Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: Методи та програмні засоби обміну даних між різноплатформеними базами даних

Виконав: студент II курсу групи 1ПІ-18 м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Розумовський Б.С.
(прізвище та ініціали)

Керівник: к. т. н., доц. Рейда О. М.
(прізвище та ініціали)

ВНТУ – 2019

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
"____"_____2019 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Розумовському Богдану Сергійовичу

1. Тема роботи – методи та програмні засоби обміну даних між різноплатформеними базами даних.

Керівник роботи: Рейда Олександр Миколайович, доц.каф ПЗ, затверджені наказом вищого навчального закладу від "__"_____ 2019 року №___

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: теорія баз даних, методи оптимізації складних запитів.

4. Зміст розрахунково-пояснювальної записки: Аналіз стану питання та постановка задачі; розробка методів та програмних засобів обміну даних між різноплатформеними базами даних; розробка баз даних; розробка адміністративної частини з використанням обміну даних; розробка скрипта поєднання; тестування адміністративної частини; економічна частина.

5. Перелік графічного матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О.М., к.т.н., доцент кафедри ПЗ		
5	Бальзан М.В., к.е.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та постановка задачі	07.10.2018 – 27.10-2019	Вик.
2	Розробка методів та програмних засобів обміну даних між різноплатформеними базами даних	28.10.2019 – 8.11.2019	Вик.
3	Розробка адміністративної частини з використанням обміну даних	9.11.2019 – 20.11.2019	Вик.
4	Тестування розробленої адміністративної частини	21.11.2019 – 1.12.2019	Вик.
5	Економічна частина	1.12.2019 – 6.12.2019	Вик.

Студент _____ **Розумовський Б.С.**

(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Рейда О.М.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В магістерській дипломній роботі проведено аналіз та порівняння різних методів та засобів зберігання даних в базу даних. Проведено дослідження сучасних актуальних технологій які дозволяють зберігати дані.

Отримані в магістерській кваліфікаційній роботі наукові та практичні результати можна використати для побудови різноплатформеної бази даних, якщо сайту потрібно зберігати велику кількість даних і ці дані повинні бути структуровані.

ABSTRACT

In the master's thesis the analysis and comparison of different methods and means of data storage in the database is carried out. The researches of modern actual technologies allowing to store data are carried out.

The scientific and practical results obtained in the master's qualification work can be used to build a multi-platform database if the site needs to store a large amount of data and this data must be structured.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Більшість інформаційних систем сьогодні зберігають дані в сховищах, що називаються базами даних. Найбільш популярні бази даних - Microsoft SQL Server, Oracle, PostgreSQL або MySQL - використовують реляційну модель представлення даних. У представлених базах даних сутності представляються у вигляді рядків таблиць, їх параметри - у вигляді стовпців, а зв'язок між сутностями різних типів здійснюється за допомогою відносин один-до-одного або один-до багатьох. [1]

Бази даних не завжди підходять для зберігання та обробки даних типу BigData з використанням ієрархічних структур, які вимагають створення комплексних запитів для роботи з даними.

NoSQL — база даних, яка забезпечує механізм зберігання та отримання даних відмінний від підходу таблиць-відношень в реляційних базах даних. NoSQL бази даних все більше і більше використовуються в задачах із застосуванням великих даних та real-time[en] web-застосунках.[6]

Мотиви нереляційного підходу включають: простоту дизайну схеми БД, значно спрощене горизонтальне масштабування на кластери машин (що є проблемою для реляційних баз даних), і тонкий контроль над доступністю. Структури даних, що використовуються в NoSQL (такі як ключ-значення, сховище з широким стовпчиком, граф, документ) є відмінними від тих, що використовуються за замовчуванням в реляційних базах, що робить тим самим деякі операції над даними значно швидшими на NoSQL. Точна відповідність використання NoSQL бази даних залежить від проблем, які треба вирішити. Іноді структури даних, які використовуються в NoSQL базах можуть розглядатись як більш гнучкі ніж таблиці реляційних моделей.

Однією з найбільших проблем є процес створення реляційного сховища, який включає в себе етап проектування моделі даних. На цій стадії можна оцінити вузькі місця обраної стратегії і спроектувати дійсно надійну і зручну систему. NoSQL рішення не вимагають визначати схему бази даних перед початком роботи, тому в

процесі розробки можна наштовхнутися на непередбачені труднощі, які можуть призвести до відмови від даного NoSQL рішення.

Тому виникає цілком обґрунтована потреба в розробці методів та програмних засобів обміну даних між різноплатформеними базами даних, що базується на комплексному підході для зберігання та обробки даних .

Актуальність роботи. Актуальність визначена відсутністю комплексних методів зберігання та обробки даних типу BigData для підвищення швидкодії доступу.

Мета та завдання дослідження. Метою роботи є підвищення швидкодії обробки запитів при роботі з структурованими даними типу BigData. За допомогою методів та програмних засобів обміну інформації між різноплатформеними базами даних.

Основними задачами дослідження є:

- Провести аналіз існуючих методів і засобів створення і реалізації баз даних;
- Розробити методи обміну даними між двома різноплатформеними базами даних;
- Розробити методи підвищення продуктивності роботи з даними;
- Розробити програмні компоненти для обміну даними між двома базами даних;
- Провести тестування розробленої адміністративної частини, яка використовує методи та програмні засоби обміну даних між різноплатформеними базами даних різноплатформеної бази даних.

Об’єкт дослідження – процес обміну даними між різноплатформеними базами даних.

Предмет дослідження – методи та програмні засоби обміну даними між різноплатформеними базами даних.

Методи дослідження. У процесі досліджень використовувались: теорія баз даних, методи оптимізації складних запитів, методи опрацювання даних та метаданих у розподілених середовищах створено за допомогою XML-технології. Для розроблення засобів аналізу, проектування реляційних баз часово-залежних даних застосовано технології аналізу та проектування інформаційних систем та програмного

забезпечення. Розроблені засоби реалізації реляційних баз часово-залежних даних ґрунтуються на методах прикладної лінгвістики та мові запитів SQL.

Наукова новизна отриманих результатів.

1. Вперше запропоновано метод обмін даними між реляційною та не реляційною базами даних,для роботи з BigData. Особливістю якої полягає в обміні даних між різноплатформеними базами даних, що дозволяє використовувати BigData, у реляційній базі даних.
2. Вперше запропоновано зберігати індексні значення в реляційній базі даних, і за допомогою них, звертатись у не реляційну базу даних, щоб отримати повну інформацію.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень розроблено метод створення та зберігання даних в різноплатформених базах даних.

Апробація результатів роботи. Результати роботи були представлені на секції програмного забезпечення XLVIII Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії 14 – 15 березня 2019 р.

Особистий внесок магістранта. Усі наукові результати отримано автором самостійно.

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ

1.1 Аналіз стану питання

Поняття «реляційний» (англ. relation – відношення) пов’язане з розробками відомого американського фахівця в області систем баз даних Е. Кодда.

Ці моделі характеризуються простотою структури даних, зручним для користувача табличним уявленням і можливістю використання формального апарату алгебри відносин і реляційного числення для обробки даних.

Реляційна модель орієнтована на організацію даних у вигляді двовимірних таблиць.

Реляційна модель даних є сукупністю взаємозв’язаних двовимірних таблиць – об’єктів моделі. [1]

Зв’язки між двома логічно зв’язаними таблицями в реляційній моделі встановлюються по рівності значень однакових атрибутів цих таблиць.

Кожна реляційна таблиця є двовимірним масивом і володіє наступними властивостями: ·

- кожен елемент таблиці являє собою один елемент даних; ·
- всі стовпці в таблиці однорідні, тобто всі елементи в стовпці мають однаковий тип (числовий, символьний тощо) і довжину;
- кожен стовпець має унікальне ім’я; ·
- однакові рядки в таблиці відсутні; ·
- порядок розміщення рядків і стовпців може бути довільним.

При описі реляційної моделі часто використовують такі терміни: відношення, кортеж, домен.

Поле, кожне значення якого однозначно визначає відповідний запис, називається простим ключем (ключовим полем). Якщо записи однозначно визначаються значеннями декілька полів, то така таблиця бази даних має складений

ключ. Між двома реляційними таблицями можуть бути сформовані зв’язки. Різні таблиці, можуть бути зв’язані між собою через загальне поле даних. [2]

Першими базами даних, які отримали широке розповсюдження, були великі БД організацій, побудовані на основі ієрархічної або мережової моделі. Через декілька років з’явилися системи, створені на основі реляційної моделі. Нові системи управління базами даних, які не є реляційними, належать до об’єктних або гібридних об’єктно-реляційних моделей.

NoSQL системи також називають «Not only SQL» (англ. not only SQL — не тільки SQL) для підкреслення того, що вони можуть підтримувати SQL-подібну структуру та мову запитів. Мотиви цього підходу включають: простоту дизайну схеми БД, значно спрощене горизонтальне масштабування на кластери машин (що є проблемою для реляційних баз даних), і тонкий контроль над доступністю. Структури даних, що використовуються в NoSQL (такі як ключ-значення, сховище з широким стовпчиком, граф, документ) є відмінними від тих, що використовуються за замовчуванням в реляційних базах, що робить тим самим деякі операції над даними значно швидшими на NoSQL. Точна відповідність використання NoSQL бази даних залежить від проблем, які треба вирішити. Іноді структури даних, які використовуються в NoSQL базах можуть розглядатись як більш гнучкі ніж таблиці реляційних моделей.[5]

Ключовим фактором, який змусив світову IT-спільноту задуматися над новими стратегіями зберігання і доступу до інформації, стало планомірне зростання обсягів даних у мережі Інтернет. У зв’язку з цим з’явилося Big Data. Big Data — це стратегія ефективної праці з величезними постійно зростаючими масивами даних.

На основі Big Data концепції чітко вимальовувалася необхідність в моделі бази даних, яка буде більше націлена на швидкість доступу і масштабованість. Потрібно було якесь більш просте рішення, аніж існуючі реляційні БД, при цьому не поступалися їм у ефективності вирішення конкретних завдань. В першу чергу, це завдання побудови хмарних сховищ, де кінцевому користувачеві в першу чергу важлива швидкість доступу і можливий обсяг інформації, що там зберігатиметься.[6]

Виділяють чотири основні типи NoSQL баз даних. Вони розрізняються моделлю даних, підходом до розподіленості і реплікації, завдяки чому можуть в різній мірі підходити під ті чи інші види конкретних завдань.

Бази Даних “ключ-значення” представляють собою найпростіший вид бази даних, будучи, по суті, асоціативним масивом, де кожному значенню відповідає свій унікальний ключ. Простота баз даних цього типу відкриває простори неймовірної масштабованості. Не потрібно ніяких схем побудови бази даних, немає ніякого зв’язку між значеннями, по суті кількість елементів асоціативного масиву обмежена лише обчислювальними потужностями. Саме тому даний вид баз даних цікавий в першу чергу компаніям, що надають хмарні послуги.

Документоорієнтована БД являє собою систему зберігання ієрархічних структур даних (документів), що має структуру дерева або лісу. Структура дерева починається з кореневого вузла і може мати кілька внутрішніх і зовнішніх вузлів. Зовнішні вузли містять кінцеві дані, які при додаванні заносяться в індекси бази, завдяки яким можна здійснювати швидкий пошук навіть при досить складною загальній структурі сховища. Фактично документоорієнтовані БД є більш складною версією сховищ “ключ-значення” — вони все ще не дуже хороші для систем, що мають на увазі безліч зв’язків між елементами, але дозволяють здійснювати вибірку за запитом без повного завантаження окремих документів у оперативну пам’ять. Механізми пошуку дозволяють знаходити як документи цілком, так і частини документів, а деревоподібна структура дозволяє організовувати окремі колекції документів одного типу або схожої тематики.

Графова модель бази даних являє собою узагальнення мережевій моделі даних і відрізняється сильними зв’язками між вузлами. Графові бази даних найкраще підходять для реалізації проектів, які передбачають природну графову структуру даних. В першу чергу соціальних мереж, а так само для створення семантичних павутини. У подібних завданнях вони сильно випереджають реляційні БД по продуктивності, простоті внесення змін і наочності подання інформації. У деяких баз даних існують механізми спеціальної оптимізації для роботи з SSD-накопичувачами.

Для роботи з досить великими графами використовуються алгоритми, які передбачають часткове приміщення графа в оперативну пам’ять. Найбільш відомі графові СУБД це ArangoDB, FlockDB, Giraph, HyperGraphDB, Neo4j, OrientDB.

Bigtable-подібні бази даних чи інакше сховища сімейств колонок містять дані, впорядковані у вигляді розрідженій матриці, рядки і стовпці якої використовуються в якості ключів. Ці сховища мають багато спільного з документоорієнтованими Базами Даних — системами керування вмістом (CRM), Event агрегаторами реєстрації на події, блоги. Не варто плутати bigtable-подібні бази даних з лінійними сховищами, які є, по суті, реляційними БД з роздільним зберіганням колонок.[10]

Як правило, ці бази даних застосовуються для веб-індексування і рішення інших завдань, які передбачають величезні обсяги даних.

1.2 Аналіз інструментів для візуального проектування баз даних

Для спрощення роботи з сервером MySQL в базовий комплект установки входить такий інструмент як MySQL Workbench. Він являє графічний клієнт для роботи з сервером, через який ми в зручному вигляді можемо створювати, видаляти, змінювати бази даних і управляти ними. Так, на Windows після установки в меню Пуск ми можемо знайти значок програми і запустити її.

Нам відкриється наступне вікно, де ми можемо побачити поле з назвою запущеного локально примірника MySQL. Якщо створено декілька баз даних, всі вони будуть виведені на екран. Це дуже спрощує роботу з декількома підключеннями. На рисунку 1.1 показане вікно при вході.

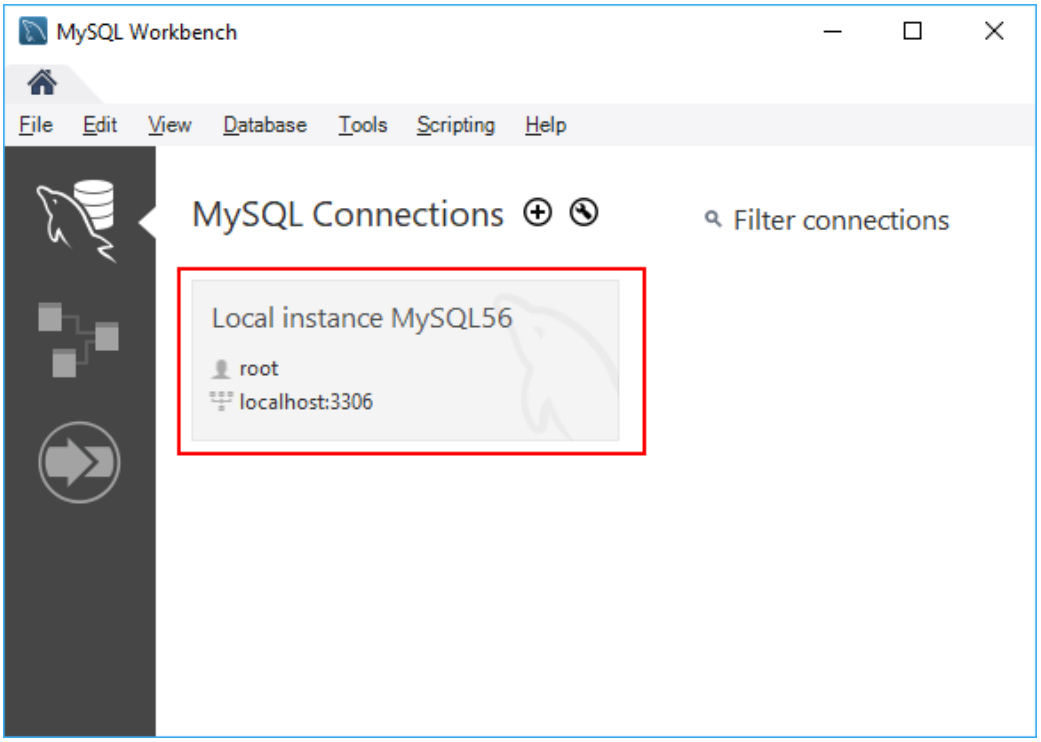


Рисунок 1.1 – Місцезнаходження значка запуску в меню Пуск

Натиснемо на нього, і нам відкриється вікно для введення пароля. Для кожного підключення краще використовувати різні паролі і імена користувачів. Тут треба ввести пароль, який був встановлений для користувача root при встановленні MySQL. На рисунку 1.3 спливаюче вікно при підключенні.

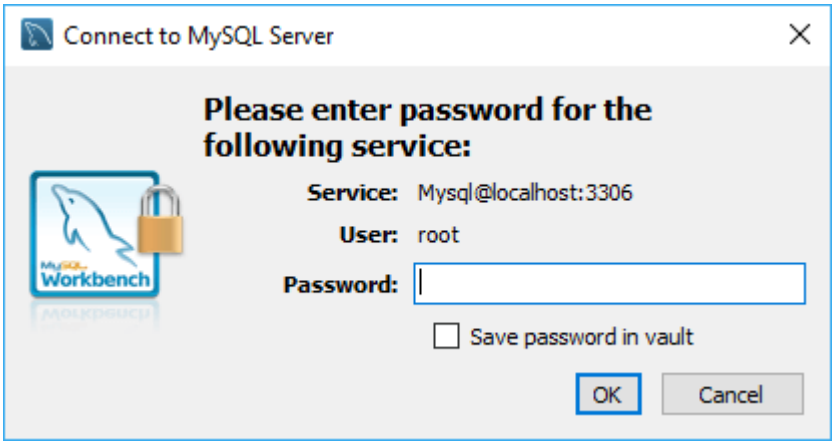


Рисунок 1.2 – Вікно вводу пароля для підключення

Після успішного логіна нам відкриється вміст сервера.

Зокрема, в лівій частині у вікні SCHEMAS можна побачити доступні бази даних.

Щоб виконувати в цій програмі запити до бд, спочатку створимо саму БД. Для цього потрібно натиснути над списком баз даних на значок "SQL" з плюсом.

Після цього в центральній частині програми відкриється вікно для введення скрипта SQL.

Дана команда створює базу даних usersdb.

Для виконання скрипта в панелі інструментів на тиснемо на значок блискавки.[13]

На рисунку 1.3 показано добавлення нової бази даних.

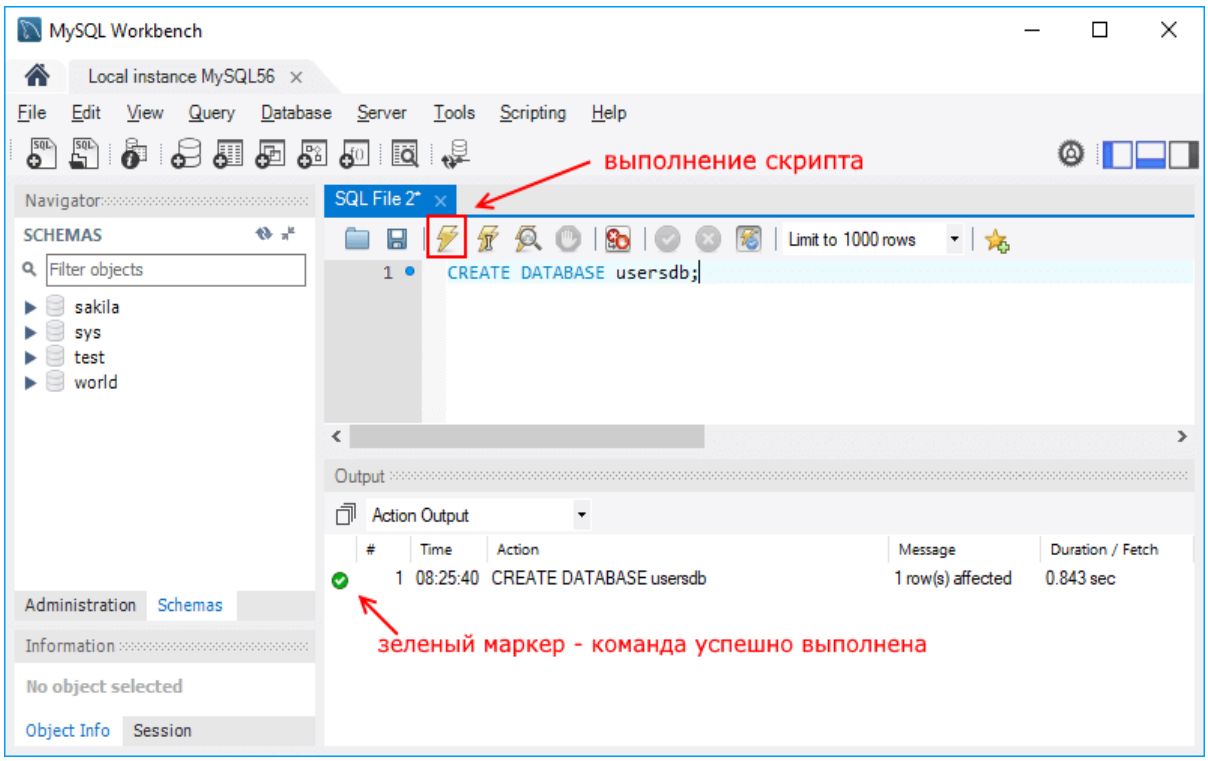


Рисунок 1.3 – добавлення нової бази даних userdb

"Postman - це потужний набір інструментів тестування API, який став необхідним для багатьох розробників. Ми робимо прекрасні продукти, що допомагають створювати приголомшливі API і покращувати продуктивність праці розробки. Postman використовується більш ніж мільйоном розробників по всьому

світу, і це число постійно зростає. Ми плануємо розробку інших продуктів, щоб надати розробникам найбільш потужне рішення для розробки і тестування API ".

Я абсолютно згоден з цим описом. Нинішня версія Postman ще більш потужна, ніж та, з якою я починав, і безліч Pro-функциональностей зараз доступні безкоштовно. Не лякайтеся, цей інструмент створений не тільки для розробників. Навіть його крута просунута функція - тестування автоматичних відповідей - не вимагає особливих знань. Це дуже простий і в той же час потужний інструмент.

Давайте пройдемся по його кращим особливостям!

Якщо ви встановлюєте Postman вперше, команда розробки надає вам колекцію "Postman Echo". Це набір збережених запитів (і відповідей), організованих логічно. Postman Echo зроблений для легкого старту тестування API із заздалегідь налаштованими запитами, від яких ви можете відштовхуватися.

"Postman Sandbox" це середовище виконання JavaScript доступна при написанні "Pre-request Script" і "Tests" скриптів. "Pre-request Script" використовується для проведення необхідних операцій перед запитом, наприклад, можна зробити запит до іншої системи і використовувати результат його виконання в основному запиті. "Tests" використовується для написання тестів, перевірки результатів, і при необхідності їх збереження в змінні. На рисунку 1.4 зображена послідовність виконання запиту.

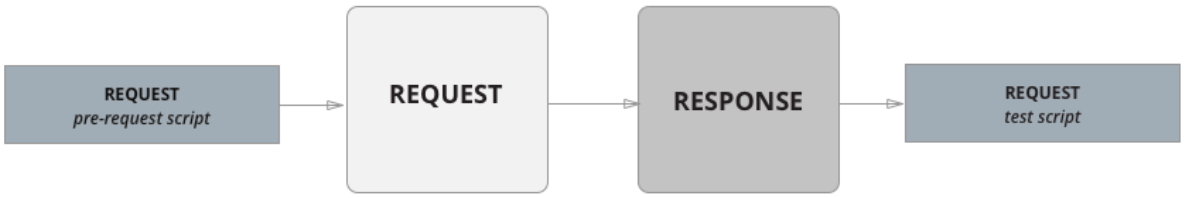


Рисунок 1.4 –Послідовність виконання запиту

Якщо відкрити цей набір, перейти в "Методи запитів" і потім в "Запити GET", всі збережені дані відобразяться в центральній частині вікна Postman. Тепер натисніть "Відправити".

На рисунку 1.5 зображено як відправити ваш перший запит і отримали першу відповідь.

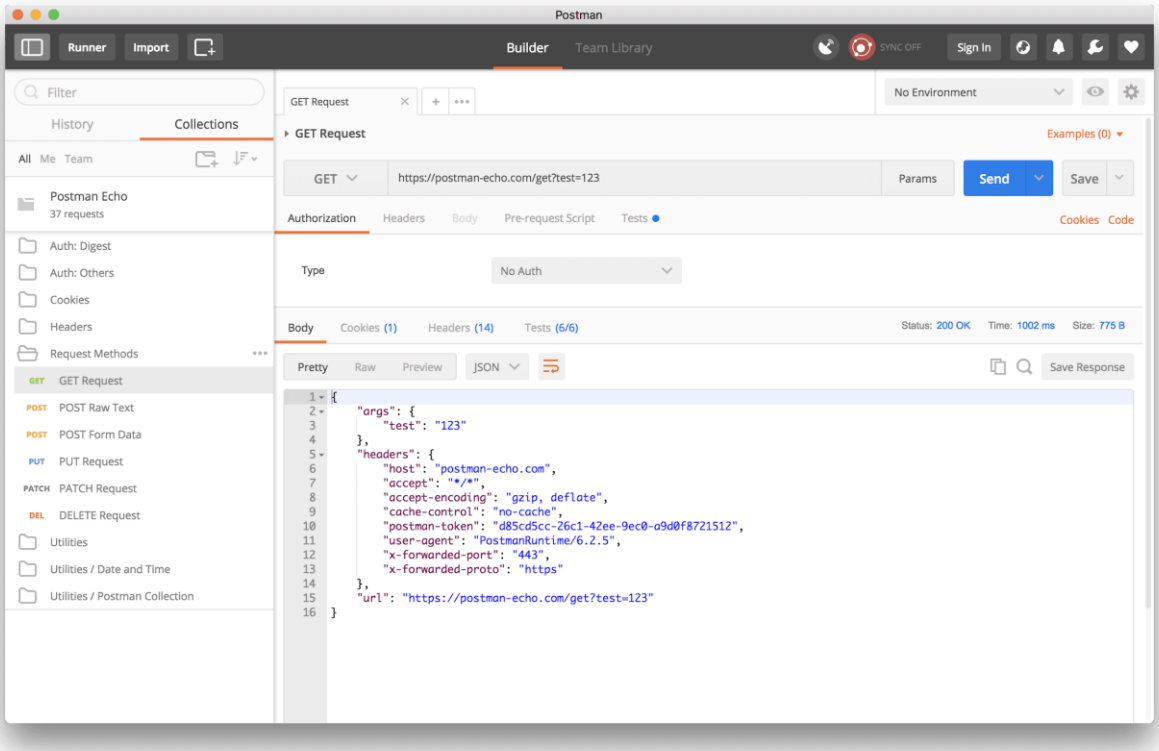


Рисунок 1.5 –Перший запит у Postman

Запити POST трохи складніші, але все одно зрозумілі і логічні.

На цей раз буде використовувати свій власний запит. Натисніть на плюсик, щоб відкрити нову вкладку, змінити тип запиту з GET на POST, і використовуйте https://jsonplaceholder.typicode.com/posts як URL запиту. Це безкоштовний REST-сервіс, який можна використовувати для фальшивих даних. Він дуже хороший для навчання, імітації роботи сервера, поширення прикладів коду.[14]

Тепер вам потрібно тіло POST-запиту. Натисніть на "Body" під URL запиту, змінити тип на "raw" і "Text" на "JSON". І на рисунку 1.5 зображено текст який потрібно вставити.

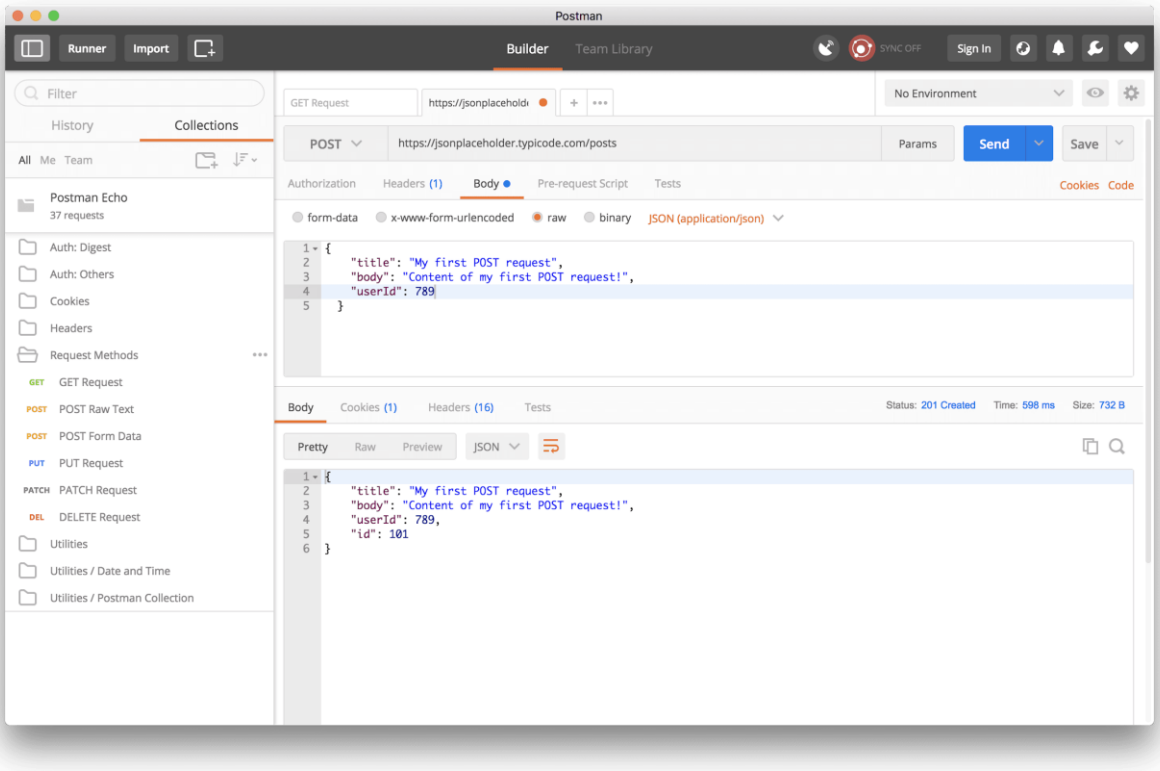


Рисунок 1.6 – POST-запит у Postman

На мою думку, було розкро не всі можливості середовищ розробки, лише ті які я використовую щодня і які спрощують і прискорюють тестування.

1.3 Порівняльний аналіз аналогів

Оскільки методи та програмні засоби обміну даних між різноплатформеними базами даних включають в себе все найкраще з двох баз можна порівняти її з використанням лише одного підходу до створення бази даних.

Недоліками, яких немає у моєму різноплатформеному обміну даних є:

- Додаток сильно прив’язується до конкретної СУБД. Мова SQL універсальна для всіх реляційних баз даних 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 складанням 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 специфічних моментів оператору бази даних доведеться освоювати “на ходу”.
- Хоча SQL і замислювався, як засіб роботи кінцевого користувача, врешті-решт він став настільки складним, що перетворився на інструмент програміста.
- Незважаючи на наявність міжнародного стандарту ANSI SQL-92, багато компаній, СУБД (наприклад, Oracle, Sybase, Microsoft, MySQL), що займаються розробкою, вносять зміни до мови SQL, вживаної в розроблених ними СУБД, тим самим відступаючи від стандарту. Таким чином з'являються специфічні для кожної конкретної СУБД діалекти мови SQL.

Отже, самі ці недоліки повинна усунути методи та програмні засоби обміну даних між різнопатформеними базами даних.

1.4 Аналіз середовищ для методів обміну даними між різноплатформеними базами даних

Для розробки мобільних додатків оглянемо такі середовища розробки:

- Visual Studio Code;
- Eclipse;
- Sublime text .

За свою довгу історію існування Microsoft випустив чимало інструментів розробки. Але так уже склалося у всіх тільки Visual Studio - велика і потужна IDE «комбайн» призначена для всього і вся. Розвивається цей продукт вже більше двох десятків років і увібрав в себе самі різні функції. Багатьом цей інструментарій подобається і іноді навіть задавали питання - чи буде перенесений Visual Studio на інші платформи. На що найчастіше отримували відповідь немає. Напевно, з якогось дива, в цілому таке портування буде дорогим і невиправдано складним, вже дуже багато всього в цій IDE зав'язано на Windows. Чи не замахуючись на всі функції повноцінної IDE, всередині Microsoft вирішили переосмислити підхід, за яким будується основний інструментарій програміста і почали з самого головного - редактора коду. Visual Studio Code це саме редактор, але при цьому володіє функціями IDE, що покладається на розширення.

Visual Studio Code відмінний вибір і має необхідний мінімум:

- непогану документацію
- автодоповнення коду (з використанням IntelliSense)
- підсвічування синтаксису
- вбудований відладчик
- розширення функціоналу за рахунок плагінів
- управління системою контролю версій git
- багатоплатформеної
- безкоштовний, з відкритим вихідним кодом

Sublime Text - це такий легкий текстовий редактор для програмістів. Щось типу Vim, тільки з людським інтерфейсом і з коробки вміє помітно більше. Також можна розглядати Sublime Text в якості IDE. До рівня IntelliJ IDEA, CLion або PyCharm йому, звичайно, як до місяця. Зате для всілякої езотерики на кшталт Go, Erlang і Haskell працює шикарно.

Багато хто оцінить легковажність редактора і швидкість його роботи. Sublime Text просто літає, в тому числі на проектах в мільйон рядків коду на хардкорних C ++ с Boost'ом і ось цим усім. При цьому саблайм має дуже низький поріг входження, чого про Vim, наприклад, сказати ніяк не можна.

Не зовсім явний момент при використанні Sublime Text – можливість відкривати не файли, а каталоги. Робиться це за допомогою File → Open Folder, або шляхом передачі імені каталогу в якості аргументу при запуску редактора з bash. При цьому у вас не тільки з'являється дуже симпатичне дерево каталогів в сайдбарі ліворуч, а й, наприклад, набагато крутіше починає працювати пошук файлу по імені (Ctr + P). Також він є платним.

Eclipse є безкоштовною програмною платформою з відкритим вихідним кодом, контролюється організацією Eclipse Foundation. Написана на мові програмування Java і основною метою її створення є підвищення продуктивності процесу розробки програмного забезпечення.

Претендує на статус найбільш популярною Java IDE і є єдиним конкурентом такої потужної платформи як NetBeans.

Але на відміну від NetBeans який для створення елементів призначеного для користувача інтерфейсу використовує від платформи незалежну бібліотеку Swing, в Eclipse використовується платформо-залежна бібліотека SWT - Standard Widget Toolkit.

IDE розроблені на базі платформи Eclipse застосовуються для створення програмного забезпечення на різних мовах програмування, так як Eclipse є платформою для розробки будь-яких інтегрованих середовищ програмування і розширень для себе ж, за принципом "Додатки для Eclipse розробляються в самій Eclipse".

Проаналізувавши платформи та їх можливості, було визначено, що Visual Studio Code являється найсучаснішою платформою для програмування на Java Script, має багато функцій, спрощує тестування, а також має багато розширень. Платформа перейняла найкращі характеристики з попередньо наведених, тому для розробки програмного продукту була обрана Visual Studio Code.

1.5 Аналіз технологій та мов програмування

Існує багато програм для написання запитів, їх обробці та виведенні інформації користувачу. Найвідоміші з них: Java, JavaScript, PHP, Ruby, Python.

Java — об'єктно-орієнтована мова програмування. У мові є купа особливостей у вигляді конструкторів класів, винятків, що призводять до падіння додатків під час роботи і інших моментів, які завжди необхідно враховувати при розробці [11]. Втім, код на Java легко читається і структурується, особливо при дотриманні прийнятих стандартів його оформлення.

JavaScript ("JS" для стислості) - це повноцінний динамічний мову програмування, який застосовується до HTML документу, і може забезпечити динамічну інтерактивність на веб-сайтах. Його розробив Brendan Eich, співзасновник проекту Mozilla, Mozilla Foundation і Mozilla Corporation.

JavaScript неймовірно універсальний. Ви можете почати з малого, з простих функцій, таких як каруселі, галереї зображень, що змінюються макети і відгук на натискання кнопок. Маючи великий досвід, ви можете створювати ігри, анімовану 2D і 3D графіку, повномасштабні програми з базами даних і багато іншого!

JavaScript сам по собі досить компактний, але дуже гнучкий. Розробниками написано велику кількість інструментів поверх основного мови JavaScript, які розблокують величезна кількість додаткових функцій з дуже великим зусиллям.

PHP - це широко використовувана мова сценаріїв загального призначення з відкритим вихідним кодом.

Говорячи простіше, PHP це мова програмування, спеціально розроблений для написання web-додатків (сценаріїв), що виконуються на Web-сервері.

Абревіатура PHP означає "Hypertext Preprocessor (Препроцесор Гіпертексту)". Синтаксис мови бере початок з C, Java і Perl. PHP досить простий для вивчення. Перевагою PHP є надання web-розробникам можливості швидкого створення динамічно генеруються web-сторінок. Детальніше про переваги PHP можна дізнатися тут.

Важливою перевагою мови PHP перед такими мовами, як мов Perl і C полягає в можливості створення HTML документів з впровадженими командами PHP. Детальніше про цю можливість дивіться тут.

Значним відмінністю PHP від якого- бо коду, що виконується на стороні клієнта, наприклад, JavaScript, є те, що PHP-скрипти виконуються на стороні сервера. Ви навіть можете конфігурувати свій сервер таким чином, щоб HTML-файли оброблялися процесором PHP, так що клієнти навіть не зможуть дізнатися, отримують чи вони звичайний HTML-файл або результат виконання скрипта.

Ruby - інтерпретована, повністю об'єктно-орієнтована мова програмування з чіткою динамічною типізацією. Він поєднує в собі Perl-подібний синтаксис з об'єктно-орієнтованим підходом. Також деякі риси запозичені з мов програмування Python, Lisp, Dylan і CLU. Кроссплатформенная реалізація інтерпретатора мови Ruby поширюється на умовах відкритого програмного забезпечення. Код, написаний на Ruby, може бути зрозумілий навіть людині, яка не розбирається в програмуванні. На RoR були створені такі проекти, як Redmine, Twitter, Shopify, Basecamp, GitHub, Kickstarter, Airbnb і інші.

Python це мова програмування загального призначення, націлений в першу чергу на підвищення продуктивності самого програміста, ніж коду, який він пише. Говорячи простою людською мовою, на Python можна написати практично що завгодно (веб- / настільні додатки, ігри, скрипти по автоматизації, комплексні системи розрахунку, системи управління життєзабезпеченням і багато багато іншого) без відчутних проблем. Більш того, поріг входження низький, а код багато в чому

лаконічний і зрозумілий навіть того, хто ніколи на ньому не писав. А з точки зору бізнесу це тягне за собою скорочення витрат і збільшення продуктивності праці співробітників.[17]

Проаналізувавши мови для написання запитів, їх обробці та вивдення інформації користувачу можна зробити висновок, що найдоцільніше використовувати мову JavaScript (Node.js, React) та PHP (Phalcon) для написання програмного продукту. Оскільки інші програми поступаються своїми можливостями та функціоналом.

1.5 Постановка задач роботи

На основі проведеного аналізу стану питання для автоматизованого ведення власних фінансів у магістерській кваліфікаційній роботі потрібно виконати такі задачі:

- проаналізувати особливості та розробити структуру бази даних;
- розробити моделі та структуру баз даних;
- розробити методи та програмні засоби обміну даних;
- розробити алгоритм функціонування адміністративної частини;
- розробити інтерфейс автоматизованої системи;
- провести тестування створеного програмного продукту.

1.5 Висновки

У першому розділі магістерської кваліфікаційної роботи проведено аналіз предметної області, також порівняно аналоги програмного продукту. Методи та програмні засоби обміну даних між різноплатформеними базами даних, будуть створені для впровадження, в будь яку сферу, яка працює з великим об'ємом даних. Проаналізовані аналоги дозволили сформулювати вимоги до методів та програмних засобів, який орієнтований на зручність розробки і подальшої підтримки.

В якості Баз данних було обрано MySql та ElasticSearch, мовою програмування – JavaScript та PHP. Для JavaScript бібліотеки React та фреймворк Node.js, Phalcon - фреймворк PHP. Середовищем програмування – середовище розробки Visual Studio Code.[9]

2 РОЗРОБКА МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ ОБМІНУ ДАНИХ МІЖ РІЗНОПЛАТФОРМЕНИМИ БАЗАМИ ДАНИХ

2.1 Проектування баз даних

Створення бази даних слід починати з її проектування (розробки). У результаті проектування має бути визначена структура бази, тобто склад таблиць, їхня структура та логічні зв'язки. Структура реляційної таблиці визначається складом стовпців, їхньою послідовністю, типом даних кожного стовпця та їхнім розміром, а також ключем таблиці. Процес проектування можна здійснювати двома підходами. За першого підходу спочатку визначають основні задачі, для розв'язання яких створюється база, та потреби цих задач у даних. За другого підходу визначають предметну область (сферу), здійснюють аналіз її даних і встановлюють типові об'єкти предметної області. Найбільш раціональним підходом проектування бази даних є поєднання обох підходів.

Зазвичай з базами даних працюють дві категорії користувачів. Перша категорія - проектувальники. Процес проектування бази даних поділяється на етапи, кожний з яких передбачає виконання певних дій. Перший етап-розробка інформаційно-логічної моделі даних предметної області, який базується на описі предметної області, одержаному в результаті її обстеження. На цьому етапі спочатку визначають склад і структуру даних предметної області, які мають міститись у базі даних та забезпечувати виконання запитів, задач і застосувань користувача. Ці дані мають форму реквізитів, що містяться в різних документах - джерелах завантаження бази

даних. Аналіз виявлених даних дозволить визначити функціональні залежності реквізитів, які використовують для виділення інформаційних об'єктів, що відповідають вимогам нормалізації даних. Подальше визначення структурних зв'язків між об'єктами дозволяє побудувати інформаційно-логічну модель.

Другий етап - визначення логічної структури бази даних. Для реляційної бази даних цей етап є значною мірою формальним, оскільки інформаційно-логічна модель відображається в структуру реляційної бази даних адекватно.

Наступний етап - конструювання таблиць бази даних, який здійснюється засобами СУБД, та узгодження їх із замовником. Структура таблиць бази даних задається за допомогою засобів опису (конструювання) таблиць у СУБД із цілковитою відповідністю інформаційним об'єктам. Крім таблиць, проектувальники розробляють й інші об'єкти бази даних, які призначені, з одного боку, для автоматизації роботи з базою, а з іншого - для обмеження функціональних можливостей роботи з базою (безпека бази даних). Після формування структури таблиць база даних може наповнюватись даними з документів-джерел. Проектувальники, як правило, не наповнюють базу конкретними даними (оскільки замовник може вважати дані конфіденційними і не надавати стороннім особам). Винятком є експериментальне наповнення модельними даними на етапі відлагодження об'єктів бази.

Друга категорія - користувачі, які працюють з базами даних. Вони отримують вхідну базу даних від проектувальників, наповнюють її та обслуговують. Користувачі не мають засобів доступу до управління структурою бази, вони мають доступ тільки до даних, при цьому тільки до тих, робота з якими передбачена на їхньому конкретному робочому місці.[5]

Проектування бази даних починається з вивчення технічного завдання на проектування бази даних, яке повинен надати замовник. Отже, бажано, щоб замовник володів відповідною термінологією і знав, принаймні в загальних рисах, технічні можливості основних СУБД. На жаль, на практиці ці побажання виконуються не завжди. Тому зазвичай розробники використовують такі підходи: демонструють замовникові роботу аналогічної бази даних, після чого узгоджують специфікацію

відмінностей; якщо аналога немає, з'ясовують коло задач і вимог замовника, після чого допомагають йому підготувати технічне завдання. Під час підготовки технічного завдання складають: перелік вхідних даних, з якими працює замовник; перелік вихідних даних, потрібних замовникові для управління структурою свого підприємства; перелік вихідних даних, які не є необхідними для замовника, але які він повинен надати іншим організаціям (у вищестоящі структури, в органи статистики, інші адміністративні і контрольні організації).

Визначивши основну частину даних, які замовник використовує, розпочинають розробку структури бази, тобто структури її основних таблиць.

1. Робота починається з визначення генерального переліку полів, який може нараховувати десятки і сотні позицій.

2. Відповідно до типу даних, що розміщуються в кожному полі, визначають тип кожного поля.

3. Розподіляють поля генерального списку по базових таблицях. На першому етапі розподіл здійснюють за функціональною ознакою. Мета - забезпечити одноразове введення даних в одну таблицю по можливості в рамках одного підрозділу, або (ще краще) - на одному робочому місці. На другому етапі розподілу полів здійснюють нормалізацію даних з метою вилучення повторів даних у таблицях бази даних.

4. Для кожної таблиці визначають ключове поле. Ключовим вибирають поле, дані в якому повторюватись не можуть.

2.2 Аналіз принципів розробки бази даних

До сучасних баз даних, на яких вони будуються, пред'являються наступні основні вимоги.

1. Висока швидкодія (малий час відгуку на запит).

Час відгуку - проміжок часу від моменту запиту до БД до фактичного отримання даних. Схожим є термін час доступу - проміжок часу між видачею команди запису (зчитування) і фактичним отриманням даних. Під доступом розуміється операція пошуку, читання даних або запису їх. Часто операції запису, видалення та модифікації даних називають операцією оновлення.

2. Простота оновлення даних.

3. Незалежність даних.

4. Спільне використання даних багатьма користувачами.

5. Безпека даних - захист даних від навмисного чи не навмисного порушення секретності, спотворення або руйнування.

6. Стандартизація побудови та експлуатації БД (фактично СУБД).

7. Адекватність відображення даних відповідної предметної області.

8. Доброзичливий інтерфейс користувача.

Найважливішими є перші два суперечливих вимоги: підвищення швидкодії вимагає спрощення структури БД, що, у свою чергу, ускладнює процедуру оновлення даних, збільшує їх надмірність.

Незалежність даних - можливість зміни логічної та фізичної структури БД без зміни уявлень користувачів.[12]

Незалежність даних передбачає інваріантність до характеру зберігання даних, програмного забезпечення і технічних засобів.Безпека даних включає їх цілісність і захист.

2.3 Етапи проектування баз даних

Проектування баз даних відбувається в чотири етапи.

На етапі формулювання й аналізу вимог встановлюються цілі організації, визначаються вимоги до БД. Вони складаються з загальних вимог, визначених у розділі 1, і специфічних вимог. Для формування специфічних вимог зазвичай

використовується методика інтерв'ювання персоналу різних рівнів управління. Всі вимоги документуються у формі, доступній кінцевому користувачу і проектувальнику БД.

Етап концептуального проектування полягає в описі і синтезі інформаційних вимог користувачів у початковий проект БД.Вихідними даними можуть бути сукупність документів користувача при класичному підході або алгоритми додатків (алгоритми бізнесу) при сучасному підході. Результатом цього етапу є високорівневе подання (у вигляді системи таблиць БД) інформаційних вимог користувачів на основі різних підходів.

Спочатку вибирається модель БД. Потім створюється структура БД, яка заповнюється даними за допомогою систем меню, екранних форм або в режимі перегляду таблиць БД. Тут же забезпечується захист і цілісність (у тому числі посилальна) даних за допомогою СУБД або шляхом побудови тригерів.

У процесі логічного проектування високорівневе подання даних у структуру використовуваної СУБД. Основною метою етапу є усунення надмірності даних з використанням спеціальних правил нормалізації. Ціль нормалізації - мінімізувати повторення даних і можливі структурні зміни БД при процедури оновлення. Це досягається розділенням (декомпозицією) однієї таблиці в дві або декілька з подальшим використанням при запитах операції навігації. Зауважимо, що навігаційний пошук знижує швидкодію БД, тобто збільшує час відгуку на запит. Отримана логічна структура БД може бути оцінена кількісно за допомогою різних характеристик (число звернень до логічних записів, обсяг даних в кожному додатку, загальний обсяг даних). На основі цих оцінок логічна структура може бути вдосконалена з метою досягнення більшої ефективності.

Спеціального обговорення заслуговує процедура управління БД. Вона найбільш проста в режимі одного користувача. У многопользовательском режимі і в розподілених БД процедура сильно ускладнюється. При одночасному доступі декількох користувачів без прийняття спеціальних заходів можливе порушення

цілісності. Для усунення цього явища використовують систему транзакцій і режим блокування таблиць або окремих записів.

Транзакція - процес зміни файлу, запису або бази даних, викликаний передачею одного вхідного повідомлення. Особливості блокування і варіанти блокування далі будуть розглянуті окремо.

На етапі фізичного проектування вирішуються питання, пов'язані з продуктивністю системи, визначаються структури зберігання даних та методи доступу.

Взаємодія між етапами проектування та словникової системою необхідно розглядати окремо. Процедури проектування можуть використовуватися незалежно в разі відсутності словникової системи. Сама словникова система може розглядатися як елемент автоматизації проектування.

Засоби проектування і оціночні критерії використовуються на всіх стадіях розробки. В даний час невизначеність при виборі критеріїв є найбільш слабким місцем у проектуванні БД. Це пов'язано з труднощами опису та ідентифікації великої кількості альтернативних рішень.

Простіше справа при роботі з кількісними критеріями, до яких відносяться час відповіді на запит, вартість модифікації, вартість пам'яті, час на створення, вартість на реорганізацію. Утруднення може викликати протиріччя критеріїв одне одному.

У той же час існує багато критеріїв оптимальності, які є невимірними властивостями, важко виразність у кількісному поданні або у вигляді цільової функції.

До якісних критеріїв можуть ставитися гнучкість, адаптивність, доступність для нових користувачів, сумісність з іншими системами, можливість конвертації в іншу обчислювальне середовище, можливість відновлення, можливість розподілу і розширення.

Процес проектування є тривалим і трудомістким і зазвичай триває кілька місяців. Основними ресурсами проектувальника БД є його власна інтуїція та досвід, тому якість рішення у багатьох випадках може виявитися низькою.[16]

Основними причинами низької ефективності проектованих БД можуть бути:

1. недостатньо глибокий аналіз вимог (початкові етапи проектування), включаючи їх семантику та взаємозв'язок даних;

2. велика тривалість процесу структурування, що робить цей процес стомлюючим і важко виконуваним при ручній обробці.

У цих умовах важливого значення набувають питання автоматизації розробки.

1. Простота оновлення даних.

2. Незалежність даних.

3. Спільне використання даних багатьма користувачами.

4. Безпека даних - захист даних від навмисного чи не навмисного порушення секретності, спотворення або руйнування.

5. Стандартизація побудови та експлуатації БД .

6. Адекватність відображення даних відповідної предметної області.

7. Доброзичливий інтерфейс користувача.

Найважливішими є перші два суперечливих вимоги: підвищення швидкодії вимагає спрощення структури БД, що, у свою чергу, ускладнює процедуру оновлення даних, збільшує їх надмірність.

Незалежність даних передбачає інваріантність до характеру зберігання даних, програмного забезпечення і технічних засобів. Вона забезпечує мінімальні зміни структури БД при змінах стратегії доступу до даних і структури самих вихідних даних. Це досягається «зсувом» всіх змін на етапи концептуального і логічного проектування з мінімальними змінами на етапі фізичного проектування.

Незалежність даних - можливість зміни логічної та фізичної структури БД без зміни уявлень користувачів.

Безпека даних включає їх цілісність і захист.

Цілісність даних - стійкість даних, що зберігаються до руйнування і знищення, пов'язаних з несправностями технічних засобів, системними помилками і помилковими діями користувачів.

Вона передбачає:

1. відсутність неточно введених даних або двох однакових записів про одне й те ж факт;

2. захист від помилок при оновленні БД;

3. неможливість видалення (або каскадне видалення) пов'язаних даних різних таблиць;

4. не псування даних при роботі в багатокористувацькому режимі і в розподілених базах даних;

5. збереження даних при збої техніки (відновлення даних).

2.4 Розробка архітектури та алгоритму роботи методів та програмних засобів обміну даних між різноплатформеними базами даних

Архітектура методів та програмних засобів обміну даних між різноплатформеними базами даних складається з системи "клієнт-сервер"(див. рис.2.1) - від клієнта до сервера баз даних передається запит на обробку бази даних, а від сервера до клієнта - релевантна запитом вибірка даних.

Тому спочатку формується адміністративна частина для клієнського зв’язку з базами даних, для отримання інформації.

Істотно скорочується трафік мережі, а коригування бази даних не вимагає монопольного доступу, внаслідок цього росте продуктивність системи. У мережному варіанті можливе як централізоване, так і розподілене зберігання бази даних.

Дані в базах даних зберігаються в структурованому вигляді. Структура даних - схема організації полів усередині запису і записів в масив (файл,дерево, таблицю тощо). Структури даних розглядаються на логічному і фізичному рівні.

Рисунок 2.1 - Система “Клієнт - сервер”.

Структурні одиниці даних логічного рівня:

1. Поле - елементарне дане (item, attribute), характеризується ім'ям (ідентифікатор поля), типом даних, ознакою обов'язкового значення й ін.

2. Запис (record) - сукупність логічно пов'язаних полів. Примірник записи - окрема реалізація записи, містить конкретні значення полів. Для ідентифікації окремого примірника запису служить ключ - одне або декілька полів. Запис має лінійну або не лінійну структуру (ієрархія, мережа).

3. Масив - сукупність екземплярів записів. Фізична запис - елементарна одиниця даних, яка може бути зчитана або записана на пристрій зберігання даних однією командою вводу-виводу.

Фізичні записи (блок або сектор) можуть об'єднувати кілька логічних записів - для них визначається порядок розташування, спосіб підтримки зв'язків між ними. Основними способами організації та методами доступу до даних є: послідовна, списковую (строкова) і довільна.

Спискова структура створюється за допомогою службової інформації у вигляді покажчиків (адреса або ключ запису), за допомогою яких підтримується потрібна послідовність записів. Покажчики зберігаються або разом із записами, або в індексному файлі. Різновидом списковому структури є мультисписок, який містить кілька покажчиків для створення різних списків із записів одного файлу. Після

створення структури реляційної бази даних, потрібно створити не реляційну базу даних, в якій буде зберігатись основна інформація.

Дані, які ми отримаємо з структурної бази даних, одразу формуємо запит на не реляційну базу даних.

Не реляційна база даних, приймає id, тих елементів, які потрібні користувачу. Оскільки всі дані в не реляційній базі даних, зберігаються під власними id. Блок-схема методу обміну даних між різноплатформеними базами даних. Взаємодія користувача з методом обміну даних між різноплатформеними базами даних наведена рисунку 2.2.

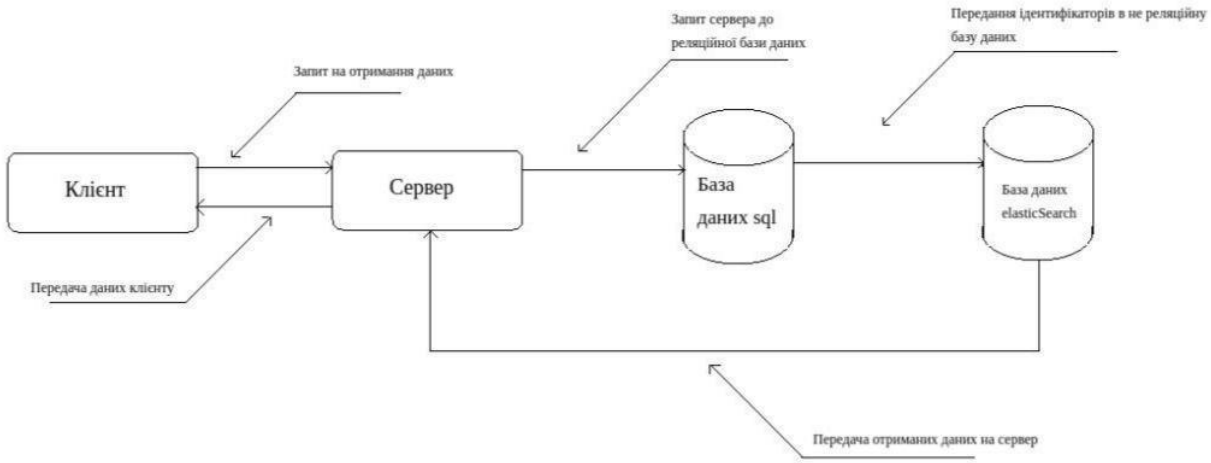


Рисунок 2.2 - Система “Клієнт - сервер”.

2.4 Висновки

У розділі проаналізовано структури реляційних та не реляційних баз даних, принципи розробки та особливості розробки бази даних. Також проаналізовано їхню архітектуру. Здійснено розробку структури баз даних та наведено блок-схему алгоритму блоку взаємодії користувача з різноплатформеною базою даних. Запити POST трохи складніші, але все одно зрозумілі і логічні.

На цей раз ви будете використовувати свій власний запит. Натисніть на плюсик, щоб відкрити нову вкладку, замініть тип запиту з GET на POST, і використовуйте https://jsonplaceholder.typicode.com/posts як URL запиту.

Тепер вам потрібно тіло POST-запиту. Натисніть на "Body" під URL запиту,змінити тип на "raw" і "Text" на "JSON". І на рисунку 1.5 зображено текст який потрібно вставити.

3 РОЗРОБКА РІЗНОПЛАТФОРМЕНИХ БАЗ ДАНИХ

3.1 Системний аналіз бази даних

Для того щоб продемонструвати зв'язок реляційної бази даних з не реляційною, було вирішено створити MySql базу даних з таблицями ідентифікаторів автотоварів, потенційних клієнтів, характеристики товарів, автомобілів для цих товарів, замовлень. А більше детальна характеристика, опис, будуть зберігатись в не реляційній базі даних ElasticSearch.

Я використав функціональний підхід до системного аналізу адже функціональний підхід застосовує рух «від задач» і використовується у тих випадках, коли заздалегідь відомі функції майбутніх користувачів БД, а також відомі всі задачі, для інформаційних потреб яких створюються БД.

Діаграма варіантів використання бази даних «Автотовари» відображає «представлення та доставка товару замовнику, підрахунок доставки». Наведу опис потоків, задач діаграми і послідовності виконання.

Це всього одна з багатьох функцій яка буде виконуватися в базі даних. Функції які будуть виконуватися в діаграмі: можливість отримання даних про товар,всі його характеристики, можливість вибирати декілька товарів.

Вихідний документ, буде формуватися у файлі бази даних: каталог товарів - буде містити усі доступні товари на даний час, термін гарантії на товар, ціну товару; список компаній які постачають товари - буде містити вид товару який постачають, його ціну, гарантійний термін.

Користувач, вибирає потрібні товари, або групи товарів з бази даних. Далі ідентифікатори товарів зберігаються у JSON форматі в спеціальній таблиці MySql, також записується час та статус.

Після цього розпочинається генерація файлу. Cron запускає скрипт, який вибирає JSON стрічку, декодує її, і створює запит у ElasticSearch, по ідентифікаторам,

які були вибрані. Далі отримані дані записуються в файл, який записується в архів. Посилання на файл, зберігається ще в одній таблиці.

Після вибору потрібних товарів, файл пройду генерацію і буде доступний для скачування у .zip форматі, всередині якого буде знаходитись xml файл.

3.2 Створення бази даних в MySQL та в ElasticSearch

Семантичне моделювання представляє собою моделювання структури даних, спираючись на зміст цих даних. В якості інструменту семантичного моделювання Використовують різні варіанти діаграм суть-зв'язок (ER - Entity - Relationship).

Суть - це клас однотипних об'єктів, інформація про які повинна бути врахована в моделі. Кожна суть винна мати найменування, вираженість іменником в однині. Прикладами суті можуть бути такі класи об'єктів як «Постачальник», «Співробітник«, «Товар».

Атрибут суті - це іменована характеристика, что є деякою властивістю суті. Найменування атрибуту має бути вираженою іменником в однині (можливо прикметник). Прикладами атрибутів суті «Товар» можуть бути Такі атрибути як «Марка», «Модель», «Наявність» і т.п.

Ключ суті - це набір атрибутів, значення яких в сукупності є унікальними для кожного екземпляру суті. Чи не надлишковість полягає в тому, что видалення будь-которого атрибуту з ключа порушує его унікальність. Суть може мати кілька різних ключів.

Зв'язок - це деяка асоціація між двома сутностями. Одна суть може бути пов'язана з іншою сутністю або сама з собою. Зв'язки дозволяють за допомогою однієї суті знаходити інші, пов'язані з нею.

Кожний зв'язок має два кінця і одне або два найменування. Найменування зазвичай виражається в невизначеній дієслівної формі: «мати», «належати» и т.п. Кожне з найменувань відноситься до свого кінця зв'язку. Іноді найменування не пишуть зважаючи на їх очевидність.

Зв'язок типу один-до-одного означає, что один екземпляр першої суті пов'язаний з одним примірником Другої суті. Зв'язок один-до-одного найчастіше свідчить про те, що насправді існує Всього одна суть, яка неправильно розділена на две .

Зв'язок типу один-до-багатьох означає, что один екземпляр першої суті пов'язаний з декількома екземпляр другої суті. Це найбільш часто використовуваний тип зв'язку. Ліва суть (з боку «один») називається батьківською, права (з боку «багато») - дочірньою.

Зв'язок типу багато-до-багатьох означає, что КОЖЕН екземпляр Першої суті може бути пов'язаний з декількома екземпляр Другої суті, и КОЖЕН примірник Другої суті може бути пов'язаний з декількома екземпляр Першої суті. Тип зв'язку багато-до-багатьох є Тимчасовим типом зв'язку, допустимі на ранніх етапах розробки моделі. Надалі цією тип зв'язку повинен бути чинний двома зв'язками типу один-до-багатьох Шляхом создания проміжної суті.

Для кожної суті вибрато атрибут чи сукупність атрибутів, что ідентифікують її однозначно і виступають ключем відповідного відношення.

Спочатку реалізувалося створення таблиць в SQL Manager Lite for MySQL. Створення таблиць, можна зробити як завдяки коду, або завдяки SQL Manager Lite for, який надає можливість створювати таблиці. Я вирішив створювати таблиці завдяки коду, тому що вважаю цей спосіб більш зручнішим.

Приклад створення таблиці, в якій буде зберігатись:

1. статус генерації файлу, буде змінюватись відносно того, в якому статусі знаходиться файл, або виникли помилки.
2. стрічка в форматі json, в якій записано унікальні ідентифікатори вибраних категорій.

3. Дата створення файлу.

Код створення даної таблиці наведено на рисунку 3.1.

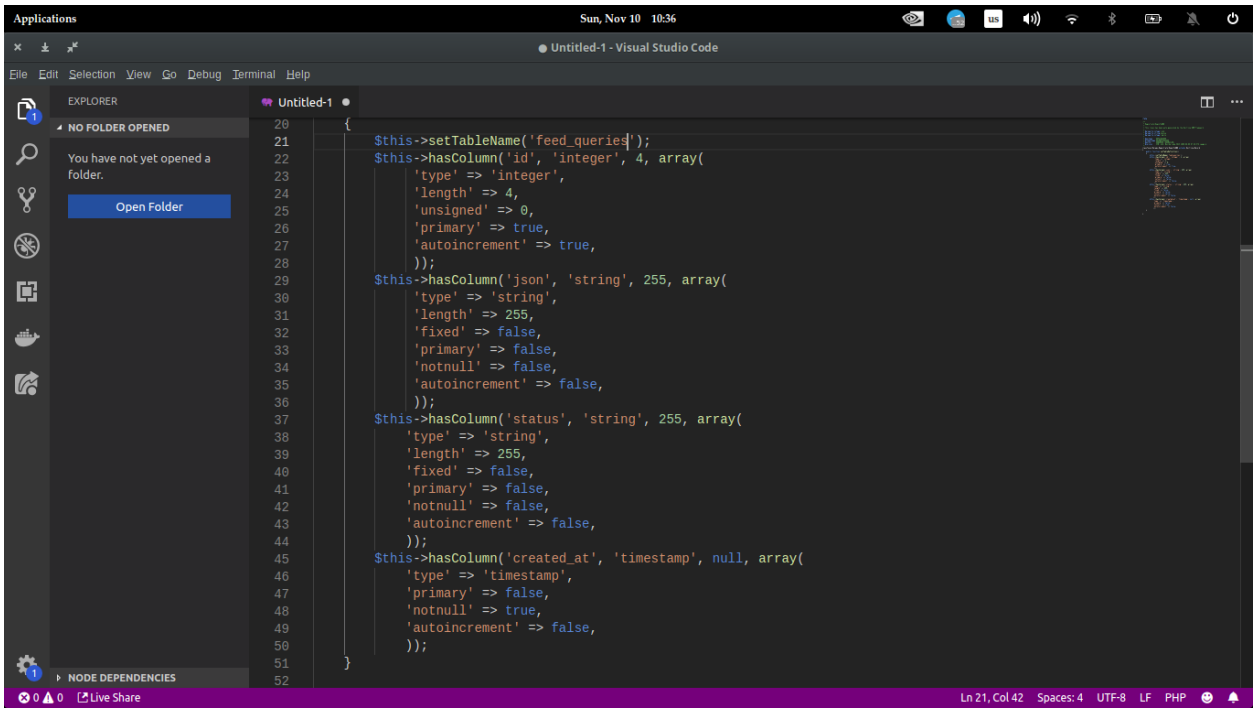


Рисунок 3.1 – Створення таблиці в мові програмування PHP.

Далі, за аналогічним принципом ми створюємо таблиці категорій, моделей, марки. Назви їх будуть зберігатись в MySQL, а більш детальна інформація буде зберігатись у не реляційній базі ElasticSearch.

Для того щоб створити товари та їх характеристики у ElasticSearch, я написав скрипт. Принципом роботи скрипта, було те, що він брав масиви, проходився по елементам і записував у POST-запит, який відправлявся у ElasticSearch. Скрипт був підключений для запуску за допомогою cron. І кожне оновлення даних автоматично записувалось в базу. Код скрипта показано на рисунку 3.2.

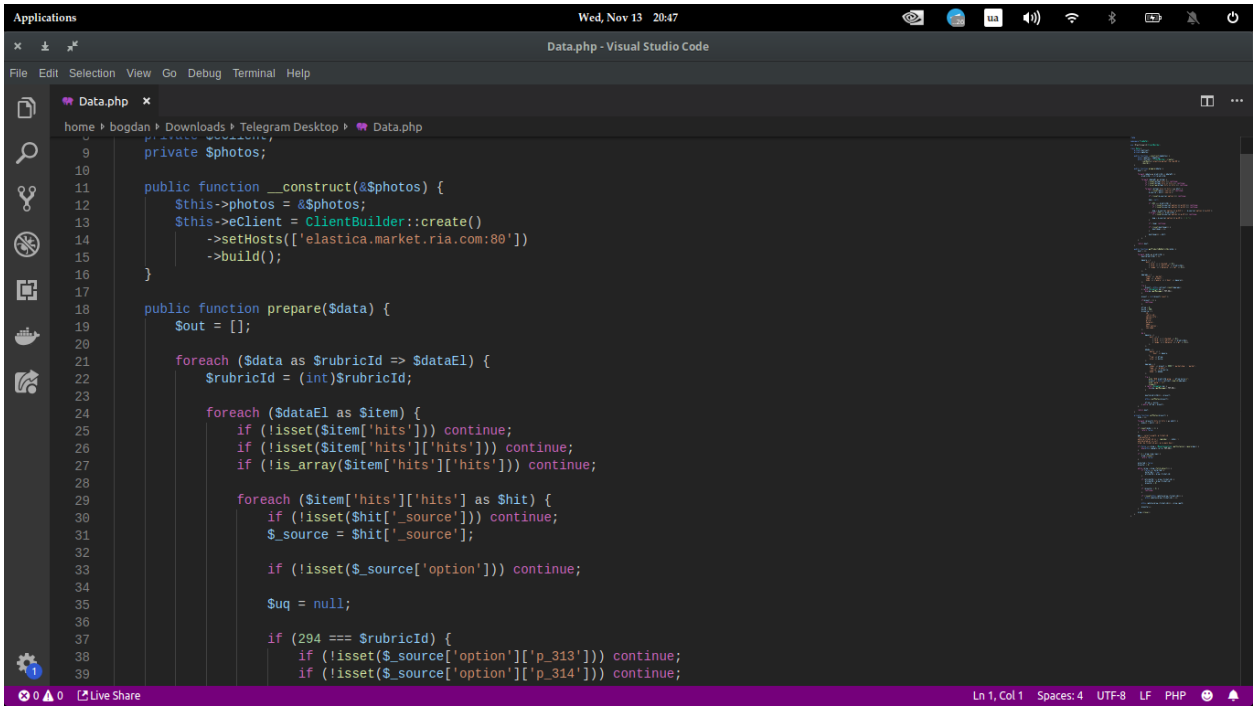


Рисунок 3.2 - Код скрипта на запис даних у ElasticSearch.

Оскільки не реляційна база даних є об'єктно орієнтованою, складності в записі не виникло. Також для кожного об'єкта було створено властивість "option_info". Це було створено для кращого пошуку характеристик товару. Після виконання скрипта, я отримав базу автотоварів, з унікальними характеристиками. Після цього, можна створювати обмін даними між базами. Вигляд одного об'єкта в базі показано на рисунку 3.3. Об'єкти зручна річ, особливо для характеристики товарів, оскільки різні товари, мають різні характеристики. Ми маємо змогу додавати в певний товар, особливі характеристики.

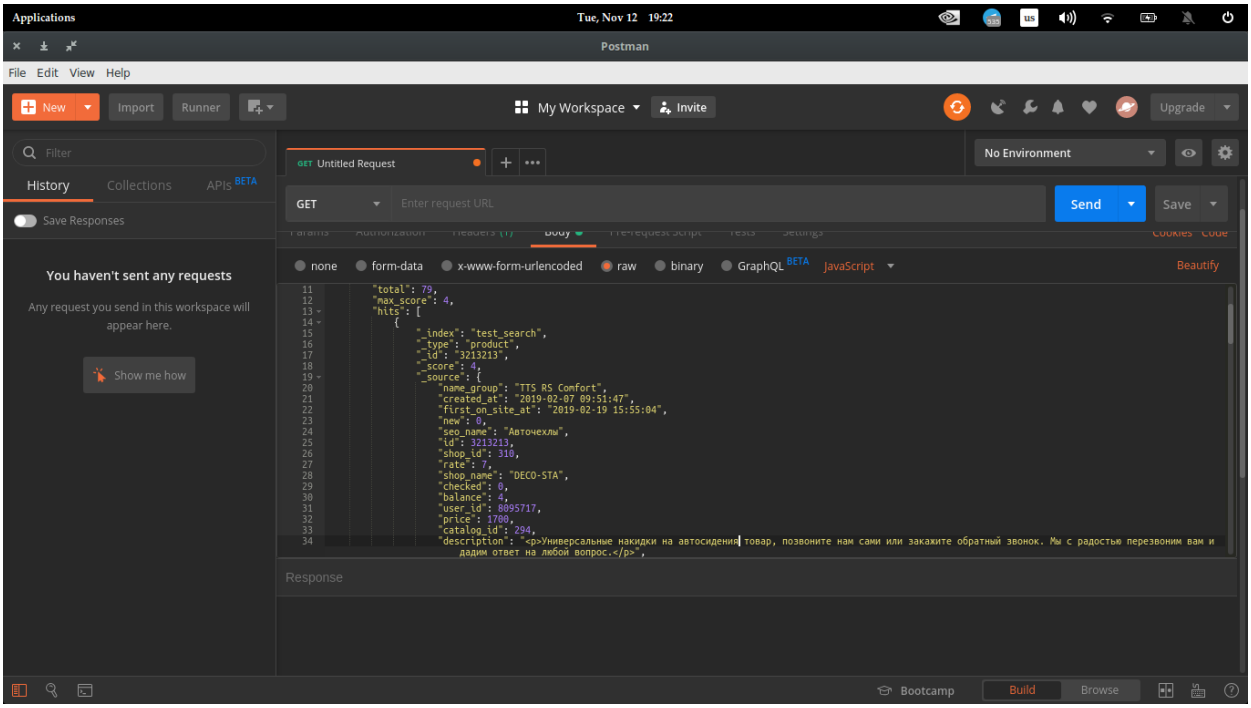


Рисунок 3.3 - Отримання одного об'єкта в Postman.

Одним з позитивних сторін Postman, те, що він безкоштовний. Ми маємо змогу зберігати наші запити, щоб в майбутньому швидко розробляти певні зміни.

3.3 Створення адміністративної частини

Для того, щоб продемонструвати методи обміну даних між різноплатформеними базами даних було створено сторінку адміністрування. На даній сторінці розміщується 4 селектора. За їх допомогою ми вибираємо товари, характеристики яких, ми хочемо отримати. Якщо потрібно більше одного товару, в правій частині екрану було створено кнопку, з її допомогою можна додати ще одне поле з селекторами(рис. 3.4). Якщо перший селектор не був використаний, решта будуть заблоковані, і поетапно розблоковуватись, коли користувач буде обирати характеристики.

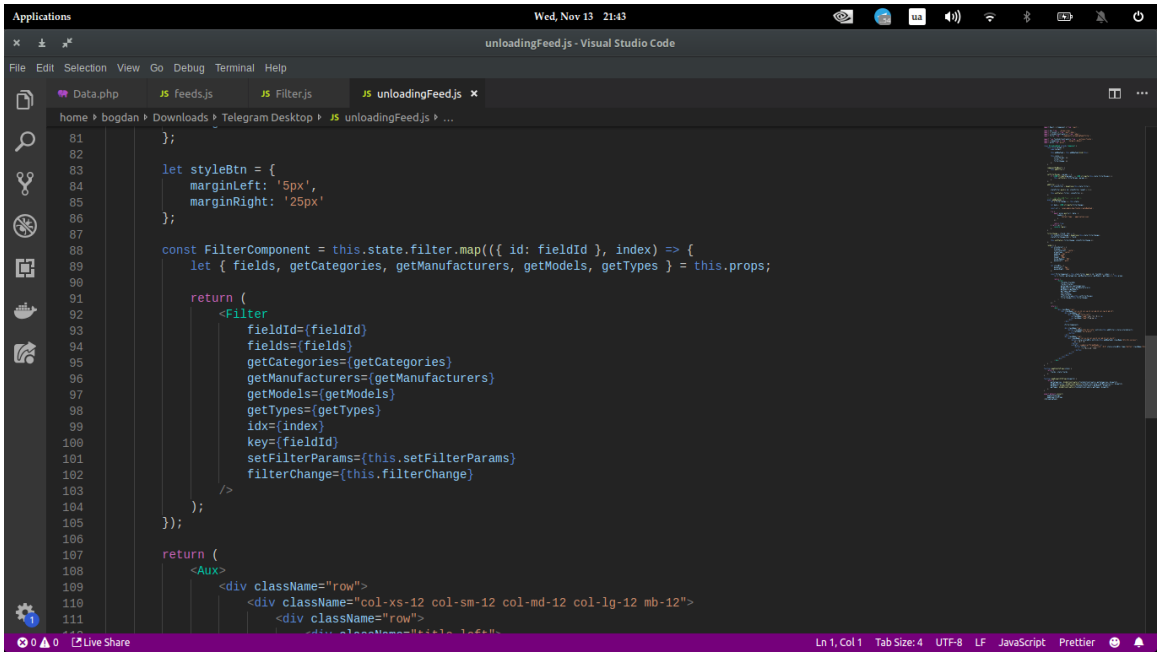


Рисунок 3.4 - Компонент відображення сторінки.

Перший селектор відображає категорії товарів. Другий показує виробника, по вибраній категорії в першому селекторі. Третіл селектор дає можливість вибрати модель, якщо потрібно скачати дані по певному товарі. Четвертий селектор опціональний, якщо користувач вибрав категорію “Шини”, то з'являється можливість вибрати сезон шин. Додано три кнопки. В правій частині екрану кнопка, яка додає ще один селектор. Нижче селекторів розташовуються дві кнопки. “Всі згенеровані фіди”, яка направляє користувача на сторінку де він має змогу скачати фід та “Згенерувати фід”, вона спочатку записує ідентифікатори товарів в реляційну базу даних та запускає скрипт, який починає генерацію фіда.

Користувач може вибрати перший селектор і натиснути на кнопку “Згенерувати фід”. Після цього будуть вибрані всі товари, які знаходяться в даній категорії.

Front-end частина даної сторінки була створена за допомогою бібліотеки “React” та “Redux”. Це найпопулярніша бібліотека, на якій написана велика кількість популярних сайтів.

Також була застосовано язик розмітки “CSS”. Код даного відображення показано на рисунку 3.5.

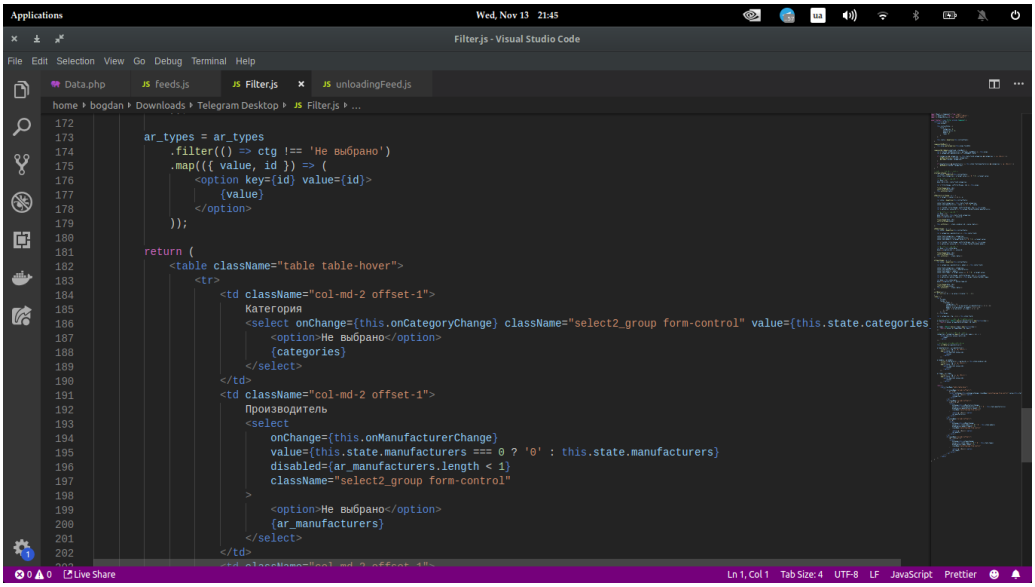


Рисунок 3.5 - Компонент відображення селектору.

Для отримання даних з реляційної бази даних, я використовував фреймворк “Node.js”. За допомогою роутів, я відправляв на back-end дані, і робив запит у базу даних, після чого відправляв потрібні дані назад. Після чого користувач бачив нові дані. Дана сторінка використовує односторінковий інтерфейс, тому все працює без перезавантаження сторінки.

При роботі з селекторами, було застосовано архітектурний шаблон “Model-view-controller”. На рисунку 3.6 продемонстровано контроллер, який зв'язує front-end частину та базу даних. Оскільки є декілька запитів до бази, то було створено декілька асинхронних функцій. Функції асинхронні, тому що дані повертаються не синхронно, і ми не впевнені, що дані зразу будуть доступні. Якщо б ми написали синхрону функцію, ми б мали великі проблеми. Контроллер дізнається про те, яку саме функцію виконувати, завдяки роутам. На реакті, ми записуємо шлях, по якому передаємо дані, на контролері ми їх отримуємо, і передаємо в модель.

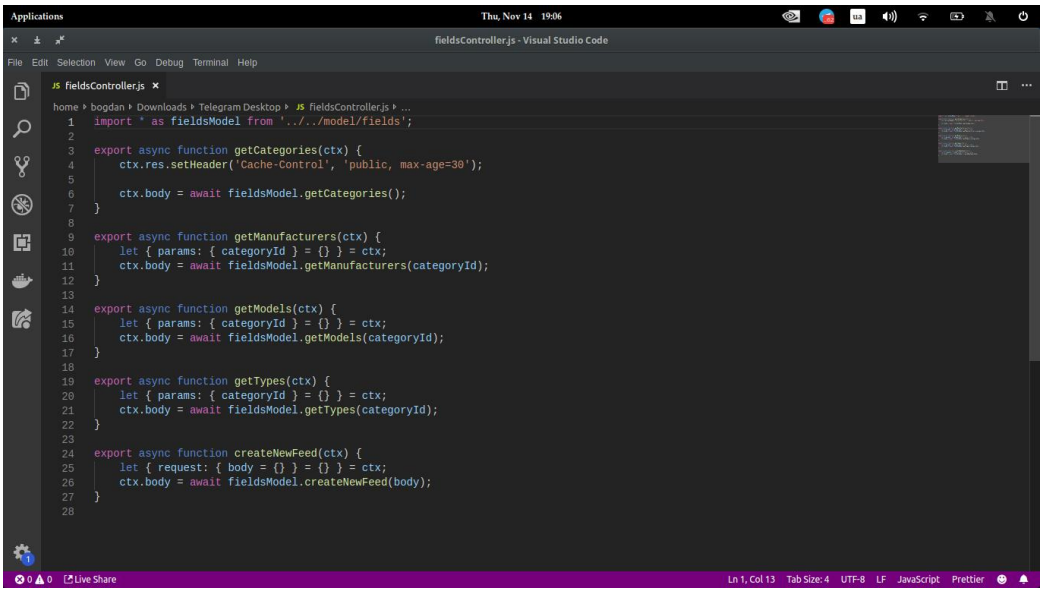


Рисунок 3.6 - Контроллер для поєднання з моделлю.

Після того, коли дані потрапляють в модель, ми перевіряємо їх наявність, оскільки, якщо вони будуть відсутні, програма працювати не буде. Після валідації даних, ми робимо запит у базу, отримуємо відповідь, перевіряємо її наявність і віддаємо назад у контроллер, який поверне дані на front-end частину сайта.

Для того, щоб було уникнути помилок, під час деструктуризації, було додано параметр по-замовчуванню. Якщо дані не були отримані, ми просто підставляємо пустий об'єкт, який нічого не поверне. І програма далі буде функціонувати без проблем.

Модель на яку посилається контроллер, зображена на рисунку 3.7.

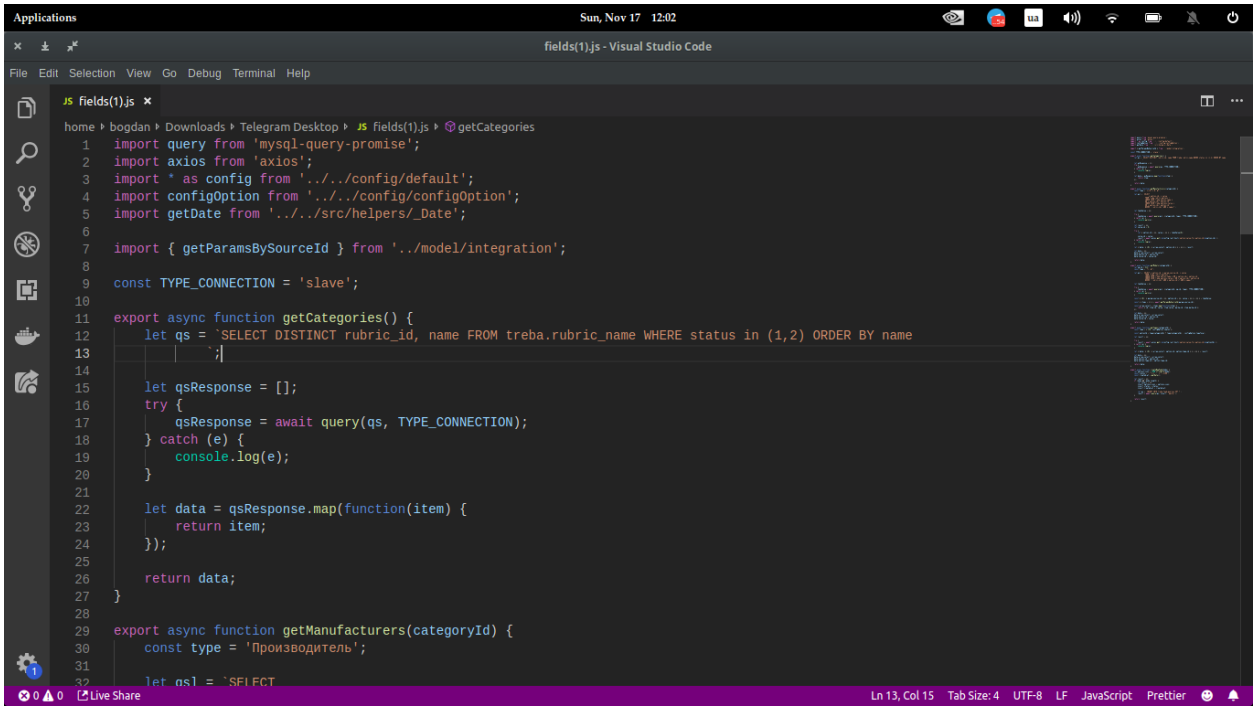


Рисунок 3.7 - Модель яка виконує запити в базу даних.

Кожна функція в файлі моделі відповідає за один лише селектор. Кожна наступна наслідує ідентифікатор попередньої, і робить запит на базу даних. Також застосована конструкція “try...catch”. Конструкція try містить блок try, в якому знаходиться одна або декілька інструкцій ({} має бути завжди використано, навіть для одиночних інструкцій), і як мінімум один блок catch. Блок catch містить інструкції, які будуть виконані, якщо в блоці try сталася помилка. Це зроблено для того, щоб була можливість обробити помилку в блоці catch, при її виникненні. Якщо будь-яка інструкція викликає помилку в try блоці, то управління негайно переходить до блоку catch. Якщо в try блоці не буде ніякої помилки, то блок catch пропустити.

Всі налаштування бази даних, знаходяться в спеціальному файлі. За допомогою імпорту даних, ми можемо отримати дані з іншого файлу. Цим ми покращуємо редагування коду, оскільки чим більший файл, тим складніше ним керувати в майбутньому.

3.4 Розробка скрипта генерації фіда

Після того, як було створено вибірку товарів, потрібно створити кнопку, після натиску на яку, створювався фід, в який записувалась б вся інформація.

Для того, щоб це зробити, потрібно додати подію “OnClick”, на кнопку. Метод який реагує на цю подію - асинхронний. Це було зроблено оскільки ми працюємо з REST-запитом, і нам повертається “Promise”.

Інтерфейс Promise (промис) являє собою обгортку для значення, невідомого на момент створення Проміс. Він дозволяє обробляти результати асинхронних операцій так, як якщо б вони були синхронними: замість кінцевого результату асинхронного методу повертається обіцянку (промис) отримати результат в певний момент в майбутньому.

Promise може перебувати в трьох станах:

1. очікування (pending): початковий стан, не виконаний і не відхилений.
2. виконано (fulfilled): операція завершена успішно.
3. відхилено (rejected): операція завершена з помилкою.

При створенні промис знаходиться в очікуванні (pending), а потім може стати виконаним (fulfilled), повернувши отриманий результат (значення), або відхиленим (rejected), повернувши причину відмови. У будь-якому з цих випадків викликається обробник, прикріплений до Проміс методом then.

Далі дані записуються в базу даних, в спеціальну таблицю (рис.3.8). Для того, щоб записати у базу дані, було створено ще одну функцію в моделі. В ній ми приймаємо дані, перекодуємо їх у JSON формат. Записуємо статус, що інформація в процесі генерації, і одину пусту таблицю, для того, щоб записувати місцезнаходження файлу після генерації. Також додається дата створення даного фіда.

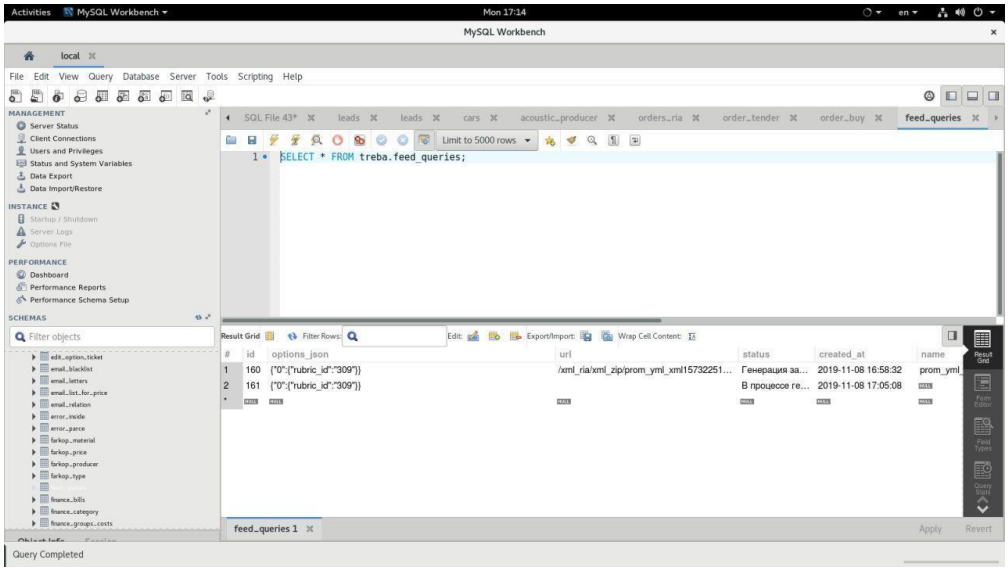


Рисунок 3.8 -Таблиця запису фідів.

Перед тим як записати у базу, ми перевіряємо наявність даних, щоб не отримати помилки. Якщо дані присутні, ми можемо записати їх у базу. На рисунку 3.9 зображено запит на запис у базу даних інформації.

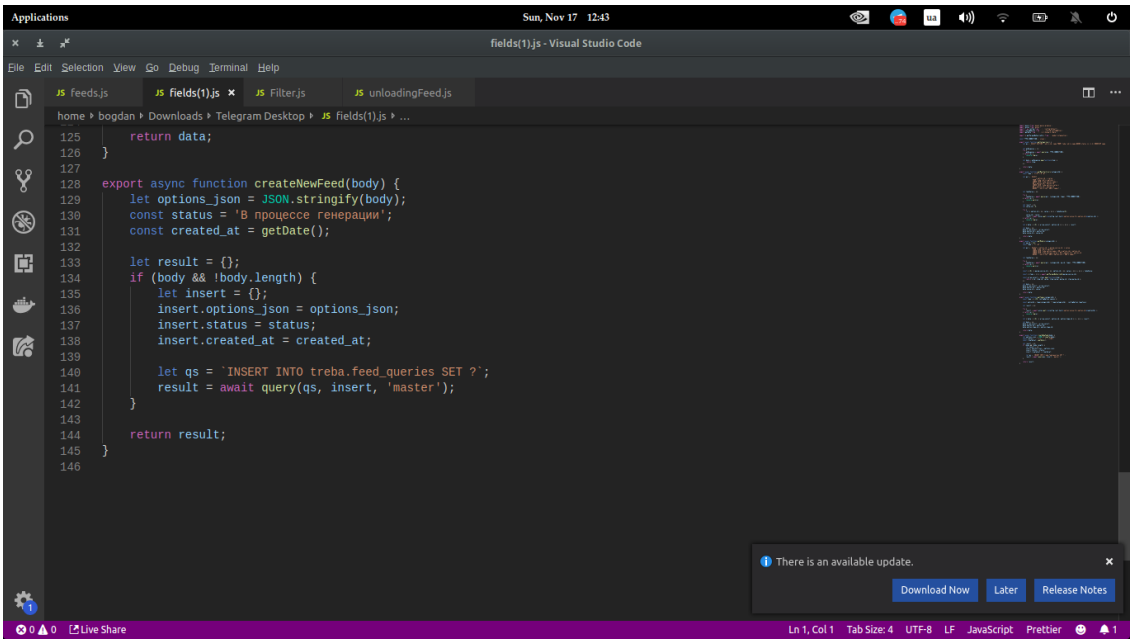


Рисунок 3.9 - Запис даних про товар у базу даних.

Далі було створено PHP-скрипт, який виконувався кожних 10 хвилин, завдяки утиліті cron. Cron - утиліта в операційних системах Unix и Linux, яка дозволяє користувач Виконувати команди або скрипти (групи команд) автоматично в задану годину.

Логіка даного скрипта, передбачає перевірку в базі даних, які и записували перед цим. Якщо там статус в процесі, і поле місцезнаходження файлу пусте, то ми вибираємо лише поле з ідентифікаторами товарів. Після цього ми декодуємо ці дані в масив, створюємо запит в ElasticSearch. Для цього ми з масиву беремо дані і підставляємо в запит. На рисунку 3.10 показано запит на отримання ідентифікаторів з MySQL бази даних, їх декодування та вставка в запит.

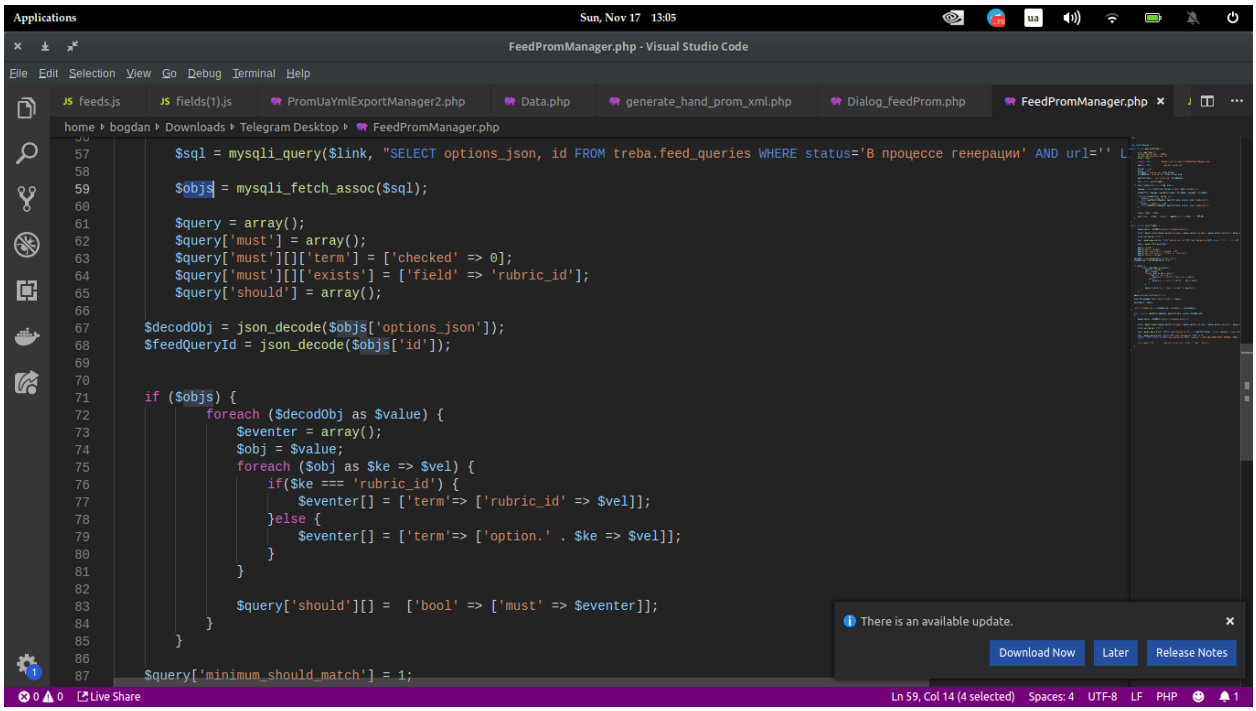


Рисунок 3.10 - Створення запиту в ElasticSearch.

Даний запит, досить гибкий, оскільки він передбачає чи користувач вибрав лише один товар, або декілька. Все відбувається динамічно і досить швидко. Після цього ідентифікатори передаються в другий файл який виконує запит в базу даних.

Даний файл формує загальний запит в базу (рис.3.11). В нього ми передаємо параметр “hande”, який вказує, що ми додаємо в запит дані, які призначені для генерації фіда.

При виконанні скрип перевіряє наявність, якщо даний параметр присутній, ми записуємо в запит ідентифікатори.

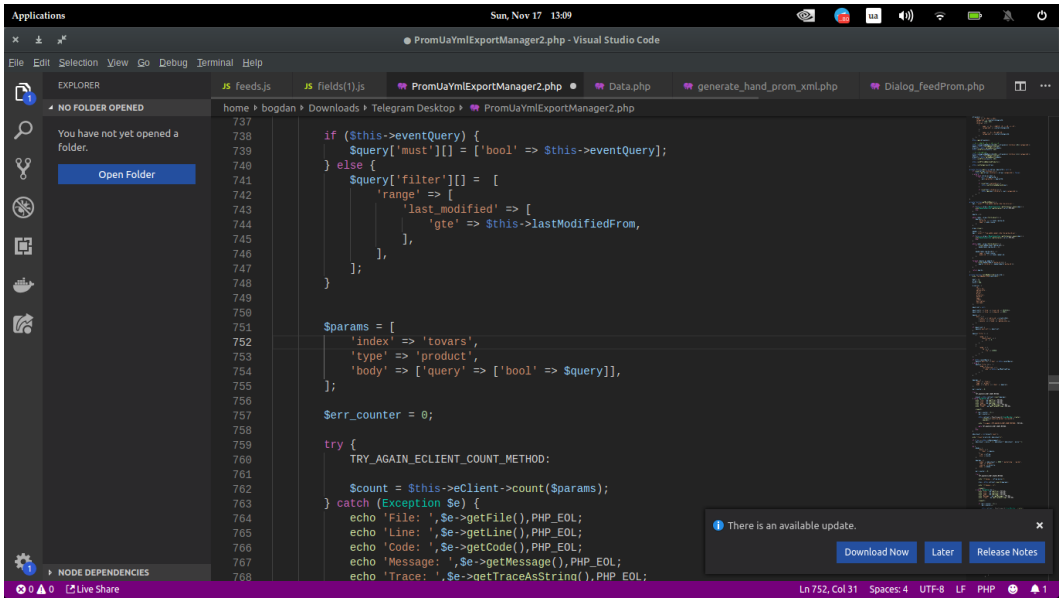


Рисунок 3.11 - виконання запиту в не реляційну базу даних.

Після того, як запит виконався, генерація пройшла успішно, ми отримуємо всі дані в змінну “manager”. Для того щоб дані записати в файл та архів, ми використовуємо метод класа “saveXmlFile”, код даного методу показано на рисунку 3.12.

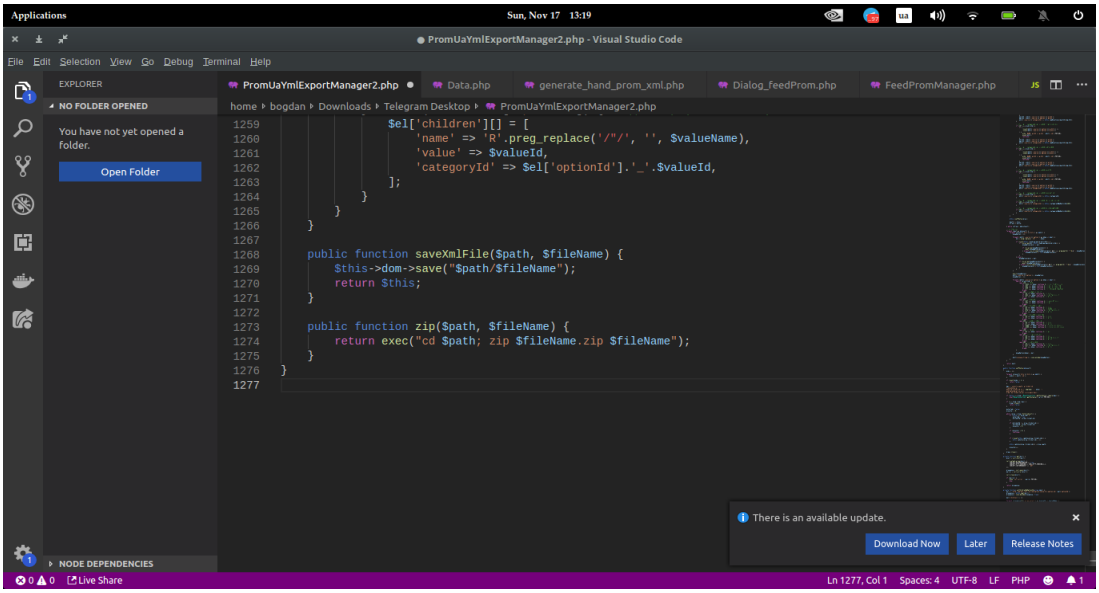


Рисунок 3.12 - Метод, який записує дані у XML файл.

Після запису в змінну, нам повертається запис в якому сказано чи фід згенерувався чи ні. Ми перевіряємо це, якщо помилок не виявлено, визиваємо метод класу “updtDb”. В нього передаємо шлях до файлу, статус, назву, і ідентифікатор (рис. 3.13).

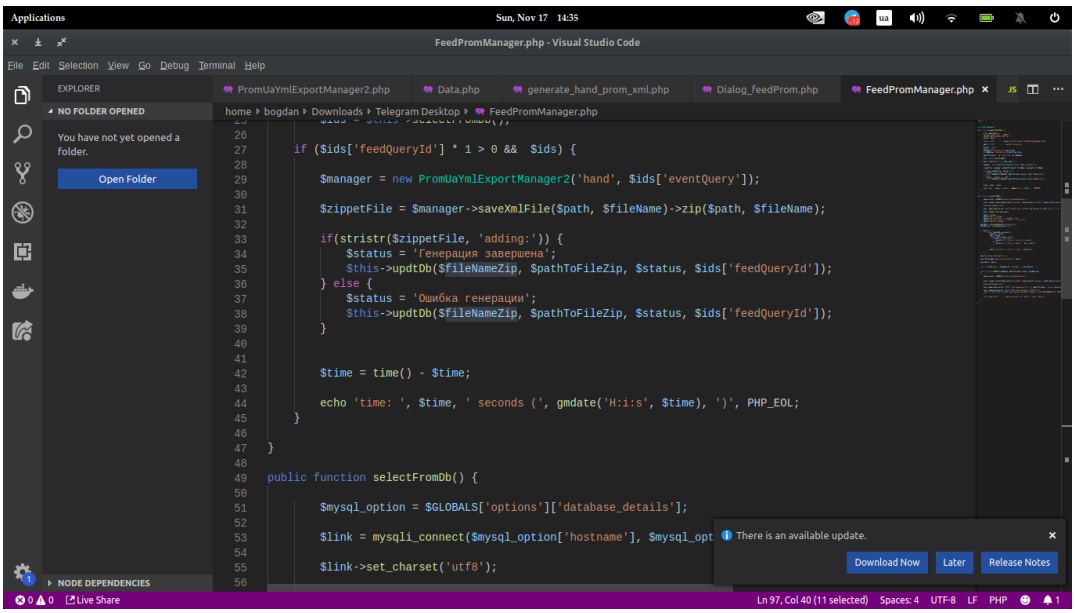


Рисунок 3.13 - Виклик методу, при успішній генерації.

Метод “updtDb” підключається до нашої реляційної бази і по ідентифікатору перезаписує дані в записі (рис.3.14). Шлях до файлу, завдяки якому, потім, ми будемо мати змогу скачувати файл.

Для того щоб зберегти память, в базі даних і на сервері. Після того як ми обновили дані, було написано ще один запит. Він видаляв запис, після 7 днів з моменту його створення. Також була написана “bash” команда, яка видаляла файл з сервера.

Завдяки цьому ми можемо не переживати, що файлів буде занадто багато, і сервер не буде перезавантажений непотрібними файлами.

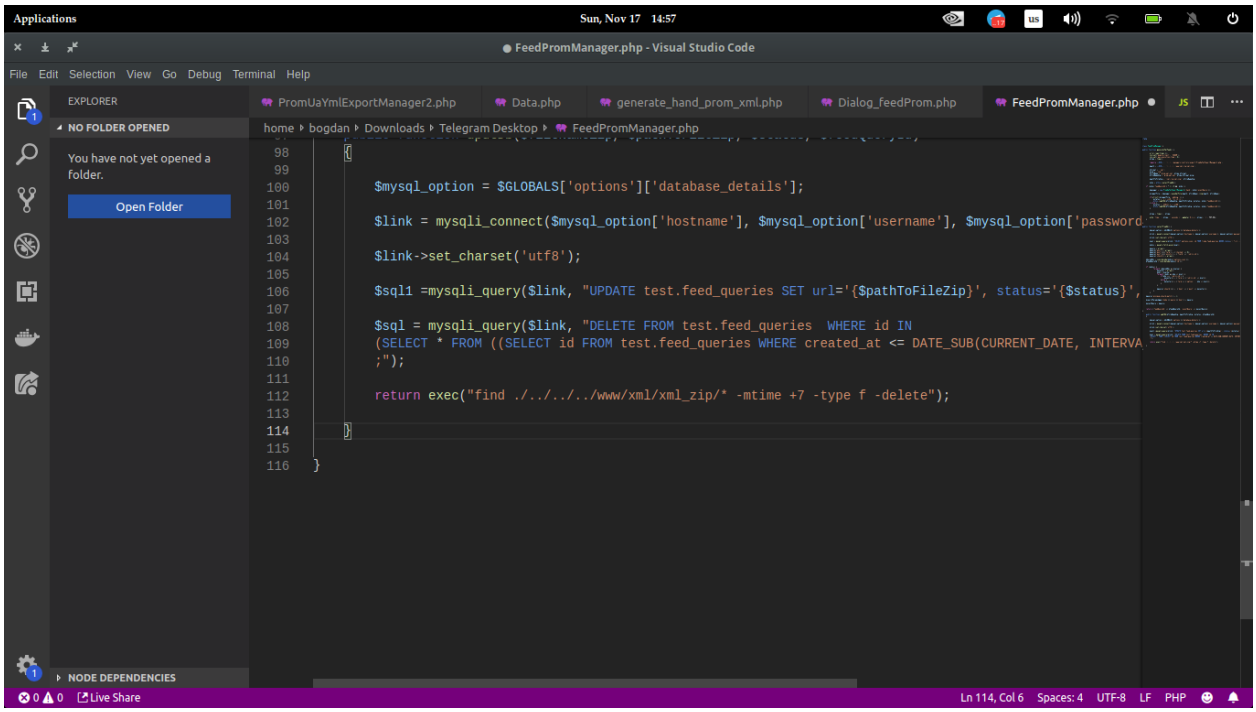


Рисунок 3.14 - Оновлення інформації в базі даних.

Після цього, було вирішено створити сторінку, на якій користувач мав б змогу переглядати статуси генерації і скачувати фіди.

Кнопку для переходу, було розташовано поряд з тією, що генерує фід. Для користувача таке розташування досить зручне, і інтуїтивно зрозуміле.

Для цього було створено контейнер “allFormedFeeds” (рис. 3.15), який списком демонструє назву файлу, статус, дату створення, і якщо фід згенерувався успішно, виводить кнопку скачування, дату створення, назву файлу. Все це реалізовано в односторінкованому інтерфейсі, з його допомогою веб-юзабіліті знаходиться на високому рівні.

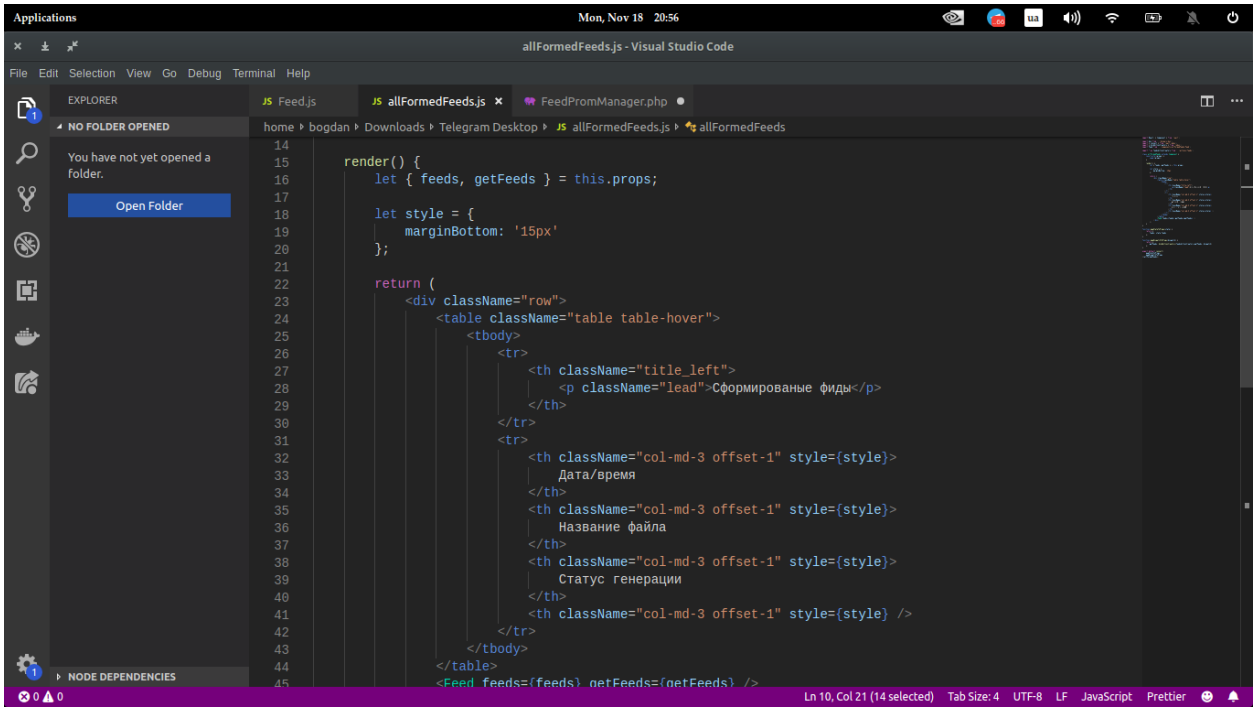


Рисунок 3.15 - Контейнер “allFormedFeeds”.

Для того щоб вивести всі фіди, по прикладу їхнього вибору, ми звертаємось по роуту до Node.js.

Викликається контроллер, яка звертається до моделі(рис. 3.16). В ньому ми звертаємось до бази даних. І отримуємо всі фіди, які є в нас в таблиці. Повертаємо все на front-end частину сайту.

Після цього користувач має змогу переглядати статуси і можливість скачати фід.

3.5 Висновки

В цьому розділі виконувалась розробка адмін панелі з вигрузки фідів, в ній були використані методи та засоби обміну даних між різноплатформеними базами даних. Під час розробки використовувались найбільш популярні і затребувані мови програмування, фреймворки та бібліотеки. Була створено дві бази даних, одна з яких зберігала ідентифікатори даних, для цього було обрано MySql базу даних. Інша база даних була не реляційною і містила повну інформацію про товар, для цього було обрано ElasticSearch, з пошуковою моделлю даних. Після цього було створено адміністративну панель. Front-end частину створено за допомогою мови програмування JavaScript, а саме бібліотек React та Redux . Для back-end частини використовувалось двы мови програмування: JavaScript з фреймворком Node.js і PHP з фреймворком Phalcon.

В Node.js ми записували ідентифікатори товарів які обрав користувач. А PHP потрібно для зв'язку між базами даних. Спочатку користувач обирав товар, дані записувались в реляційну базу, після натиску кнопки “Згенерувати фід”, програма брала по даті створення ідентифікатори товару і виконувала запит в не реляційну базу даних. Коли дані були отримані, вони автоматично записувались в файл та архів. Якщо користувач перейде на сторінку “Всі згенеровані фіди”, він буде мати змогу скачати архів з фідом.

4 ТЕСТУВАННЯ АДМІНІСТРАТИВНОЇ ПАНЕЛІ

4.1 Тестування, види тестувань

Тестування програмного забезпечення - процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Може оцінюватись:

1. відповідність вимогам, якими керувалися проектувальники та розробники
2. правильна відповідь для усіх можливих вхідних даних
3. виконання функцій за прийнятний час
4. практичність
5. сумісність з програмним забезпеченням та операційними системами
6. відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення багів (помилок або інших дефектів).[6]

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів, так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових

програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

Тестування «білої скриньки» - об'єктом тестування тут є не зовнішня, а внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним. Зазвичай аналізуються керуючі зв'язки елементів, рідше — інформаційні зв'язки. Тестування за принципом «білої скриньки» характеризується ступенем, в якому тести виконують або покривають логіку (вихідний текст) програми.

Тестування «чорної скриньки» - основне місце програми тестів «чорної скриньки» — інтерфейс ПЗ.

Ці тести демонструють:

1. Як виконуються функції програми.
2. Як приймаються вихідні дані.
3. Як виробляються результати.
4. Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе. Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 1010. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми. Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ІТ);

Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його ОТ).

Будь-який спосіб тестування «чорної скриньки» повинен:

Виявити такі вхідні дані, які з високою ймовірністю належать набору IT;

Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.[18]

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

1. Некоректних чи відсутніх функцій;
2. Помилок інтерфейсу;
3. Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;
4. Помилок характеристик (необхідна ємність пам'яті і т. д.);
5. Помилок ініціалізації та завершення.

Подібні категорії помилок способами «білої скриньки» не виявляються.

За ступенем автоматизації:

1. Ручне тестування (manual testing)
2. Автоматизоване тестування (automated testing)
3. Напівавтоматизоване тестування (semiautomated testing)

Ручне тестування (Manual testing) — це процес ручної перевірки програмного забезпечення на помилки. Тестувальник має відігравати роль користувача програми й використовувати властивості програми для знаходження помилок у роботі програми. Тому для перевірки було обрано ручне тестування.[19]

4.2 Ручне тестування

Для того щоб протестувати програму вручну, я перейшов на сторінку вибору товару (рис.3.11).

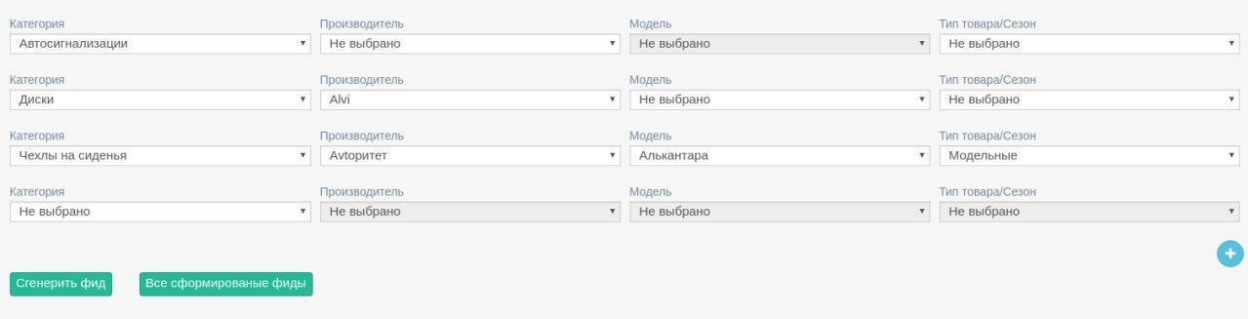


Рисунок 4.1 - Адміністративна панель вибору товару.

Для того, щоб протестувати весь функціонал, в кожному селекторі ми обирали різні товари, і різну вложеність вибору. Далі ми натискаємо кнопку “згенерувати фід”.

Після цього, ми заходимо у базу даних, щоб бути впевненими, що дані збереглись у базі даних (рис.3.12).

На рисунку, показано два фіда в базі. Один з яких згенерований і має посилання на файл в якому зберігається вся інформація. В другого, в полі посилання на файл пусто, оскільки скрипт генерації, ще не виконувався, тому статуси в фідів різні.

Але головне для нас, що ідентифікатори товарів успішно записуються в базу, тому ми можемо продовжити тестування.

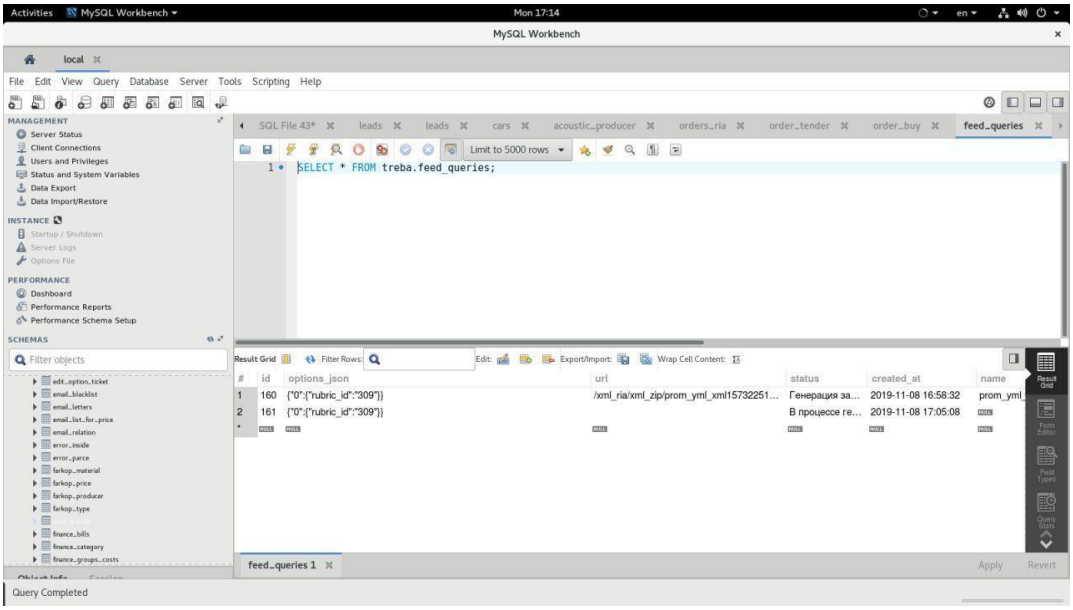


Рисунок 4.2 - База даних після запису даних.

Після того, як ми впевнелись в тому, що дані записуються, ми очікуємо 3 хвилини, щоб скрипт запустився, і розпочалась генерація файлу.

Для того, щоб впевнитись, що файл згенерувався, ми перейдемо на сторінку “Всі згенеровані фіди”. Після десяти хвилин очікування, ми можемо побачити, що наш фід успішно згенерився без помилок, і ми можемо скачати його (рис.4.13).

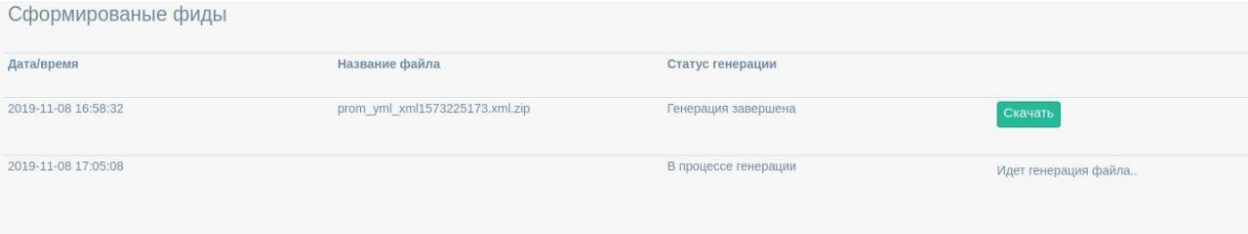


Рисунок 4.2 - Сторінка перегляду фідів.

Оскільки кнопка “Скачати” з'явилась, це може означати те, що фід згенерувався без проблем і ми можемо успішно скачати його.

Як ми бачимо на рисунку 4.3, всі дані записані у відповідні поля. Товари відповідають нашій вибірці.

4.3 Висновки

Отже, в результаті тестування, ми отримали файл з даними, які були обрані. Під час проходження тесту, помилок не виникало, все інтуїтивно зрозуміло. Весь процес автоматизований, і не потребує від користувача знання програмування.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Рейда О.М. та Войтко В.В..

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою.

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Експерт 1	2. Експерт 2
	Бали, виставлені експертами:	
1	4	4
2	3	3
3	3	4
4	4	3
5	3	4
6	4	4
7	3	3
8	4	4
9	4	3
10	4	3
11	3	4
12	3	4
Сума балів	$СБ_1 = 43$	$СБ_2 = 43$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 43$	

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

$Т_р$ - кількість робочих днів у місяці, $Т_р$ = 22 дні;

t - число днів роботи розробника, t = 50 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2 – Розрахунки основної заробітної плати

Працівник	Оклад М, грн	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплат праці, грн.
Науковий керівник	6000	272,72	5	1363,5
Інженер-програміст	3400	154,54	50	7727
Всього:				9090,5

Розрахуємо додаткову заробітну плату:

$$Здод = 0{,}1 \cdot 9090{,}5 = 909{,}05\ (грн.)$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати:

$$Нзп = (Зо + Зр) \cdot \frac{\beta}{100}, \qquad (5.2)$$

$$Нзп = (9090{,}5 + 909{,}05) \cdot \frac{36{,}3}{100} = 3629{,}83\ (\text{грн.}).$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$A = \frac{Ц \cdot H_a}{100} \cdot \frac{T}{12}, \qquad (5.3)$$

де Ц – балансова вартість обладнання, грн;

H_a – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.3 – Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань,грн

Персональний комп’ютер	10000	25	3	625
Всього:				625

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n \text{Н}_\text{і} \cdot \text{Ц}_\text{і} \cdot \text{К}_\text{і}, \qquad (5.4)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	180	1	180
Пачка паперу	уп.	130	1	130
Ручка	шт.	10	1	10
Всього з урахуванням транспортних витрат				352

Витрати на силову електроенергію розраховуються за формулою:

$$B_e = B \cdot П \cdot Ф \cdot K_n \cdot \beta, \qquad B_e = B \cdot П \cdot Ф \cdot K_n ; \qquad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,7 грн/кВт);

П – установлена потужність комп’ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп’ютера (Ф=200 год.);

$К_п$ – коефіцієнт використання потужності ($К_п$< 1, $К_п = 0,7$).

$$Ве = 1,7 \cdot 0,6 \cdot 200 \cdot 0,7 = 142,8 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (1..3) \cdot (З_о + З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1 * (9090,5 + 909,05) = 9999,55 \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

В = 9090,5+909,05+3629,83+625+352+142,8+9999,55= 24748,73 (грн.)

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{24748,73}{1} = 24748,73$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{24748,73}{0,9} = 27498,58 \text{ (грн.)}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки.

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n \quad (\Delta П_я \cdot N + П_я \Delta N)_i \quad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технології зменшаться на 25 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 25 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 150 користувачів, протягом другого року – на 125 користувачів, протягом третього року – 100 користувачів. Реалізація інформаційної технології до впровадження результатів наукової розробки складала 500 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 400 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $^{\Delta П_1}$ протягом першого року складатиме:

$$\Delta П_1 = 25 \cdot 500 + (400 + 25) \cdot 150 = 76250 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 25 \cdot 500 + (400 + 25) \cdot (150 + 125) = 129375 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 25 \cdot 500 + (400 + 25) \cdot (150 + 125 + 100) = 171875 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$вкладених інвестицій розраховується за формулою:

$$Е_{абс} = (ПП - PV), \qquad (5.10)$$

де $^{\Delta П_i}$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

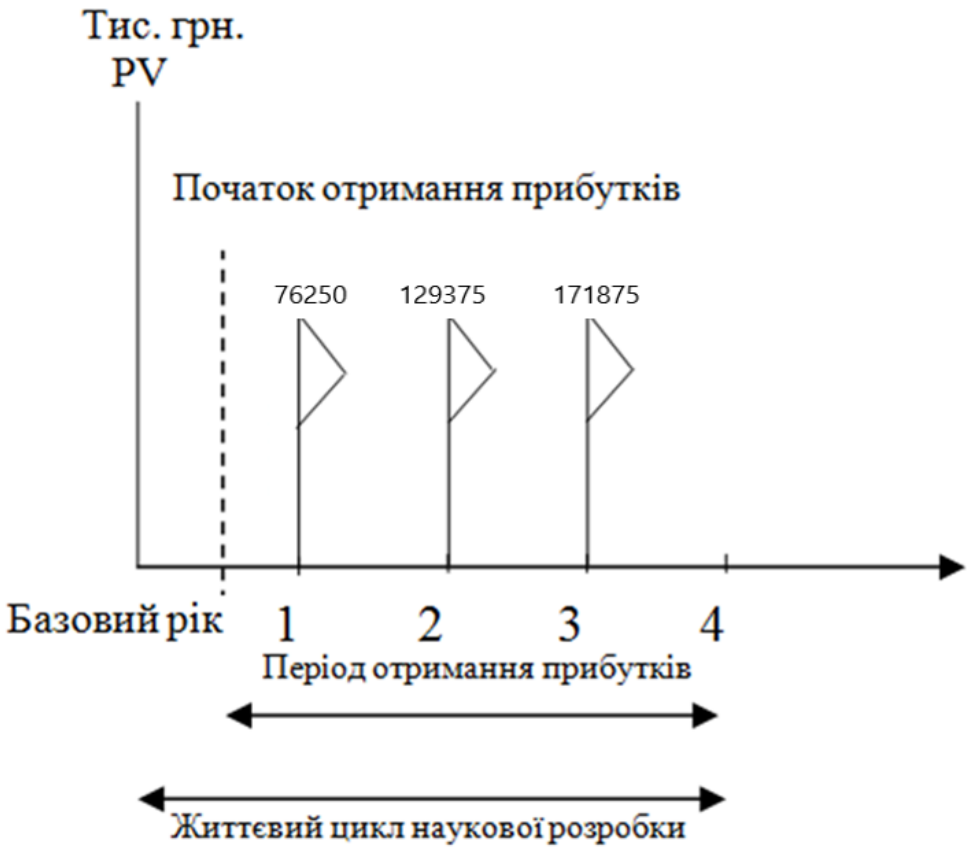


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{27498{,}58}{(1+0{,}1)^0} + \frac{76250}{(1+0{,}1)^2} + \frac{129375}{(1+0{,}1)^3} + \frac{171875}{(1+0{,}1)^4} = 305109{,}38 \text{ (грн.)}$$

Тоді розрахуємо $E_{абс}$:

$$E_{абс} = 305109{,}38 - 27498{,}58 = 277610{,}8 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$E_в = \sqrt[T]{1 + \frac{E_{абс}}{PV}} - 1 \quad (5.12)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$E_{\text{в}} = \sqrt[3]{1 + \frac{277610{,}8}{27498{,}58}} - 1 = 1{,}23 \text{ або } 123\ \%$$

Далі, розраховану величина $E_{\text{в}}$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{\text{мін}}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{\text{мін}}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0{,}2 + 0{,}1 = 0{,}3$$

Оскільки $E_{\text{в}}$ = 123% > $\tau_{\text{мін}}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{\text{ок}}$ розраховується за формулою:

$$T_{\text{ок}} = \frac{1}{\text{E}_{\text{в}}}$$

$$T_{\text{ок}} = \frac{1}{1{,}23} = 0{,}81 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

ВИСНОВОК

У ході виконання магістерської кваліфікаційної роботи було розроблено методи та програмну програмні замоби обміну даних між різноплатформеними базами даних.

В результаті виконання кваліфікаційної роботи, було отримано наступні результати:

1. Проведено аналіз предметної галузі, в результаті якого було зроблено висновок про можливість обміну даних між різноплатформеними базами даних.

2. Проведено аналіз існуючих аналогів, який підтвердив актуальність розробки власного програмного додатку, який буде нівелювати недоліки існуючих та реалізовувати розроблені методи.

5. Розроблено структурну схему створення баз даних. Було розроблено програмний систему за допомогою мов програмування: PHP,JavaScript. Framework Node.js, Phalcon, а також бібліотеки React та Redux в середовищі розробки Visual Studio.

6. Проведено тестування роботи розробленого веб додатку, яке підтвердило його стабільність та придатність до використання.

7. Проведено технологічний аудит розроблених методів і програмних засобів обміну даних між різноплатформеними базами даних, в ході якого було встановлено високий технічний рівень та комерційний потенціал розробки. Також, виконано розрахунок економічного ефекту від можливого впровадження розроблених методів і засобів, в ході якого підтверджено доцільність фінансування цієї наукової розробки.

Підсумовуючи усе вищесказане, можна зробити висновок, що задачі магістерської кваліфікаційної роботи було виконано у повному обсязі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Database [Електронний ресурс]. // Wikipedia. – 2017. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Database. - Назва з екрану.
2. NoSQL [Електронний ресурс]. – Режим доступу: https://www.quality-assurance-group.com/mobilnyj-dodatok-vnosymo-rozuminnya-u-znachennya-terminu/ - Назва з екрану.
3. Г.В. Табунщик, Т.І. Каплієнко, О.А Петрова Проектування та моделювання програмного забезпечення сучасних інформаційних систем. Запоріжжя: ЗНТУ. 2016. - 230 ст.
4. SQL. [Електронний ресурс]. – Режим доступу: http://bourabai.kz/dbt/SQL0 - Назва з екрану.
5. Прамодкумар Дж. Садаладж і Мартін Фаулер NoSQL. Методологія розробки нереляційних баз даних. - Київ,2016. - 112 ст.
6. А.В Вайгент Big data. Вся технологія в одній книзі. Київ, 2016. - 179 ст.
7. Томас Ерл Основи Big Data : концепції, алгоритми та технології. Київ, 2016. - 116 ст.
8. Андреас Вайгенд BIG DATA. Вся технологія в одній книзі. “Ексмо-Бомбора”. 2016.
9. JavaScript. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/JavaScript_basics. - Назва з екрану.
10. Бэрон Шварц, Вадим Ткаченко, Петр Зайцев MySQL по максимуму. Оптимізація, реплікація, резервне копіювання. Санкт-Петербург: “Питер” .2016.
11. React. [Електронний ресурс]. – Режим доступу: https://ru.reactjs.org/. - Назва з екрану.
12. Я.Г. Куваєв, О.А. Жукова, І.А. Сечкін Організація реляційних баз даних. Дніпро: НГУ. 2017.
13. MySql Workbench. [Електронний ресурс]. – Режим доступу: https://metanit.com/sql/mysql/1.3.php. - Назва з екрану.

14. ElasticSearch. [Електронний ресурс]. – Режим доступу: https://tproger.ru/blogs/why-elasticsearch-is-a-good-choice/ - Назва з екрану.
15. Томас Коноллі, Каролін Бегг Бази даних: проектування, реалізація і супровід. Теорія та практика. 2017. - 50 ст.
16. Д.П. Осіпов Технології проектування баз даних. Москва, 2017. - 220 ст.
17. Алекс Янг. Бредлі Мек. Майк Кантелон Node.js в дії. 2-е видання. Санкт-Петербург, 2015. - 300 ст.
18. Тестування. [Електронний ресурс]. // Wikipedia. – 2017. – Режим доступу:https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення - Назва з екрану.
19. Ручне_тестування. [Електронний ресурс]. // Wikipedia. – 2017. – Режим доступу: https://uk.wikipedia.org/wiki/Ручне_тестування - Назва з екрану.
20. Козловський В. О. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт. Вінниця: ВНТУ. 2012.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"____ "_____2019 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Методи та програмні засоби обміну даних між різноплатформеними базами даних» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доц. О.М. Рейда

"____"_____2019 р.

Виконав:

_____студент гр.1ПІ-18м Б.С. Розумовський

"____"_____2019 р.

Вінниця – 2019 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Методи та програмні засоби обміну даних між різноплатформеними базами даних». Згідно отриманого завдання кінцевий програмний продукт може використовуватись офіційною організацією.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення - протокол №_____ від _____.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення швидкодії обробки запитів при роботі з структурованими даними типу BigData. За допомогою методів та програмних засобів обміну інформації між різноплатформеними базами даних.

Об'єктом дослідження є процес обміну даних між різноплатформеними базами даних.

Для досягнення поставленої мети мети в роботі вирішуються такі завдання:

- провести аналіз існуючих методів і засобів створення і реалізації баз даних;
- розробити методи обміну даними між двома різноплатформеними базами даних;
- Розробити методи підвищення продуктивності роботи з даними;
- розробити програмні компоненти для обміну даними між двома базами даних;
- провести тестування розробленої адміністративної частини, яка використовує методи та програмні засоби обміну даних між різноплатформеними базами даних різноплатформеної бази даних.

3. Технічні вимоги

Вихідні дані до роботи: браузер Google Chrome; реляційна база даних SQL, нереляційна база даних NoSql.

Вихідні дані до модифікації роботи: бібліотека React та Redux, фреймворки Phalcon та Node.js., мененджер пакетів npm або yarn.

4. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги веб додатку.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз стану питання та постановка задачі	07.10.2018 – 27.10-2019
2	Розробка методів та програмних засобі обміну даних між різноплатформеним базами даних	28.10.2019 – 8.11.2019
3	Розробка адміністративної частини використанням обміну даних	9.11.2019 – 20.11.2019
4	Тестування розробленої адміністративно частини	21.11.2019 – 1.12.2019
5	Економічна частина	1.12.2019 – 6.12.201

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. Лістинг коду

```
<?php
class FeedPromManager {
public function generateYmlFeed() {
        error_reporting(-1);
        ini_set('memory_limit', '5000M');
        ini_set('max_execution_time', 0);
        $time = time();

require__DIR__. '/../../managers/controls/export/PromUaYmlExportManager2.php';
```

```
        $path = __DIR__ . '/../../../www/xml/xml_zip';

        $format = '.xml';
        $zip = '.zip';
        $fileName = 'prom_yml_xml'.$time.$format;
        $fileNameZip = 'prom_yml_xml'.$time.$format.$zip;

        $pathToFileZip = '/xml/xml_zip/'.$fileNameZip;

        $ids = $this->selectFromDb();

      if ($ids['feedQueryId'] * 1 > 0 &&  $ids) {

        $manager = new PromUaYmlExportManager2('hand', $ids['eventQuery']);

        $zippetFile = $manager->saveXmlFile($path, $fileName)->zip($path, $fileName);

        if(stristr($zippetFile, 'adding:')) {
          $status = 'Генерация завершена';
          $this->updtDb($fileNameZip, $pathToFileZip, $status, $ids['feedQueryId']);
        } else {
          $status = 'Ошибка генерации';
          $this->updtDb($fileNameZip, $pathToFileZip, $status, $ids['feedQueryId']);
        }

        $time = time() - $time;

        echo 'time: ', $time, ' seconds (', gmdate('H:i:s', $time), ')', PHP_EOL;
```

```
        }

    }

    public function selectFromDb() {

        $mysql_option = $GLOBALS['options']['database_details'];

        $link = mysqli_connect($mysql_option['hostname'], $mysql_option['username'], $mysql_option['password'], $mysql_option['database'], $mysql_option['port']);

        $link->set_charset('utf8');

        $sql = mysqli_query($link, "SELECT options_json, id FROM treba.feed_queries WHERE status='В процессе генерации' AND url='' LIMIT 1");

        $objs = mysqli_fetch_assoc($sql);

        $query = array();
        $query['must'] = array();
        $query['must'][]['term'] = ['checked' => 0];
        $query['must'][]['exists'] = ['field' => 'rubric_id'];
        $query['should'] = array();

    $decodObj = json_decode($objs['options_json']);
    $feedQueryId = json_decode($objs['id']);


    if ($objs) {
```

```
            foreach ($decodObj as $value) {
                $eventer = array();
                $obj = $value;
                foreach ($obj as $ke => $vel) {
                    if($ke === 'rubric_id') {
                        $eventer[] = ['term'=> ['rubric_id' => $vel]];
                    }else {
                        $eventer[] = ['term'=> ['option.' . $ke => $vel]];
                    }
                }

                $query['should'][] =  ['bool' => ['must' => $eventer]];
            }
        }

    $query['minimum_should_match'] = 1;

    $searchParamsAggs['body']['query']['bool'] = $query;

    $eventQuery = $query;

    return ['feedQueryId' => $feedQueryId, 'eventQuery' => $eventQuery];
}

    public function updtDb($fileNameZip, $pathToFileZip, $status, $feedQueryId)
    {

        $mysql_option = $GLOBALS['options']['database_details'];
```

```
            $link = mysqli_connect($mysql_option['hostname'], $mysql_option['username'], $mysql_option['password'], $mysql_option['database'], $mysql_option['port']);

            $link->set_charset('utf8');

            $sql1 =mysqli_query($link, "UPDATE treba.feed_queries SET url='{$pathToFileZip}', status='{$status}', name='{$fileNameZip}' WHERE id = {$feedQueryId}");

            $sql = mysqli_query($link, "DELETE FROM treba.feed_queries  WHERE id IN
            (SELECT * FROM ((SELECT id FROM feed_queries WHERE created_at <= DATE_SUB(CURRENT_DATE, INTERVAL 7 DAY))) AS fq)
            ;");

            return exec("find ./../../../www/xml/xml_zip/* -mtime +7 -type f -delete");

        }

    }
```

Додаток В.Ілюстративний матеріал

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д. т. н., професор _____О. Н. Романюк

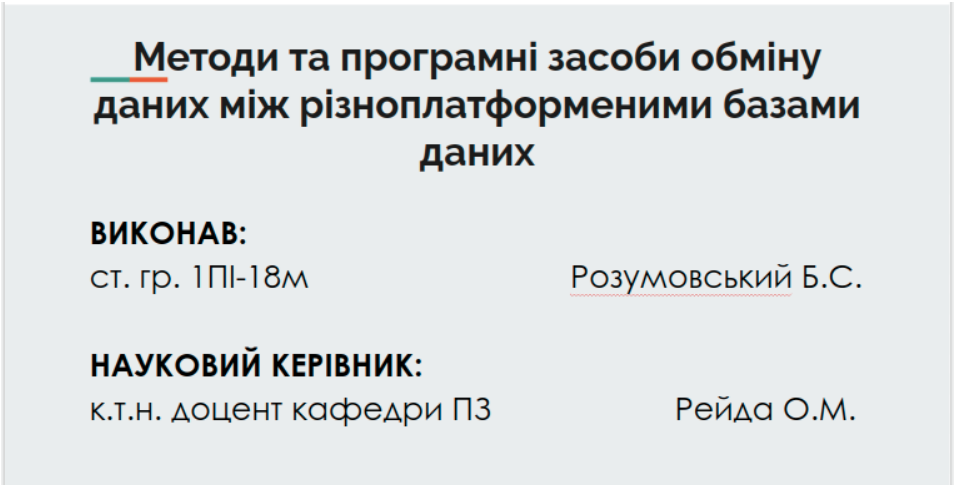
Науковий керівник, к. т. н., проф кафедри ПЗ_____О.М. Рейда

Рецензент, к.т.н, доцент кафедри КН_____ Я.В. Іванчук

Нормоконтроль, к. т. н., проф. кафедри ПЗ _____О.М. Рейда

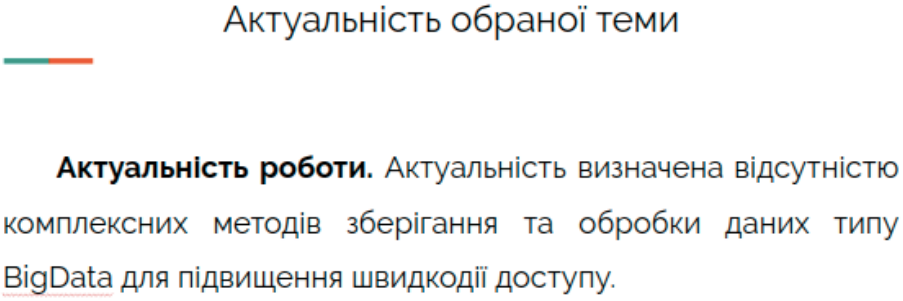
Виконавець, студент групи 1ПІ-18м _____ Б.С. Розумовський

Слайд 1 – Тема і автор МКР



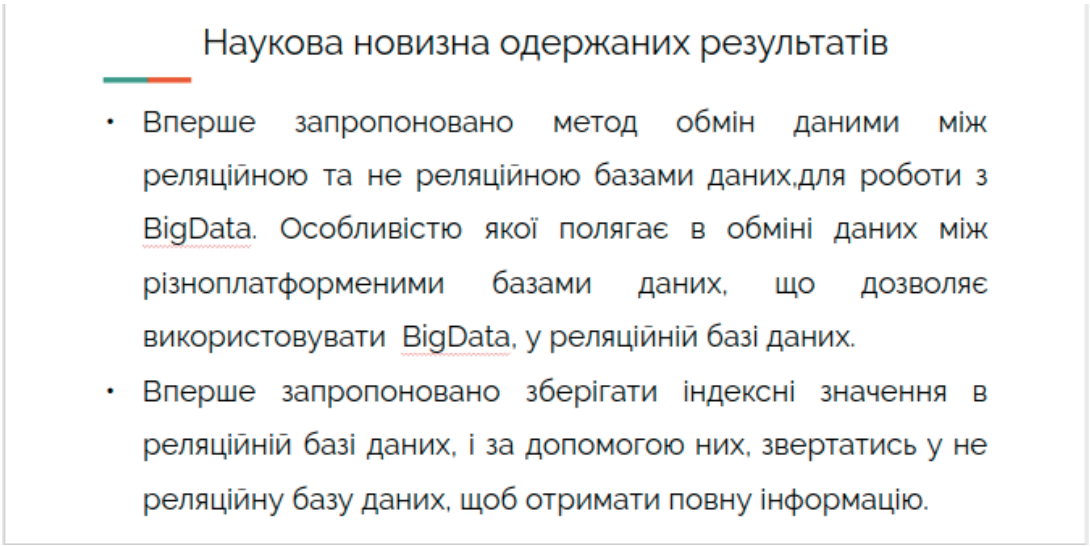
Слайд 2 – Мета роботи та предмет і об'єкт дослідження

Слайд 3 – Актуальність обраної теми



Слайд 4 - Основні задачі дослідження

Слайд 5 - Наукова новизна одержаних результатів



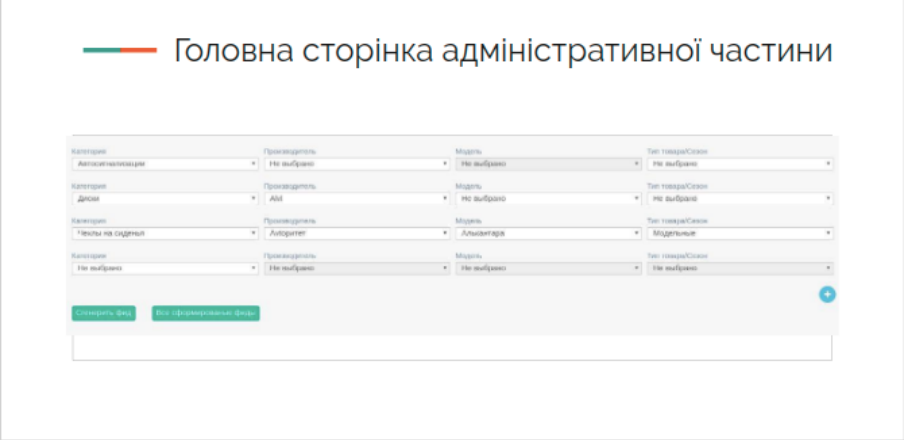
Слайд 6 - Практичне значення одержаних результатів

Практичне значення одержаних результатів

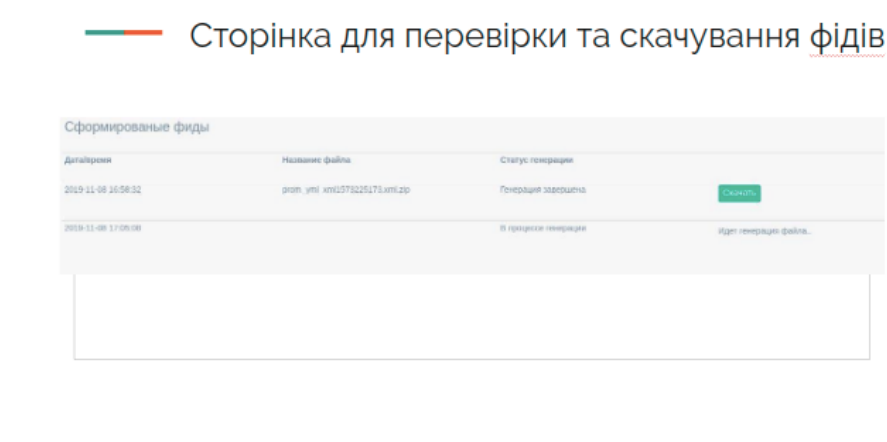
На основі отриманих в магістерській кваліфікаційній роботі теоретичних положень розроблено метод створення та зберігання даних в різноплатформених базах даних.

Слайд 7 - Схема роботи адміністративної частини генерації фідів за допомогою методів обміну даних між різноплатформеними базами даних

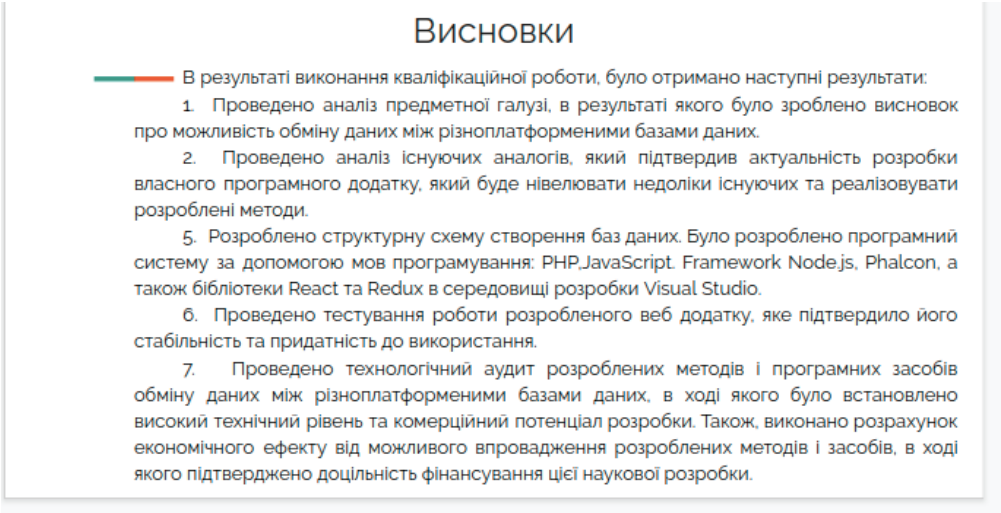
Слайд 8 - Головна сторінка адміністративної частини



Слайд 9 - Сторінка для перевірки та скачування фідів



Слайд 10 - Сторінка «Висновки»



Слайд 11 - Сторінка «Дякую за увагу»