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

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 681.3.06

«До захисту допущено»
Завідувач кафедри
_____ І.А. Дичка
«___»_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 “Інженерія програмного забезпечення”

на тему: «Метод аналітичної обробки великих даних для сервісу аналітичних досліджень міжнародних торгівельних даних»

Виконала:
студентка VI курсу, групи КП-61м
Феліндаш Валерія Віталіївна _____

Керівник:
Ст. викладач ПЗКС, к.ф.м.н.,
Гречко А.В. _____

Консультант з аналітичної обробки даних:
Ст. науковий співробітник
Інституту кібернетики ім. В.М.Глушкова, к.ф.м.н.,
Мелащенко А.О. _____

Рецензент:
доцент кафедри ММСА ІПСА, к.т.н., доц,
Дідковська М.В. _____

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

Київ – 2018 року

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

HS code – тематичний параметр для пошукової системи, розроблений для класифікації товарів імпорту/експорту;

Big data (укр. Великі дані) – сукупність підходів, інструментів і методів обробки структурованих і неструктурованих даних величезних обсягів;

Система керування базами даних (СКБД) – комплекс програмного забезпечення, що надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних;

Amazon Web Services (AWS) – інфраструктура платформ хмарних веб-сервісів, представлена компанією Amazon. В AWS представлені сервіси оренди віртуальних серверів, надання обчислювальних потужностей, зберігання даних (файловий хостинг, розподілених сховищ даних) і т.п.;

Amazon RDS (укр. Служба реляційної бази даних Amazon) – це веб-служба, яка спрощує настройку, роботу і масштабування реляційної бази даних в хмарі;

REST API (укр. Від англ. Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого додатка в мережі;

iOS – мобільна операційна система для смартфонів, електронних планшетів, носяться програвачів і деяких інших пристроїв, що розробляється і випускається американською компанією Apple.

ВСТУП

На сьогодні в усьому світі спостерігається високий інтерес до технологій класу Big Data, пов'язаний з постійним зростанням об’ємів даних, якими доводиться оперувати великим компаніям. Важливим активом для будь-якої компанії є накопичена інформація, проте з кожним роком стає все складніше і дорожче обробляти її і отримувати від неї користь.

Отже об’єктом дослідження є процес обробки великих даних для сервісу аналітики. Предметом дослідження є метод для обробки та аналізу великих даних.

Мета роботи - розроблення методу для обробки та аналізу великих даних для сервісу аналітики міжнародних торгівельних даних на основі міжнародного HS коду.

Існує багато архітектурних рішень, які дозволяють досягнути бажаних завдань. В магістерський дисертації пропонується розглянути найбільш поширені принципи роботи з великими даними та проаналізувати архітектури, які дозводяють виконувати поставлені задачі. Також планується проаналізувати методи роботи з великими даними, які можуть бути використані для побудови аналітичних звітів, а також робота з даними у розподілених файлових системах. Для розуміння задачі необхідно розглянути основні аспекти аналітичної обробки даних, зокрема вилучення та обробка денормалізованих даних з існуючих сховищ та провести дослідження швидкодії роботи різних баз даних, а саме AWS Dynamo DB та PostgreSQL.

В ході аналізу передбачається порівняння баз даних та архітектур побудови серверних застосунків. Спираючись на отримані дані буде знайдено оптимальний рішення для побудови системи для роботи з великими даними.

В результаті базуючись на дослідження буде побудована система для роботи з аналітичними даними яка буде доступна для в магазині додатків AppStore та Google Play.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РОБОТИ З ВЕЛИКИМИ ДАНИМИ

В першу чергу, під терміном "BigData" розуміється величезний набір інформації. Причому обсяг даних настільки великий, що обробка великих обсягів інформації стандартними програмними і апаратними засобами представляється вкрай складною. Існують труднощі зі зберіганням та обробкою великих обсягів даних. Саме тому необхідно проаналізувати методи та інструменти, які необхідні для роботи з такими даними.

1.1. Проблематика роботи з великими даними

Отже можна виділити основну проблему – це складність обробки потоку великих даних та візуалізації їх.

Причинами виникнення даної проблеми є:

- Обмежена доступність – існуючі альтернативні програмні забезпечення обмежують вхідні параметри запиту, тобто не дозволяють шукати по всім странам.
- Обмежена швидкодія – великі обсяги даних, які необхідні для аналізу результатів, є причиною високих витрат по швидкодії.
- Відсутність аналогів, які візуалізують отримані результати – так як результат роботи подається в табличному вигляді, є причиною того, що користувачеві важко працювати з цими даними в подальшому.

Складність обробки потоку великих даних та візуалізації їх, у свою чергу, породжує наступний ряд проблем:

- Відсутність бази даних – громіздка база даних та важкість у підтримуванні бази, є причиною того, що дані зберігаються у файлах, що призводить до складності формування та обробки запиту.

- Недостатня швидкодія – розголудженність вхідних даних та відсутність логіки при формуванні запиту збільшують час обробки запиту.

1.2. Основні підходи для збереження та обробки великих даних

До найбільш поширених підходів обробки даних відносяться такі інструменти.

- SQL – мова структурованих запитів, що дозволяє працювати з базами даних. За допомогою SQL можна створювати і модифікувати дані, а управлінням масиву даних займається відповідна система управління базами даних.
- NoSQL – термін розшифровується як Not Only SQL (не тільки SQL). Включає в себе ряд підходів, спрямованих на реалізацію бази даних, що мають відмінності від моделей, що використовуються в традиційних, реляційних СУБД. Їх зручно використовувати при постійно мінливій структурі даних. Наприклад, для збору і зберігання інформації в соціальних мережах.
- MapReduce – модель розподілу обчислень. Використовується для паралельних обчислень над дуже великими наборами даних (петабайт * і більше). У програмному інтерфейс не дані передаються на обробку програмі, а програма – даними. Таким чином запит є окремою програмою. Принцип роботи полягає в послідовній обробці даних двома методами Map і Reduce. Map вибирає попередні дані, Reduce агрегує їх.
- Hadoop – використовується для реалізації пошукових і тематичних механізмів високонавантажених сайтів – Facebook, eBay, Amazon і ін. Відмінною особливістю є те, що система захищена від виходу з ладу будь-якого з вузлів кластера, так як кожен блок має, як мінімум, одну копію даних на іншому вузлі.

- SAP HANA – високопродуктивна NewSQL платформа для зберігання і обробки даних. Забезпечує високу швидкість обробки запитів. Ще одним відмітним ознакою є те, що SAP HANA спрощує системний ландшафт, зменшуючи витрати на підтримку аналітичних систем [1].

Найбільш поширені технології для роботи з великими даними зображені на рис. 1.

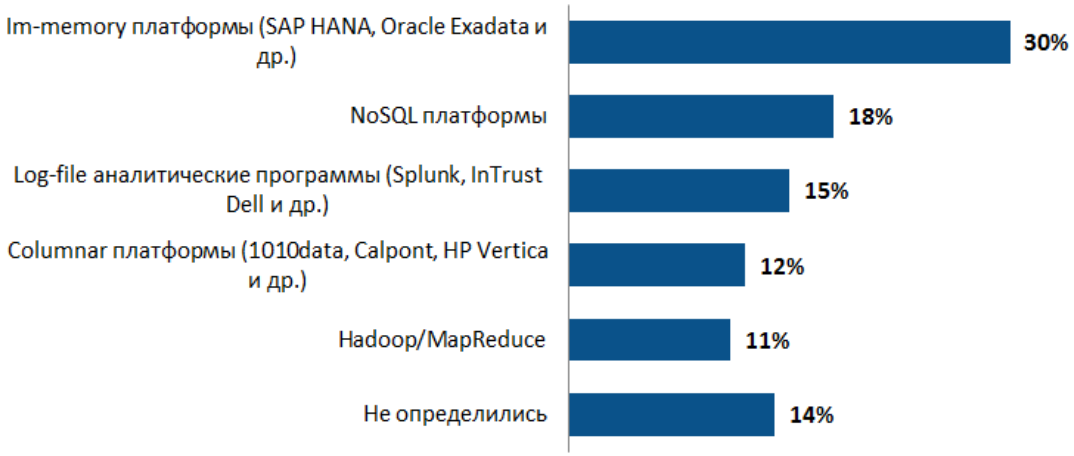

Рис.1. Основні технології для роботи з BigData

Проаналізувавши рис.1, можна зробити висновки, що найбільшою популярністю користуються такі технології Big Data, як вбудовані платформи компаній SAP, HANA, Oracle та ін. Результати опитування T-Systems показали, що їх вибрали 30% опитаних компаній. В основу популярності стали NoSQL платформи (18% користувачів), також компанії використовували аналітичні платформи компаній Splunk і Dell, їх вибрали 15% компаній. Найменш корисні для вирішення проблем великих даних за результатами опоросу є продукти Hadoop / MapReduce.

1.3. Аналогічні системи з використанням великих даних

Сфера використання технологій Великих Даних обширна (рис. 2.). Так, за допомогою Великих Даних можна дізнатися про переваги клієнтів, про ефективність маркетингових кампаній або провести аналіз ризиків.

Нижче ви побачите результати опитування IBM Institute, про напрямки використання Big Data в компаніях.

Рис. 2. Сфери використання великих даних

Як видно з рис.2, більшість компаній використовують великі дані в сфері клієнтського сервісу, другий за популярністю напрямок – операційна ефективність, в сфері управління ризиками великі дані менш поширені на поточний момент. Саме тому слід детально дослідити існуючи аналогічні рішення.

1.3.1. The Atlas of Economic Complexity

The Atlas of Economic Complexity – це потужний інструмент візуалізації даних, який дозволяє людям вивчати глобальні торгові потоки на ринках, відслідковувати цю динаміку з часом та відкривати нові можливості для зростання в кожній країні. "Атлас" визначає промислові можливості та розвинені країни в центрі перспектив зростання, де різноманітність та складність існуючих можливостей сильно впливають на те, як відбувається зростання (рис. 3).

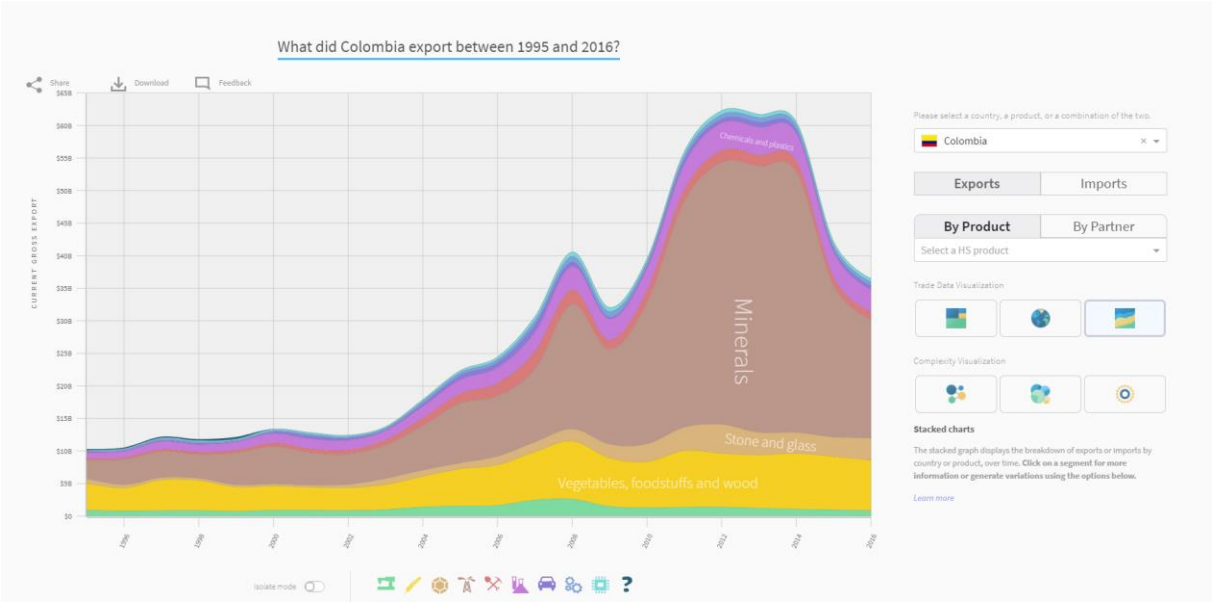

Рис. 3. Результат The Atlas of Economic Complexity

Розроблений у Центрі міжнародного розвитку Гарвардського університету, Атлас поєднує дані про торгівлю з синтезованими статистичними даними з дослідження CID щодо економічної складності у доступному та інтерактивному вигляді. Як динамічний ресурс, інструмент постійно розвивається з новими даними та функціями [2].

Технології, які були використані при реалізації:

- Власні розробки.
- Hadoop.

Переваги:

- Є можливість комбінувати поля, або вибирати одне з них.
- Пошук відкритий, без реєстрації.

Недоліки:

- Не є достатньо гнучкою для забезпечення подальшого аналізу отриманих результатів.

1.3.2. The International Trade Centre – Nepal

Міжнародний торговельний центр (ІТЦ) – сервіс аналітики експорту/імпорту в Непалі для покращення їх експорту (рис. 4) [3].

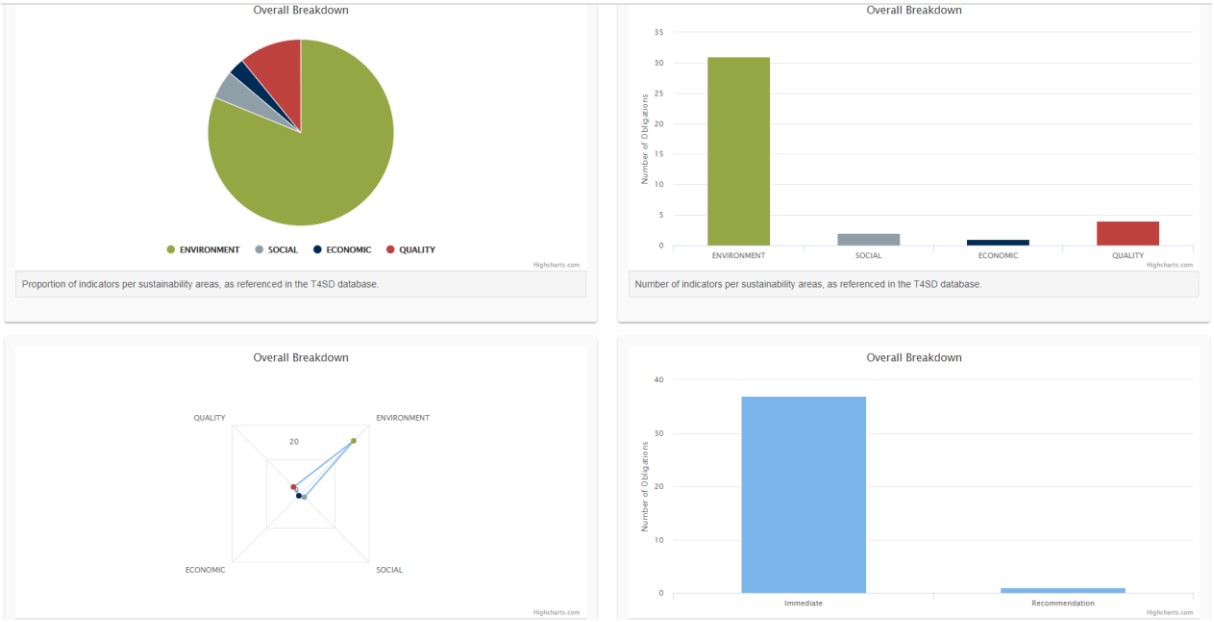

Рис. 4. Результат аналітики Міжнародний торговельного центру в Непалі

Технології, які були використані при реалізації:

- Власні розробки;
- Hadoop.

Переваги:

- є можливість комбінувати поля, або вибирати одне з них.

Недоліки

- аналітика відбувається лише на даних про експорт/імпорт Непалу;
- необхідна реєстрація, більш того ускладнена авторизацією через телефон.

1.3.3. International Trade Business Intelligence System

Система міжнародної торгівлі бізнес-аналітики (it-biS) – це онлайн платформа експортної аналітики, яка надає:

- Інформація про міжнародну торгівлю в режимі інтерактивного режиму.
- Інформація про ціни на 14000 товарів (світові, регіональні).
- Інформація про обсяги торгівлі та ринків збуту.
- Інформація по 28 параметрам, важливим для прийняття рішень по українському експорту (рис. 5).
- Аналітичний доповідь експорту та міжнародної торгівлі по необхідним параметрам и товарам [4].

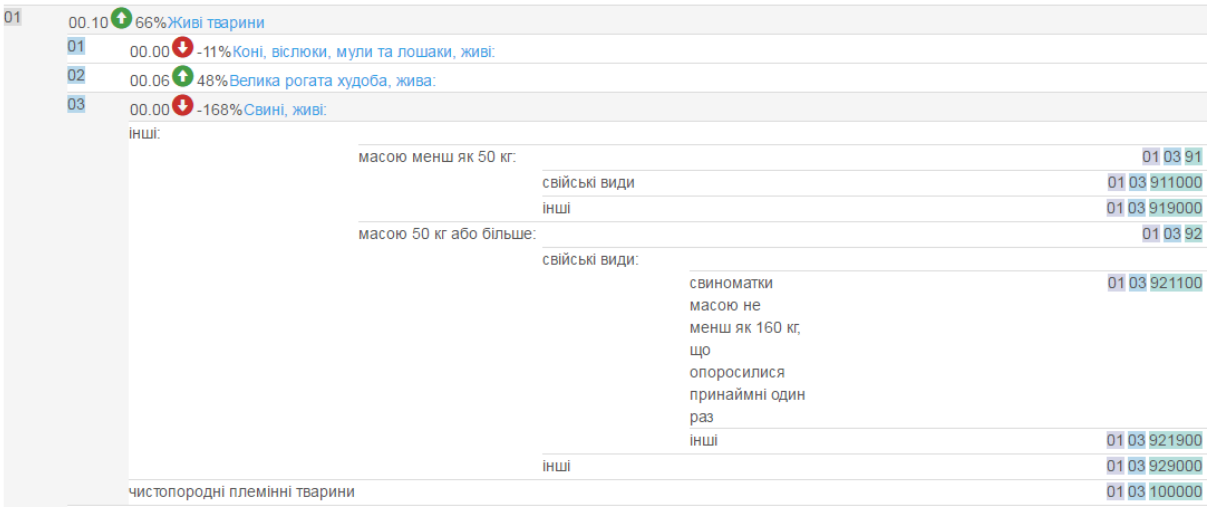

Рис. 5. Головна сторінка International Trade Business Intelligence System

Технології, які були використані при реалізації:

- Власні розробки;
- SAP HANA;
- Oracle Exadata.

Переваги:

- можливість відстежувати експортну та імпортну діяльність в режимі реального часу.

Недоліки:

- не є достатньо гнучкою для забезпечення подальшого аналізу отриманих результатів;

- час виконання запиту у деяких випадків складає більше 30 секунд.

1.4. Висновки

Проведений аналіз існуючих проблем та рішень показав, що хоча всі розроблені на сьогодні системи аналоги включають широкі можливості для здійснення пошуку та фільтрації даних, проте вони не є достатньо гручкими для забезпечення подальшого аналізу отриманих результатів, тому значну частину роботи присвячено огляду підходів для їх вирішення.

Дане дослідження передбачило розроблення серверної частини веб-сервісу, а також клієнтської частини. Були проаналізовані основні методи для побудови системи для роботи з великими даними та аналогічні сервіси та іх реалізація.

2. МЕТОДИ РОБОТИ З ВЕЛИКИМИ ДАНИМИ

Обробка великих даних завжди було непростою задачею, існує велика кількість підходів та засобів для опрацювання великих даних. Вибір остаточно способу має грунтуватися на кількості даних, кінцевої мети, та бюджету.

2.1. Робота з даними у розподілених файлових системах

Існує безліч комбінацій програмного і апаратного забезпечення, які дозволяють створювати ефективні рішення Big Data для різних бізнес дисциплін: від соціальних медіа та мобільних додатків, до інтелектуального аналізу і візуалізації комерційних даних. Важливе значення Big Data – це сумісність нових інструментів з широко використовуваними в бізнесі базами даних, що особливо важливо при роботі з крос-дисциплінарними проектами.

Послідовність роботи з Big Data складається з збору даних, структурування отриманої інформації за допомогою звітів і аналітика (dashboard). Так як робота з Big Data має на увазі великі витрати на збір даних, результат обробки яких заздалегідь невідомий, основним завданням є чітке розуміння, для чого потрібні дані, а не те, як багато їх є в наявності. В цьому випадку збір даних перетворюється в процес отримання виключно потрібної для вирішення конкретних завдань інформації.

Big Data зазвичай зберігаються і організовуються в розподілених файлових системах.

У загальних рисах, інформація зберігається на декількох (іноді тисячах) жорстких дисках, на стандартних комп'ютерах.

Так звана «карта» (map) відстежує, де (на якому комп'ютері і / або диску) зберігається конкретна частина інформації.

Для забезпечення відмовостійкості та надійності, кожну частину інформації зазвичай зберігають кілька разів, наприклад – тричі.

За допомогою стандартного устаткування і відкритих програмних засобів для управління цією розподіленою файловою системою наприклад, Hadoop, порівняно легко можна реалізувати надійні сховища даних в масштабі петабайт.

Найпростіший спосіб запустити MapReduce-програму на Hadoop – скористатися streaming-інтерфейсом Hadoop. Streaming-інтерфейс передбачає, що map і reduce реалізовані у вигляді програм, які приймають дані з stdin і видають результат на stdout [5].

MapReduce – це модель розподіленої обробки даних, запропонована компанією Google для обробки великих обсягів даних на комп'ютерних кластерах (рис. 6).

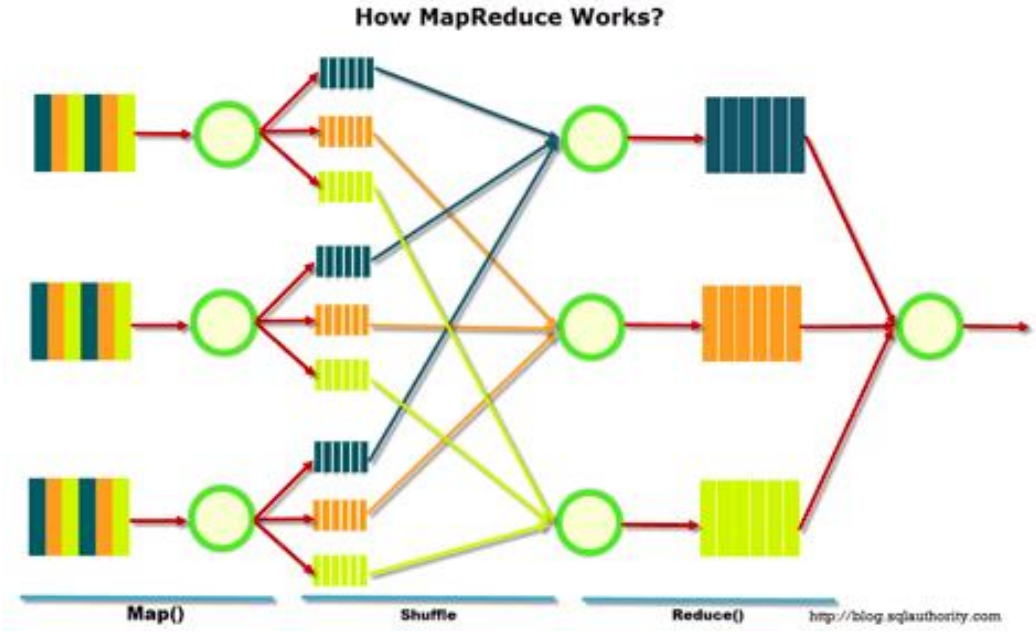

Рис. 6. Принцип роботи з MapReduce

Обробка даних відбувається в 3 стадії:

Перша стадія – стадія Map. На цій стадії дані оброблюються за допомогою функції map(), яку визначає користувач. Робота цієї стадії полягає в передобробці і фільтрації даних. Робота дуже схожа на операцію map в функціональних мовах програмування – призначена для користувача функція застосовується до кожної вхідного відрізку.

Функція map () застосована до однієї вхідного відрізку і видає безліч пар ключ-значення.

Друга стадія – стадія Shuffle. Проходить непомітно для користувача. У цій стадії висновок функції map «розбирається по кошиках» – кожний кошик відповідає одному ключу виведення стадії map. Надалі ці кошики послужать входом для reduce.

Третя стадія – стадія Reduce. Кожний «кошик» із значеннями, сформований на стадії shuffle, потрапляє на вхід функції reduce ().

Але не менш важливий вибір сервісу або бази даних для збереження великої кількості даних.

На сьогодні існує багато хмарних рішень для збереження та опрацьовання великої кількості даних. Кожне рішення має свої сервіси систем керування базами даних. Так серед усіх існуючих варто виділити сервіси Amazon, оскільки вони є лідерами на ринку.

Підхід map-reduce можна використоувавти в основних базах даних, які підтримують багатокластерність.

2.2. Аспекти аналітичної обробки даних

Для оптимізації роботи з великими даними варто провести аналітичну роботу над тим, як можна оптимізувати дані для зменшення затратності їх опрацювання.

Тому далі буде висвітлено основні проблеми, які доводиться вирішувати при створенні аналітичних систем на прикладі публічні закупівлі України, зокрема вилучення та обробка денормалізованих даних з існуючих сховищ.

Також дослідження можливості аналізу та обробки даних. При цьому варто відслідкувати основні проблеми побудови аналітичної системи тендерних закупівель та розглянути підходи до їх розв’язання, зокрема шляхи нормалізації текстових даних [6].

Першочерговим завданням при створенні будь-якої аналітичної системи є вибірка та фільтрація даних про суб’єктів з БД (рис. 7). Було ідентифіковано відсутність сталої форми введення інформації, що призвело до запису у таблиці дублів замовників та переможців. Ці записи відрізнялись лише мовою введення, вживаними абревіатурами або знаками пунктуації. Для розв’язання цієї задачі виконано не тривіальний пошук унікальних значень та видалення їх повторень.

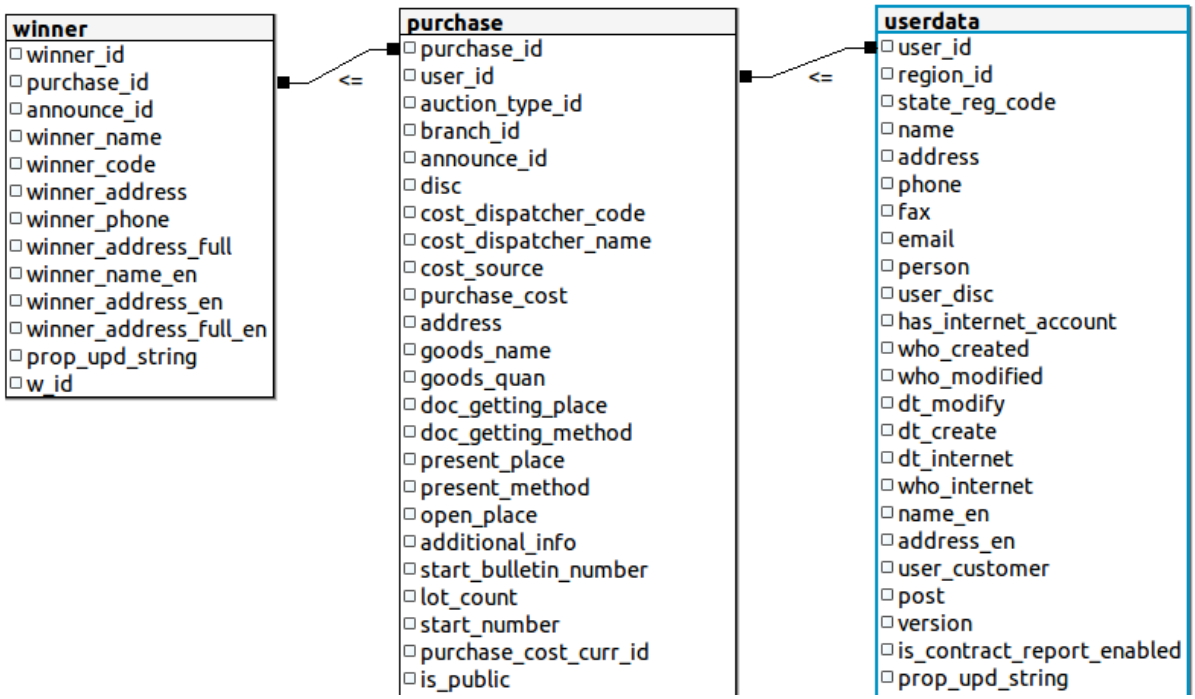

Рис. 7. Фрагмент схеми БД tender.me.gov.ua

Було передбачено ряд правил для заміни повних назв типів суб'єктів господарювання на їх абревіатури та приведення абревіатур з різних мов до єдиного набору. Зокрема здійснено наступні заміни: "товариство з обмеженою відповідальністю" : "тов", "товариство обмеженою відповідальністю" : "тов", "приватне підприємство" : "пп", "державна установа" : "ду", "державне підприємство" : "дп", "приватне акціонерне товариство" : "пат" і т.д..

Це дозволило зменшити кількість дублів замовників вдвічі, а переможців вчетверо, оскільки через одночасне використання повних назв

та їх абревіатур деякі організації та установи зустрічалися у відповідних таблицях БД по декілька разів.

Кожному замовнику або переможцю в БД було присвоєно унікальний ідентифікатор. Відфільтровані та ідентифіковані дані збережені в таблицях замовників, переможців та закупівель в базі даних PostgreSQL.

2.3. Дослідження швидкодії роботи різних баз даних

На сьогодні в усьому світі спостерігається великий інтерес до технологій класу Big Data, пов'язаний з постійним зростанням даних, якими доводиться оперувати великим компаніям. Важливим активом будь-якої компанії є накопичена інформація, проте з кожним днем стає все складніше і дорожче обробляти її і отримувати від неї користь.

На сьогодні існує багато хмарних рішень для зберігання та опрацювання великої кількості даних. Кожне рішення має свої сервіси систем керування базами даних. Так серед існуючих варто виділити сервіси Amazon, оскільки вони є лідерами на ринку, DynamoDB та Aurora PostgreSQL.

В роботі пропонується порівняти час обробки запитів для баз даних AWS Dynamo DB та AWS Aurora PostgreSQL. Також для більшої ілюстративності вирішено продемонструвати швидкість читання з використанням механізму індексації та без нього.

Крім того, так як метою роботи є продемонструвати взаємозв'язок між розміром даних і швидкістю доступу при наявності індексів і без них, також врахована кількість проіндексованих стовпців в кожному окремому випадку. Усі результати дослідження продемонстровано в таблично-графічному вигляді.

2.3.1. Обробка запитів на локальній машині

На швидкість обробки запитів впливає багато різних факторів. По-перше, це звісно структура та механізми самої бази даних, на яких вона заснована, оскільки кожна БД має свій алгоритм читання та структуру зберігання даних. Наприклад, одна СКБД може показувати кращі результати при читанні даних, а інша при записі. По-друге, фізичне розташування комп'ютера, на якому зберігається база даних. Так, результати з локального комп'ютера будуть отримані швидше, ніж з датацентру, розташованого на іншому континенті в іншій півкулі. По-третє, на швидкість впливають розміри результату. Наприклад, результуюча таблиця з 10 записами буде отримана швидше у порівнянні з таблицею з такою ж схемою даних, проте кількістю рядків у сотню раз більшою.

Також не останню роль у швидкості читання даних відіграють механізми оптимізації баз даних. Так, більшість СКБД підтримують механізм індексації. Індекс формується зі значень одного чи кількох стовпчиків таблиці та вказівників на відповідні рядки таблиці і, таким чином, дозволяє знаходити потрібний рядок за заданим значенням. Прискорення роботи з використанням індексів досягається в першу чергу за рахунок того, що індекс має структуру, що оптимізована для пошуку, наприклад, збалансованого дерева.

Для підвищення ефективності запитів індекси зазвичай створюються на тих стовпцях таблиці, які часто використовуються в запитах. Для однієї таблиці можуть бути створені кілька індексів. Однак збільшення числа індексів уповільнює операції додавання, оновлення, видалення рядків таблиці, оскільки при цьому необхідно оновлювати самі індекси. Крім цього, індекси займають додатковий обсяг пам'яті.

В роботі пропонується порівняти час обробки запитів для баз даних AWS Dynamo DB та AWS Aurora PostgreSQL. Також для більшої

ілюстративності вирішено продемонструвати швидкість читання з використанням механізму індексації та без нього.

Для тестових даних обрано набір фільмів з публічної бібліотеки imdb. Тестова база даних має достатній набір даних, щоб відстежити залежність у часі виконання запиту для різної кількості рядків у результуючій таблиці.

Оскільки тестування проводиться з використанням мережі Інтернет, затримка у з'єднанні може негативно вплинути на кінцевий результат. Для того, щоб цього уникнути, було прийнято виконувати кожен тестовий запит 5 разів і за остаточний результат вважати час, що є середнім арифметичним часу виконання усіх запитів.

Тестування AWS DynamoDB відбувалося, як на локальній машині з використанням версії DynamoDB для розробників, так і з хмарною версією. Для виконання тестів вибрано мову програмування JavaScript та платформу Node.js. Для встановлення з'єднання з базою даних використано SDK aws-sdk. Засобом доступу до бази даних обрано застосування PgAdmin, яке надає графічний інтерфейс та SQL-консоль.

Перед початком роботи слід завантажити тестові дані до бази даних. Для цього було створено SQL-скрипт, що записує усі дані в сховищі.

В ході тестування на тестових даних було виконано 5 запитів, які відрізнялися кількістю рядків у результуючий таблиці – 10, 100, 1000, 2000, 4000 рядків відповідно.

Таблиця 1

Читання для локальної AWS DynamoDB з індексацією та без

Кількість записів	**Час, мілісекунди**	
	AWS DynamoDB без індексації	AWS DynamoDB з індексацією
10	49	43
100	53	52

1000	138	98
2000	309	141
4000	691	192

Виміряний при цьому час доступу до даних для кожного дослідженного випадку наведено у табл. 1. Для наочності дані представлені у графічному вигляді на рис. 8.

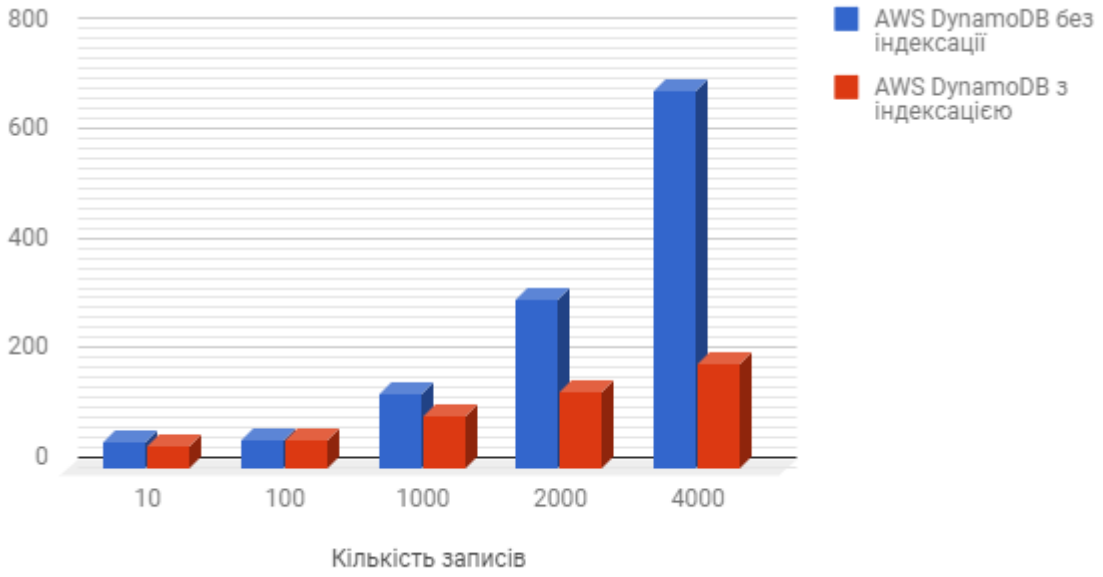

Рис. 8. Візуалізація результатів для локальної AWS DynamoDB з індексацією та без

Проаналізувавши дані з таблиці та на графіку, можна зробити висновки, що на невеликих об’ємах даних читання таблиці з індексом та без займає приблизно однаковий час. Проте при збільшенні кількості рядків в результатах запиту час доступу до даних у підході з використанням таблиці з індексом суттєво знижується.

2.3.2. Обробка запитів на машині на сервісах Amazon

Аналогічне дослідження виконано для AWS Aurora PostgreSQL – сумісної з MySQL і PostgreSQL реляційної бази даних, створеної для

надання швидкості та доступності, притаманних найсучаснішим комерційним базам даних, та одночасно забезпечення простоти та економічності, що характеризують БД з відкритим вихідним кодом.

Даний сервіс не надає механізм для відладки застосувань на машині розробника, тому тестування здійснювалося лише на базі даних, що розміщується на серверах Amazon.

AWS Aurora PostgreSQL неможливо розгорнути локально, тому в подальшому будуть порівнюватись AWS DynamoDB та AWS Aurora PostgreSQL з індексацією та без (таблиця 2).

Таблиця 2

Читання з не локальних: AWS DynamoDB та AWS Aurora PostgreSQL з індексацією та без

Кількість записів	**Час, мілісекунди**			
	AWS DynamoDB без індексації	AWS DynamoDB з індексацією	AWS Aurora PostgreSQL з індексацією	AWS Aurora PostgreSQL без індексації
10	277	217	120	121
100	315	241	167	157
1000	898	396	248	300
2000	1842	507	371	445
4000	3978	712	480	793

Аналогічно до результатів локальних AWS DynamoDB з індексацією та без дані з табл. 2 представлені у графічному вигляді на рис. 9.

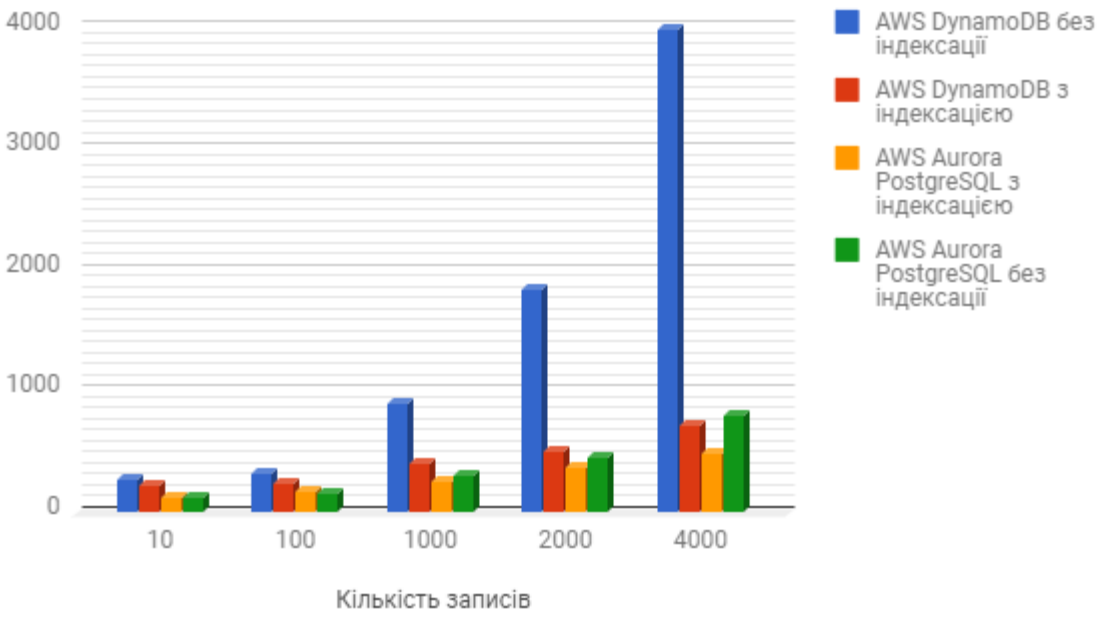

Рис. 9. Графік результатів читання з не локальних: AWS DynamoDB та AWS Aurora PostgreSQL з індексацією та без

Як можна побачити, виконання запитів на локальній машині завжди відбувається швидше, ніж у хмарі Amazon. Індекс не дає суттєвого приросту при невеликій кількості рядків результуючої таблиці. Проте зі збільшенням кількості запитів можна побачити залежність у вигляді зменшення часу опрацювання запиту для таблиці з індексом.

Проаналізовано швидкодію читання даних з використанням сервісів Amazon Dynamo DB, Aurora PostgressSQL з метою дослідження швидкодії роботи. Для обох баз даних тестування виконувалося з використанням індексації та без, на великих та малих об’ємах даних.

Також було підтверджено такі гіпотези, як:

- Стуктурування вхідних даних спростить оброблення та покращить швидкодію обробки великих даних.
- Використання підходу map-reduce – дозволить будувати аналітичні звіти в реальному часі із прямим використанням потоку даних.

В роботі проаналізовано швидкість читання даних з використанням сервісів Amazon Dynamo DB, Aurora PostgressSQL. Для обох баз даних тестування виконувалося з використанням індексації та без, на великих та малих об’ємах даних.

Крім того було проаналізовано кількість проіндексованих стовпців в кожному окремому випадку, та вплив кількості рядків на остаточний результат.

Проаналізувавши результати можна сказати, що виконання запитів на локальній машині завжди відбувається швидше, ніж у хмарі Amazon.

Проаналізувавши дані з таблиці та на графіку для локальної бази даних, можна зробити висновки, що на невеликих об’ємах даних читання таблиці з індексом та без займає приблизно однаковий час. Проте при збільшенні кількості рядків в результатах запиту час доступу до даних у підході з використанням таблиці з індексом суттєво знижується

Проте на машині у хмарі Amazon індекс не дає суттєвого приросту при невеликій кількості рядків результуючої таблиці. Проте зі збільшенням кількості запитів можна побачити залежність у вигляді зменшення часу опрацювання запиту для таблиці з індексом.

2.4. Порівняння сервісів, які опрацьовують запити

Не менш важливу роль у швидкодії сервісу відіграє не тільки база даних як сховище але і сервіси, які опрацьовують запити клієнтів. Тренд останніх років показує що застосунок побудований на мікросервісах має перевагу у швидкодії перед монолітною архітектурою.

Існує декілька сервісів, які надають послуги із використанням Serverless технологій. Найбільшими серед них є Google із сервісом Firebase Function та Amazon AWS Lambda. У роботи пропонується порівняти ці два сервіси.

Сервіси будуть порівнюватися такими характеристиками, як час написання функцій, тестування і розгортання, а також вартість експлуатації та середній час опрацювання запитів.

2.4.1. Дослідження проблем створення функцій сервісів

Для побудови серверної частини застосунку варто розглянути Serveless технологію. Це модель хмарних обчислень, в якій платформа динамічно керує виділенням машинних ресурсів.

Основна відмінність від звичайних серверних технологій полягає в тому, що розробник не займається такими проблемами, як підключення до бази даних, налаштовування портів тощо, а лише функцію, яка відповідає за конкретну задачу. Тоді все застосування складається з декількох функцій, які і надають основні можливості. Розглянемо більш детально створення функції для Amazon AWS Lambda та Firebase Function [9].

Функції з Lambda можна створити двома способами: шляхом прямого доступу до Lambda Console або через Serverless framework. В роботі використовується другий підхід, оскільки це набагато спрощує розгортання функції.

Serverless рекомендує централізувати всю конфігурацію функцій у єдиний файл YML. У нього заноситься назва функції, оскільки вона буде відображатися на консолі Lambda, ім'я функції у кодовій базі та деяка конфігурація на час виконання.

Нижче наведено відповідний фрагмент конфігурації з файлу YML для створення нового користувача (лістинг 1). У даному випадку виконується функція вхідного HTTP-запиту за допомогою методу GET.

Лістинг 1. Фрагмент конфігурації з файлу YML для створення нового користувача.

```
functions:
 getData:
   handler: handler.getData
```

```
    events:
       http:
          path: getData
          method: get
          integration: lambda-proxy
          cors: true
```

Одним з цікавих аспектів AWS Lambda є те, що можна вказувати будь-який тип події, що запускає виклик функції, а не лише вхідний HTTP-запит, надісланий клієнтом. Іншими активатори можуть бути завантаження файлів на S3 (лістинг 2) або розгортання інших служб на AWS [10].

Лістинг 2. Завантаження файлів на S3 або розгортання інших служб на AWS.

```
const dataSource = require('./dataSource);
const helpers = require('./helpers');
const handleError = helpers.handleError;
const handleSuccess = helpers.handleSuccess;
module.exports = function(event, context, callback) {
 const body = JSON.parse(event.body);
 if (!body.uid) {
   return handleError(context, { error: 'Bad Input' });
 }
 const uid = String(body.uid).replace(/[^\d]/g, "");
 dataSource.get({
   uid
 })
   .then(user => handleSuccess(context, { uid }))
   .catch((err) => handleError(context, { error: 'Something went wrong'
}));
}
```

Слід звернути увагу, що Lambda не буде обробляти кодування або декодування JSON, це все необхідно виконати вручну.

Створення проекту з хмарними функціями виявилося набагато простішим. Найбільш поширеним випадком використання Firebase Function

є обробка вхідних HTTP-запитів, тому тут немає великої кількості конфігурацій для маршрутизації конкретної події до певної функції.

Початковий проект розроблено за допомогою CLI Firebase. Інтерфейс CLI створює проект Firebase, який дозволяє розміщувати важливу конфігурацію, а не повністю покладатися на редактор правил консолі.

Всередині файлу Javascript виконується визначення функцій, при цьому кожен експорт вважається функцією (лістинг 3). Фактичне створення функції було набагато простішим (лістинг 4).

Лістинг 3. Визначення функцій

```
exports.getData = functions.https.onRequest(getData);
```

Лістинг 4. Фактичне створення функції

```
const admin = require('firebase-admin');
module.exports = function(req, res) {
  if (!req.body.phone) {
    return res.status(422).send({ error: 'Bad Input' });
  }
  const uid = String(req.body.uid).replace(/[^\d]/g, "");
  admin.getData({ uid })
    .then(data => res.send(data))
    .catch(err => res.status(422).send({ error: err }));
}
```

2.4.2. Порівняння швидкодії сервісів

Також було проведено аналіз швидкодії опрацювання запитів. Це є дуже суб’єктивна характеристика, оскільки на це впливає багато факторів (фізичне розміщення дата центрів, навантаження мережі, апаратне забезпечення що використовується дата цетнрами тощо).

У порівняльному аналізу різницю між двома сервісами майже не було виявлено. З мінімальним розривом краще себе показав Firebase Function (рис. 10).

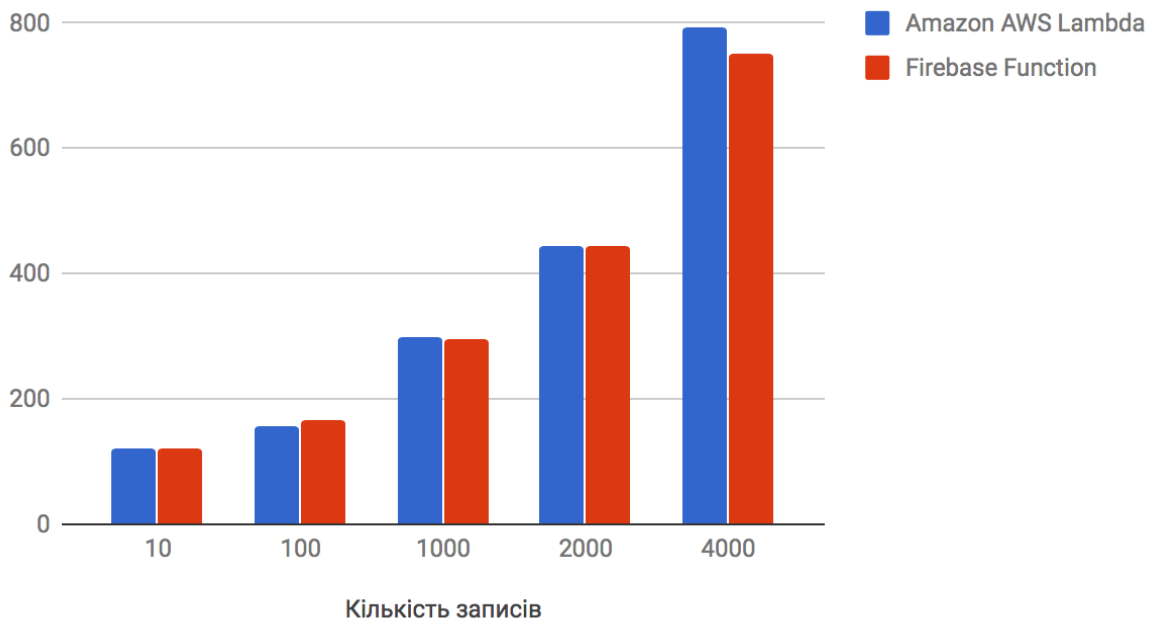

Рис. 10. Графік результатів читання використання AWS Lambda та Firebase Function

2.4.2. Порівняння вартості сервісів

Загалом, можна розрахувати вартість викликів функцій на основі двох показників: кількості викликів і кількості часу, потрібного кожному виклику для виконання.

Вартість послуг Serverless технології формується на основі реальної кількості споживаних ресурсів, а не на передплаченому об'ємі ресурсів.

На момент написання цієї статті функції Firebase коштували $0,40 за мільйон викликів (після двох мільйонів безкоштовних), тоді як вартість функцій Amazon API Gateway становила $3,50 за мільйон викликів (після мільйона безкоштовних). Отже можна зробити висновок, що Firebase Function є майже у дев'ять разів дешевшою за Amazon API Gateway.

2.4.3. Результати порівняння сервісів

У порівнянні за показником зручність використання кращим виявився Firebase. Для нього потрібна менша кількість попередніх налаштувань, а

також характерним є набагато більш приємний інтерфейс. Об'єм конфігурації Firebase менший, оскільки у ньому доступно менше тригерів функцій.

Щодо зручності розгортання обидві технології виявилися майже ідентичними: функції запускаються викликом serverless deploy для AWS та firebase deploy для Firebase з консолі [11].

Для перевірки швидкості відповіді використано застосування Postman. Оскільки затримка мережі могла суттєво вплинути на результати тестування, тестування виконувалося 10 разів та і в якості значення обраховувалося середнє арифметичне отриманих результатів. AWS Lambda показав середнє значення 237 мілісекунд, а Firebase – 245 мілісекунд. Судячи з цих результатів, можна зробити висновок, що обидва сервіси мають однакову швидкість виконання запитів.

2.5. Висновки

Передбачається, що кінцевий результат буде представляти собою програмне забезпечення і буде застосовуватися для аналітики міжнародних торгівельних даних імпорту/експорту між країнами всього світу та Україною.

Запропоноване програмне забезпечення буде виконувати поставлені задачі, використовуючи оптимізацію машинних ресурсів, що прискорить швидкодії обробки запитів, можливості графічної візуалізації результатів запиту.

Основним характеристики рішення:

- Підвищити швидкодію шляхом структурування вихідних даних.
- Зберігання даних в базі даних.
- Використовувати комбіновані методи обробки вихідних даних.

Було проаналізовано методи роботи з великими даними, які можуть бути використані для побудови аналітичних звітів, а також робота з даними у розподілених файлових системах

Було розглянуто основні аспекти побудови аналітичних систем, зокрема вилучення та обробка денормалізованих даних з існуючих сховищ. Можна з впевненістю зробити висновки, що для підвищення швидкодії опрацювання запитів однім із важних етаапів є структурування вхідних даних, які знаходяться в БД.

Дослідження швидкодії роботи різних баз даних, а саме DynamoDB та PostgreSQL показало, що на великих обсягах даних при однакових умовах PostgreSQL виконує запити швидше, ніж DynamoDB. Хоча на невеликих обсях обидві СКБД показували однаковий результат. Але оскільки магістерська дисертація націлена на роботу із великими обсягами даних, було прийнято рішення використовувати PostgreSQL в якості бази даних.

Був проаналізований процес роботи із найпопулярнішими serverless технологіями, таким як Amazon Lambda та Google Cloud Function на основі об’єктивних (часових, фінансових) та необ’єктивних характеристиків (зручність використання). На основі даного дослідження можна сказати, що Google Cloud Function надає кращі сервіси для роботи.

3. АРХІТЕКТУРА СИСТЕМИ

Архітектура системи проста на перший погляд, але все ж є деякі нюанси, які дозволяють стверджувати про застосування складних конструкцій. Перш за все необхідно сказати про загальну архітектуру.

Система є дуже розгалуженою, оскільки використовує дуже широкий спектр технологій: дві бази даних, два REST API та клієнтській мобільний застосунок. Для більш кращого розуміння системи слід розглянути кожен із пунктів окремо.

3.1. Бази даних

Враховуючи попереднє дослідження було вибрано використовувати базу даних PostgresSQL оскільки вона показала найкращі результати. Також була отримана база із HS кодами системи “My Comtrade” у вигляді резервної копії бази даних ArangoDB. Завдяки HS коду можна легко дізнатися правильну митну класифікацію для товарів, які можуть імпортуватись або експортуватись.

Ободві бази даних є абсолютно різними між собою по своїй структурі збереження даних. Так PostgresSQL є реляційною базою даних, а ArangoDB документ орієнтована база даних.

Розглянемо обидві СКБД детальніше.

3.1.1 База даних PostgresSQL

PostgreSQL – це вільно поширювана об'єктно-реляційна система управління базами даних. Ця база даних є найбільш розвинена серед open-source аналогів та становить альтернативу для комерційних рішень.

Переваги PostgreSQL:

- Надійність.

- Продуктивність. Грунтується на використанні індексів, інтелектуальному планувальнику запитів, системи блокувань, системі управління буферами пам'яті і кешування, чудовою масштабованості при конкурентній роботі.
- Можливість розширення. База даних підтримує як горизонтальне так і вертикальне розширення. Також існує велика кількість функцій обробки даних. Таким чином розробник може перенести обробку даних із серверної частини в базу даних що в свою чергу дозволить оптимізувати серверний застосунок.
- Мовою запитів даних є SQL. Це забезпечую простоту у доступі та використання СКБД.
- Багатий набір типів даних.
- Великий набір вбудованих даних та операторі, що довзоляє оперувати даними на рівні бази даних.
- Має широку підтримку кодування виключаючи UNICODE та ASCII. Це довзволяє зберігати широкий набір текстових даних.
- База даних може повідомляти про помилки на більш ніж 20 мовах.[13].

Базу даних було розгорнуто на хмарному сревісі AWS RDS. Це дозволило уникнути проблем пов’язаних із хостингом та підтримкую.

3.1.1 База даних ArangoDB

ArangoDB – відкрита система керування базами даних, що надає різні способи зберігання даних у форматі ключ та значення. Для роботи з базою можна використовувати 2 мові запитів: SQL-подібну або спеціальне розширення мови JavaScript. База даних підтримує горизонтальне та вертикальне маштабованість. Також СКБД має веб інтерфейс який надає функції управління [14].

Так до переваг даної СКБД можна віднести:

- Відсутність заданіої схеми даних. Дані в базі зберігаються у документах із різними форматами даних.
- ArangoDB має вбудовану підтримку доступу до бази через REST/Web API. Таким чином базу даних можна використовувати у ролі сервера.
- JavaScript використовується для запитів до бази даних та виконується безпосередньо на CPU.
- Багатопоточність що дозволяє розподіляти навантаження між ядрами CPU.

Дана база була розміщена на Amazon Elastic Computer Cloud, оскільки сервіс RDS не підтримує дану СКБД.

3.2. Серверна частина

Серверний застосунок надає інтерфейс доступу до даних для побудови графіків та аотодоповнення вводу. Так отраминня даних для побудови графіків здійснюється по HS коду, а автокомпліт за введеним користувачем текстом у полі введення даних.

Для написання серверної частини була вибрана мова програмування Java Script. Для запуску коду викорустовується платформа Node.js.

Node.js – open-source платформа для виконання застосунків, написаних мовою Java Script. Це є основним середовищем виконнання JavaScript коду за виключенням браузера.

Node.js характеризується такими властивостями:

- асинхронна однопотокова модель виконання запитів,
- неблокуючий ввід та вивід,
- система модулів CommonJS,
- рушій JavaScript Google V8 [15].

Для менеджера пакетів використовувався NPM.

NPM (Node Package Manager) – це менеджер заоежностей для мови програмування Java Script. Середовища виконання є Node.js. Менеджер пакунків має однойменний консольний клієнт через який здійснюється доступ до реєстрів пакунків. Окремо слід виділити реєстер пакунків – це база даних в якому містяться дані про усі доступні пакети npm, версії, авториів точки доступу тощо. Також доступні пакунки можна переглядати та шукати через веб-сайт NPM.

Менеджер пакунків та реєстр керуються NPM, Inc.

Для побудови REST API використоввався фркймворк Express.

Express.js, або просто Express – програмний каркас розробки веб-застосунків для Node.js, реалізований як open-sorce проект та публікується під ліцензією MIT. Він створений для побудови створення веб-застосунків. Сам фреймворе має обмежені можливості проте вони можуть бути розширені за рахунок численних плагінів, які можуть бути підключені.

Перевагою фреймворку є те що він не використовує багато ресурсів а його можливості можна розширити завдяки численим плагінам.

Даний стек технологій використовувався для розробки обох прикладних інтерфейсів. Суттєва відмінність між ними полягає в комунікації із базою даних. Пропоную розглянути обидва API більш детальніше.

3.2.1 REST API для побудови графіків.

Для побудови графіків використовулася база даних PostgresSQL. Для комунікації із базою даних використовувався пакет node-pg. Для зручності використання було створено єдину точну доступу, яка надавала інтерфейс доступу до бази даних. За це відповідає модуль executeSQLCode який приймає стрічку запиту SQL у рядку. Враховуючи складність моделі даних було прийнято рішення робити запиту до бази даних використовуючи виключно SQL.

Принцип побудови REST API використовуючи фреймворк Express є дуже простий. Програміст має описати функцію, яка буде опрацьовувати запити API та відправляти відповідь. Для створення API використовується об’єкт app, що створюється функцією Express. Функція Express імпортується із самого фреймворка [16].

Для створення методу REST API використовується той же екземпляр app. Назва методу означає метод запиту, перший параметр отримує стрічку яка відповідає за url. Другий параметр – це функція, яка і буде відповідати за опрацювання запиту. Так приклад можна побачити у лістингу 5.

Лістинг 5. Створення методу REST API

```
app.get('/rest/graph3/data', function(req, res, next) {
  var years = req.query.year.split(',');
  if (!req.query.cmdcode) {
    return res.send([]);
  }
  dataQ(req.query.gnum, req.query.cmdcode, years)
    .then(function(result) {
      var d = {"data":result};
      return res.send(d);
    }).fail(function(error) {
      logger.error(error);
    });});
```

Для запуску сервера використовується метод listen, який отримує в якості параметру порт, який був призначений, та callback функцію, що буде викликана після запуску сервера (лістинг 6). Оскільки дана api буде запущена на хмарному сервісі AWS, іде перевірка на задання системного порта для запуску. Якщо такого значення немає (програма запущена на локальній машині), то використовується порт по замовчуванню 3000.

Лістинг 6. Метод listen, якій використовується для запуску сервера

```
var port = process.env.PORT || 3000;
var server = app.listen(port, function () {
    console.log('Server running at http://127.0.0.1:' + port + '/');
});
```

В подальшому застосунок буде працювати та опрацьовувати запити, які приходитимуть як локально так із-за меж поточного комп'ютера.

Розглянуваний серверний застосунок запущений на хмарному сервісі AWS EC2. За рахунок цього вдалося уникнути проблем із налаштуванням та підтримкою серверного обладнання.

3.2.1 REST API для автодоповнення

Для побудови інтерфесу автодоповнення також використовувався фреймворк express. З основними принципами роботи з ним ми ознайомилися в попередньому розділі.

Своєрідність яка присутня для даного застосунку полягає у взаємодії із базою даних ArangoDB. Для цього завдання використовувався пакет arangojs. За допомогою його виконувалися запити. Приклад такого запиту можна побачити у лістингу 7. Оскільки запит виконується асихронно, то функція повертає promise, який буде опрацьований після завершення запиту.

Лістинг 7. Запіт до ArangoDB

```
module.exports.findName = function(name) {
  name = 'prefix:' + name;

  var query =
'for e IN FULLTEXT(export_nodes,\'name\',@node) for n in
GRAPH_SHORTEST_PATH(\'exports\', {_key:\'v-1\'}, e, {includeData:
true}) return n';
  return getData(query, name, '');
};
function getData(query, param, year) {
  return db.query(query).then(function(cursor) {
    return cursor.all()
  });
}
```

Як і у випадку із REST API для побудови графіків даний серверний застосунок виконується на хмарному сервісі AWS EC2.

3.3. Клієнтський мобільний застосунок

Мобільний застосунок і є кінцевим продуктом розробки. Безпосередньо з чим буде працювати користувач. Для зручності користувача було прийнято рішення розробити кросплатформерний мобільний застосунок.

Для розробки кросплатформеного додатку є цілий комплект технологій. Кожна з яких має свої плюси і мінуси. Варто розглянути кожен спосіб детальніше.

Cordova дозволяє створювати універсальні мобільні застосунки, що працюють на різних мобільних платформах, з використанням стандартних веб-технологій (таких як HTML5, CSS3 і JavaScript). Використання Apache Cordova дозволяє створювати програми, що функціонують на широкому спектрі мобільних платформ [17].

Проте існує ряд недоліків у цій технології.

Виконання програми у веб-переглядачі уповільнює застосунок у порівнянні із рідними програмами.

Якщо з’явилися нова функціональність у одної з мобільних ОС, то пройде певний час поки вона буде підтимуватися фреймворком.

Гібридні застосунки важко розробляти, якщо необхідно використовувати складні функції телефону: push notification, gps тощо.

Переваги:

- один застосунок для декількох мобільних ОС.

Недоліки:

- складність розробки;
- швидкодія.

React Native – фреймворк, що базується на основі React, та дозволяє розробляти мобільні застосунки для ОС iOS та Android.

Мовою програмування є також JavaScript. Основною перевагою є те, що розробник не створює мобільний веб-додаток, гібридний додаток чи HTML5 додаток [18]. Розробник створює застосунок, який ніяк не відрізняється від нативного, що створений із Objective-C чи Java. React Native використовує ті самі функціональні блоки графічного інтерфейсу, що і звичайні iOS та Android додатки. Розробник просто складає ці блоки разом використовуючи JavaScript та React.

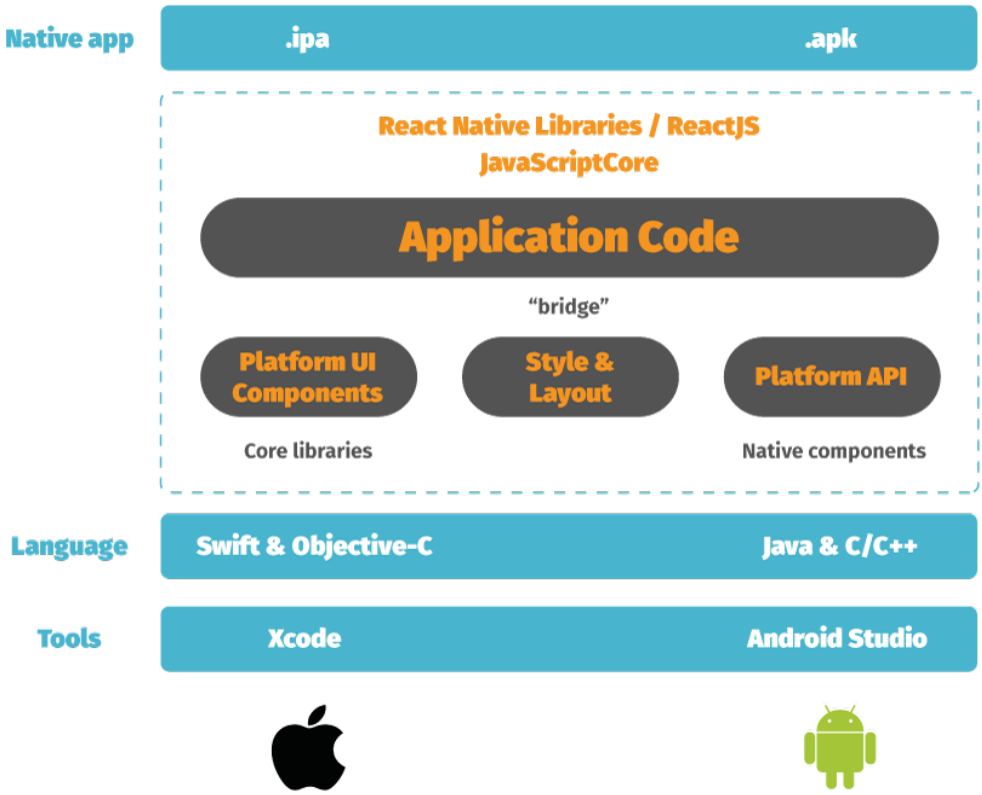

Рис. 11. Загальна структура проекту React Native

Недоліки цієї технології полягають в тому, що нові нативні api можуть бути недоступні у React Native. Для вирішення цього недоліку розробники фреймворку надають можливість розробити недоступні модулі окремо для кожної платформи.

Окремо хотілося б виділити те, що дана технологія, є дуже новою і динамічною на сьогоднішній день. Складність полягає в неідеальній

документації та помилки які часто можна помітити на сторінці проекту в github.

Фреймворк є повністю відкритим проектом, який розробляється спільнотою однодумців та управляється в більшій мірі працівниками Facebook.

Переваги:

- один застосунок для декількох мобільних ОС;
- безкоштовний для використання.

Недоліки:

- динамічність.

Ionic – це платформа із відкритим кодом для розробки мобільних додатків, яка полегшує створення власних та прогресивних веб-програм найвищої якості за допомогою веб-технологій.

Найстаріша версія вийшла в 2012 році і базувалася на AngularJS та Apache Cordova. Найновіша версія фреймворка надає засоби та застосунки, що включають CSS та HTML5.

Додаток може бути створеним із цими веб-технологіями та встановлений на пристрій із використанням Cordova. Також доступне IDE ionic creator для візуального створення додатка.

Безкоштовним є лише базова версія інструменти. Решта інтструментів є платними.

Переваги:

- один застосунок для декількох мобільних ОС.

Недоліки:

- динамічність;
- гібридний веб-застосунок;
- платні інстументи.

Нативний застосунок із на Objective-C/Swift чи Java/Kotlin. Не варто упускати, що кожна із платформ надає інструменти для створення застосунків.

Рідними є додатки які спеціально розробляються для ОС Android чи iOS. Вони мають доступ до усіх можливостей пристрою: gps, акселерометр, гіроскоп тощо. Середовищем розробки для рідних додатків є Android Studio IDE для Android та Xcode для iOS, спеціально розроблена компанією Goggle та Apple для створення застосунків.

Обидві надають декілька мов програмування на вибір. Для android це може бути Java/Kotlin в свою чергу для iOS це Objective-C/Swift. Для будівництва екранів вигляду – мова розмітки XML. У нативних додатках програмний код компілюється у спеціальний байт код а потім виконується на віртуальній машині кожної платформи окремо. Це надає переваги у швидкодії.

Переваги:

- швидкість виконання;
- повний доступ до API пристрою;
- легкість у підримуванні коду.

Недоліки:

- можуть запускатися лише на одній із платформ.

Враховуючи плюси ти мінуси усіх типів застосунків, найгіднішим є React Native. Це надасть гнучкості застосунку та легкість у майбутньому розширенні та використанні.

Таким чином питання з мовою програмування також вирішується.

JavaScript (JS) – скриптова, динамічна, об'єктно-орієнтована мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з

користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та веб-сторінки [19].

React Native використовує специфікацію мови ESM6. ECMAScript – стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262. Найвідомішими реалізаціями стандарту є мови JavaScript, JScript та ActionScript, які широко використовується у Вебі.

JavaScript має багато властивостей об'єктно-орієнтованої мови, але не можна сказати що вони широко застосовуються розробникми. Фреймворк react відмовляється від унаслідування класів а широко застосовує вкладеність об’єктів. Крім того, JavaScript має ряд властивості функціональних мов.

JavaScript має С-подібний синтаксис, але зруштеє має наступні суттєві відмінності:

- об'єкти, з можливістю інтроспекції і динамічної зміни типу через механізм прототипів;
- функції як об'єкти першого класу;
- обробка винятків;
- автоматичне приведення типів;
- автоматичне прибирання сміття;
- анонімні функції.

Існує дуже широкий спектр мобільних пристроїв. Кожен із пристроїв має свій екран із своєю специфікою: розширення, густина пікселів на дюйм тощо. Через це досить гостро стоїть питання відображення вигляду екрану на різних пристроях. Для створення графічних елементів React Native використовує JSX. Компоненти React зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP, XHP.

Для побудови мобільного застосунку були використані готові рішення, слід розглянути найбільш важливі із них.

3.3.1. Бібліотека React-navigation

React Navigation надає максимальний відсоток для реалізації нативної навігації для обох платформ iOS та Android. JavaScript API – є простим і крос-платформенним – кожен компонент навігації має у своїх властивостях об’єкт за допомогою якого здійснюється навігація: відкривається новий екран, повертається на попередній екран, змінюється вибрана вкладка чи відкривається бокове меню [20].

Дана бібліотека повністю сумісна із Redux, таким чином навігація відбувається в створювачах подій Redux.

Необхідно описати ієрархію екранів в об’єкті JSON та передати її батьківський компонент. Після цього використовуючи ті самі ключі, які були зазначені у описі можливо відкрити новий екран. У лістингу 8 наведений ієрархію екранів застосунку.

Лістинг 8. Опис ієрархії екранів

```
import { StackNavigator } from 'react-navigation';
import UkraineAndOtherScreen from '../screens/UkraineAndOtherScreen';
import ChartScreen from '../screens/ChartScreen';

const RootStack = StackNavigator(
  {
    UkraineAndOtherScreen: {
      screen: UkraineAndOtherScreen,
    },
    ChartScreen: {
      screen: ChartScreen,
    },
  },
  {initialRouteName: 'UkraineAndOtherScreen',
  }
);
export default RootStack;
```

Для додавання екрану в стек необхідно зазначити ключ, що був вказаний у об’єкті вище.

Дана бібліотека поширюється за умовами ліцензії BSD-2-clause та є вільною у використанні.

3.3.2. Бібліотека react-native-charts-wrapper

Для побудови графіків кожна із платформ має готові високофунціональні рішення. Для Android це MPAndroidChart, а для iOS – Charts. Таким чином дана бібліотека надає єдину точку доступу до побудови графіків на обох платформах.

Для цього проекту використовувався графік типу BarChart за пободову тф реалізацію його відповідає однойменний компонент.

За побудову графіку в додатку відповідає компонент ChartComponent. Як і для любого компонента react відображення графічних даних відбувається у методі render. У лістингу 9 наведений метод render.

Лістинг 9. Метод render, що відповідає за побудову графіку в додатку

```
render() {
     const dataForShwoing = {
       data: {
         dataSets: [{
           values: this.props.data,
           label: 'Bar dataSet',
           config: {
           color: processColor('#d0d0e2'),
           barSpacePercent: 40,
           barShadowColor: processColor('lightgrey'),
         }
         }],
       },
       xAxis: {
        valueFormatter: this.props.countries,
        granularityEnabled: true,
        granularity : 1,
      }
     };

     return (
       <View style={{flex: 1}}>

         <View style={styles.container}>
           <BarChart
             style={styles.chart}
```

```
            data={dataForShwoing.data}
            animation={{durationX: 2000}}
            legend={this.state.legend}
            gridBackgroundColor={processColor('#ffffff')}
            drawBarShadow={false}
            drawValueAboveBar={true}
            drawHighlightArrow={true}
            xAxis={dataForShwoing.xAxis}
            onChange={(event) => console.log(event.nativeEvent)}
          />
        </View>
      </View>
    );
  }
```

Так дані для відображення передається компоненту у властивостях. Дані формуються у необхідній для відображення формат та зберігаються в об’єкті dataForShowing. В блоці return повертаються JSX компоненти із заданим стилем та форматом даних.

3.3.3 Реалізація застосунку

Застосунок складається із двох екранів: екрану вибору товару закупівлі та графіку. Останній уже був описаний при розгляді бібліотеки для побудови графіків. За вибір елемента відповідає компонент UkraineAndOtherCountryScreenComponent. Давайте розглянемо його детальніше.

Для відображення списку автодоповнення використовувався компонент AutocompleteComponent. Він здійснює запит до апі після кожної зміни введеної інформації та відображає варіанти у списку під текстовим полем.

Після введення інформації викликається функція queryTextHasBeenChanged яка запитує варіанти для автодоповнення та змінює стан компонента (лістинг 10).

Лістинг 10. Компонент автодоповнення

```
queryTextHasBeenChanged(text) {
 fetch('http://dyplomachart-env.guqvteptpb.eu-central-
1.elasticbeanstalk.com/rest/auto/' + text)
      .then(function(response) {
        console.log("Autocomple Response:");
        console.log(response);
        return response.json();
      })
      .then(function(myJson) {
        this.setState({data: myJson});
      }.bind(this));
  }
```

Після зміни стану компоненту викликається функція render яка перемальовує базуючись на новому стані. У return стейтменті методу render повертаються JSX компоненти із заданими стилем та даними. Також там присутні функції для відображення кожного варіанту автодоповнення та розділювача між ними. У лістингу 11 наведений метод render компоненту автодоповнення.

Лістинг 11. Метод render компоненту автодоповнення

```
render() {
    return (
      <Autocomplete
          inputContainerStyle={{
            paddingLeft: 4, paddingRight: 4,
            marginLeft: 20, marginRight: 20}}
          style={{height: 40, width: 300}}
          renderSeparator={_ => <View
      style={{
                  height: 1, backgroundColor: '#ffffff',
                  marginLeft: 20, marginRight: 20}}
          />}
          renderItem={obj => (
    <View style={{padding: 4, marginLeft: 20,
      marginRight: 20, backgroundColor: '#ffffff'}}>
              <TouchableOpacity onPress={() => {
                this.setState({selectedItem: obj.term,
```

```
                inputValue: obj.term, data: []});
              this.props.callback(obj);
            }}>
              <Text style = {{fontSize: 18}}>{obj.term}</Text>
            </TouchableOpacity>
          </View>)
        }
        data={this.state.data}
        renderTextInput={_ => (
            <View style={{backgroundColor: '#ffffff'}}>
            <TextInput
              style={{height: 50, marginLeft: 40 }}
              value={this.state.inputValue}
              onChangeText={text => {
                this.setState({inputValue: text});
                this.queryTextHasBeenChanged(text);
              }}
            />
          </View>)
        }
      />);
  }
```

Після того як користувач вибрав варіант для перегляду відправляється запит до REST API для побудови графіків. Дані формуються у необхідний формат та відкривається екран побудови графіка із отриманими даними. Для цього використовується функція requestDataForChart куди передається об’єкт який вибраний із списку. Окрім текстових даних із API автодоповнення отримуються ідентифікатор продукту, базуючись на ньому робиться запит даних для побудови графіку. У лістингу 12 наведена функція запиту даних для побудови графіка.

Лістинг 12. Функція запиту даних для побудови графіка

```
 requestDataForChart(obj) {
const chartUrl = 'http://dyplomachart-env.guqvteptpb.eu-central-
1.elasticbeanstalk.com/
rest/graph3/data?gnum=202&year=2014&cmdcode=73'
    console.log(chartUrl + obj.code);
    fetch(chartUrl + obj.code)
```

```
.then(function(response) {
  return response.json();
})
.then(function(myJsonArr) {
  const reduceFunction = (accumulator, currentValue) => {
    const intValue = parseInt(currentValue.contract_volume);
    if (accumulator[currentValue.country_to]) {
      accumulator[currentValue.country_to] += intValue;
    } else {
      accumulator[currentValue.country_to] = intValue;
    }
    return accumulator;
  };
  const dictValue = myJsonArr.data.filter(item =>
      item.country_from === 'Україна')
    .reduce(reduceFunction, {})
  this.props.navigation.navigate('ChartScreen', dictValue)
}.bind(this));
```

3.4. Висновки

Враховуючи попереднє дослідження було вибрано використовувати базу даних PostgresSQL, оскільки вона показала найкращі результати. Також була отримана база із HS кодами системи “My Comtrade” у вигляді резервної копії бази даних ArangoDB.

Для написання серверної частини була вибрана мова програмування Java Script. Для запуску коду використовується платформа Node.js.

Мобільний застосунок і є кінцевим продуктом розробки. Враховуючи плюси ти мінуси усіх типів застосунків, було обрано React Native. Це надасть гнучкості застосунку та легкість у майбутньому розширенні та використанні.

Отже було розглянуто основну архітектуру системи та сформульовано основні методичні підходи до розроблення структур баз даних, серверної частини та клієнтського мобільного застосуноку для розв’язання задач побудови аналітичних звітів.

4. РОЗГОРТАННЯ СИСТЕМИ ТА ТЕСТУВАННЯ

4.1. Розгортання системи

4.1.1. Розгортання бази даних

Для хостингу бази даних був вибраний сервіс Amazon RDS. Завдяки цьому ресурсу вдалося уникнути проблем із підтримкою апаратного забезпечення.

Amazon Relational Database Service або Amazon RDS – це розподілена реляційна база даних від Amazon.com. Це хмарний сервіс, який забезпечує користувачів реляційних базами даних для використання в додатках. Amazon RDS дозволяє здійснювати швидке розгортання, просте обслуговування та легке масштабування. Такі складні процеси, як оновлення програмного забезпечення БД, проведення резервного копіювання, повернення до ранніх станів (відновлення) проводяться автоматично.

Найпростіший спосіб створити екземпляр DB – це використання консолі RDS. Після створення примірника БД ви можете використовувати стандартні утиліти SQL клієнта для підключення до екземпляру DB, наприклад таку як утиліта pgAdmin.

Після вибору в консолі сервісів RDS створення екземпляру та завдання рушія PostgresSQL слід перейти до більш детальних налаштувань:

- License Model – postgresSQL має лише одне ліцензію, яка вже вибрана.
- DB Engine Version – версія PostgresSQL яка буде використана.
- DB Instance Class – це тип машини, яка буде використовуватися для бази даних. Так Amazon надає дуже широкий вибір. Для економії коштів слід вибрати db.t2.micro.

- Multi-AZ Deployment – слід встановити так для продакшин, що надасть більшу стійкість.
- Storage Type – тип жорсткого диску, для надійності слід вибрати SSD.
- Allocated Storage – тут слід вказати розмір сховища для БД.

Останнім параметрами має бути заданий ім'я користувача, що матиме root права та пароль до його. Використовуючи цей логін і пароль можна буде підкоючитися до бази даних.

Після створення екрепляра бази вона буде доступна у консолі AWS RDS у вкладці Inseances (рис. 12). Поле status позначає поточний статус бази даних.

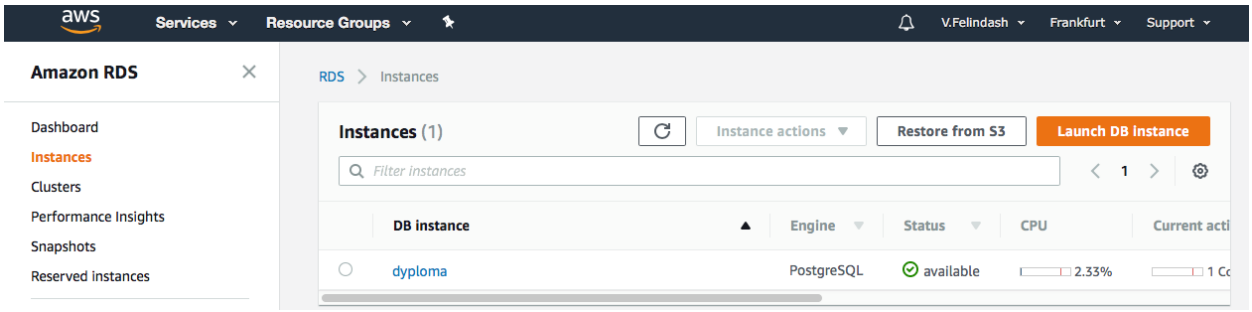

Рис. 12. База даних, яка доступна у консолі AWS RDS у вкладці Inseances

Після вибору новоствореного insteance має відкритися консолі із моніторингом та налаштуванням (рис. 13).

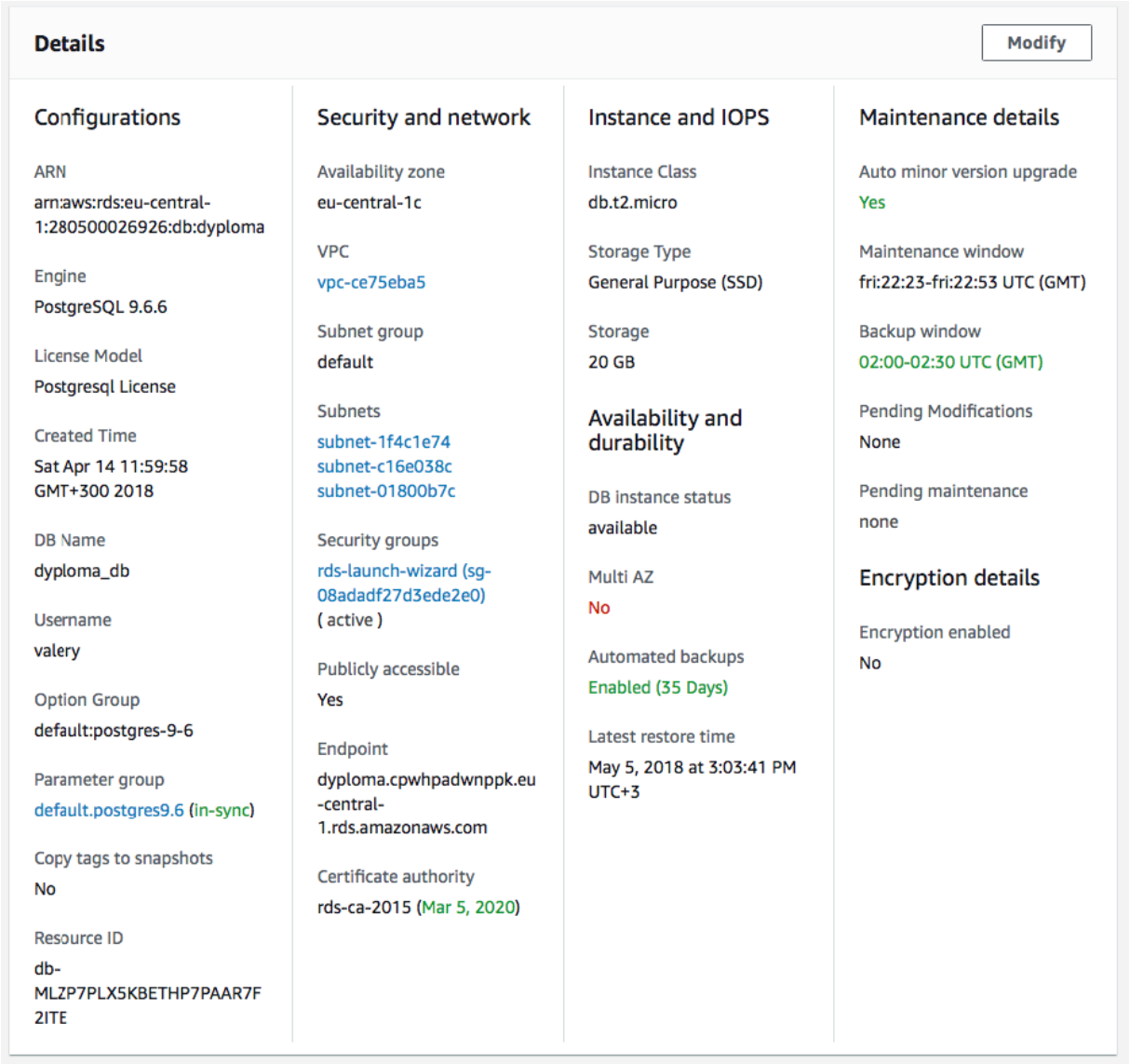

Рис. 13. Налаштування нового insteance для бази даних

Проте після створення екземпляру підключитися до неї не вдасться. Сервіси Amazon мають дуже строгі вимоги до приватності Insteance. Так для налаштування доступу слід вибрати Security Group в деталях та добавити Inbounde Rules з IP адресом машини, з якою буде здійснюватися доступ (рис. 14).

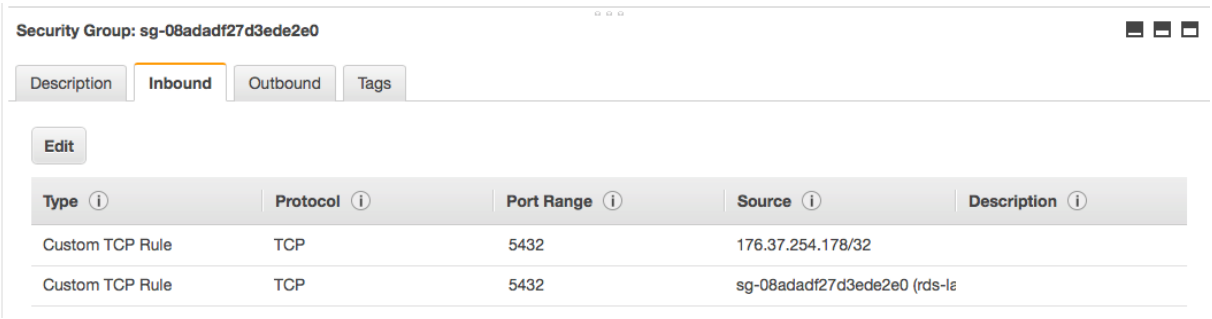

Рис. 14. Налаштування security group для бази даних

Після цього до ресурсу можна буде підключитися, використовував утиліту pgAdmin (рис. 15).

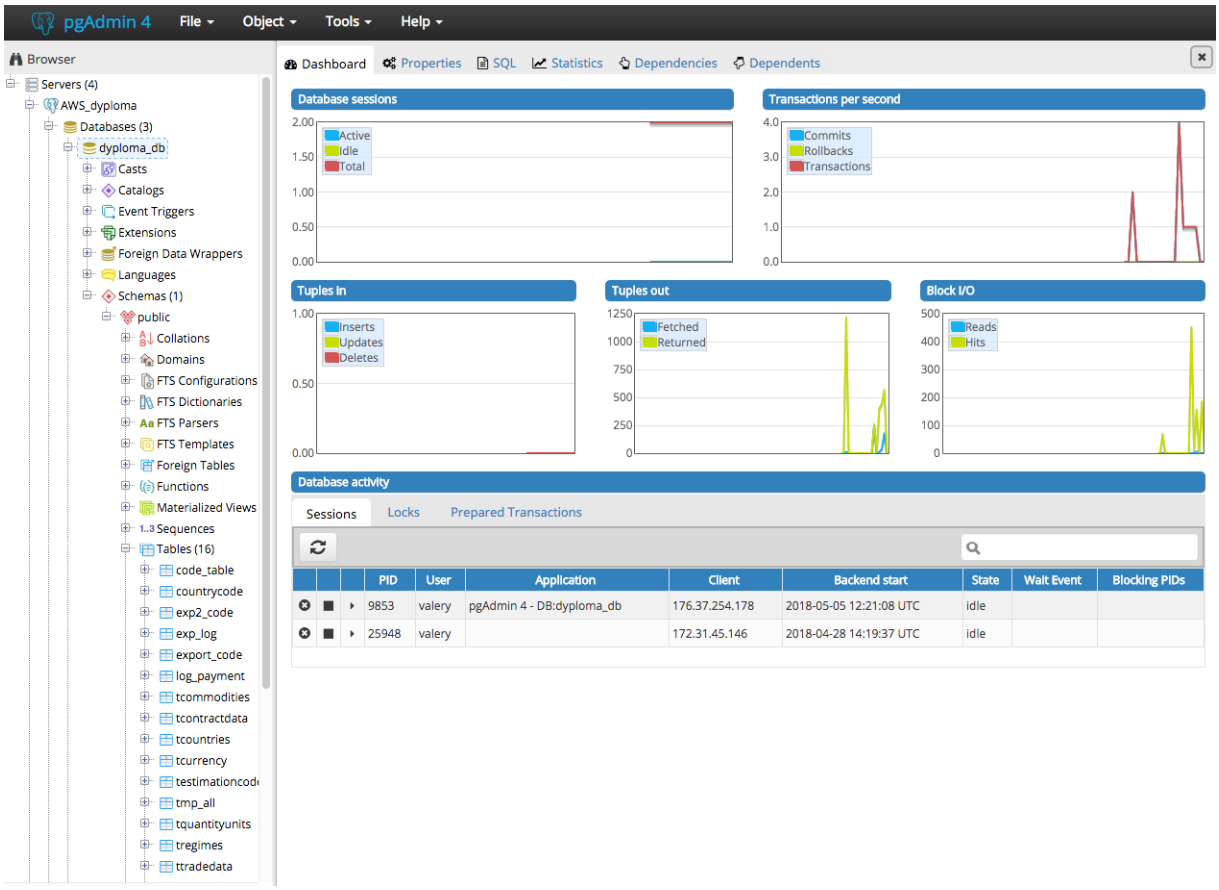

Рис. 15. Утиліта pgAdmin для підключення до ресурсу

4.1.2. Розгортання бекенду сервісу

Для хостингу сервісу бекенду було вибрано сервіс Amazon EC2. Це є хмарний сервіс, який надає комп'ютер, на якому буде виконуватися програмне забезпечення. Amazon має зручніший сервіс для розгортання

хмарних рішень, мова іде про Elastick Beanstalk. Даний сервіс є автомаштабований та вирішує більшість питань із налаштуваннями виконуваного середовища. Але через специфіку бази даних ArangoDB ним скористатися нам не вдасться.

Amazon Elastic Compute Cloud (EC2) забезпечує високу обчислювальну потужність у веб-обладнанні Amazon Web Services. Використання Амазон EC2 усуває необхідність вкладати кошти в апаратну частину, щоб надає змогу швидше розробляти та розгортати програми. Amazon EC2 дозволяє збільшувати або зменшувати обсяги змін у вимогах або збільшення попит, що зменшує потребу в прогнозуванні трафіку.

Користувач може створити та запустити екземпляр в консолі EC2. При створення варто вказати тип екземпляру, щоб дуже означає, які характеристики віртуальна машина буде мати.

Після запуску також варто налаштувати приватність. Це необхідно для того, щоб даний екземпляр міг мати змогу підключитися до бази даних PostgresSQL. Для слід задати той самий security group, що вибраний для бази даних.

Останнім кроком у створенні є завдання ключа доступу Без його у розробника не буде можливості доступу до створеного екземпляру. Ви можете створити новий ключ, або використовувати існуючий. Варто помітити, що після першого запуску ключ не може бути створений або доданий.

Після завершення стартових налаштувань можна запускати машину використовуючи кнопку launch. Розгортання та запуск віртуальної машини може зайняти якийсь час.

Після завершення в консолі можна буде знайти список усіх віртуальних машин, який зображений на рис. 16.

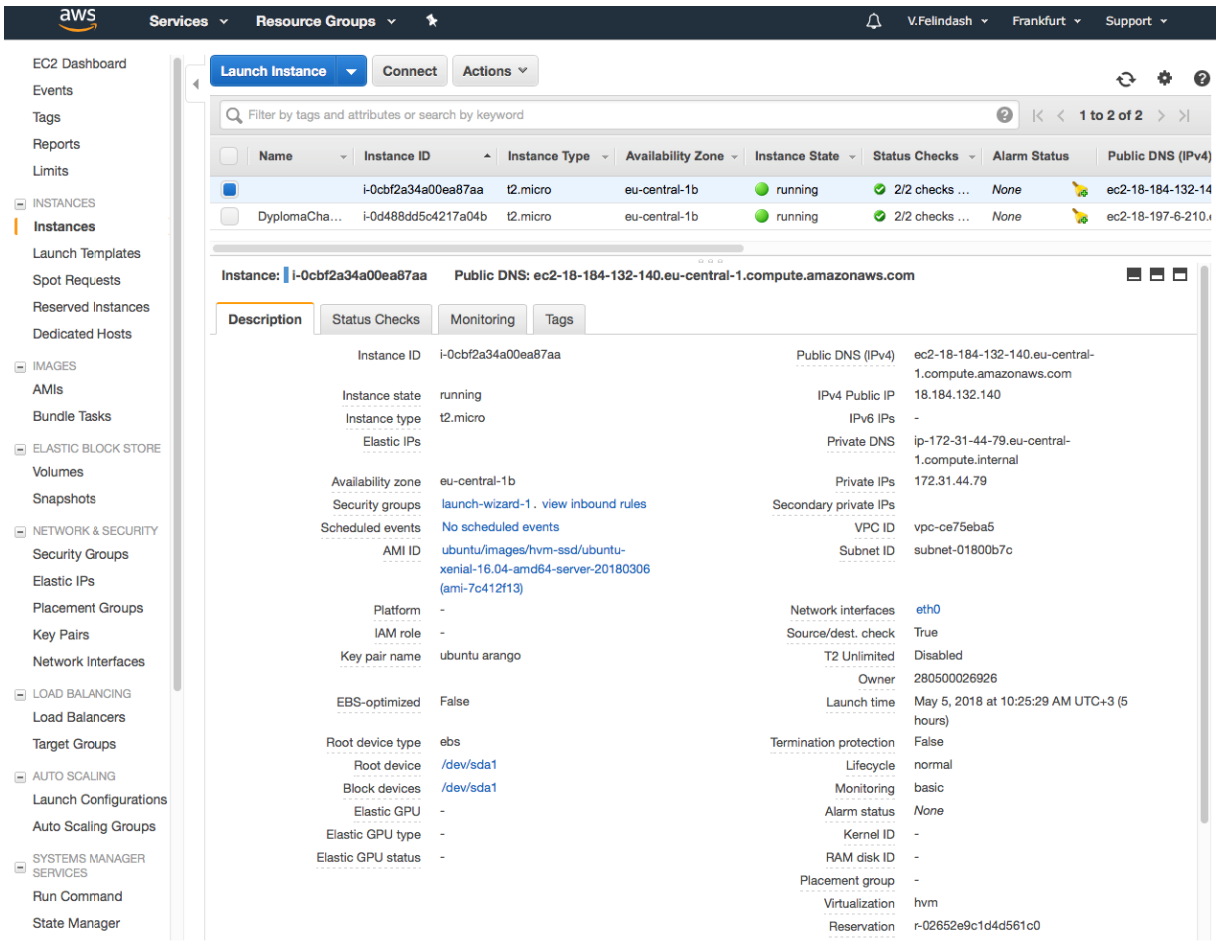

Рис. 16. Список усіх віртуальних машин, а також їх налаштовування

Для доступу до створеної віртуальної машини слід використовувати утиліту SSH. Дана утиліта є доступною у кожній операційній системі сімейства UNIX та використовує однойменний протокол для віддаленого керування операціною системою. Дані що передаються повністю шифруються. Крім того, протокол дозволяє не тільки використовувати безпечний віддалений shell на машині, але і туннелювати графічний інтерфейс, тобто X Tunnelling. Проте це не є необхідним для даної задачі.

Для підключення необхідно мати приватний ключ та знати публічний DNS адрес віртуальної машини. Приватний ключ був нами створений при ініціалізації екземпляру, а публічний DNS можна знайти в консолі AWS. Виклик утиліти SSH має виглядати наступним чином (лістинг 13).

Лістинг 13. Виклик утиліти SSH

```
ssh -i <private-key>.pem <user>@<public-DNS>
```

Після виконання команди ви матимете доступ до хмарної машини (рис. 17).

Рис. 17. Доступ до хмарної машини

Для запуску програмного забезпечення слід встановити базу даних ArangoDB та середовище виконання node. Це можна здійснити за допомогою системної утиліти apt-get.

Для передачі файлів можна використовувати той же протокол SSH, проте доведеться це робити з іншою утилітою scp. Для неї також необхідний приватний ключ та публічне DNS ім'я.

За допомогою цієї утиліти слід передати бекап бази та серциві коди застосунку. Після встановлення залежностей застосунок готовий до роботи.

Проте сервіс не може бути доступний за звичайнім протоколом http. Для цього слід змінити налаштування приватності для екземпляра, на якому запущене програмне забезпечення. В security groups треба добавити правило для протокола http та підмаски 0.0.0.0/0 (рис. 18).

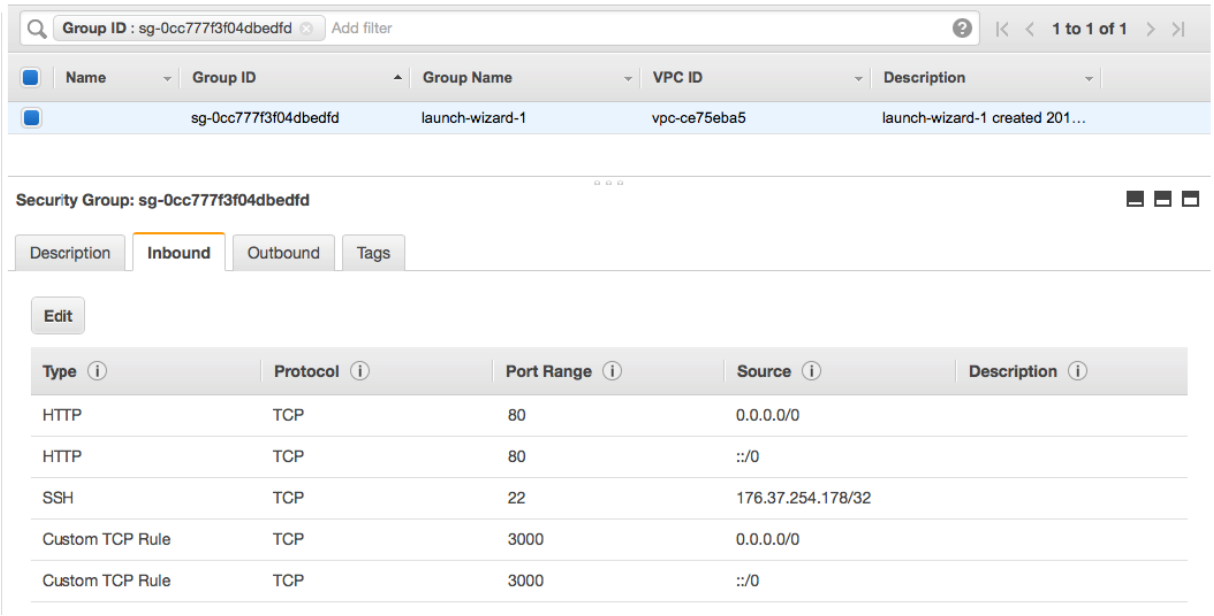

Рис. 18. Правило для протокола http та підмаски 0.0.0.0/0

Після цього сервіс буде доступний публічно на рис. 19 зображено результат запросу до сервису, що розгорнутий на Amazon.

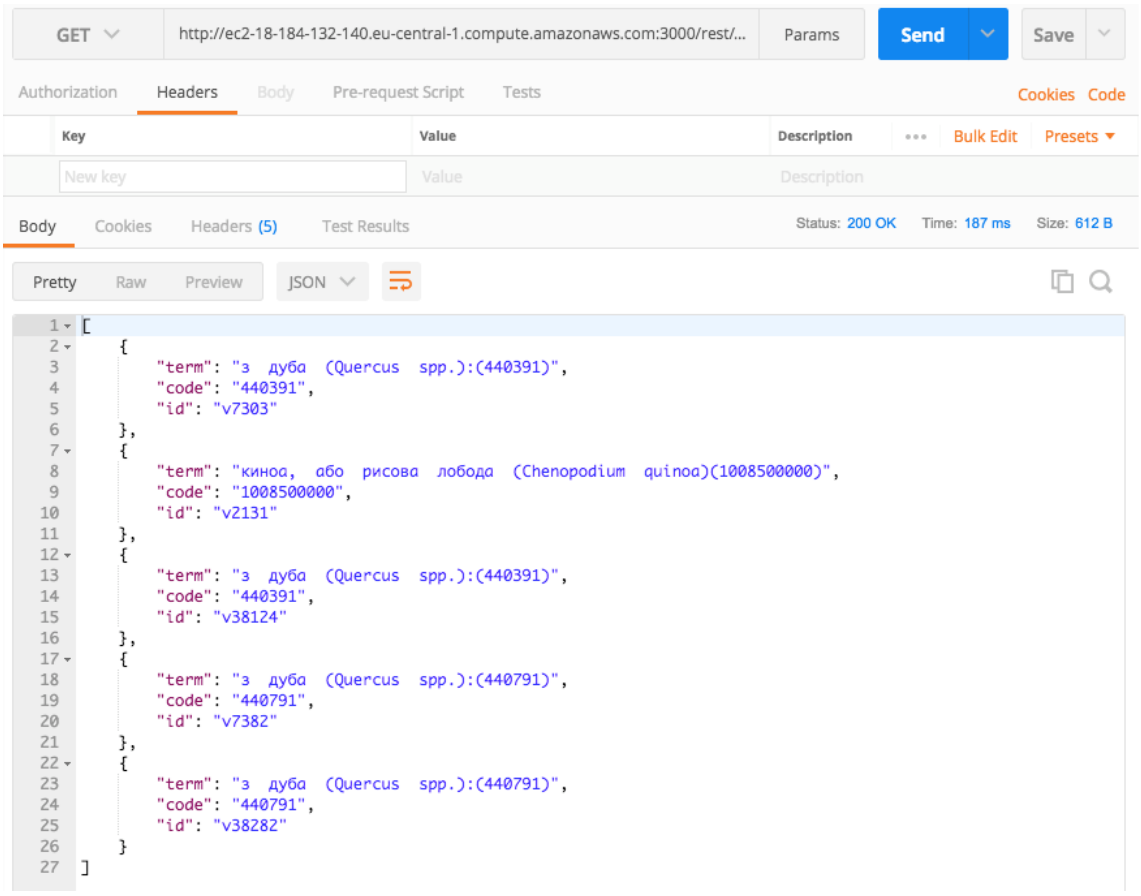

Рис. 19. Результат запросу до сервису, що розгорнутий на Amazon.

4.1.3. Розгортання мобільного застосунку

Оскільки мобільний застосунок був розроблений з використанням фреймворку ReactNative є можливість розмістити його в магазині додатків Play Market або Apple Store. Був вибраний магазин Apple Store із-за популярності планшетів компанії Apple.

Для публікації додатку розробнику необхідно бути членом програми розробників. У веб-інтерфейсі iTunes Connect на вкладці my Application необхідно створити додаток для публікації та заповнити необхідні дані (рис. 20).

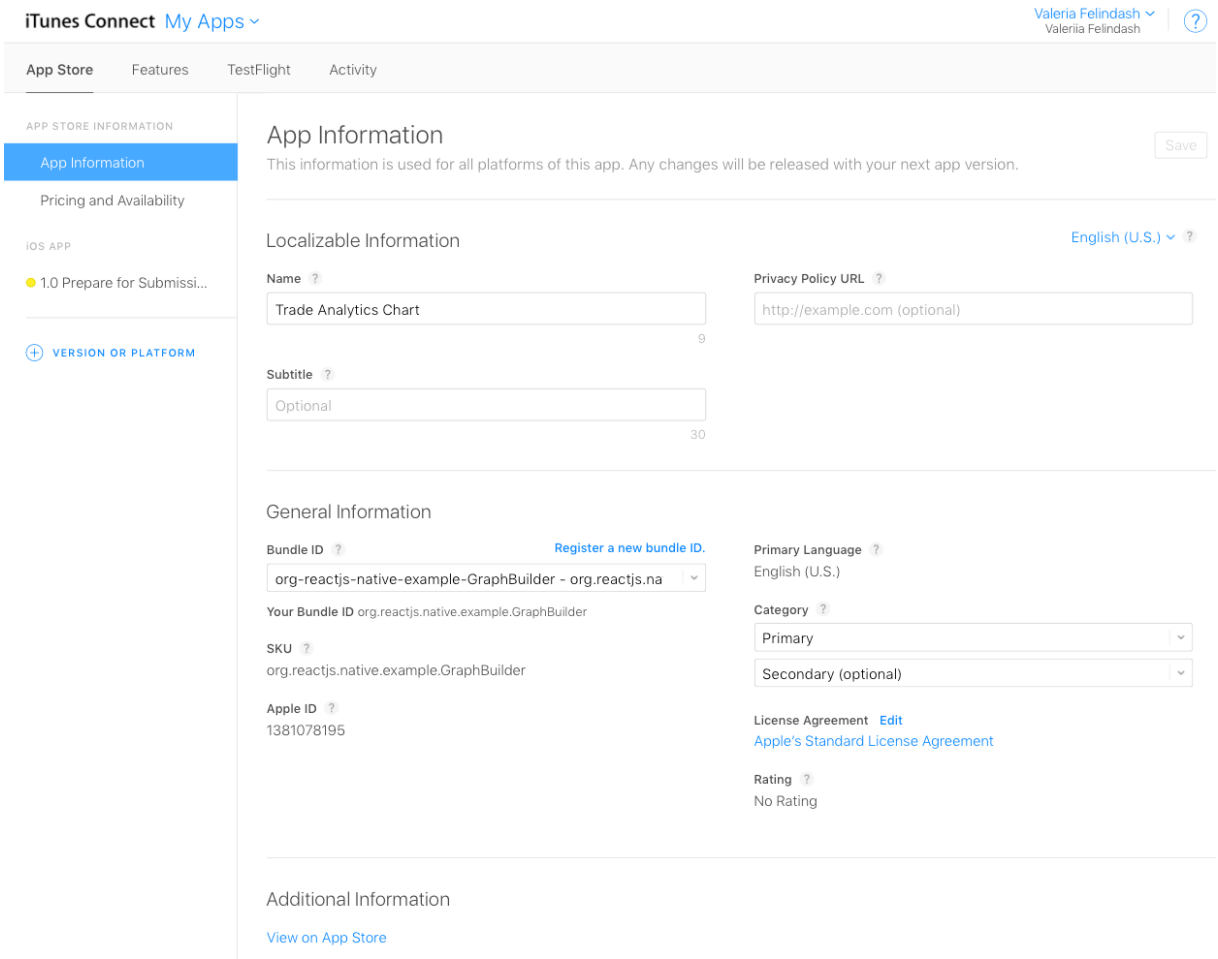

Рис. 20. Створення додатку для публікації в Apple Store

Кожен дадоток та кожна версія оновлення проходить ретельний огляд. Це робиться з метою того, щоб не допустити розміщення контенту, який

суперечить політиці компанії. Для проходження огляду необхідно вказати контактне обличчя та підтвердити відсутність протиправного контенту.

Для розміщення додатку в магазині додатки також слід описати застосунок та надати скріншоти застосунку (рис.21).

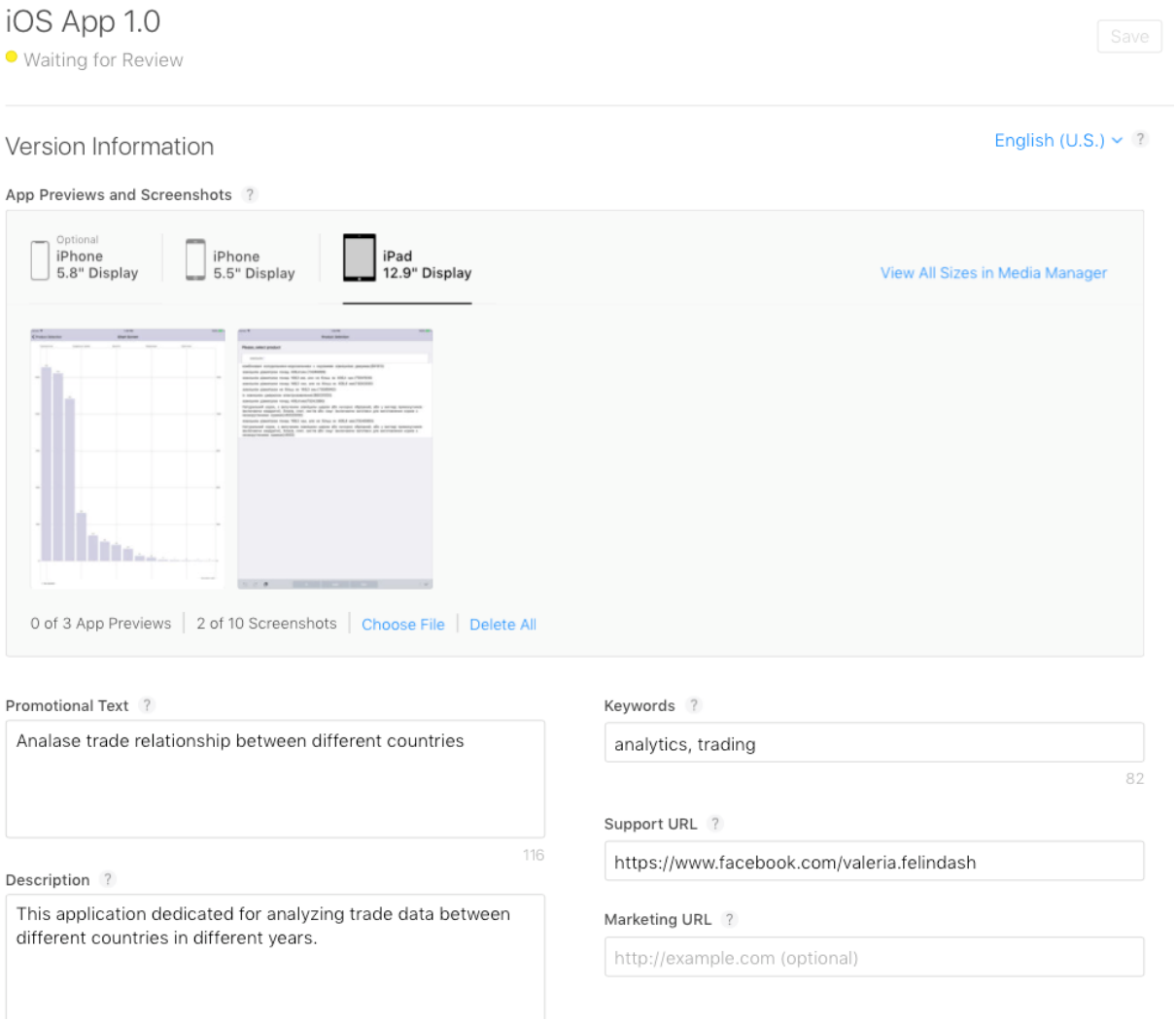

Рис. 21. Інформація про застосунок

Для завантаження самого додатку у магазин слід скористатися середовищем розробки Xcode. Оскільки фреймворк React Native має свої особливості розробки, необхідно налаштувати відповідне середовище. Ці кроки детально описані не будуть.

Попередньо до Xcode слід приєднати аккаунт розробника, від чийого імені додаток буде опублікований в Apple Store. Це можна зробити в Menu > Preferences > Accouns.

Для створення білда у верхньому меню необхідно вибрати Product -> Archive. Через декілька хвилин білд буде готовий для завантаження. Після його створення з'явиться утиліта органайзер білдів, з чиєю допомогою можна буде завантажити додаток в AppleStore.

Ця ж утиліта підписує додаток та усі залежності та завантажує програму в AppleStore (рис. 22).

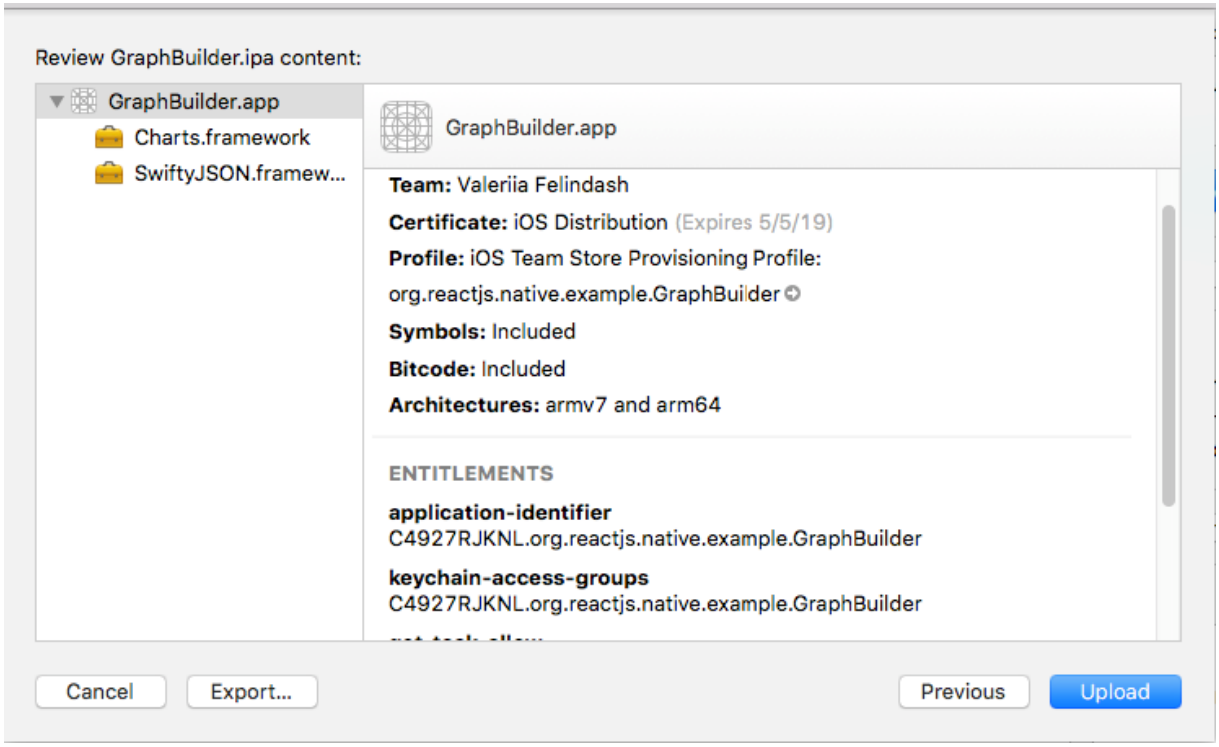

Рис. 22. Підписання та завантаження додатку

Після того, як пройде огляд додаток буде доступний для публічного завантаження в Apple Store (рис. 23)

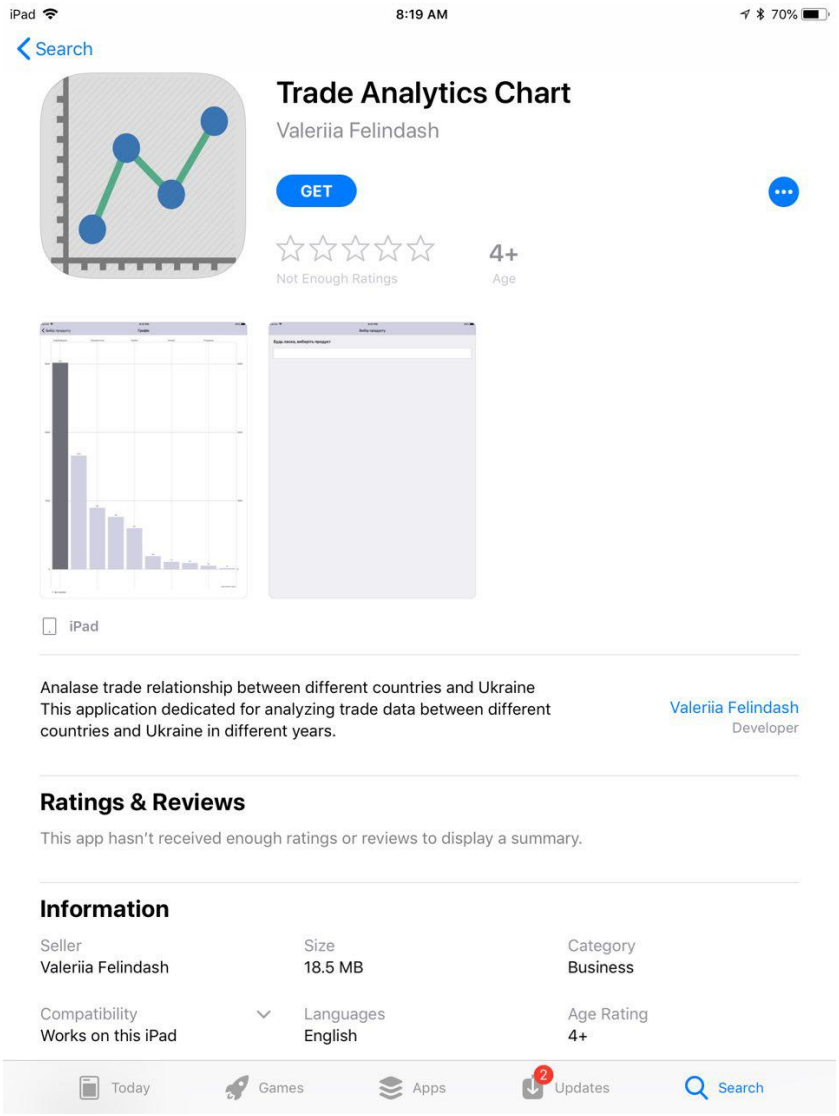

Рис. 23. Завантажений додаток в Apple Store

4.2. Тестування системи

Для коректної роботи системи необхідно протестувати функіональні можлівості. Для цього була проаналізована основна класифікація видів тестування, було обрано необхідні тести в рамках системи, прогенеровано тести різних видів, а також реалізованно їх системі.

4.2.1. Димове тестування

Димове тестування у тестуванні програмного забезпечення означає мінімальний набір тестів на явні помилки. Цей тест зазвичай виконується

самим програмістом. Програму, що не пройшла такий тест, немає сенсу передавати на глибше тестування [21].

Перш за все потрібно провести тестування на проходження основного шляху використання програми. Тобто потрібно використати головні функції системи і перевірити, чи вони коректно працюють. Тестувати дані функції потрібно, використовуючи мобільний пристрій:

- вмикаємо застосунок і бачимо поле для вводу даних;
- починаємо вводити дані;
- з’являється список із можливими кодами та товарами;
- вибираємо один із товарів;
- після відкриття нового екрану бачимо графік із даними про країни, з якими Україна мала найбільші торги;
- перевірка маштабування графіку.

Отже, димове тестування основного функціоналу пройшло успішно. Всі інші функції під час даного тестування не розглядаємо, оскільки вони є менш важливими.

4.2.2. Функціональне тестування

Мета функціонального тестування – виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог.

Функціональні тести повинні охоплювати всі реалізовані функції з урахуванням найбільш ймовірних типових помилок. Тестові сценарії, що поєднують окремі тести, орієнтовані на перевірку якості розв'язку функціональних задач.

Функціональні тести створюються за зовнішніми специфікаціями функцій, проектної інформації і за текстом на МП, що стосуються його функціональних характеристик і застосовуються на процесі комплексного

тестування й іспитів для визначення повноти реалізації функціональних задач і їхньої відповідності вхідним вимогам [22].

Тестування для даної реалізації слід проводити на декількох пристроях із різними форматами екранів в портретному та горизонтальному режимі. Для того щоб покрити якомога більше можливих форматів екрану слід використати апарати із форматами планшетними та мобільними форматами екранів.

Окремо варто звернути увагу на сценарії які не покриваються димовим тестуванням проте можуть мати місце при використанні застосунки:

- введення даних невадідних даних із передуючими пробілами та пробілами після вводу даних;
- перевірка частоти відправки запитів;
- побудова графіків при отриманні невалідних даних із сервера (хоч такий випадок і є малоімовірним, одна на його варто звернути увагу також).

Функціонлаьне тестування пройшло успішно. Було виявлено та виправлено декілька дефектів.

4.2.3. Модульне тестування

Модульне тестування – це процес тестування програмного забезпечення, який полягає в окремому тестуванні кожного та всіх модулів коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні – інтерфейс, клас.

Модульні тести, або Unit-тести, розробляються в процесі розробки програмістами та, іноді, тестувальниками білої скриньки (white-box testers).

Зазвичай Unit-тести застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку [23].

Метою модульного тестування являється ізоляція кожної частини програми та впевненість у тому, що кожна окрема частина (тобто модуль) є коректною. Модульний тест забезпечує детальний контроль, за яким має працювати тестований код. Як результат, це надає деякі переваги. Насамперед модульне тестування допомагає знайти помилки раніше в циклі розробки програмного забеспечення, що робить розробку більш дешевшою та швидшою.

Модульне тестування дозволяє програмісту, коли він буде змінювати код (проводити рефакторинг) бути впевненим, що модуль працює вірно (регресивне тестування). Оскільки модульне тестування вимагає написання тестів для всіх функцій та методів у програмі, помилки швидко локалізуються та виправляються.

Модульне тестування дипломного проекту здійснювалося за допомогою фреймворка для тестування Jest. Він допомагає налаштувати середовище для тестування та проводити тести. Даний фреймворк не є унікальним тільки для React Native його також застосовують для тестування будь яких додатків які побудовані за допомогою технології React.

Фреймворк Jtest доданий до проекту по замовчуванню, це відбувається при ініціалізації проекту для будь якого React Native починаючи з версії 0.38.

Так у даному дипломному проекті модульними тестами покриті усі виклики Api. Для того щоб не робити запити до реального сервера кожного раз як відбувається тестування було задану стандартну відповідь для кожного із запитів. Це зроблено за допомогою використання бібліотеки Jtest (лістинг 13).

Лістинг 13. Тестування запиту для отримання списку товарів

```
import get Facts from '../actions';
```

```
describe('#getFacts() using Promises', () => {
  it('should load facts data', () => {
    return getFact()
    .then(data => {
      expect(data).toBeDefined()
      expect(data.entity[0].name).toEqual('Test 1')
      expect(data.entity[1].name).toEqual('Test 2')
      expect(data.entity[2].name).toEqual('Test 3')
      expect(data.entity[0].photos.length).toEqual(3)
      expect(data.entity[1].photos.length).toEqual(1)
      expect(data.entity[2].photos.length).toEqual(5)
      expect(data.entity[0].coords).toBeDefined()
      expect(data.entity[1].coords).toBeDefined()
      expect(data.entity[2].coords).toBeDefined()
    })
  })
})
```

4.4.4. Інтеграційне тестування

Інтеграційне тестування (англ. integration testing) – це така частина тестування ПЗ, під час якої окремі модулі програми інтегруються один з иншім та тестуються разом, у взаємодії.

Інтеграційне тестування виконується після модульного тестування та перед верифікацією та валідацією програмного забеспечення. Якщо розглядати цей процес як систему, то на вхід їй подаються модулі, які вже пройшли модульне тестування; потім модулі групуються в більші частини, виконуються тести передбачені розробніком, а на виході системи – інтегрована система, що готова до наступного етапу тестування.

Таким чином було реалізоване тестування зміни стану додатку та перевірки правильного відображення змін. Для цього створювалася подія для зміни графіку і очікувалося оновлення графічного інтерфейсу.

Основною метою інтеграційного тестування є перевірити видповідальність вимог до функціональних можливостей, продуктивності,

надійності до основних компонентів програми. Ці компоненти (а саме групи модулів), тестуються методом чорної скриньки, успішні та неуспішні тест-кейси симулюються відповідними вхідними параметрами [24].

Якщо у процесі модульного тестування зазвичай тестують окремі функції, то интеграційному тестуванні випробовуються взаємодію модулів між собою.

Таким чином було реалізоване тестування зміни стану додатку та перевірки правильного відображення змін. Для цього створювалася подія для зміни графіку і очікувалося оновлення графічного інтерфейсу.

Інтеграційне тестування не виявило проблем, так як тести такого типу зазвичай виявляють проблеми після внесення змін до системи або її рефакторингу.

4.3. Висновки

Отже в результаті роботи був отриманий програмний продукт, який доступний для публічного використання. У ході розгортання було підвердженну високий рівень безпеки, що стало можливим завдяки використання найновіших сервісів компанії Amazon.

Був розроблений мобільний додаток, який повністю відповідає остнннім вимогам компанії Apple та має зручний на інтуїтивно зрозумілий інтерфейс. Мобільний клієнт розміщений у магазині додатків Apple Store. Враховуючи специфіку фреймворка React Native без допрацювання мобільний застосунок так само може бути розміщений у магазині додатків Google Play.

Тестування показало, що був розроблений надійний програмний продукт, який стійкий до помилок та зовнішнього втручання. Розроблені Unit тести дозволять не зруйнувати існуючі можливості застосунку, а інтеграційні тести дозволяють бути впевнений у сумістності програмного продукту з використовуваними залежностями.

5. ПОБУДОВА БІЗНЕС МОДЕЛІ

5.1. Опис проблеми

В першу чергу, під терміном "BigData" розуміється величезний набір інформації. Причому обсяг її настільки великий, що обробка великих обсягів даних стандартними програмними і апаратними засобами представляється вкрай складною. Існують труднощі зі зберіганням та обробкою гігантських обсягів даних.

Тому основна проблема – це складність обробки потоку великих даних та візуалізації їх.

Причинами виникнення даної проблеми є:

- Обмежена доступність – існуючі альтернативні програмні забезпечення обежують вхідні параметри запиту, тобто не дозволяють шукати по всім странам, а лише з одною.
- Обмежена швидкодія – великі обсяги даних, які необхідні для аналізу результатів, є причиною високих витрат по швидкодії.
- Відсутність аналогів, які візуалізують отримані результати – так як результат роботи подається в табличному вигляді, є причиною того, що користувачеві важко працювати з цими даними в подальшому.

Складність обробки потоку великих даних та візуалізації їх, у свою чергу, породжує наступний ряд проблем:

- Відсутність бази даних – громіздка база даних та важкість у підтримуванні бази, є причиною того, що дані зберігатьються у файлах, що призводить до складності формування та обробки запиту.
- Недостатня швидкодія – розголудженність вхідних даних та відсутність логіки при формуванні запиту збільшують час обробки запиту.

Вище перелічені проблеми, що виникають при обробки потоку великих даних та візуалізації їх представлені на рис. 24 у вигляді дерева проблем.

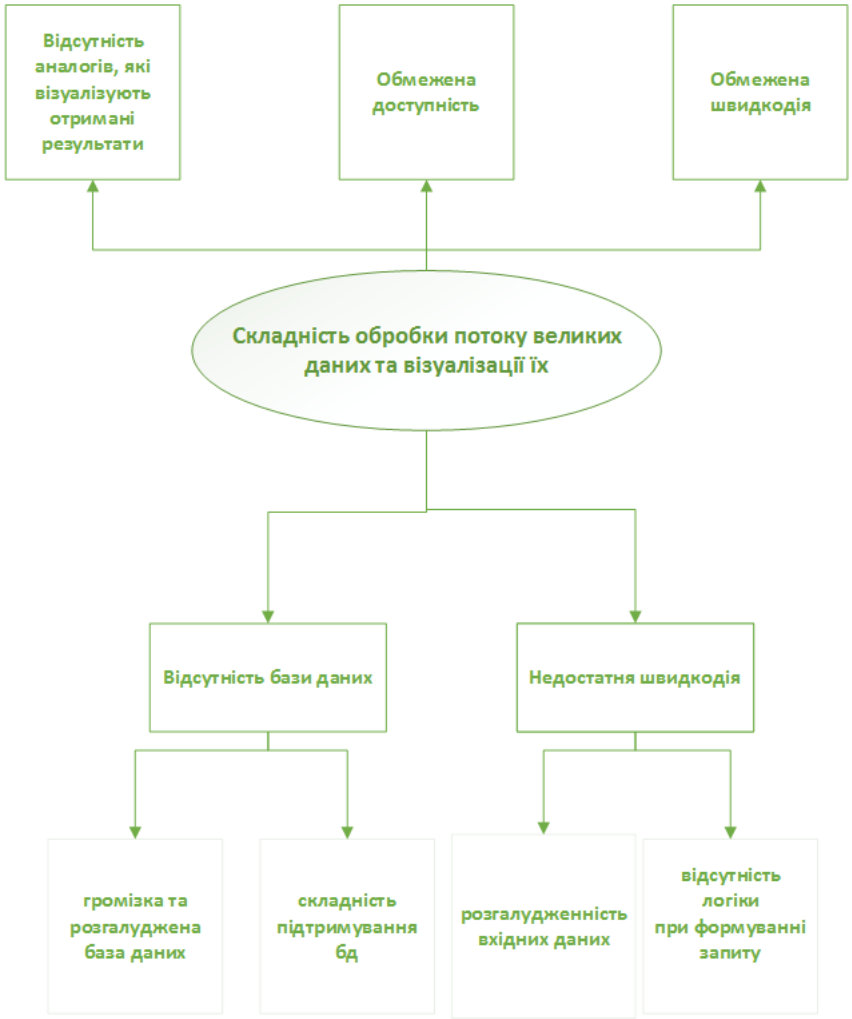

Рис. 24. Дерево проблем

5.2. Зацікавлені сторони

Зацікавленими у розробці програмного забезпечення для сервісу аналітики міжнародних торгівельних даних можуть бути компанії та наукові заклади.

У вирішенні проблем дослідження та аналізу оптимізації машинних ресурсів зацікавленні насамперед структури та організації, які безпосередньо працюють із обробкою великих даних, проводять аналітику та намагаються структурувати їх.

У ході аналізу предметної галузі, були виявлені наступні групи зацікавлених сторін:

- компанії, які займаються аналітикою на основі big data;
- наукове співтовариство, яке займаються оптимізацією машинних ресурсів;
- навчальні заклади.

До цього переліку слід додати студентів, школярів, звичайних людей, які зацікавлені в інформації, що надає продукт.

Хоча останні дві групи і не працюють безпосередньо з оптимізацією машинних ресурсів, проте вони теж можуть бути зацікавлені у розробці відповідного продукту та мають великий вплив.

У табл. 3 зведено усі групи зацікавлених сторін, їх інтереси, вплив та стратегії їх приваблення.

Таблиця 3

Зацікавлені сторони продукта

Назва зацікавленої сторони	**У чому зацікавлені**	**Вплив**	**Стратегії впливу на сторони**
компанії, які займаються аналітикою на основі big data	в мінімізації машинних ресурсів, необхідних для обробки запитів	Великий	-Презентація власного продукту, демонстрація функціональних можливостей. -Надання можливості тимчасового пробного використання. -Забезпечення технічної підтримки та навчання.
наукове співтовариство, яке займаються оптимізацією машинних ресурсів	в мінімізації машинних ресурсів	Великий	
навчальні заклади	в інформації з імпорту / експорту товарів між країнами ЄС	Середній	
студенти, школярі, звичайні люди	в інформації з імпорту / експорту товарів між країнами ЄС	Середній	

5.3. Рішення. Основні характеристики

Відповідно до вище зазначених проблем, можна описати кінцевий продукт, що має їх вирішувати.

Передбачається, що кінцевий продукт буде представляти собою програмне забезпечення для персонального комп'ютера і буде застосовуватися для аналітики міжнародних торгівельних даних імпорту/експорту країн.

Запропоноване програмне забезпечення буде виконувати поставлені задачі, використовуючи оптимізацію машинних ресурсів, що прискорить швидкодії обробки запитів, можливості графічної візуалізації результатів запиту.

Основним характеристики рішення:

- Підвищити швидкодію шляхом структурування вихідних даних.
- Зберігання даних в базі даних.
- Використовувати комбіновані методи обробки вихідних даних.

Окрім вище зазначених можливостей, кінцевий продукт повинен мати наступні характеристики:

- розроблене програмне забезпечення має працювати на звичайних користувацьких телефонах із середніми технічними параметрами або вище:
    - обсяг оперативної пам'яті – не менше 2Гб;
    - обсяг зайнятої пам'яті жорсткого диску – 1Гб.
- час обробки запиту не повинен перевищувати – 5 сек.
- розроблене програмне забезпечення має працювати на сучасних поширених операційних системах телефонів iOS.

5.4. Конкретні переваги рішення

До ряду переваг можна віднести підвищення швидкості формування аналітичних звітів в режимі реального часу, а також мінімізація машинних

ресурсів. Також перевагою є продукт, який надає інформацію про обсязі даних, тобто доступна інформація по імпорту / експорту всіх продуктів всіх країн ЄС, яка структурована по роках і країнам.

Також до переваг слід віднести візуалізацію отриманих результатів у вигляді діаграм, графіків, які зручні у використанні та зрозумілі для всіх користувачів.

А також доступність і зручність реалізованого продукту серед клієнтів.

5.5. Клієнти та сегменти ринку споживання.

З огляду на встановлені проблеми та результати аналізу зацікавлених сторін можна зробити висновок, що запропоноване програмне забезпечення не представляє інтересу для звичайних кінцевих споживачів, які будуть використовувати продукт у наукових цілях.

Тому кінцеві споживачі (студенти, школяри, звичайні люди тощо) не можуть розглядатися як окремий сегмент ринку, оскільки їх зацікавленість є опосередкованою, а вплив на розвиток продукту невеликий.

Натомість, запропоноване рішення може зацікавити державні та приватні компанії, які займаються оптимізацією машинних ресурсів. Дані сегменти мають великий вплив та зацікавленість і їх слід вважати основними складовими цільового ринку.

Тому слід зосередити зусилля на приваблені вище згаданих сегментів та застосовувати відповідні розроблені стратегії у взаємовідносинах із ними (див. табл. 3).

5.6. Унікальна ціннісна пропозиція

Програмне забезпечення для оптимізації машинних ресурсів повинно задовольняти цілому ряду вимог. Найважливіше це звісно швидкодія.

Для вирішення даної задачі використовуються спеціальні підходи, зокрема:

- Підвищити швидкодію шляхом структурування вихідних даних.
- Зберігання даних в базі даних.
- Використовувати комбіновані методи обробки вихідних даних.

При побудові рішення необхідно проаналізувати, які технології використовуються при роботі з BigData, а також дослідити оптимізацію існуючих методів та дати оцінку результатам. Однією з ще одних важливих вимог є можливість перегляду результатів запитів за допомогою візуалізації результатів у вигляді діаграм, графіків.

Метою застосування вище вказаних методів є досягнення компромісу між програмною швидкодією та візуалізацією результатів обробки запитів.

В результаті запропоновано програмний продукт, який задовольняє вище означеним вимогам.

Представлене рішення надає кінцевому користувачеві простоту у використанні, високий рівень швидкодії, доступність та легкість у аналізі отриманих результатів.

5.7. Доходи і витрати

Програмного забезпечення працює з даними, які знаходяться у відкритому доступі.

Пропонується розробити декілька версій програмного забезпечення, що будуть відрізнятися доступними функціональними можливостями:

- базова версія – надає користувачеві основні засоби для роботи із аналітикою даних, але обмежує результати роботи програмного забеспечення;
- розширена – користувач матиме змогу використовувати всі функціональні можливості продукту без обмежень.

Кожна із запропонованих версій буде відноситися до окремої цінової категорії, для розширеної будуть надаватися додаткові послуги із технічної підтримки. Основним джерелом доходу є продаж платних версій продукту, надання послуг із підтримки, а також реклама для базової версії.

Для розробки продукту та його супроводу передбачаються наступні витрати:

- витрати на розробку продукту, його супровід та підтримку (на користування базою даних, придбання необхідного апаратного та програмного забезпечення, закупівля необхідних ресурсів та оплата послуг);
- витрати на оплату праці;
- витрати на забезпечення інфраструктури та інші.

Прогноз доходів і витрат наведено в табл. 4 та табл. 5.

Таблиця 4

Прогноз доходів

№	Статті доходів	Доходи за місяць (у. о.)					
		1-й, $	**2-й, $**	**3-й, $**	**4-й, $**	**5-й, $**	**6-й, $**
1	Продаж базової версії	-	-	-	-	-	-
2	Продаж розширеної версії продукту	-	-	-	2000	2000	2000
3	Реклама	-	-	-	200	200	200
4	Надання підтримки	-	-	-	1000	1000	1000
	Результат:				**3200**	**3200**	**3200**

№	Статті доходів	Доходи за місяць (у. о.)					
		7-й, $	**8-й, $**	**9-й, $**	**10-й, $**	**11-й, $**	**12-й, $**
1	Продаж базової версії	-	-	-	-	-	-

2	Продаж розширеної версії продукту	2000	2000	2000	2000	2000	2000
3	Реклама	200	200	200	200	200	200
4	Надання підтримки	1000	1000	1000	1000	1000	1000
	Результат:	**3200**	**3200**	**3200**	**3200**	**3200**	**3200**

Таблиця 5

Прогноз витрат

№	**Статті видатків**	**Витрати за місяць (у. о.)**					
		1-й, $	**2-й, $**	**3-й, $**	**4-й, $**	**5-й, $**	**6-й, $**
1	витрати на забезпечення інфраструктури	100	100	100	100	100	100
2	витрати на розробку продукту	7000	7000	7000	-	-	-
3	плата за підтримку	-	-	-	500	500	500
	Результат:	**7100**	**7100**	**7100**	**600**	**600**	**600**

№	**Статті видатків**	**Витрати за місяць (у. о.)**					
		7-й, $	**8-й, $**	**9-й, $**	**10-й, $**	**11-й, $**	**12-й, $**
1	витрати на забезпечення інфраструктури	100	100	100	100	100	100
2	витрати на розробку продукту	-	-	-	-	-	-
3	плата за підтримку	500	500	500	500	500	500
	Результат:	**600**	**600**	**600**	**600**	**600**	**600**

Розрахунок маржинальних прибутків протягом 12 календарних місяців наведено в табл. 6.

Таблиця 6

Маржинальні прибутки

	Витрати за місяць (у. о.)					
	1-й, $	**2-й, $**	**3-й, $**	**4-й, $**	**5-й, $**	**6-й, $**
Прибуток **(у. о.)**	**-7100**	**-7100**	**-7100**	**2600**	**2600**	**2600**

	Витрати за місяць (у. о.)					
	7-й, $	**8-й, $**	**9-й, $**	**10-й, $**	**11-й, $**	**12-й, $**
Прибуток **(у. о.)**	**2600**	**2600**	**2600**	**2600**	**2600**	**2600**

5.8. Бізнес-модель

В результаті проведеного аналізу були зібрані дані для побудови бізнес-моделі, які можна подати наступним чином, відповідно до канви бізнес-моделі:

- проблема – Відсутність бази даних, недостатня швидкодія;
- рішення – Програмне забезпечення для візуалізації даних великого обсягу із можливістю подальшої аналітики;
- унікальна ціннісна пропозиція – Можливість візуалізації даних великого обсягу із можливістю подальшої аналітики;
- споживачі – компанії, які займаються аналітикою на основі big data, наукове співтовариство, яке займається оптимізацією машинних ресурсів, навчальні заклади ;
- структура витрат;
- потоки доходів.

Подана канва містить розділи, такі як прихована перевага, ключові метрики, та канали збуту, відповідно до різновиду канви бізнес-моделі lean canvas. Описані вище розділи представлені у табл. 7.

Таблиця 7

Канва бізнес-моделі

<table>
<tr>
<td rowspan="2">Проблема

-Відсутність бази даних
-Недостатня швидкодія</td>
<td>Рішення
Програмне забезпечення для візуалізації даних великого обсягу із можливістю подальшої аналітики</td>
<td rowspan="2">Унікальна ціннісна пропозиція
Можливість візуалізації даних великого обсягу із можливістю подальшої аналітики</td>
<td>Прихована перевага
Використання розроблених методів для оптимізаціх машиних ресурсів</td>
<td rowspan="2">Споживачі

-компанії, які займаються аналітикою на основі big data
-наукове співтовариство, яке займаються оптимізацією машинних ресурсів
-навчальні заклади</td>
</tr>
<tr>
<td>Ключові метрики

-обсяг продажу продукту;
-швидкодія
-кількість звернень до техпідтримки;
-кількість переглянутої реклами</td>
<td>Канали збуту

-власний web-ресурс;</td>
</tr>
<tr>
<td colspan="3">Структура витрат
- розробка продукту;
- супровід та підтримка;
- оплата праці;
- адміністративні витрати;
- підтримка інфраструктури;
- інші.</td>
<td colspan="2">Потоки доходів
-продаж розширеної версій продукту;
-надання підтримки;
-реклама.</td>
</tr>
</table>

5.9. Висновки

Запропонована бізнес-модель має забезпечити позитивний економічний ефект та вирішити виявлені проблеми дослідження та візуалізації даних великого обсягу із можливістю подальшої аналітики. Для цього пропонується розробити відповідне програмне забезпечення, що задовольнятиме вище зазначеним вимогам та вирішуватиме поставлені задачі.

Дане програмне забезпечення вносить на ринок унікальну ціннісну пропозицію та має свої конкурентні переваги у порівнянні із аналогами.

Розроблена модель потребує додаткового, більш глибокого, аналізу та доробки. Проте, виходячи із попередніх розрахунків, розробка та реалізація запропонованого програмного продукту є рентабельною і здатна покрити

витрати на його створення та почати приносити дохід вже через півроку після завершення розробки.

ВИСНОВКИ

Основна мета роботи була досягнута в ході виконання роботи. Було сформовано та в подальшому реалізовано метод для обробки та аналізу великих даних для сервісу аналітики міжнародних торгівельних даних на основі HS коду.

В ході виконання роботи було сформовано наукову новизну, яка полягає в наступному:

1. Дослідження баз даних на основі фінансових показників та показників швидкодії та вибір оптимальної.

2. Запропоновано метод для нормалізації даних в базі даних для аналітичної системи.

3.Порівнянно сервіси, які опрацьовують запити, грунтуючись на характеристиках ціни, швидкодії та зручності використання.

У першому розділі розглянуто основну проблематику роботи з великими даними, основні підходи для збереження та обробки великих даних і аналогічні системи з використанням великих даних та технології, які були використані для реалізації подібних систем.

У другому розділі проаналізовано методи роботи з великими даними, які можуть бути використані для побудови аналітичних звітів, а також робота з даними у розподілених файлових системах; розглянуто основні аспекти аналітичної обробки даних, зокрема вилучення та обробка денормалізованих даних з існуючих сховищ; проведено дослідження швидкодії роботи різних баз даних, а саме AWS Dynamo DB та PostgreSQL.

У третьому розділі розглянуто основну архітектуру системи; сформульовано основні методичні підходи до розроблення структур баз даних, серверної частини та клієнтського мобільного застосуноку для розв’язання задач побудови аналітичних звітів.

У четвертому розділі наведено опис процессу розгортання додатку та всіх його складових, які використовувались для програмної розробки, такі як бази даних, серверної части та саме мобільного застосунку. Зроблено висновки по проведеному аналізі результатів отриманих під час тестування методу і програмної реалізації.

У п’ятому розділі наведено бізнес-ідею та бізнес-план створення і розвитку стартапу на основі розробленого методу.

Робота має практичну цінність, а саме запропоновані методи та засоби дають змогу виявити цілісну картину побудови аналітичної системи, а також реалізувати надійну систему. Розроблені методи та програмне забезпечення для аналітичної системи розгорнуті на Amazon та доступні всім користувачам операційної системи iOS.

Основні положення і результати роботи були представлені та обговорювались на X науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг» ПМК-2018 (Київ, 21-23 березня 2018 р.).

ДОДАТОК 1
КОПІЇ ГРАФІЧНИХ МАТЕРІАЛІВ

ДОДАТОК 2
ЛІСТИНГ ПРОГРАМИ

Лістинг 2.1. Модуль побудови графіків у мобільному застосунку

```
import React, { Component } from 'react';
import {
  AppRegistry,
  StyleSheet,
  Text,
  View,
  processColor
} from 'react-native';

import {BarChart} from 'react-native-charts-wrapper';

class ChartComponent extends Component {
  constructor() {
     super();

     this.state = {
       data: {
         dataSets: [{
           values: [{y: 100}, {y: 105}, {y: 102}, {y: 110}, {y: 114}, {y:
109}, {y: 105}, {y: 99}, {y: 95}],
           label: 'Bar dataSet',
           config: {
             color: processColor('teal'),
             barSpacePercent: 40,
             barShadowColor: processColor('lightgrey'),
             highlightAlpha: 90,
             highlightColor: processColor('red'),
           }
         }],
       }
     };
   }

   handleSelect(event) {
     let entry = event.nativeEvent
     if (entry == null) {
       this.setState({...this.state, selectedEntry: null})
     } else {
       this.setState({...this.state, selectedEntry: JSON.stringify(entry)})
     }

     console.log(event.nativeEvent)
   }


   render() {
     const dataForShwoing = {
       data: {
         dataSets: [{
           values: this.props.data,
           label: 'Bar dataSet',
           config: {
           color: processColor('#d0d0e2'),
           barSpacePercent: 40,
           barShadowColor: processColor('lightgrey'),
         }
         }],
       },
```

```
       xAxis: {
        valueFormatter: this.props.countries,
        granularityEnabled: true,
        granularity : 1,
       }
      };

      return (
        <View style={{flex: 1}}>

          <View style={styles.container}>
            <BarChart
              style={styles.chart}
              data={dataForShwoing.data}
              animation={{durationX: 2000}}
              legend={this.state.legend}
              gridBackgroundColor={processColor('#ffffff')}
              drawBarShadow={false}
              drawValueAboveBar={true}
              drawHighlightArrow={true}
              xAxis={dataForShwoing.xAxis}
              // onSelect={this.handleSelect.bind(this)}
              // highlights={this.state.highlights}
              onChange={(event) => console.log(event.nativeEvent)}
            />
          </View>
        </View>
      );
    }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#ffffff',
    height: 100,
    padding: 20,
  },
  chart: {
    flex: 1,
    backgroundColor: '#ffffff',
  }
});

export default ChartComponent;

import React, { Component } from 'react';
import {
  View, Text, TouchableOpacity, StyleSheet, TextInput
} from 'react-native';
import ChartComponent from '../components/ChartComponent'

class ChartScreen extends Component {
  static navigationOptions = {
    title: 'Графік',
    headerTitleStyle: {
      fontWeight: 'bold',
    },
    headerStyle: {
      backgroundColor: '#d0d0e2',
```

```
    },
    headerTintColor: '#000000',
  };
  render() {
    const data = this.props.navigation.state.params;
    const sortedArr = Object.keys(data)
      .map(key => ({ [key]: data[key] }))
      .sort((a, b) => {
        console.log(Object.values(a)[0], Object.values(b)[0]);
        return Object.values(a)[0] - Object.values(b)[0];
      })
      .reverse();
    const values = sortedArr.map(x => Object.values(x)[0])
    const keys = sortedArr.map(x => Object.keys(x)[0])

    return (
      <View style={{flex: 1}}>
        <ChartComponent
          style={{height: 300}}
          data={values}
          countries={keys}
        />
      </View>
    )
  }
}

export default ChartScreen;

import React, { Component } from 'react';

import {

  View, Text, TouchableOpacity, StyleSheet, TextInput

} from 'react-native';

import Autocomplete from 'react-native-autocomplete-input';



class AutocompleteComponent extends Component {



  constructor(props) {

    super(props);

    this.state = {

      data: [],

      selectedItem: '',

      inputValue: '',

    };

    this.onClearPress = this.onClearPress.bind(this);
```

```
  }


  /**
  Request autocomplete options
   */
  queryTextHasBeenChanged(text) {
    fetch('http://ec2-18-184-132-140.eu-central-
1.compute.amazonaws.com:3000/rest/auto/' + text)
      .then(function(response) {
        return response.json();
      })
      .then(function(myJson) {
        this.setState({data: myJson});
      }.bind(this));
  }


  onClearPress() {
    this.setState({
      inputValue: '',
      data: []
    });
  }


  render() {
    return (
      <View>
      <Autocomplete
          inputContainerStyle={{paddingLeft: 4, paddingRight: 4,
marginLeft: 20, marginRight: 20}}
          style={{height: 40, width: 300}}
          renderSeparator={_ => <View
```

```
            style={{height: 1, backgroundColor: '#ffffff', marginLeft: 20,
marginRight: 20}}

          />}

          renderItem={obj => (

            <View

              style={{padding: 4, marginLeft: 20, marginRight: 20,
backgroundColor: '#ffffff'}}>

              <TouchableOpacity onPress={() => {

                this.setState({selectedItem: obj.term, inputValue:
obj.term, data: []});

                this.props.callback(obj);

              }}>

                <Text style = {{fontSize: 18}}>{obj.term}</Text>

              </TouchableOpacity>

            </View>)

          }

          data={this.state.data}

          renderTextInput={_ => (

              <View style={{backgroundColor: '#ffffff', flexDirection:
'row'}}>

                <TextInput

                  style={{height: 50, marginLeft: 40, flex: 1 }}

                  value={this.state.inputValue}

                  onChangeText={text => {

                    this.setState({inputValue: text});

                    this.queryTextHasBeenChanged(text);

                  }}

                />


                { this.state.inputValue !== "" ?

                <TouchableOpacity

                  onPress={this.onClearPress}
```

```
                    style={styles.clearButtonStyle}>
                      <Text style={{color: 'gray', fontWeight: 'bold'}}>
                        Очистити
                      </Text>
                    </TouchableOpacity>
                    : <View />}
              </View>)
            }
        />
        </View>
      );
  }
}

const styles = StyleSheet.create({
  clearButtonStyle: {
    height: 50,
    width: 100,
    marginLeft: 50,
    justifyContent: 'center'
  }
});

export default AutocompleteComponent;
```

ДОДАТОК 3
КОПІЯ ПРЕЗЕНТАЦІЇ