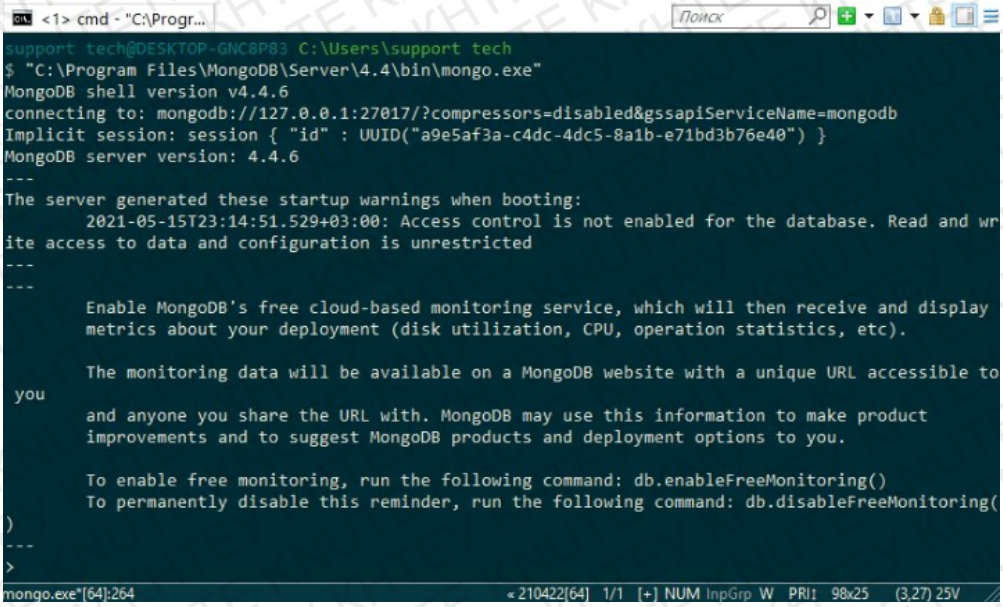

Рис. 3.3. Запуск MongoDB за допомогою командного рядка

Створимо базу даних за допомогою командного рядка який підключений до MongoDB за допомогою команди *use school* на рис 3.4:

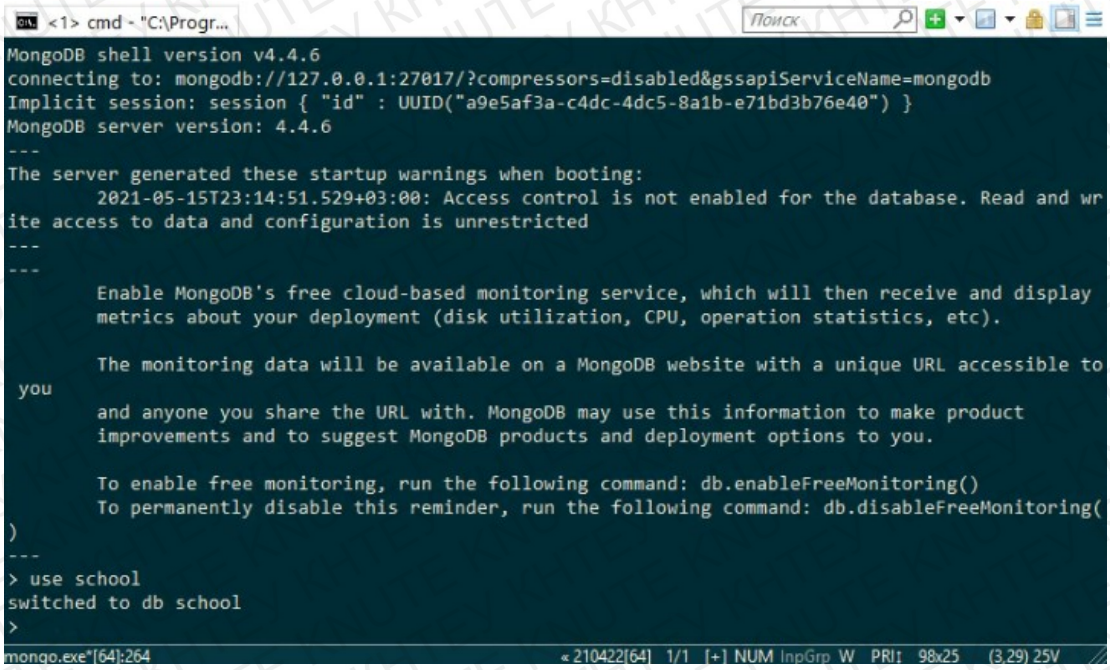

Рис. 3.4. Створення бази даних

Якщо в традиційному світі SQL є таблиці, то в світі MongoDB є колекції. І якщо в реляційних БД таблиці зберігають однотипні жорстко структуровані об'єкти, то в колекції можуть містити найрізноманітніші об'єкти, що мають різну структуру і різний набір властивостей.

Для створення колекції потрібно ввести команду на рис. 3.5:

db.createCollection(“classbook”)

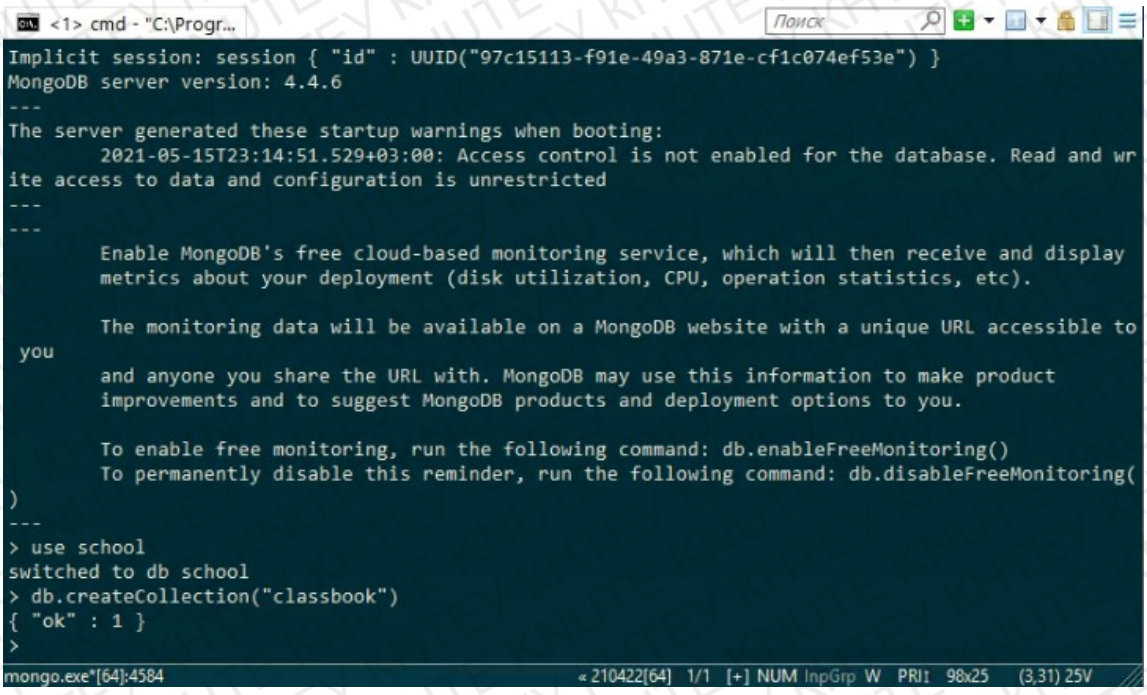

Рис. 3.5. Створення колекції

Графічний клієнт Compass

Для роботи з MongoDB можна використовувати офіційний графічний клієнт Compass на рис. 3.6.

При запуску програми нам треба ввести ряд даних, щоб підключитися до сервера.

- Hostname: хост сервера
- Port: порт, по якому запущений сервер
- SRV Record: специфікація, яка визначає розташування сервера (наприклад, адреса і порт).
- Authentication: тип застосовуваної аутентифікації. Здеь є такі опції:
  - None
  - Username / Password
  - X.509
  - Kerberos
  - LDAP
- Replica Set Name: назва репліки MongoDB, до якої відбувається підключення
- Read Preference: визначає, як Compass управляє операціями читання. Може приймати такі опції: Primary, Primary Preferred, Secondary, Secondary Preferred і Nearest.
- SSL: вказує, чи буде використовуватися захищене підключення
- SSH tunnel: чи слід підключатися до кластеру MongoDB через тунель SSH

Наприклад, підключимося до локального сервера. Для цього спочатку запустимо сам сервер MongoDB. У Compass залишимо все настройки підключення за замовчуванням (вони якраз націлені на локальний сервер, який запускається за адресою localhost: 27017) і для підключення натиснемо на кнопку Connect.

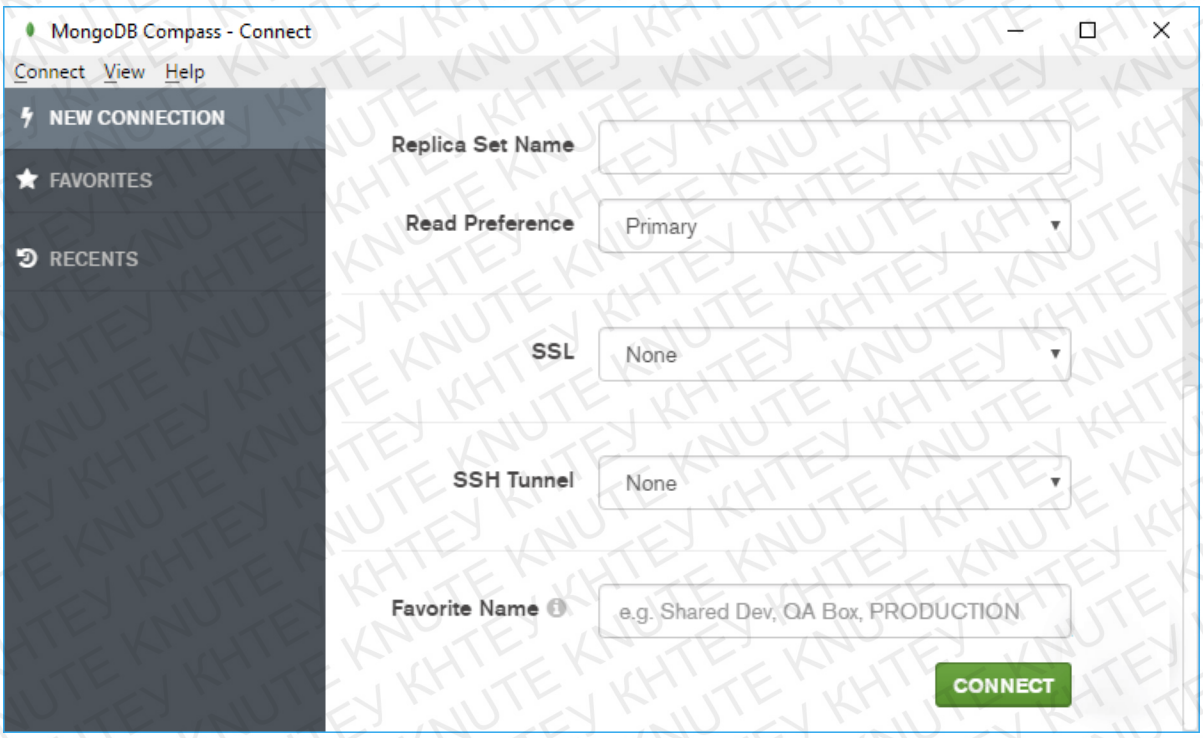

Рис. 3.6. Інтерфейс налаштування MongoDB

Після цього нам відкриється список баз даних, які є на сервері на рис. 3.7:

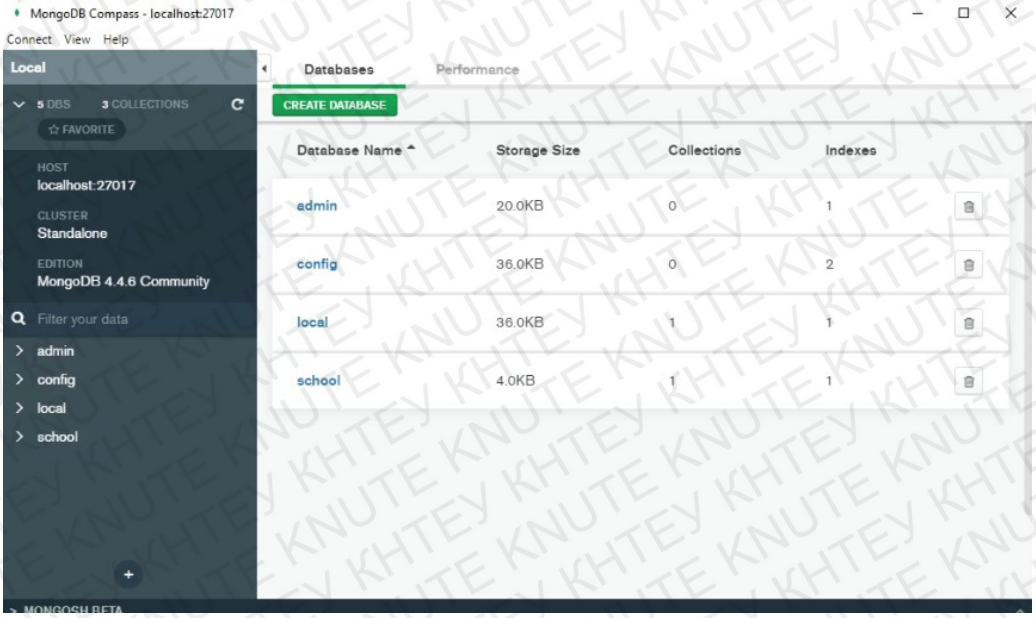

Рис. 3.7. Список створених баз даних

Ми можемо вибрати певну базу даних і отримати по ньому інформацію, зокрема, побачити набір колекцій в бд, скільки вони займають даних на рис. 3.8.

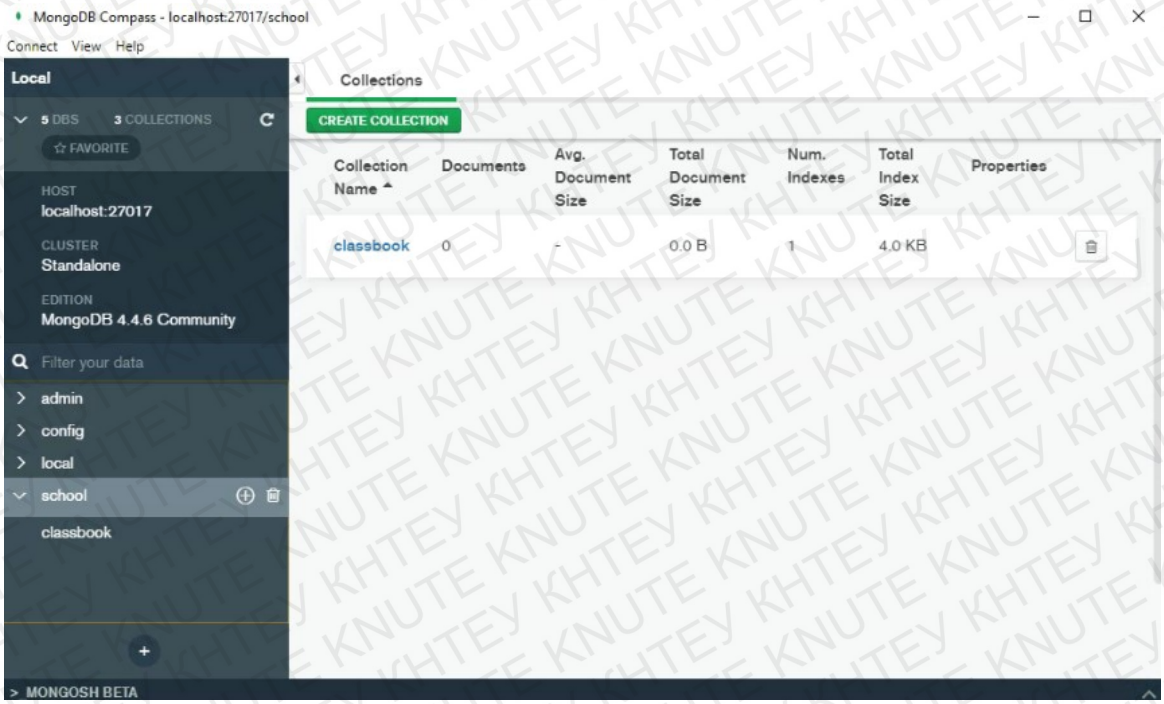

Рис. 3.8. Список набору колекції в базі даних

Натиснувши на певну колекцію, можна побачити графічно всі дані, які є в колекції на рис. 3.9:

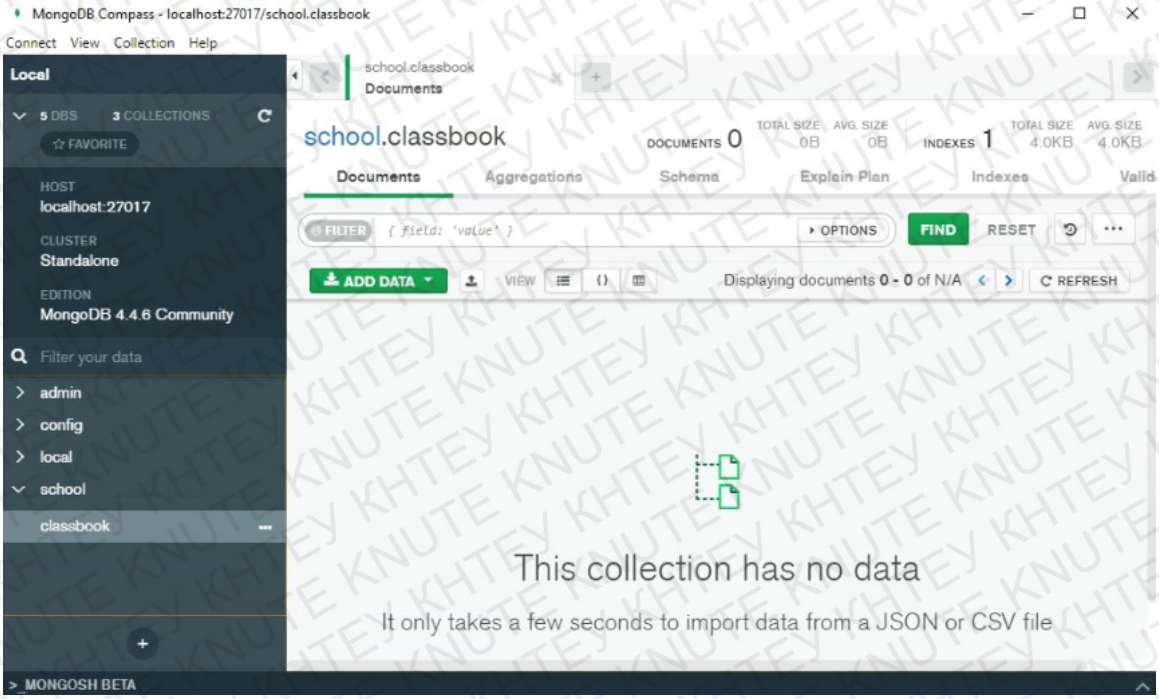

Рис. 3.9. Перегляд даних в колекції

Використовуючи графічний інтерфейс програми Compass, ми можемо управляти цими даними, додавати, змінювати, видаляти їх.

3.2. Розробка API для перетворення електронної таблиці в формат бази даних NoSQL

Розробка API

Flask - це веб-фреймворк для Python, що означає, що він надає функціональні можливості для створення веб-додатків, включаючи управління HTTP-запитами і шаблони рендеринга. У цьому розділі наведено розробка API на базі Python Flask. У наступному розділі буде наведена демонстрація функціонування готового додатку API .

MongoDB надає драйвери і інструменти для взаємодії зі сховищем даних MongoDB з використанням різних мов програмування, включаючи Python, JavaScript, Java, Go і C #.

PyMongo є офіційним драйвером MongoDB для Python, і ми будемо використовувати його для створення простого скрипта, який ми будемо використовувати для маніпулювання даними, що зберігаються в нашій базі даних school.

Використовуючи PyMongo, ми збираємося написати простий скрипт, який ми можемо виконати для виконання різних операцій з нашою базою даних MongoDB

Підключення бібліотек для розробки API

На базі підключених бібліотек буде будуватися наше API на рис. 3.10.

```
import json
from collections import OrderedDict
from itertools import islice
from openpyxl import load_workbook
import pymongo
from pymongo import MongoClient
from flask import Flask, render_template
from flask_restful import Api
from pymongo import MongoClient
from flask_pymongo import PyMongo
```

Рис. 3.10. Підключення бібліотек

Підключення Flask та бази даних MongoDB

```
app = Flask(__name__, template_folder="C:/Users/support tech/Desktop/API/template")
api = Api(app)

client = MongoClient('localhost', 27017)
db = client['school']
collection_base = db['classbook']
```

Рис. 3.11. Набір команд для створення API та його підключення до БД

Скрипт для перетворення вмісту електронних таблиць в формат NoSQL бази даних(JSON)

Python для створення файлів JSON з бази даних SQL. Але так само просто використовувати Python для створення JSON з електронних таблиць Excel. Ключовим компонентом є бібліотека Python openpyxl, яка пропонує набір функцій для читання і запису книг Excel .xlsx.

Скористаємося Python для генерації JSON.

Тепер претворим наш зразок електронної таблиці в JSON на рис. 3.12.

```
@app.route('/converting', methods=['POST'])
def converting():
    file_table = load_workbook('classbook.xlsx')
    sheet = file_table['book']
    table_list = []

    for row in islice(sheet.values, 1, sheet.max_row):
        cell = OrderedDict()
        cell['_id'] = row[0]
        cell['student_name'] = row[1]
        cell['class_student'] = row[2]
        cell['teacher'] = row[3]
        cell['subjects'] = row[4]
        cell['rating'] = row[5]
        table_list.append(cell)

    j = json.dumps(table_list, ensure_ascii=False)

    with open('classbook.json', 'w', ) as fl:
        fl.write(j)
    return 'THE XLSX SPREADSHEET HAS BEEN CONVERTED TO THE JSON FILE FORMAT'
```

Рис. 3.12. Скрипт перетворення

Скрипт для завантаження та оновлення даних

Тепер коли дані електронної таблиці були перетворені в формат NoSQL потрібно їх завантажити до БД, внаслідок запиту скрита API підключається за допомогою локального хоста до MongoDB та дає можливість завантажувати дані до MongoDB або оновлювати дані, якщо вони були змінені на рис. 3.14.

```
@app.route('/update_data', methods=['POST'])
def update_data():
    client.drop_database(db)

    with open('classbook.json') as f:
        file_data = json.load(f)
        collection_base.insert_many(file_data)
```

Рис. 3.14. Скрипт для завантаження даних у MongoDB

Скрипт відображення даних MongoDB на веб-сторінці Flask

Після перетворення файлу формату xlxs в формат JSON та завантаження таблиці у MongoDB експортуємо дані на веб-стронку для перегляду за допомогою скрипта який перебирає усі дані за допомогою масива та завантажує на сторінку у вигляді JSON файлу на рис. 3.15.

```
@app.route("/view", methods = ['GET'])
def view():
    data = collection_base.find()

    result = []
    for i in data:
        result.append([i['_id'],i['student_name'],i['class_student'],i['teacher'],i['subjects'],i['rating']])

    return render_template('base.html', name=result)
```

Рис. 3.15. Скрипт для перегляду даних на веб-сторінці

Скрипт для пошуку документа по імені у MongoDB

Дані були завантажені та відображенні на веб-сторінці, але якщо ми хочемо переглянути документ певного учня? Для цього є скрипт який був створений для пошуку учня по його ПІБ на рис. 3. 16, за допомогою запита можливо переглянути усі дані учня, а тобто: клас, предмет, вчитель, оцінка.

```
@app.route("/find_name/<s_name>", methods = ['GET'])
def find_name(s_name):
    search = collection_base.find({'student_name': s_name})

    for document in search:
        return render_template('base.html',name=document)
```

Рис. 3.16. Скрипт для пошуку учня по його ПІБ

3.3 Результат роботи розробленого API

Перевірка скрипта на зміну формату документа

У нас є репозиторії для збереження створеного формату, він зберігається в директорії де зберігається розроблений API під назвою - “app.py” на рис. 3.17.

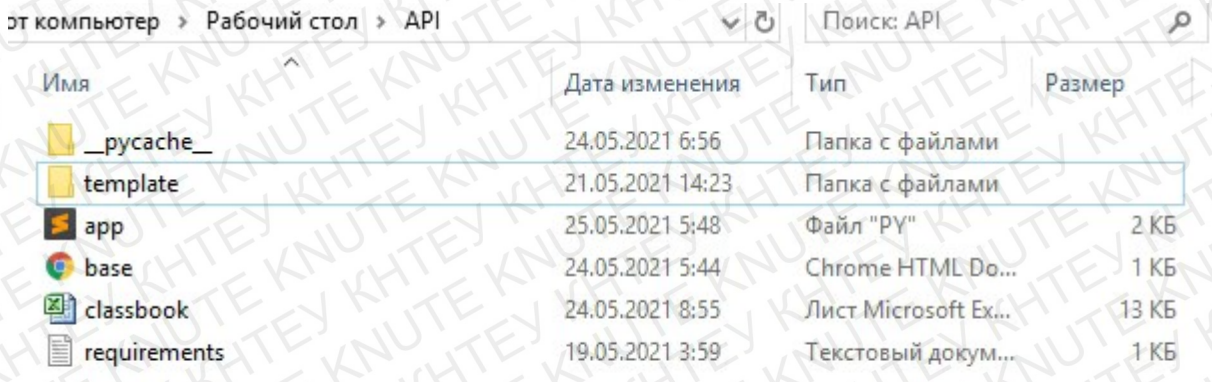

Рис. 3.17. Папка для збереження API

Після запиту методу: “/converting”, у папці де зберігається API було створено за допомогою скрипта файл з розширенням JSON. Тепер API здатний перетворити вміст електронних таблиць в формат NoSQL бази даних на рис. 3.18.

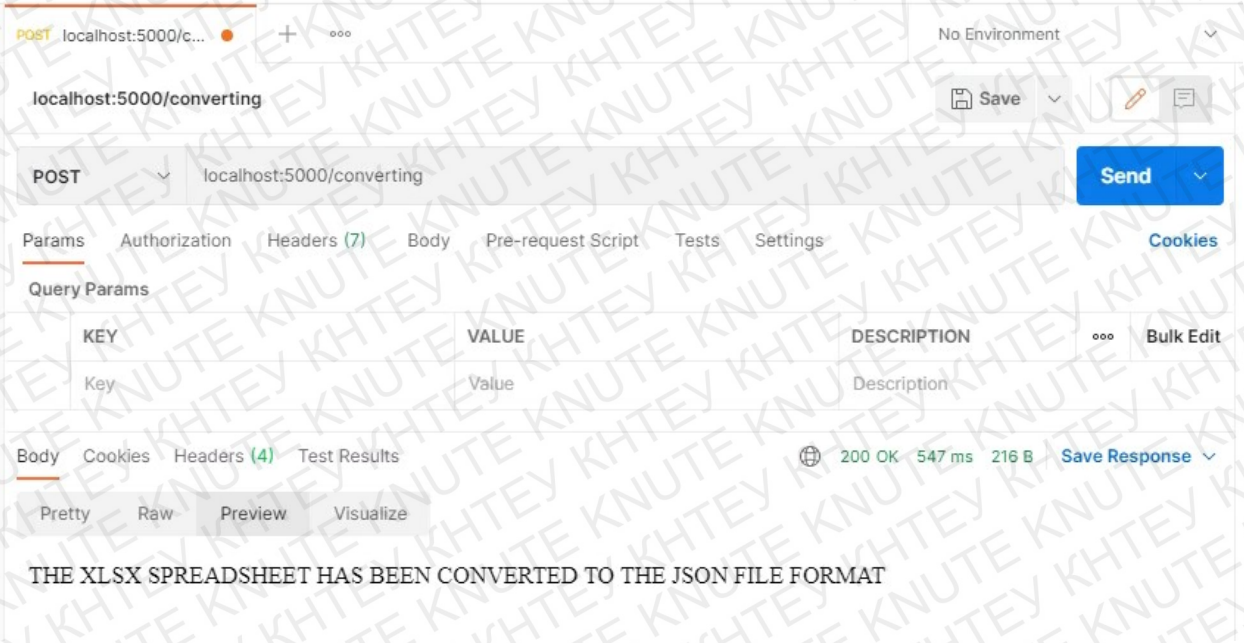

Рис. 3.18. Тестування запиту: “/converting” та отримання повідомлення про успішність виконання

Після виконання запиту: “/converting” за допомогою скрипта надходить повідомлення: “THE XLSX SPREADSHEET HAS BEEN CONVERTED TO THE JSON FILE FORMAT ”, воно повідомляє про успішність та зберігає файл у директорії де зберігається API на рис. 3.19.

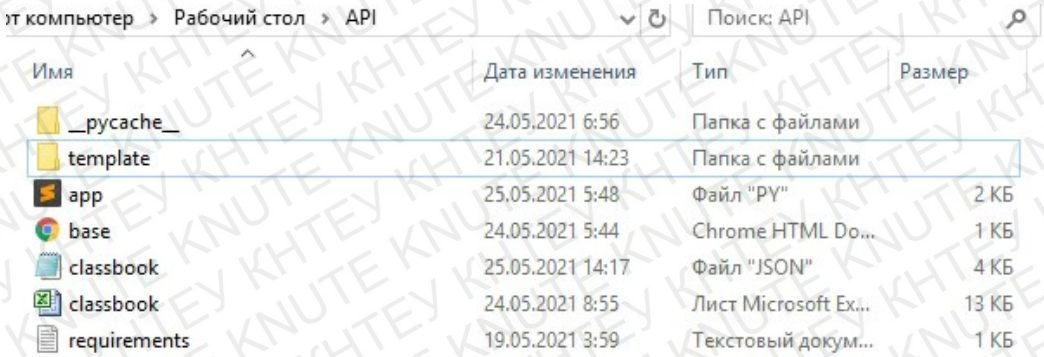

Рис. 3.19. Місце збереження файлу після виконання запиту

Тестування скрипта завантаження файлу у MongoDB

Скрипт для завантаження файлу у MongoDB та оновлення даних у разі їх змінення можливо при виклику запиту: “/update_data”. Запит очищує базу даних та завантажує нові змінені дані про успішність виклику API нам повідомляє про успішність повідомленням: “DATABASE HAS BEEN UPDATED AND READY FOR WORK” на рис. 3.20.

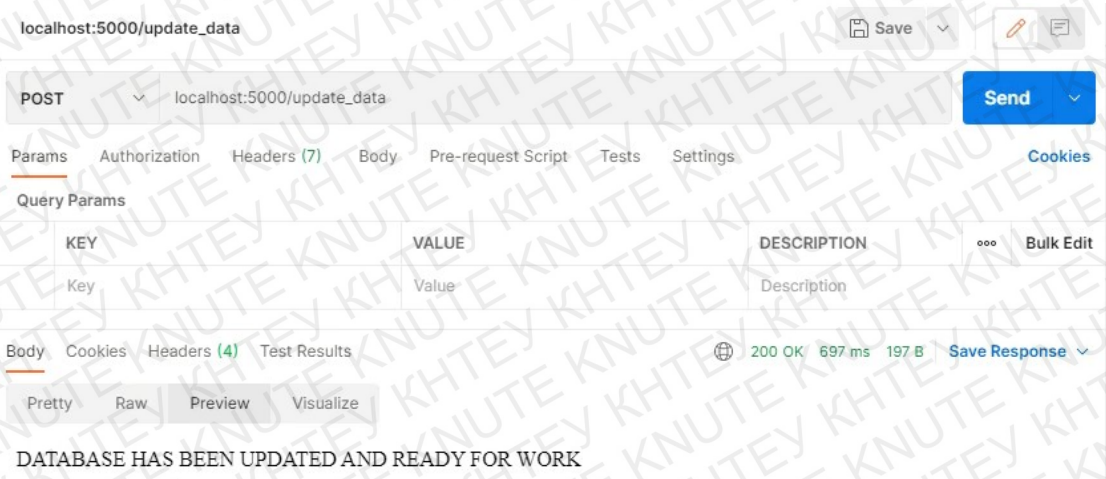

Рис. 3.20. Тестування запиту: "/update_data" для завантаження даних до БД

Після виклику запиту скрипт вилучає стару версію даних за допомогою команди *drop_database* та завантажує нову підключаючись по локальному хосту. Вигляд БД після завантаження на рис. 2.21.

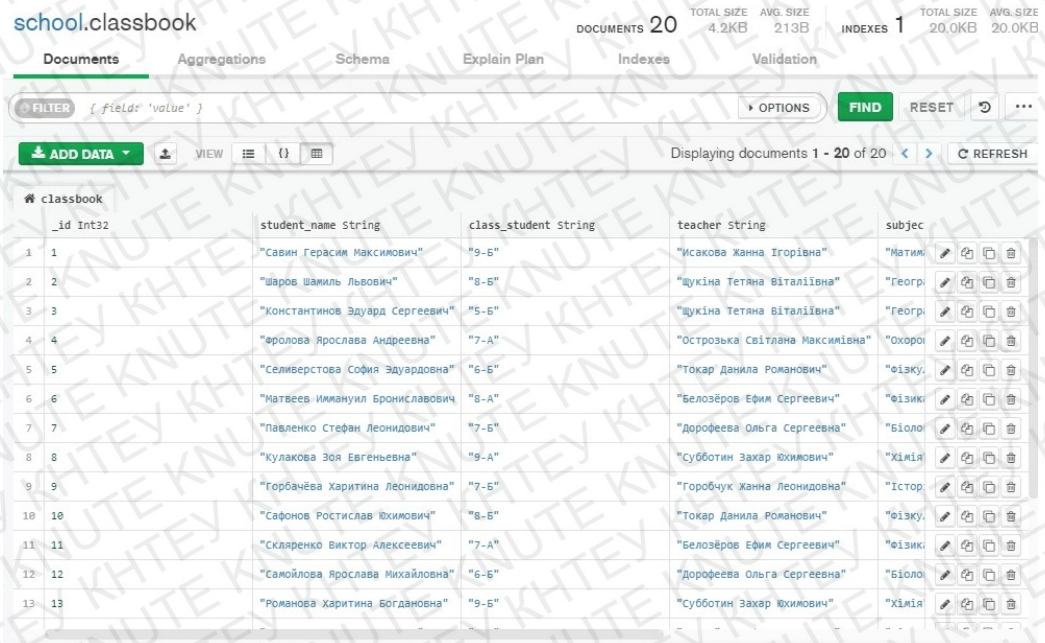

Рис. 3.21. Завантаження даних за допомогою перетвореного файлу JSON

Тестування скрипта для відображення даних MongoDB у веб-сторінку

Для перегляду даних не за допомогою електроного файлу або MongoDB можливо переглянути також за допомогою запиту URL: " /view " на рис. 3.22. Для цього нам потрібно щоб дані були вже завантажені у БД та були готові до роботи. Дані будуть відображені у вигляді JSON файлу.

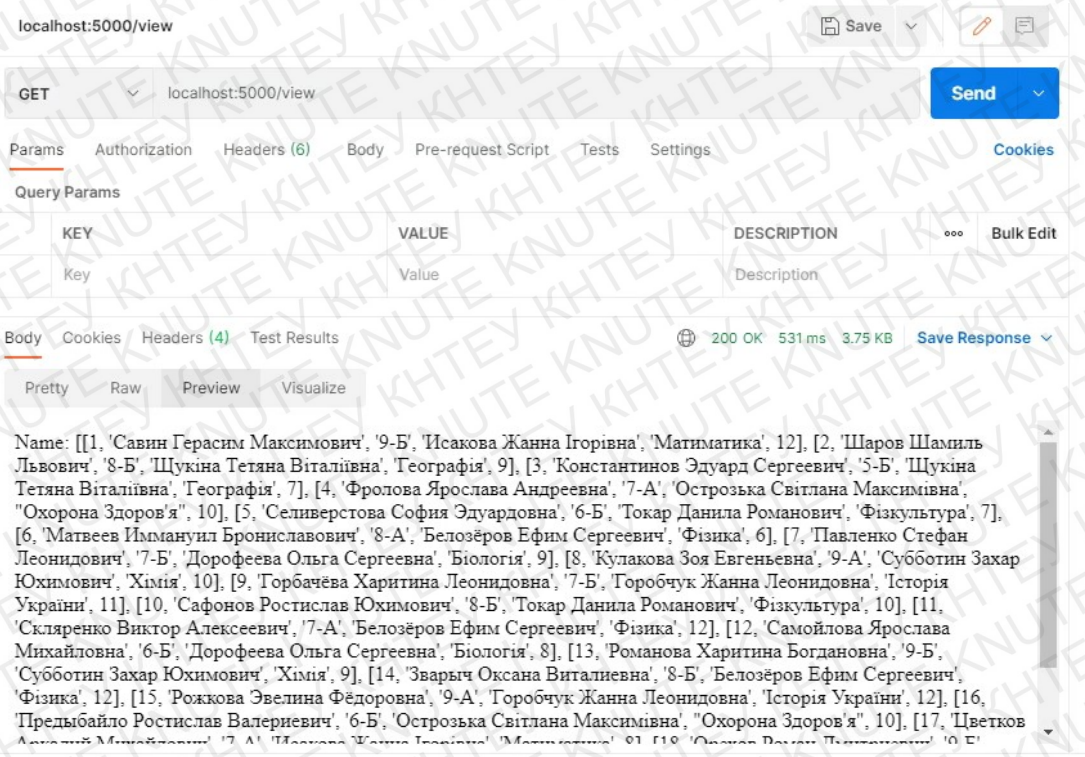

Рис. 3.22. Тестування запиту: "/view" для перегляду даних у веб-сторінці

Пошук даних за допомогою ПІБ учня

Що якщо нам потрібно отримати не усі записи цілком, а тільки запис певного учня для перегляду його оцінки за предмет. Для пошуку учня по його ПІБ було створено запит: " /find_name/<s_name> ", який виділяє певний запис з MongoDB та відображає його на веб-сторінці на рис. 3.23. Як ми бачимо спочатку ми викликаємо запит, в потів вписуємо ПІБ учня.

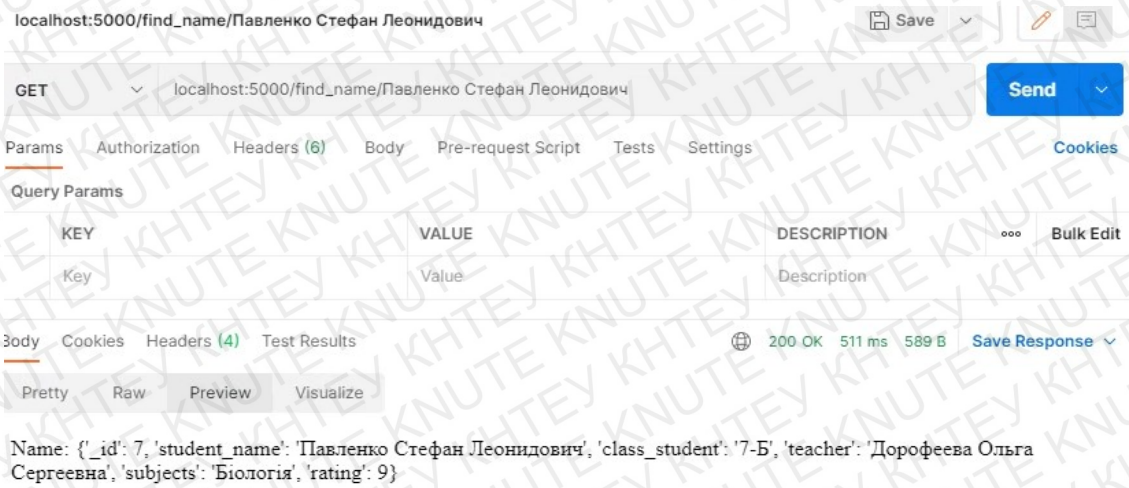

Рис. 3.23. Тестування запиту: "/find_name/Павленко Стефан Леонидович" для відображення певного запису учня

Звіт успішності проведених запитів по HTTP

За допомогою командного рядка можливо отримати звіт успішності викликаних запитів, тобто це історія викликаних запитів за допомогою локального хоста на рис. 3.24. У разі успішності API повертає нам число 200 – це означає, що запит виконано успішно та не має помилок, але якщо сталась помилка API не обробить запит та поверне нам число 500.

Рис. 3.24. Історія викликаних запитів відображених за допомогою командного рядка

У третьому розділі випускній кваліфікаційній роботі розв’язано завдання розроблення API для перетворення вмісту електронних таблиць в формат NoSQL бази даних. У результаті виконання одержано такі результати.

1. Проаналізовано методи та засоби створювання бази даних у MongoDB та користування інструментом графічного інтерфейсу MongoDB Compass для керування даними баз даних. Обгрунтована розробка API додатку для розв’язання завдання перетворення вмісту електронних таблиць у формат даних NoSQL для подальшого завантаження вмісту у СУБД.

2. Розроблено метод запиту */converting* для конвертації електронних таблиць формату Excel (xlsx) у формат JSON - текстовий формат для обміну даними з СУБД.

3. Розроблено метод запиту */update_data* для завантаження даних у СУБД MongoDB. Після конвертації можливо завантаження даних, API встановлює зв'язок між обраною БД у MongoDB та починає передачу даних у форматі JSON

4. Розроблено метод запиту */view* для перегляду даних у певній БД за допомогою веб-сторінки в форматі JSON.

5. Розроблено метод запиту */find_name/<s_name>* для знаходження певного документа за методом ключового слова. В даному методі можливо знайти потрібний документ за допомогою ПІБ (прізвище, ім’я, по батькові), саме ПІБ є аргументом змінної *s_name*.

ВИСНОВКИ

У випускній кваліфікаційній роботі було розроблений метод розробки API для перетворення вмісту електронних таблиць в формат NoSQL бази даних. Метод дозволяє конвертувати електроні файли формату xlsx у формат json, а також завантаження даних у базу даних MongoDB після конвертування. Для демонстрації роботи методу був розроблений API для веб-додатку.

Проаналізовано методи, моделі та засоби створювання API додатків, створення, формування та організація даних СУБД MongoDB. Обгрунтовано актуальність впровадження та використання API додатків у СУБД.

Для реалізації додатку було обрано текстовий редактор Sublime Text, мову програмування Python, СУБД MongoDB та фреймворк для створення веб-додатків Flask. Роботу API протестовано за допомогою HTTP-клієнта POSTMAN. Проведене тестування підтвердило ефективність та правильність роботи розробленого методу.

Виконано конвертування електронних таблиць в формат файлу який буде сумісний з БД NoSQL та завантаженням конвертованих даних у СУБД MongoDB. Також, було розроблення таких функції як перегляд даних БД на веб-сторінці у форматі JSON та пошуку певного документу в БД методом спів падання даних. Встановлено, що розроблений метод має програмну доцільність та ефективність.

СПИСОК ВИКОРИСТОВУВАНИХ ДЖЕРЕЛ

1. Трояновська Т. І. Метод розробки API / Т. І. Трояновська, М. Ф. Маховик // «Інформаційні технології та взаємодії» : IV Міжнародна науково-практична конференція, 8-10 листопада 2017 р. м. Київ. – С. 131-132.
2. API в веб-приложениях : [Електронний ресурс] – режим доступу до ресурсу : https://mkdev.me/posts/chto-takoe-api-v-veb-prilozheniyah-i-zachem-on-nuzhen.
3. Working with Web API : [Електронний ресурс] – режим доступу до ресурсу : https://launchschool.com/books/working_with_apis.
4. RESTful API Server – Doing it the right way: [Електронний ресурс] – режим доступу до ресурсу : http://blog.mugunthkumar.com/articles/restful-api-serverdoing-it-the-right-way-part-1/.
5. Подходы к проектированию RESTful API: [Електронний ресурс] – режим доступу до ресурсу : https://dataart.ru/news/podhody-k-proektirovaniyu-restfulapi/.
6. A Brief Introduction to REST: [Електронний ресурс] – режим доступу до ресурсу : https://www.infoq.com/articles/rest-introduction.
7. Mongoose ODM: [Електронний ресурс] – режим доступу до ресурсу : http://mongoosejs.com/docs/2.7.x/docs/embedded-documents.html.
8. MongoDB – NoSQL Database: [Електронний ресурс] – режим доступу до ресурсу : http://www.impressico.com/2015/09/30/mongodb-nosql-database/.
9. Коротко про API та його тестування: [Електронний ресурс] – режим доступу до ресурсу : https://qagroup.com.ua/publications/korotko-pro-ari-ta-jogo-testuvannia/.
10. Що таке API?: [Електронний ресурс] – режим доступу до ресурсу : https://seo24.kiev.ua/rozrobka/shho-take-api/
11. Керівництво по NoSQL для розробників: [Електронний ресурс] – режим доступу до ресурсу : https://javarush.ru/groups/posts/467-rukovodstvo-po-nosql-dlja-razrabotchikov-

12. Реляційні і Дані NoSQL: [Електронний ресурс] – режим доступу до ресурсу : https://docs.microsoft.com/ru-ru/dotnet/architecture/cloud-native/relational-vs-nosql-data

13. Создание веб-приложения с помощью Flask в Python 3 : [Електронний ресурс] – режим доступу до ресурсу : https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3-ru

14. Що потрібно знати про базах даних? : [Електронний ресурс] – режим доступу до ресурсу : http://datareview.info/article/chto-nuzhno-znat-o-bazah-dannyih-chast-2/

15. Створення API : [Електронний ресурс] – режим доступу до ресурсу : https://metanit.com/web/nodejs/4.11.php

16. Чотири API для бази даних : [Електронний ресурс] – режим доступу до ресурсу : https://habr.com/ru/post/541860/

17. NoSQL підхід в зберіганні даних : [Електронний ресурс] – режим доступу до ресурсу : https://jazzteam.org/ru/technical-articles/nosql-storage-approach/

18. Поради тестування API в Postman : [Електронний ресурс] – режим доступу до ресурсу : https://qagroup.com.ua/publications/porady-testuvannia-ari-v-postman/

19. Що таке бази даних NoSQL? : [Електронний ресурс] – режим доступу до ресурсу : https://aws.amazon.com/ru/nosql/

20. Навіщо потрібні нереляційних бази даних в big data: історія появи і розвитку : [Електронний ресурс] – режим доступу до ресурсу : https://www.bigdataschool.ru/wiki/nosql

21. Postman - як інструмент тестування API : [Електронний ресурс] – режим доступу до ресурсу : https://medium.com/effective-developers/postman-

22. Проектування баз даних : [Електронний ресурс] – режим доступу до ресурсу : https://ru.wikipedia.org/wiki

ДОДАТОК

Програмний код реалізації API додатку

```
import json
from collections import OrderedDict
from itertools import islice
from openpyxl import load_workbook
import pymongo
from pymongo import MongoClient
from flask import Flask, render_template
from flask_restful import Api
from pymongo import MongoClient
from flask_pymongo import PyMongo

app = Flask(__name__, template_folder="C:/Users/support tech/Desktop/API/template")
api = Api(app)

client = MongoClient('localhost', 27017)
db = client['school']
collection_base = db['classbook']


@app.route('/converting', methods=['POST'])
def converting():
file_table = load_workbook('classbook.xlsx')
sheet = file_table['book']
table_list = []

for row in islice(sheet.values, 1, sheet.max_row):
    cell = OrderedDict()
    cell['_id'] = row[0]
    cell['student_name'] = row[1]
    cell['class_student'] = row[2]
    cell['teacher'] = row[3]
```

```
    cell['subjects'] = row[4]
    cell['rating'] = row[5]
    table_list.append(cell)

j = json.dumps(table_list, ensure_ascii=False)

with open('classbook.json', 'w', ) as fl:
        fl.write(j)
return 'THE XLSX SPREADSHEET HAS BEEN CONVERTED TO THE JSON FILE FORMAT'

@app.route("/view", methods = ['GET'])
def view():
data = collection_base.find()

result = []
for i in data:
        result.append([i['_id'],i['student_name'],i['class_student'],i['teacher'],i['subjects'],i['rating']])

return render_template('base.html', name=result)

@app.route("/find_name/<s_name>", methods = ['GET'])
def find_name(s_name):
search = collection_base.find({'student_name': s_name})

for document in search:
        return render_template('base.html',name=document)

@app.route('/update_data', methods=['POST'])
def update_data():
client.drop_database(db)

with open('classbook.json') as f:
    file_data = json.load(f)
    collection_base.insert_many(file_data)

return 'DATABASE HAS BEEN UPDATED AND READY FOR WORK'
```

```
if __name__ == "__main__":
    app.run(debug=True)
```