МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Система трекінгу користувачів веб-ресурсу шляхом взаємодії з GraphQL»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Петров С.О.**

Студента групи ІН – 72 — **Пахота Ю.О.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедри Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-72 спеціальності "Інформатика" денної форми навчання Пахоти Юрія Олександровича.

Тема: "**Система трекінгу користувачів веб-ресурсу шляхом взаємодії з GraphQL**"

Затверджена наказом по СумДУ

№_____ от _____2021 г.

Зміст пояснювальної записки: 1) аналітичний огляд методів відслідковування користувачів; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, технологій, бібліотек і критеріїв, що використовуються додатком для відслідковування та обробки користувачів деякого веб-ресурсу; 5) розробка інформаційного й програмного забезпечення для трекінгу користувачів шляхом взаємодії з GraphQL; 6) аналіз результатів.

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ Петров С.О..

Завдання прийняв до виконання _____ Пахота Ю.О.

РЕФЕРАТ

Записка: 55 стор., 20 рис., 3 табл., 6 додатків, 16 джерел.

Об’єкт дослідження — навчальний веб-ресурс E-Olymp.

Мета роботи — розробка програмного забезпечення для трекінгу та виведення інформації щодо студентів Сумського Державного Університету на навчальному веб-ресурсі E-Olymp.

Методи дослідження — метод спостереження.

Результати — розроблено програмне забезпечення (веб-додаток) для відслідковування користувачів системи E-Olymp, а саме – студентів Сумського Державного Університету. Отримання та фільтрація даних відбувається за допомогою технологій GraphQL Apollo, які слугують серверною частиною проекту. Передача інформації від інтерфейсу прикладного програмування до клієнтської частини, а також її відтворення, реалізовано за допомогою бібліотеки Apollo Client та JavaScript-фреймворку React.

ВЕБ-ДОДАТОК ДЛЯ ВІДСЛІДКОВУВАННЯ КОРИСТУВАЧІВ,
GRAPHQL ІНТЕРФЕЙС ПРИКЛАДНОГО ПРОГРАМУВАННЯ,
МЕТОД СПОСТЕРЕЖЕННЯ, АНАЛІТИЧНА СИСТЕМА,
СТАТИСТИЧНА КАРТКА.

ЗМІСТ

ВСТУП

Випускна робота присвячена проблемі отримання та обробки необхідної інформації з її подальшим використанням. Об’єктом дослідження слугує веб-ресурс E-Olymp. Це безкоштовний навчальний сайт по програмуванню, де бажаючі можуть спробувати здобути, вдосконалити або перевірити вже наявні знання по написанню коду. Веб-ресурс надає різного рівня складності задачки та змагання для розв’язання.

На цьому порталі також і студенти Сумського Державного Університету програмують. Для них і було створено даний дипломний проект. Саме ці користувачі веб-ресурсу відслідковуються та відтворюються на клієнтському інтерфейсі сайту для подальшого аналізу результатів їх роботи у вигляді рейтингу та графіку.

Проблема є гострою, оскільки необхідно перевіряти лише потрібних користувачів системи. У даному випадку це є студенти Сумського Державного Університету. Так як веб-ресурс відображає у рейтингу всіх програмістів підряд по рангу, то знайти конкретних людей виявиться складною задачею. Це буде гарною новинкою, яка зможе відобразити навички шуканих учнів, які можуть потім використати цю статистику у своїх профілях, резюме, особистих картках тощо.

Веб-додаток використовує технологію GraphQL - мову запитів та маніпулювання даних. Це є важливою бібліотекою та впровадженням у програмне забезпечення, оскільки веб-ресурс E-Olymp має цю мову в своєму розпорядженні також. Це є великим плюсом, вона не тільки облегшує та пришвидшує обмін даними між веб-додатками, але й дозволяє «обирати» лише необхідну інформацію із тих, що пропонує їх інтерфейс прикладного програмування. Автоматично відбувається «фільтрація» інформації – залишаються лише студенти Сумського Державного Університету. Завдяки цьому також знімається додаткове навантаження на систему, пришвидшується робота програмного забезпечення. Це все в купі забезпечує користувачу веб-сайту найкращий досвід.

ПОСТАНОВКА ЗАДАЧІ

Головна мета цього проекту – це створення аналітичного сайту програмістів Сумського Державного Університету шляхом співпраці з веб-ресурсом E-Olymp через технологію GraphQL. Студенти, які навчаються, беруть участь у олімпіадах, або просто виконують завдання для розвитку кар’єри розробника програмного забезпечення, будуть відслідковуватися цим веб-додатком та викладені з подальшою детальною аналітикою їх діяльності. Останнє передбачає особисту картку користувача, із статистикою вирішених завдань або олімпіад, кількістю потрачених рішень на це та рейтингом успішності щодо інших учнів. Крім цього, буде також викладений детальний графік студента, який показуватиме активність учасника протягом всього часу програмування.

У планах реалізувати веб-сайт, який буде складатись із серверної частини, в яку входить мова маніпулювання даних GraphQL, для здійснення запитів на сторону E-Olymp за інформацією про студентів, та клієнтської – інтерфейс для виведення отриманих даних для звичайного користувача системи. Сайт матиме дві сторінки, одна з яких – головна – буде виводити таблицю із всіма отриманими програмістами і їх статистикою. Інша – згадана вище особиста сторінка користувача, яка міститиме картку студента із зібраними даними щодо його праці на E-Olymp протягом всього часу. Отримана інформація потім також виводиться у вигляді графіка та демонструє весь пройдений шлях розробника програмного забезпечення.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Одна із головних задач даного проекту полягає у тому, щоб отримати потрібні дані з певного веб-ресурсу. Це означає, що об’єкт розробив та використовує у своєму розпорядженні інтерфейс прикладного програмування, або скорочено API. Він дозволяє «звертатись» до розробника та «запитувати» про необхідну інформацію, функції чи особливості сайту. За допомогою цієї технології будуть отримуватись відомості про студентів для подальшого розпорядження ними. Це означає посилання запитів за потрібними даними для їх використовування у програмному забезпеченню, що розробляється, у вигляді таблиці та графіку, ніби це бібліотека для загального застосування.

1.1 Серверна частина.

Реалізація може бути виконана різними способами. Розглянемо найбільш компетентні з них, які могли б застосовуватись на проекті:

1. Java.
2. .NET Core.
3. Node.js.

Перше рішення – це типізована об’єктно-орієнтована мова програмування, яка використовує свою власну віртуальну машину для запуску програмного забезпечення. Застосовується при розробці різних додатків, починаючи від настільних та веб, закінчуючи мобільною сферою. До речі про останнє, це основна мова для кодування програм на операційну систему Android.

Java використовує фреймворк Spring для реалізації серверної частини додатків [2]. Вона підходить для більш серйозних проектів, оскільки є багато-потоковою, що означає можливість виконання декілька операцій одночасно, але застосовує на це по одному потоку кожного разу, що передбачає додаткове навантаження та більше витрачання ресурсів. Об’єктно-орієнтований підхід дозволяє застосовувати класи та об’єкти знову і знову, але не матиме практичного використання у коді даного проекту. Java – це нагромадження

складних систем та інтерфейсів, які не мають місця у цій дипломній роботі, оскільки не будуть застосовуватись. Мова програмування є строго типізованою, що не підходить для початкових замислів та бажання використати GraphQL із описанням своїх типів даних, тому було прийнято рішення відмовитись від цього пункту.

.NET [3] – це веб-фреймворк з відкритим вихідним кодом, який на сьогоднішній день один із найпопулярніших рішень для реалізації серверної частини проекту. Він пропонує можливості для більш гнучкого налаштування бази даних, збереження інформації та її захисту. Також, як і попереднє рішення, багатомовна технологія .NET призначена більше для потенційно об’ємних проектів. Крос-платформні можливості значно гірші, ніж у свого конкурента Node.js, а продуктивність додатку буде менше, оскільки остання мова програмування виграє у розробці малих та середніх програмних забезпечень.

Рішення, яке залишилось останнім у списку, але не по значенню – JavaScript-фреймворк Node.js [1]. JavaScript [4] – це мова програмування, яка застосовується до HTML документу та дозволяє «оживити» будь-яку сторінку веб-сайту. Динамічне змінювання даних, інтерактивність із будь-якими елементами, відправлення форм з інформацією тощо, без необхідності оновлювати чи переходити до іншої сторінки – це все завдяки цій мові.

Чому саме Node.js як сервера частина цього проекту? Він виграє у своїх попередніх конкурентів тим, що він одно-потоковий та дозволяє асинхронне введення і виведення інформації. Це «грає на руку», оскільки поставлені цілі мають невеликі потреби у вигляді лише декількох асинхронних запитів до бази даних. Швидкодія веб-додатку буде максимальною завдяки технології Node.js, а відповідь від сховища інформації – моментальною. Веб-фреймворк дозволяє установлювати потрібні бібліотеки одразу у додатку, без необхідності додатково щось скачувати, а модулі коду можуть використовуватись знову і знову.

Node.js підходить не тільки завдяки цим перевагам, а ще й тому, що він згоден для створення нового інтерфейсу прикладного програмування, який, власне, і розробляється на базі технології GraphQL [6], а також чудово поєднується зі згаданою раніше мовою запитів. Програмна платформа буде слугувати оболонкою для потреб створити серверну частину за допомогою бібліотеки GraphQL і Apollo Server, що поєднуватиме це все з інтерфейсом на стороні клієнту. Це найпопулярніша і найефективніша зв'язка технологій для праці з інтерфейсом програмування додатку E-Olymp та реалізації поставлених цілей і потреб проекту.

Задача дипломної роботи базується на використанні мови запитів GraphQL, але є ще пара альтернативних рішень, які могли б застосуватись при розробці додатку.

1. Розбір сайту безпосередньо власноруч.

2. Технологія REST.

Перший метод передбачає використання так званий розбір веб-ресурсу. Простими словами – це витяг всієї інформації з певної сторінки, або навіть цілого сайту, за допомогою деякого спеціально запрограмованого під потреби бота [4]. Він сканує об'єкт для дослідження, дістає необхідні дані та зберігає їх у базі даних або файлі. Все, що отримується, зазвичай передається у виді JSON – текстовому форматі для обміну даних, що оснований на JavaScript та часто використовується саме з цією мовою програмування [13].

```
{
    "1": {
        "name": "John Doe",
        "username": "johnnyD",
        "problems": 5,
        "solves": 17,
        "rating": 4,
        "chart": [
            { "year": 2014, "solves": 3 },
            { "year": 2015, "solves": 13 },
            { "year": 2016, "solves": 11 },
            { "year": 2017, "solves": 9 },
            { "year": 2018, "solves": 7 },
            { "year": 2019, "solves": 10 },
            { "year": 2020, "solves": 14 },
            { "year": 2021, "solves": 6 }
        ],
        "photo": "https://avatars1.githubusercontent.com/u/45543904?v=4"
    },
    "2": {
        "name": "Bill Jason",
        "username": "billy",
        "problems": 14,
        "solves": 69,
        "rating": 7,
        "chart": [
            { "year": 2014, "solves": 7 },
            { "year": 2015, "solves": 13 },
            { "year": 2016, "solves": 9 },
            { "year": 2017, "solves": 7 },
            { "year": 2018, "solves": 12 },
            { "year": 2019, "solves": 15 },
            { "year": 2020, "solves": 8 },
            { "year": 2021, "solves": 4 }
        ],
        "photo": "https://avatars1.githubusercontent.com/u/45543905?v=5"
    },
}
```


Рисунок 1.1 – Приклад даних у форматі JSON

Один із мінусів цього методу – це те, що він може бути досить повільним для виконання порівняно із застосуванням GraphQL, установленого для цих цілей. Також безкоштовний бот для розбору сайту буде досить не універсальним, а тому доведеться створювати власний під свої потреби, що віднімає ресурси та час, на відміну від готового інтерфейсу прикладного програмування. Це стосується не тільки сканування даних, а й їх вивід, оскільки формат JSON може не підійти. Наприклад, коли потрібно зберігати інформацію відразу у базу даних. Це все в сукупності не приваблює, тому було вирішено відмовитись від цього методу.

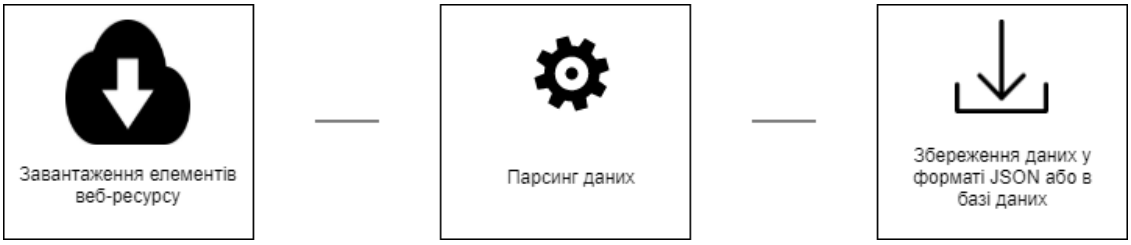


Рисунок 1.2 – Приклад сканування даних

Друге рішення застосовується, якщо веб-ресурс користується REST інтерфейсом прикладного програмування як основним [11]. Його функція

полягає у тому, щоб на кожен ресурс проекту мати свою кінцеву точку та ідентифікацію по URL-адресі. Тобто, якщо у нас є деякі люди, а у них свій ідентифікатор та унікальні статті, то щоб скористатися можливістю та витягнути дані, знадобиться перейти по кожному із цих URL-адресів окремо. На практиці це виглядатиме так:

- */api/user/:id/* - для знаходження деякого користувача;
- */api/article/:id/* - для знаходження деякої статті;
- */api/articles/* - для знаходження всіх статей веб-ресурсу;
- */api/user/:id/articles* – для знаходження всіх статей деякого користувача;
- */api/user/:id/article/:id* – для знаходження деякої статті деякого користувача.

HTML – це мова гіпертекстової розмітки [5]. Вона застосовується для того, щоб браузер, на стороні користувача системи, зміг інтерпретувати цей код та відтворити на його основні контент веб-сторінки. JavaScript допомагає в цьому та виконує складні операції, які сприяють відтворенню, оновленню чи взагалі роботі компонентів на сайті. Це магія яка дозволяє динамічно «працювати» з веб-сторінкою. Без нього нам постійно довелося б перезавантажувати сайт для того, щоб, наприклад, змусити 2D, 3D анімації рухатись, показувати відео в програвачі або бачити переміщення елементів по сторінці. У простому варіанті, все це виглядає приблизно так:

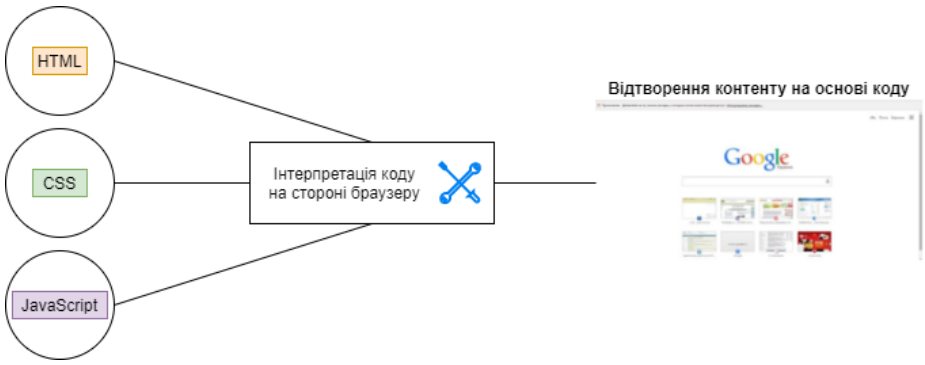


Рисунок 1.4 – Відтворення роботи браузеру

CSS – це каскадні таблиці стилів [14]. Мова, за допомогою якої, можливо «прикрасити» зовнішній вигляд сторінки та веб-ресурсу в цілому. Вона надає

гарний вид контенту, створеному завдяки HTML. Це може бути колір тексту, його шрифт, вирівнювання або адаптивність сайту під кожен девайс.

Для того, щоб скористатися інтерфейсом прикладного програмування будь-якого веб-ресурсу та здобути необхідну інформацію для деяких цілей, знадобиться зробити HTTP-запит. Це запит на серверну частину веб-об'єкту з подальшою відповіддю від нього у вигляді даних, що відображаються у браузері.

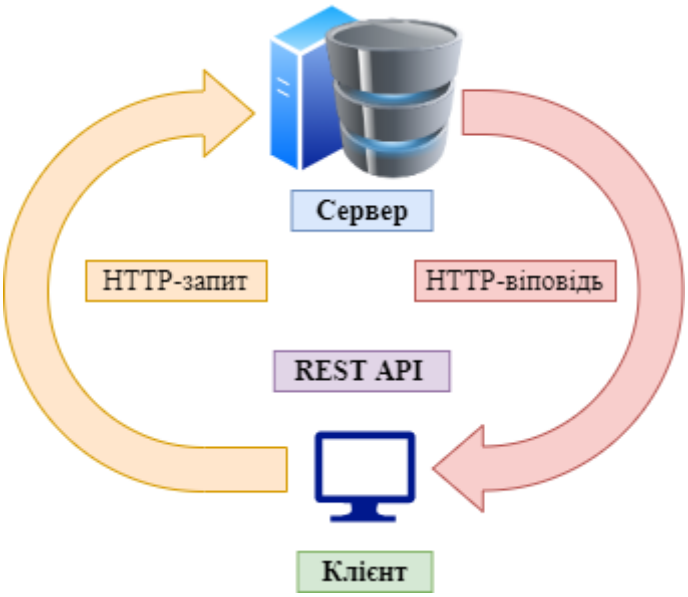


Рисунок 1.3 – Приклад HTTP-запиту та відповіді

Використання HTTP-запитів буває різне. Зазвичай це одне із чотирьох можливих варіантів звернення до сервера, а саме:

- GET;
- POST;
- PUT;
- DELETE;

Розглянемо перший випадок. Він застосовується для того, щоб отримати або відправити деякі дані з серверу. У останній функції приховується нюанс у цьому методі. Він полягає в тому, що запит змінює URL-адресу у браузері користувача на щось подібне:

```
/index.js?login=Admin&password=12345
```

Так зовсім не безпечно, адже кожний перехожий може заглянути у монітор відправника запиту та побачити ці дані, а потім застосувати їх для проникнення на ресурс під чужим іменем. Для таких цілей застосовують другий варіант POST, а GET – для лише отримання деякої інформації або функції сайту.

Крім вище наведених методів, POST також може ще й може редагувати дані. В плані відправлення інформації, це є більш безпечним варіантом, оскільки відбувається все в прихованому від користувача місці, на стороні браузера, але тільки в тих випадках коли це не впливає на вміст самої сторінки. Наприклад, виконання деякого скрипту, варіанти яких ми можемо бачити нижче:

- вхід до мережі;
- оплата товару платіжною карткою;
- відправлення повідомлення;

PUT-запит створює або оновлює вже існуючі дані на сервері. Останнє працює на інформацію створену за допомогою попереднього POST-запиту та відбувається через відправлення по URL-адресі готового ресурсу. DELETE-запит видаляє будь-які дані, задані по URL-адресі, які присутні на сайті.

На всі запити до серверу повертається «відповідь» у вигляді специфічного коду, що називається статус код. Він відображає результат виконання того чи іншого запиту на серверну частину веб-ресурсу. Це включає не тільки віддачу сигналу, але й виконання потрібних функцій, що вимагає сам запит. Зазвичай статус коди мають п’ятизначну цифру та позначаються на п’ять класів, які розглянуто детальніше нижче:

Таблиця 1.1 Визначення статус кодів

Статус код	**Визначення**
1xx	Тимчасові інформаційні коди, що сповіщають про прийняття запиту та його обробку.
2xx	Сповіщення про успішну обробку запиту.
3xx	Це відповідь на необхідність подальших дій для успішного виконання запиту (перенаправлення на інший сайт)
4xx	Сповіщення про невдале виконання запиту на стороні клієнта (користувача).
5xx	Відповідь про невдале виконання запиту на стороні серверу (користувач виконав все правильно).

Проаналізувавши всі плюси і мінуси переглянутих вище технологій, було прийнято рішення використовувати технологію GraphQL для рішення поставлених цілей та задач по створенню серверної частини проекту. Для розроблення власного інтерфейсу прикладного програмування та поєднання із оболонкою для бібліотек Node.js, мова запитів підходить найефективніше.

1.2 Клієнтська частина.

Питання відтворення отриманої інформації про студентів може бути вирішене різними конкуруючими технологіями. Нижче розглянуто декілька із них. Це включає як звичайний JavaScript за замовчуванням, так і його веб-фреймворки.

1. AngularJS.
2. Vue.js.
3. React Redux.
4. React native.

Одні з них використовуються для простих «легеньких» одно-сторінкових веб-ресурсів, які не мають на меті створити якийсь великий проект. Інші ж, застосовують у більш опрацьованих багатосторінкових сайтах, де відбувається швидкий обмін великих об’ємів даних та роботи, відтворення складних об’єктів та процедур.

Один із плюсів AnjularJS полягає в тому, що він може ефективно працювати з HTML-кодом та дозволяє безпосередньо прямо в нього вставляти будь-який JavaScript-код та виконувати скрипти [8]. Працює цей веб-фреймворк за схемою Model-View-Controller (скорочено MVC), або модель-вид-контролер, - розділення програмного забезпечення на три окремі частини, що можливо змінювати незалежно один від одного. Нижче наведений опис кожного із цих пунктів:

Таблиця 1.2 Опис MVC-моделі

Назва	**Призначення**
Модель	Відповідає на запити контролеру та надає загальну інформацію.
Вид	Реагує на зміни та має зобов’язання відображати дані.
Контролер	Помічає дії користувача та сповіщає модель про це.

Це дозволяє зручно розподіляти роботу на великих проектах та працювати різним людям одночасно над різними етапами програмного забезпечення. Мінусом цього фреймворку буде те, що він відносно повільний, тому що має постійно відслідковувати всі моделі програмного забезпечення та їх зміни. До того ж, для AngularJS треба використовувати TypeScript, тому що вся документація та ресурси написані переважно саме для цієї мови програмування. Він схожий з JavaScript, але має свої відмінності в типізації даних та, наприклад, поліпшеного об’єктно-орієнтованого програмування. У даному проекті немає місця для мови TypeScript, тому було прийнято рішення

відмовитись від бібліотеки AngularJS. Він попросту не потрібен, а його використання лише забирає зайві ресурси та час.

Vue.js – це ще один можливий розвиток для вирішення задач створення веб-ресурсу [9]. Він схожий з іншими веб-фреймворками, але має свої відмінності. Як мінімум, його функціонал дозволяє підійти до проекту на рівні представлення, тобто view, що дозволяє легко інтегрувати додаток до інших бібліотек. Звідси і пішла назва додатку – Vue (схожість з англійською view з вимови). Концепція веб-фреймворку складається з компонентів – різних елементів DOM [4]. Останнє – це об'єктна модель документу, тобто «дерево» HTML-, XHTML- та XML-документів. Змінюючи їх за допомогою коду, на виході отримується готовий сайт. Завдяки цьому присутня можливість «розкласти» проект на маленькі частини та працювати з ними окремо, а потім, якщо є така необхідність, використовувати їх заново. Все це виглядає як віртуальний конструктор всередині веб-ресурсу. Це гарна практика для великих проектів також, як і у попередньому варіанті.

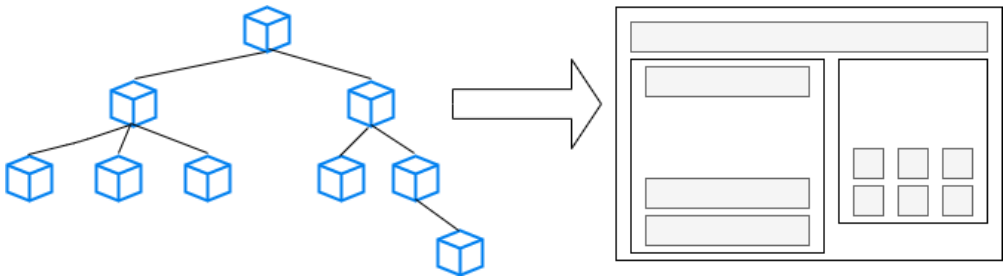


Рисунок 1.5 – Розбиття компонентів

Звісно ж, на крупних замислах, поділ на частини є невід'ємною частиною розробки програмного забезпечення. Vue.js також використовується і для рішення більш складних одно-сторінкових веб-сайтів. Підхід у бібліотеки до компонентів не настільки гнучкий, як у свого конкурента React, а викладання елементів на інтерфейс у останнього більш функціональний. Vue.js потребує лишні трати ресурсів і часу на його вивчення, особливо коли має язиковий бар'єр у документації. Також він не є настільки популярним, як його конкуренти, оскільки є відносно новою бібліотекою, а тому має менше

технологій, які могли б застосуватись у проекті при необхідності. Відмова від застосування даного варіанту, на користь останньої альтернативи - React, є очевидною.

Одне з технологій, які інколи застосовуються разом із бібліотекою React, це Redux [7]. Вона являє собою додаток для менеджменту стану програмного забезпечення. Хоча React і має своє особисте управління ресурсами, але в більш складних та об'ємних роботах виникають деякі труднощі з «рідним» керуванням, а також масштабуванням. Виправляється це за допомогою функції сховища у бібліотеці Redux. Воно передбачає керування стану додатку. Працює по схожій на бази даних схемі – має статус програмного забезпечення, по запиту може його змінити або відобразити, а також відслідковує це все.

На «чистому» React построєні не всі веб-ресурси. Даний випадок також не є виключенням. Но для цього проекту вистачить і «корінного» варіанту бібліотеки. Redux досить складний у його розумінні і використовуванні, передбачає написання більшої кількості коду, а також не дуже інтуїтивно зрозумілий для новачка. У цій дипломній роботі застосовується не так багато компонентів, вони не є складними, а половина з них – це звичайні функції. Сенсу витрачати ресурси на більш глибоке керування станом немає, як і, власне, на імплементацію бібліотеки Redux.

Для створення системи трекінгу користувачів було вирішено використати веб-фреймворк React [10]. Ця JavaScript-бібліотека має найбільшу популярність у створені сайтів та веб-сфері в цілому. Із трьох варіантів, вона залишається остання та підходить найбільше для рішення поставлених задач.

1.3 Сховище інформації.

На даному етапі вже відомо, що серверна сторона проекту відправляє запити на веб-ресурс, дістає звідти інформацію про студентів та повинна зберігати її. Для останнього потрібно обрати одне із рішень, що сприяє цьому [12]. Є декілька варіантів для збереження даних:

- локально у файлі;
- локально на базі даних;
- хмарне сховище інформації.

Перший метод передбачає створення та запис інформації у локальний файл, на комп’ютері або будь-якому іншому девайсі, із подальшими зверненнями до нього за даними. Це, звісно ж, один із нескладних та швидких рішень, але у нього є декілька мінусів. Незручна перевірка даних без належного інтерфейсу властивим усім базам та необхідність кожного разу на всіх девайсах створювати файл і записувати в нього інформацію для роботи додатку – це одні із найголовніших проблем цього методу. Це випливе у те, що загальний доступ до програмного забезпечення буде обмежений, а різні файли на різних комп’ютерах можуть містити різні версії даних, що спровокує конфлікт у роботі програми. Не кажучи вже про те, що при цьому файл не зможе містити великі дані, якщо додаток того потребує, а швидкодія буде на маленькому рівні. Файл не має методів для захисту інформації яку зберігає, так само як і перевірку на запити які надходять. Останнє заставить розробника писати алгоритм для їх роботи безпосередньо у додатку, що коштує додаткових ресурсів та сил на виконання. Такі досить вагомі аргументи змушують творця програми відмовитись від цього методу.

Ідентифікатор	Ім'я	Вік	Вага	Зріст
1	Петренко Вікторія Тарасович	29	45.7	189
2	Тамара Олександрович Пономаренко	36	70.6	155
3	Дмитро Миколайович Василенко	59	104.3	151
4	В'ячеслав Олексійович Крамаренко	43	90.1	174
5	Дмитренко Ярослава Андрійович	35	96.3	204
6	Анастасія Тарасович Іванченко	18	81.3	178
7	Мірошниченко Леонід Михайлович	37	58.7	178
8	Євгенія Євгенійович Іванченко	43	96.2	154
9	Кравченко Єлизавета Євгенійович	45	100.2	154
10	Романченко Софія Володимирович	27	91.4	184

Рисунок 1.6 – База даних у файлі на прикладі Excel

Розглянемо ще одне рішення – збереження даних локально, але вже с використанням бази даних. Воно передбачає установлення бази даних на комп’ютері, або будь-якому іншому девайсі, де знаходиться сам додаток. Це зручний вихід для тестування додатку на етапі розробки, або для демонстрування того, що уже зроблено і як працює, але не для повноцінних готових проектів. Як і в попередньому методі, цей містить ті ж самі проблеми, але до них додаються ще й нові. Одна із них – це потреба у встановленні кожного разу локальну базу даних з її подальшим налаштуванням для підключення до програмного забезпечення та його роботи. Не кажучи вже про те, що це в принципі неможливо на деяких пристроях. Звісно ж, на серйозних проектах про це й мови бути не може.

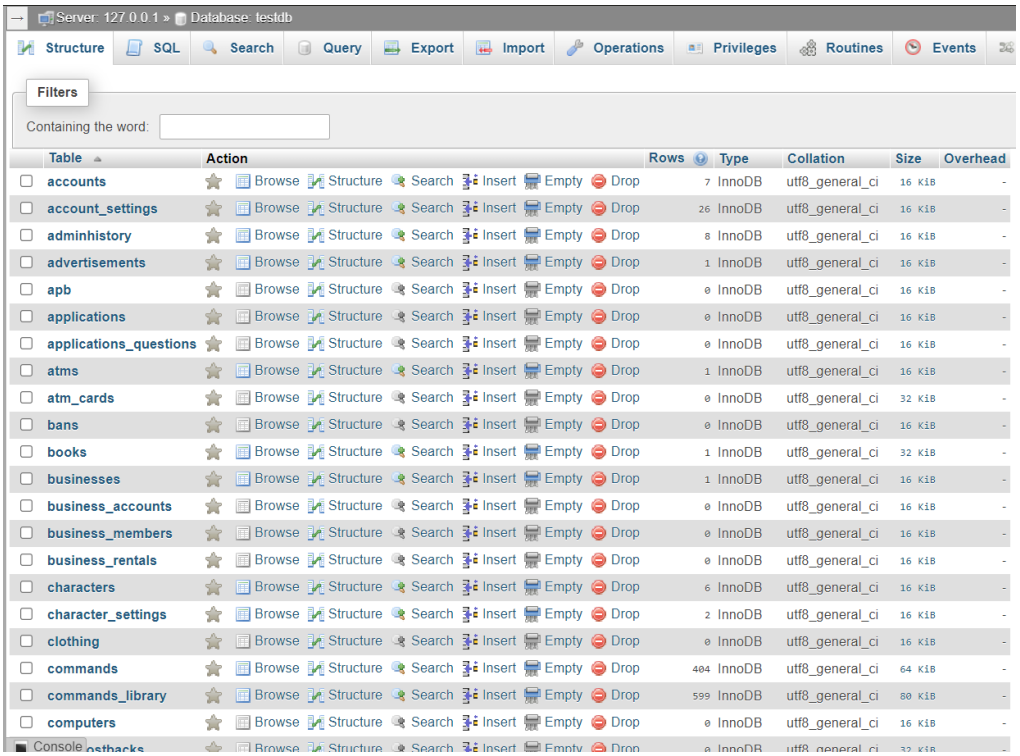


Рисунок 1.7 – Локальна база даних

Останнє рішення виглядає найбільш оптимальним, оскільки попередні два не задовольняють потреби проекту. Хостинг бази даних на хмарному сховищі інколи потребує коштів та сил, якщо це великі об’єми інформації на громіздкому проекті, але завжди є варіант використати безкоштовні сховища

з певними базами даними. На останньому потрібно зупинитись, тому розглянемо найголовніші з них для подальшого використовування.

1. MySQL.

2. PostgreSQL.

3. SQLite.

4. Oracle.

5. MongoDB.

Перший варіант – це технологія, що належить до систем управління реляційними базами даних (скорочено СУРБД). Тобто, для цього використовується деяке програмне забезпечення, що забезпечує більш зручне та структуроване зберігання і застосовування даних. Це може бути, наприклад, реалізовано як у локальному хостингу, приклад якого ми бачили вище. MySQL відноситься до таких званих SQL баз даних, тобто вона реляційна, а це означає, що утримання інформації відбувається за допомогою таблиць. Дані статичні, структуровані та мають визначену наперед схему, тому проходять строго по ній.

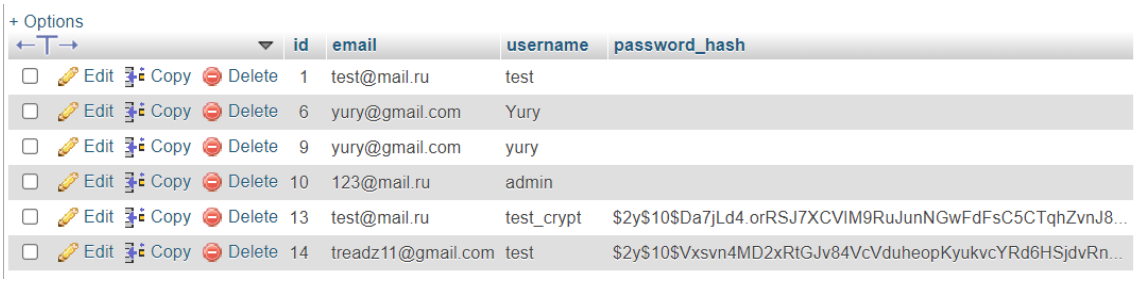


Рисунок 1.8 – Приклад таблиці з даними

Кожен запис складається із атрибутів – стовпців – по яким і записується інформація. Вони мають свій тип даних. Типів буває декілька і всі слугують для описання різних даних:

Таблиця 1.3 Опис типів даних

Назва	**Призначення**
Рядкові	Дані, які складаються із символів.
Цілі	Цілі числа.
Із плаваючою точкою	Числа, у яких змінюється абсолютна величина.
Логічні	Мають два варіанти: так або ні (true – false).
Перечислення	Позначаються як «enum». Список позначених ідентифікаторів.
Дата	Для позначення дати.
null або undefined	Дані відсутні або не були присвоєні.
Структурні	Дані у вигляді набора елементів будь-якого типу. Для об’єктно-орієнтованих баз даних.
Класові	Для об’єктно-орієнтованого програмування та баз даних.

Також повинен бути присутній ідентифікатор – зазвичай це ціле число – для позначення наведеного запису. Це називають ключем, за допомогою якого таблиці реляційних баз даних взаємопов’язані. Наприклад, це наочно показано на рисунку 1.6.

MySQL – одна із найбільш популярних реляційних баз даних, але має й свій ряд недоліків. Більшість функціоналу, який мають SQL бази даних, не буде використовуватись та реалізовуватись на повну у даному проекті, а тому в цьому попросту немає сенсу. Це означає, що MySQL у розробці буде досить повільна та програвати своїм конкурентам по цьому параметру. Варто також згадати, що статичне описання даних, присутнє MySQL, із подальшим обмеженням, не знайде ніякого застосування тут. Навпаки, для вирішення поставлених проблем потрібна не реляційна база даних – NoSQL – що має свій

ряд переваг для праці з невеликими веб-додатками, прикладом яких і є цей проект.

PostgreSQL – це ще одна популярна SQL база даних. І вона найбільше підходить до цього терміну із усіх перелічених сховищ інформації, оскільки має більше всього функціоналу для роботи з реляційною технологією. Це свого роду «професіонал» серед перелічених SQL баз та максимально розкриває потенціал цього терміну. PostgreSQL також «виправляє» одну із проблем SQL сховищ даних – паралельність. Вона досягається за допомогою технології багатоверсійносто паралельного контролю, яку використовує PostgreSQL, позволяючи виконувати одночасне зчитування та запис інформації, працювати з великими об’ємами даних та багато-поточністю. Сховище інформації має чудову підтримку об’єктно-орієнтованого підходу до баз даних, на випадок коли розроблюваний додаток має об’єктно-орієнтований критерій та виконувався за допомогою підходящої мови програмування. Типи даних, які використовуються у таких баз даних, описані у таблиці 1.3. Це зручно, коли об’єктно-орієнтоване програмне забезпечення має свою власну модель даних, і потрібне таке сховище даних, яке не буде його «переписувати» або придумувати нове, а тому збереже їх у такому ж самому типізованому виді. Гарний плюс PostgreSQL, але ця особливість зовсім неважлива для розробників даної дипломної роботи. База даних також підтримує технологію збережених технологій, які можуть оброблювати запит один раз, а потім зберігатись на сервері та використовуватись знову для уникнення лишніх затрат ресурсів. Це відчутне спрощення застосування постійних повторюваних операцій.

PostgreSQL – досить таки потужна система, але знову ж таки, як і попередня база даних, більшість її хороших та корисних технологій попросту не буде використовуватись в даному випадку. Сховище даних переважно створене для реляційних та об’єктно-орієнтованих рішень, які потребують саме ці методи застосування. Для кожного додатку потрібно ретельно

вибирати базу даних, і тому було вирішено не застосовувати саме цю. Вона буде сповільнювати запити, а продуктивність буде нижче ніж у конкурентів.

До цього було розглянуто лише мережеві сховища даних. SQLite пропонує використання локального варіанту звернення до інформації – безпосередньо через файли. Завдяки цьому, вона має велику швидкість при роботі з даними, а сама база легко вбудовується в будь-який додаток. Це гарні аргументи при виборі сховища даних, і тому може здатися, що це ідеальний вибір. Але у нього також є свої недоліки, якщо брати конкретно даний веб-додаток. Із-за своєї особливості записувати у файл виникає проблема, яку було описано при розгляді локальних файлових сховищ інформації – необхідність кожного разу перезаписувати та переносити записи, які ще й можуть містити різні версії даних. Зазвичай, SQLite добре себе проявляє у невеликих настільних програмах, а також при тестуванні з масштабністю, але не веб-додатках. Так, база даних є реляційною та підтримує SQL технологію. Хоча вона себе добре проявляє у схожих проектах, але для поставлених розробником потреб є більш підходяще сховище інформації.

Розглянемо ще одне із рішень - реляційна база даних Oracle. Це сучасне та популярне рішення для розробки додатків. Відрізняється від інших SQL технологій тим, що має свій власний діалект розрахований на використання збережених процедур, про які згадано вище, та вбудованих функцій. Oracle може містити великі об’єми даних, а її сумісність з девайсами та операційними системами на високому рівні, але й вимагає це сховище інформації в обмін чимало. Використовується воно в організаціях просунутого рівня, а оскільки це комерційний проект - програмне забезпечення коштує дорого. Зроблена технологія на професіональному рівні для масштабних проектів.

```sql
CREATE TABLE test(id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(20), age INTEGER, weight VARCHAR(5), height INTEGER);
INSERT INTO test(name, age, weight, height) VALUES
    ('Петренко Вікторія Тарасович', 29, '45.7', 189),
    ('Тамара Олександрович Пономаренко', 36, '70.6', 155),
    ('Дмитро Миколайович Василенко', 36, '104.3', 151),
    ("В'ячеслав Олексійович Крамаренко", 59, '90.1', 174),
    ('Дмитренко Ярослава Андрійович', 43, '96.3', 204),
    ('Анастасія Тарасович Іванченко', 35, '81.3', 178),
    ('Мірошниченко Леонід Михайлович', 18, '58.7', 178),
    ('Євгенія Євгенійович Іванченко', 37, '96.2', 154),
    ('Кравченко Єлизавета Євгенійович', 43, '100.2', 154),
    ('Романченко Софія Володимирович', 45, '91.4', 184);
SELECT * FROM test WHERE id = 1;
```

Рисунок 1.7 – Приклад використання SQL бази даних

Залишається лише варіант використання NoSQL бази даних для рішення поставленої задачі. Деякі з них, а саме Google Cloud BigTable, Apache HBase, мають гарні технології для застосування, тримають великі об’єми даних та виконують гарний дата менеджмент. Звертаються до цих сховищ інформації, як і у випадку з Oracle, лише крупні компанії, яким потрібні саме ці переваги та визначені заздалегідь методи. До того ж, ці рішення потребують коштів. Одне з чудових рішень для цього проекту є застосування третього варіанту - не реляційної бази даних (скорочено СУБД) MongoDB.

Query

```
db.collection.find({
  name: "John Doe"
})
```

Result


```
[
  {
    "_id": ObjectId("5a934e000102030405000000"),
    "chart": [
      {
        "solves": 3,
        "year": 2014
      },
      {
        "solves": 13,
        "year": 2015
      },
      {
        "solves": 11,
        "year": 2016
      },
      {
        "solves": 9,
        "year": 2017
      },
      {
        "solves": 7,
        "year": 2018
      },
      {
        "solves": 10,
        "year": 2019
      },
      {
        "solves": 14,
        "year": 2020
      },
      {
        "solves": 6,
        "year": 2021
      }
    ],
    "name": "John Doe",
    "photo": "https://avatars1.githubusercontent.
    "problems": 5,
    "rating": 4,
    "solves": 17,
```


Рисунок 1.8 – Застосування NoSQL MongoDB

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 GraphQL – як покращений варіант REST.

Використання технології GraphQL для рішення поставлених задач має свої плюси. Вона пропонує більш гнучкі варіанти для обробки та отримання необхідних даних. Давайте порівняємо мову запитів та REST на прикладі відправлення запитів. При використанні останнього, для того щоб знайти деякого користувача веб-ресурсу, його статті та, наприклад, послідовників, нам доведеться зробити три окремі вимоги на три різні кінцеві точки.

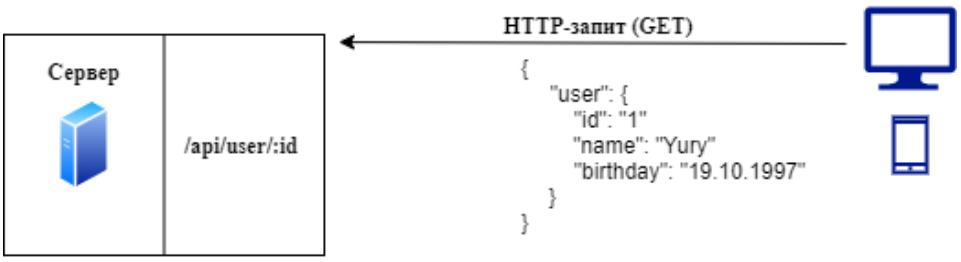


Рисунок 2.1 – REST GET-запит на користувача

Таким чином, було відправлено GET-запит на сервер для того, щоб отримати JSON дані про деякого користувача. Для знаходження тієї чи іншої статті потрібно зробити те ж саме, але окремо на іншу URL-адресу:

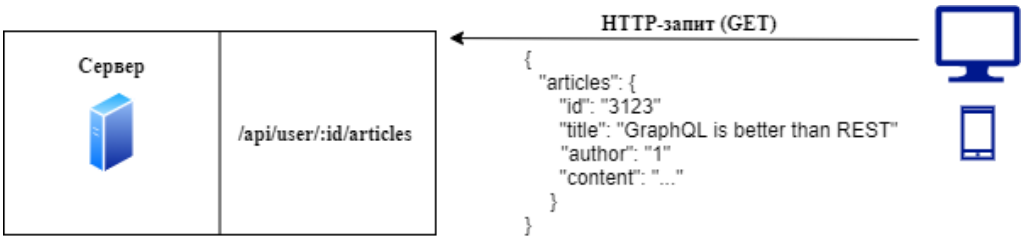


Рисунок 2.2 – REST GET-запит на статтю користувача

Як і для послідовників даного користувача:

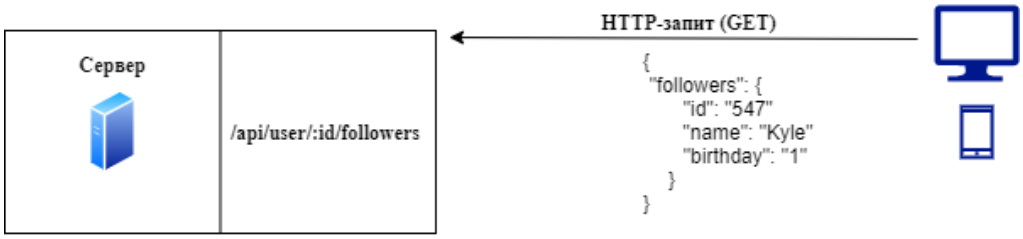


Рисунок 2.3 – REST GET-запит на послідовників користувача

Більш того, крім цієї інформації, надходять й інші зовсім непотрібні дані. Сказати, що це додаткове навантаження, довше виконання роботи та й взагалі зайве, - не сказати нічого. GraphQL допомагає в рішенні цієї проблеми. За допомогою нього потрібно відправити лише один GET-запит на сервер, замість трьох, і всі необхідні дані надійдуть до відправника без додаткового витрачання ресурсів.

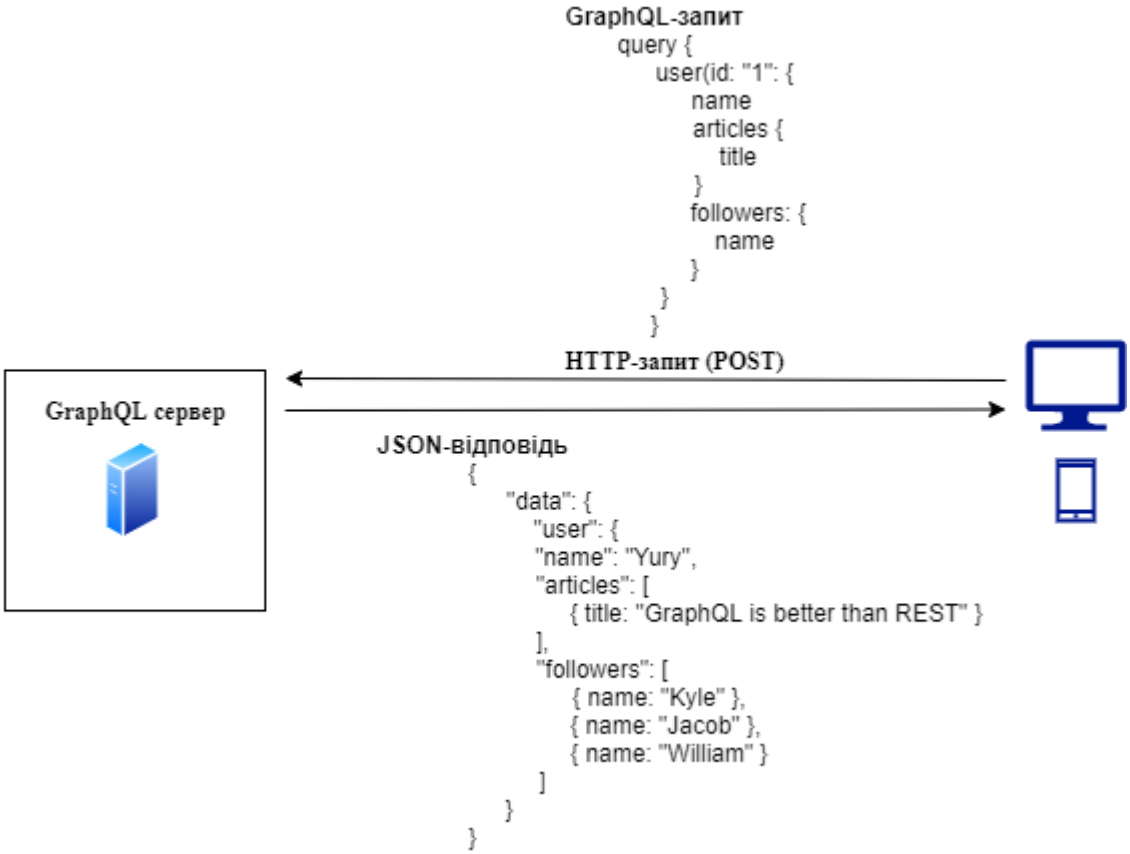


Рисунок 2.4 – GraphQL POST-запит

Результатом стане JSON-відповідь із усіма конкретними даними які ми шукали. Одним запитом одразу знайдено користувача, його опубліковані статті та послідовники. Це наступна ступінь після технології REST. Відбувається це за допомогою заздалегідь визначених типів даних у схемі GraphQL. Технологія задає свої типи у системі, щоб позначити структуру всього інтерфейсу прикладного програмування. Приклад опису цього наведений нижче:

```
type User {
    id: Int!
    name: String!
    birthday: String!
}
```

Так, наприклад, виглядає визначення типу користувача в схемі. Веб-ресурс буде зберігати лише ідентифікатор, ім’я та день народження юзера. А при витягуванні даних з іншого ресурсу гнучкість системи дозволить отримувати саме цю інформацію. Це можуть бути як всі три поля, так і лише, наприклад, ім’я. У даному випадку це грає ключову роль.

У системі із технологією GraphQL також присутні так звані «розпізнавачі схеми». За допомогою них відбувається опис тих даних, які сервер поверне до відправника запиту. До кожної схеми обов’язково повинні бути свої розпізнавачі. Наприклад, якщо необхідно узнати і вивести всіх користувачів веб-ресурсу, буде зроблено такий запит:

```
Query: {
    totalUsers: () => users.length
    allUsers: () => users
}
```

totalUsers відповідає за виведення кількості користувачів в цілому, а allUsers – за їх відтворення. Якщо виникне потреба десь використати цю інформацію, система вже знатиме що саме повертати у відповідь на запити.

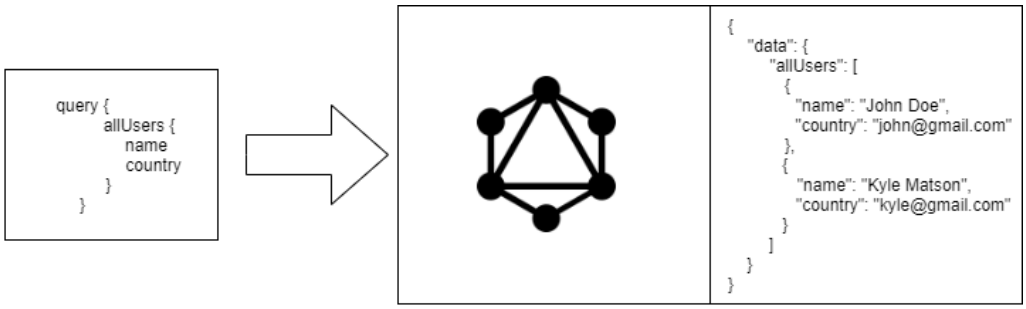


Рисунок 2.5 – GraphQL запит

Завдяки цьому круговороту подій, веб-розробники на сторонах серверу та інтерфейсу можуть більше не спілкуватись між собою, адже схема визначена. Вона є таким собі мостом між програмістами, оскільки виконавці заздалегідь знають структуру даних і що буде відправлено по мережі. На стороні клієнту навіть не потрібна допомога серверу при тестуванні продукту. Функція підробки інформації зробить все за них. Це також є плюсом GraphQL, адже робота пришвидшується та спростовується.

GraphQL технологія також пропонує функцію так званих «мутацій». За допомогою неї можливо робити такі дії:

- створювати нові дані;
- оновлювати існуючі;
- видаляти існуючі;

Вони мають таку ж саму структуру, як і звичайні запити, але починаються зі слова «mutation» замість «query». Приклад цього наведений нижче:

```
mutation {
    createUser(name: "John", birthday: "27.10.1996") {
        id
        name
        birthday
    }
}
```

Таким чином був створений новий користувач системи з іменем “John” та днем народження у вигляді строкових даних “27.10.1996”. В круглих дужках ми

вводимо аргументи, які сервер потім використовує для формування нового учасника. На цей запит сервер відповість наступною JSON-відповіддю:

```
"createUser": {
    "id": 2,
    "name": "John",
    "birthday": "27.10.1996"
}
```

У схемі після дужок вказано саме яке повернення бажано бачити від системи і тому отримано 3 пункти. Звісно ж, як і для звичайного запиту, повинний бути заздалегідь описаний розпізнавач для мутації:

```
type Mutation {
  createUser(name: String!, birthday: String!): User!
}
```

Як видно з цього прикладу, в коді ім’я та день народження позначені як строкове значення. Детальніше про типи даних описано вище, у таблиці 1.3. Такий вид запитів може використовуватись, наприклад, для створення нового користувача в системі після його реєстрації на веб-ресурсі. Прикладом для зміни існуючих даних буде заміною старого пароля на новий, якщо учасник захоче цього. А видалення - це, наприклад, коли потребується очистити базу даних серверу або сайту в цілому.

Мінус REST не тільки у тому, що він може віддавати забагато інформації у відповіді, коли це не потрібно, але й навпаки – мало. Це називається «проблема (n+1)-запиту». Уявимо, що для рішення задачі необхідно роздобути все ті ж самі статті користувача. У випадку з REST, спочатку потрібно зробити запит на кінцеву точку */api/users/:id* для отримання учасника, а потім уже, для кожного із них, відправити запит на кінцеву точку */api/users/:id/articles* і уже вивести саме шуканні статті. І так доведеться робити кожен раз.

GraphQL – це чудове рішення для проблем пов’язаних із отриманням необхідних даних щодо студентів Сумського Державного Університету. E-Olymp використовує його у своєму інтерфейсі прикладного програмування, а тому це облегшує задачу. Можливість робити запити безпосередньо до навчального веб-ресурсу і виводити інформацію, надає нам максимальну

швидкодію і працездатність, позбавляє зайві витрати часу на це. Глядач розробленої веб-системи для відслідковування студентів отримає найкращій досвід користування додатком завдяки плавній і гнучкій реалізації.

2.2 Інтерфейс прикладного програмування.

React є одним із найпопулярніших та найсучасніших фреймворків для роботи над веб-ресурсом. Він поєднує в собі швидкість, простоту, гнучкість і надійність. Все що потрібно, адже саме ці параметри грають ключову роль при виборі технології. Також гарним плюсом є те, що він не містить в собі великих нагромаджень різних технологій та пристосований до будь-яких проектів, особливо невеликих та одно-сторінкових, як уже було сказано раніше.

React – це ефективний та гнучкий фреймворк для будування веб-інтерфейсів. Користувацький інтерфейс «збирається» із так званих компонентів, що було вже розглянуто на прикладі технології Vue.js (рис. 1.5).

Кожен компонент відповідає за свій «кусочок» веб-сторінки. Один може відтворювати меню, деякий показує блок із головним контентом, таким як новини, а інший, наприклад, відсік із даними для контактування, або коментарями. Всі вони виконують свою задачу, а поєднуючись - утворюють один єдиний сайт.

Кожен із компонентів повертає готовий опис того, що він має показувати на екрані веб-сторінки. Сам фреймворк зчитує ці дані та відтворює на сайті. Приклад коду для відтворення елементу наведений нижче:

```
class HelloWorld extends React.Component {
    render() {
        return React.createElement(
            "div",
            null,
            "Привіт світ!"
        );
    }
}

ReactDOM.render(React.createElement(HelloWorld),
document.getElementById('example'));
```

Але всі програмісти в основному використовують JSX – спеціальний синтаксис для покращення розуміння та читабельності коду. Це також

дозволяє структурувати його. Із використанням JSX-синтаксису, попередній приклад виглядатиме так:

```
class HelloWorld extends React.Component {
    render() {
        return (
            <div>
                Привіт, світ!
            </div>
        );
    }
}

ReactDOM.render(<HelloWorld />, document.getElementById('example'));
```

Компоненти бувають двух типів:

- функціональні;
- класові;

Перший випадок потребується коли необхідно повернути нескладний фрагмент сторінки і для цього необов'язково використовувати складні нагромадження коду. Прикладом буде звичайна функція, яка повертає невелику кількість коду і не вимагає використання класів:

```
function HelloName(props) {
    return <h1>Hello, {props.name}</h1>;
}
```

Останні ж, навпаки, популярні там, де необхідно розробити елемент сайту як одну цілу структуру, наприклад, таблицю чи якусь дошку. Позначаються вони за допомогою слова "class":

```
class App extends React.Component {
  constructor(props) {
    super(props);
  }
  state = {
    isLoading: true
  }
  componentDidMount() {
    this.setState.isLoading = false;
  }
  render() {
    return (
      <div>
        {this.state.isLoading ? <h1>loading...</h1> : <h1>Hello,
{this.props.name}</h1>
      </div>
    )
  }
}
```

Даний випадок також називається компонентом зі станом. Тобто, окрім вхідних даних (props), що передаються до класу чи функції, елемент також підтримує внутрішній стан змінних (state), які доступні через *this.state*. Плюсом даного типу компонента є те, що він визиває візуалізацію даних знову та оновлює змінену інформацію. Приклад вище поєднує в собі і стан, і вхідні дані одразу. У функціональних компонентів немає стану, лише вхідні дані. Вони повертають лише те, що повинно бути візуалізовано від самого початку, без подальших змін.

Таким чином, керуючи циклом життєдіяльності кожного компонента, за допомогою станів і вхідних даних, розробник контролює всі елементи веб-ресурсу окремо один від одного, а поєднуючи утворює швидкий, гнучкий та функціональний сайт з усіма необхідними інструментами та зручностями для користувача. React найбільше підходить в даному випадку для виконання усіх цих задач одночасно.

Для поєднання клієнтської і серверної частини, було інтерпретовано технологію Apollo [6]. Це бібліотека на JavaScript мові, створена спеціально для вирішення проблеми з’єднання серверу на GraphQL із інтерфейсом програмного забезпечення, що в цьому випадку є React. Створена для останньої версії React, Apollo Client слугує «обгорткою» для нього та дозволяє отримувати дані від серверу і застосовувати їх у своїх потребах – створенні компонентів та наповнення їх коректною інформацією. Бібліотека відправляє запити до інтерфейсу прикладного програмування та отримує їх без необхідності впливати на React і відслідковувати зміну стану компонента. Приклад коду ми можемо бачити нижче:

```
const Users = () =>
    <Query
        query={gql'
            {
                totalUsers
                allUsers {
                    id
                    name
                    age
                }
            }
        '}
    >
        {({ data, loading }) => loading ?
            <h1 className="text-center">loading users...</h1> :
            <UserList count={data.totalUsers} users={data.allUsers} />
        }
    </Query>;
```

Одне із чудових рішень для поставленої задачі – це встановлення бази даних MongoDB. Найбільш зручна документо-орієнтована безкоштовна модель даних для настільних та веб-додатків. Особливо в парі з використанням JavaScript мови. MongoDB є NoSQL сховищем інформації, а тому не потребує будування графіків або SQL-запитів, визначення схем та ідентифікаторів, у вигляді зовнішніх ключів, для обробки даних. Розробник в змозі сам динамічно змінювати інформацію як побажає. Це невеликий проект, який робить нескладні запити до СУБД, а тому їх швидкість виконання буде на максимальному рівні, особливо коли немає нагромаджень технологій, як у конкурентів із реляційним направленням. Оскільки необхідні дані надходять у вигляді JSON-файлу, тобто вони не структуровані як у інших СУРБД, то не реляційний підхід до рішення проблеми є найбільш прийнятим.

Початкова задача – діставати інформацію із ресурсу E-Olymp, зберігати їх у базі даних та передавати на клієнтську частину. Необхідні технології були обрані для реалізації цих пунктів, але що з приводу частоти відправлення запитів? Щоб не «закидати» зверненнями веб-об’єкт, було вирішено робити вимогу раз в деякий час - наприклад, раз в годину або добу. Зазвичай, коли необхідно запланувати виконання роботи на деякий час, або протягом певного інтервалу, імпортують різні додатки до програми. У даному випадку це не є виключення, а оскільки Node.js застосовується як обгортка, а JavaScript – як

основна мову, то ефективніше всього буде використання бібліотеки cron. Додаток до програми ідеальніше всього підходить для подібних цілей, особливо при моніторингу певної інформації.

Установлення бібліотеки відбувається за допомогою обгортки Node.js та його менеджеру пакетів – скорочено npm. Приклад команди для імпорту додатку знаходиться нижче:

```
npm i cron
```

Відбувається процес за такою схемою:

Рисунок 2.6 – Приклад роботи бібліотеки cron

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Описання роботи веб-додатку.

Система трекінгу користувачів веб-ресурсу шляхом взаємодії з GraphQL – це проект, який дозволяє переглядати студентів Сумського Державного Університету на розробленому веб-додатку. Програма буде відслідковувати їх успіхи на науковому повчальному ресурсі під назвою E-Olymp. Останнє - це система яка проводить заходи та спортивні олімпіади по програмуванню, а також заохочує всіх бажаючих брати участь в них. Але це не все, оскільки об’єкт ще містить збірник задачок по написанню коду у розмірі більше десяти тисяч. Всі, хто цікавиться розробкою додатків, навіть якщо не мають бажання брати участь у змаганнях, можуть спробувати свої сили у рішенні простих завдань та «прокачувати» свої навички, починаючи від легких рішень до більш складних. Це сприяє не тільки поліпшенню майстерності у написані коду, але й витривалості, розвиває мислення та здатність до аналізу поставленої задачі. Тому E-Olymp є чудовим науковим ресурсом не тільки для тих, хто програмує, але й любить розвиватись та вирішувати задачки з різних галузей математики. Система має свій рейтинг користувачів, з кількістю балів нарахованих за проходження той чи іншої задачки. Також, у кожного учасника відображається кількість пройдених запитань різного рівня складності, скільки знадобилося рішень в цілому на рішення цього всього та ранг, який відображає дійсні навички програміста. Веб-ресурс усіма зусиллями заохочує своїх учнів до проходження задач по написанню коду, спортивних змагань та олімпіад для різного рівня школярів, студентів і просто всіх бажаючих перевірити свої знання.

Звісно ж там є і студенти Сумського Державного Університету з їх особистим рангом та рейтингом, кількістю вирішених задачок та спроб. Головна мета системи трекінгу як раз і полягає у відслідковуванні саме цих учасників та виведенні їх результатів роботи на веб-ресурсі у виді таблиці та особистої сторінки з детальною інформацією про них. Проект матиме дві

сторінки, одна з яких буде головною, що показуватиме загальну таблицю з користувачами, їх рангом, рейтингом і цифрами пов'язаними з рішенням задачок.

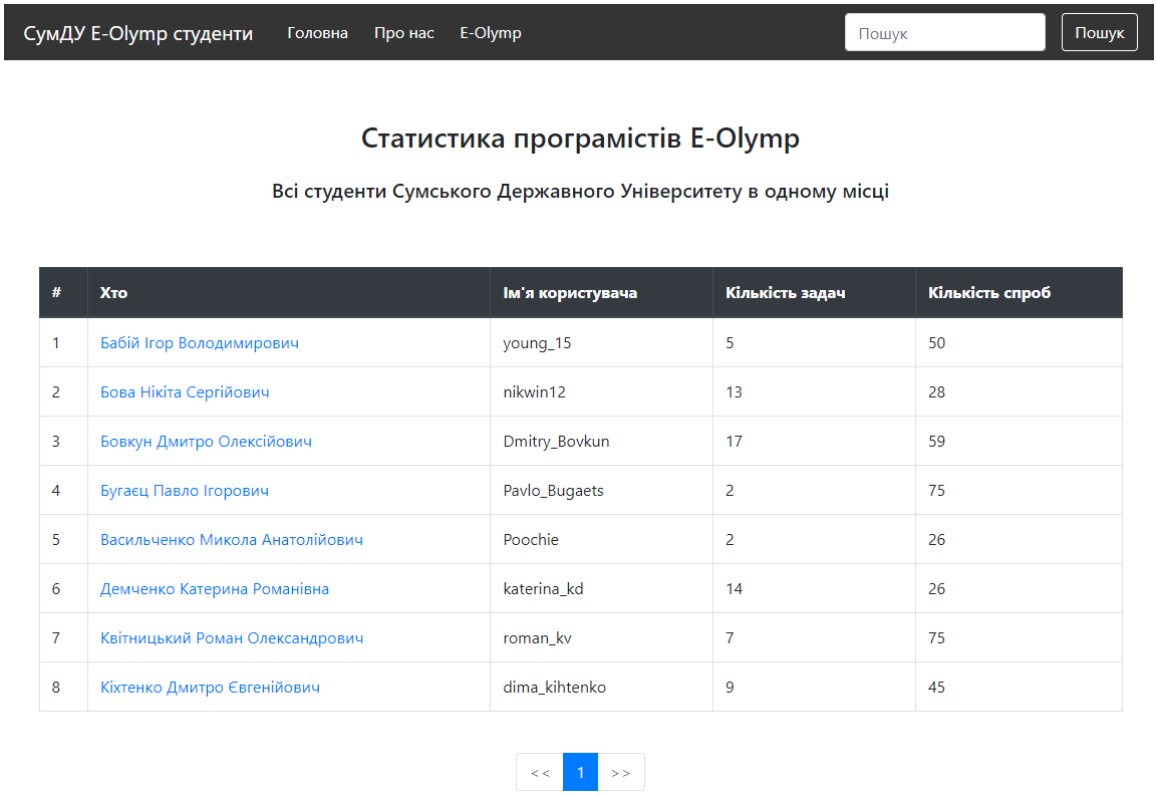


Рисунок 3.1 – Головна сторінка веб-сайту

А інша ж – це особиста сторона студента, на якій буде детально показано вирішення поставлених запитань на E-Olymp з кількістю витрачених рішень на це.

Рисунок 3.2 – Особиста картка користувача

Сторінка також складається з графіку, присутнього на всіх користувацьких панелях, який показуватиме інформацію про те коли саме задачки були зроблені програмістом, кількість зайнятих рішень на це та складність самого запитання. Це допоможе більш детально розкрити темп та ріст студента як фахівця, а також його знання.

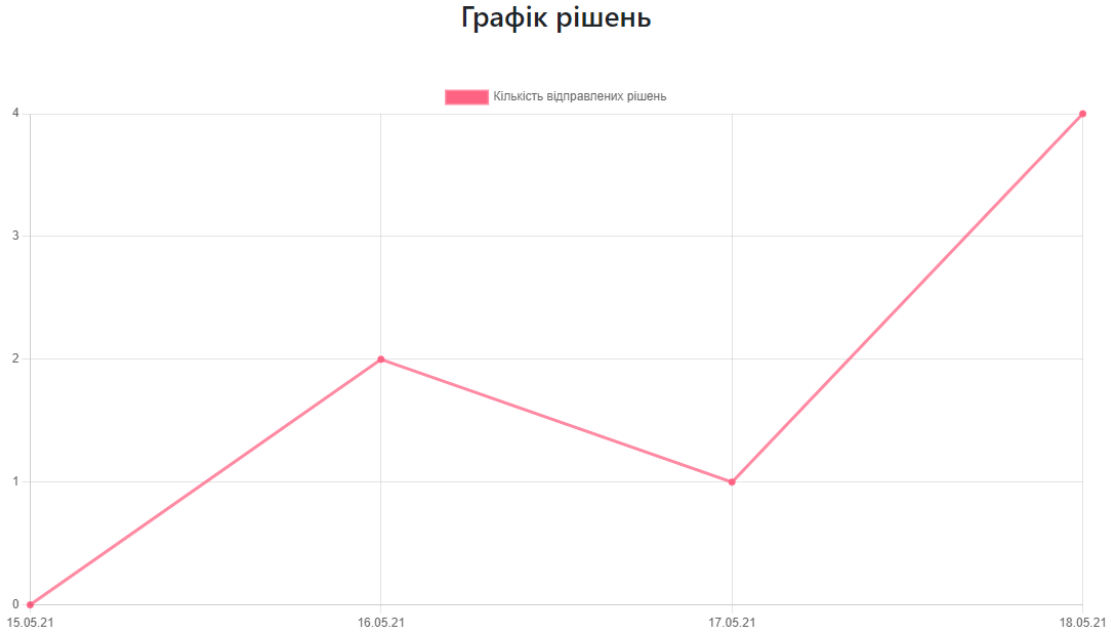


Рисунок 3.3 – Графік рішень програміста

3.2 Технології для рішення поставленої задачі.

Головна мета – це сканувати веб-ресурс E-Olymp на явність студентів Сумського Державного Університету раз в якийсь період часу. Наприклад, кожного дня або години. За допомогою технології GraphQL діставати звідти інформацію про них, оскільки навчальний сайт використовує саме його як інтерфейс прикладного програмування, зберігати її та показувати на розробленому веб-додатку у вигляді аналітичної статистики та графіку. Все це можна розбити на декілька шагів:

- запит до E-Olymp за даними;
- зберегти їх до бази даних;
- витягнути з бази даних та відправити на клієнтську частину;
- вивести це у вигляді компонентів (таблиця, статистика, графік).

Для початку потрібно налаштувати проект. Він має GraphQL Apollo технології як засіб для формування серверної частини, а React Apollo – для створення інтерфейсу веб-додатку. Нижче наведений код запуску серверу із підключенням GraphQL Apollo:

```
const { ApolloServer } = require('apollo-server-express');
const express = require('express');
const typeDefs = readFileSync('./src/schema.graphql', 'utf-8');
const resolvers = require('./src/resolvers');
const expressPlayground = require('graphql-playground-middleware-
express').default;

const app = express();
const server = new ApolloServer({
    typeDefs,
    resolvers
});

server.applyMiddleware({ app });
app.get('/', (req, res) => res.end('Welcome to the GraphQL application'));
app.get('/playground', expressPlayground({ endpoint: '/graphql' }));
app.listen({ port: 4000 }, () => console.log(`GraphQL Service Running @
http://localhost:4000${server.graphqlPath}`));
```

Apollo Client застосовується також і на клієнтській стороні, оскільки завдяки ньому відбувається “спілкування” між ним і сервером. Код запуску React та підключення Apollo Client до нього наведений нижче:

```
import ApolloClient, { gql } from 'apollo-boost';
import { ApolloProvider } from 'react-apollo';

const client = new ApolloClient({
  uri: 'http://localhost:4000/graphql'
});

ReactDOM.render(
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>,
  document.getElementById('root')
);
```

Технологія відправляє запити, необхідні для відтворення певної інформації, до підготовленого GraphQL, який, у свою чергу, робить вимогу до бази даних, та отримує дані від нього.

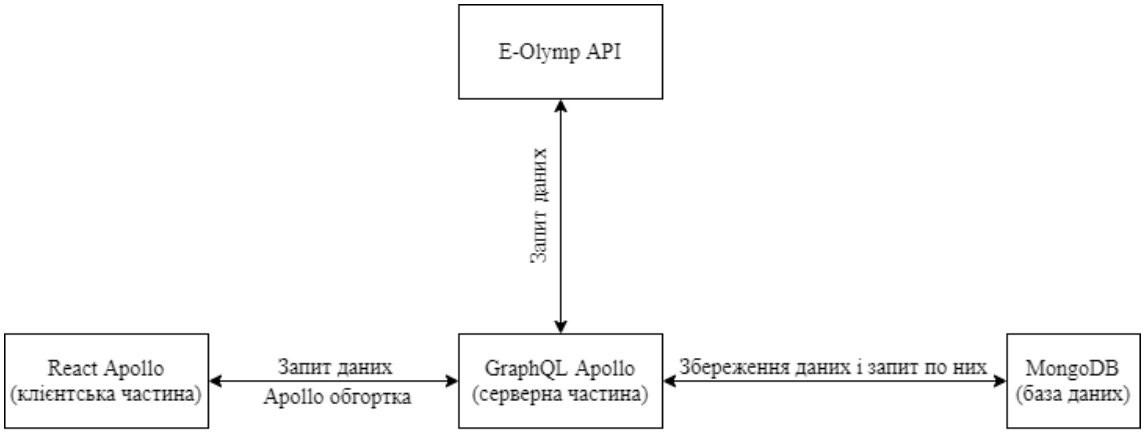


Рисунок 3.4 – Схема передачі даних

Це відтворює наші пункти із списку. По перше нам необхідно зробити запит до інтерфейсу прикладного програмування веб-ресурсу E-Olymp та запросити дані звідти. Код виконання даної дії наведений нижче:

```
cron.schedule('* * * * *', () => {
  fetch('https://api.e-olymp.com/oauth/token', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: 'grant_type=password&username=serg_pet&password=O08gjM9E'
  })
  .then(res => res.json())
  .then((data) => {
    fetch('https://api.e-olymp.com/graphql', {
      method: 'POST',
      headers: {
        'Authorization': 'Bearer ${data.access_token}',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        query: '
          {
            cognito {
              user1: user(id: 1) {
                name
              }
              user2: user(id: 5522) {
                name
              }
              user3: user(id: 5525) {
                name
              }
            }
          }
        ',
      }),
    }).then(res => res.json()).then(json => console.log(json.data.cognito));
  })
});
```

Код одночасно виконує і наступний шаг нашого списку, а саме збереження даних до бази даних MongoDB. Це відбувається відразу після отримання даних від веб-ресурсу для облегшення рішення задачі та пришвидшення процесу.

В прикладі також можна побачити використання бібліотеки cron, про яку було згадано у пункті 2. Додаток сканує кожної доби веб-ресурс E-Olymp та оновлює інформацію про студентів, тим самим дозволяючи графіку та статистиці більш детально розкривати діяльність учнів.

Наступні шаги виконуються за допомогою React Apollo. Перший використовує цю можливість та отримує інформацію, що надходить від інтерфейсу прикладного програмування. Бібліотека, завдяки обгортці Apollo,

створює компоненти і наповнює їх динамічними даними розміщуючи на веб-додатку. Наглядний код розміщення користувачів на головній сторінці через відправлення запиту на сервер наведений нижче:

```
export const ROOT_QUERY = gql'
  query {
    totalCognito
    allCognito {
      id
      name
      username
      problems
      solves
    }
  }
';

const Users = () =>
    <Query query={ROOT_QUERY} fetchPolicy="cache-and-network">
        {({ data, loading }) => loading ?
            <h1 className="text-center">Завантаження користувачів...</h1> :
            <div className="container">
                <h3 className="text-center">Статистика програмістів E-
Olymp</h3>
                <h5 className="text-center">Всі студенти Сумського Державного
Університету в одному місці</h5>
                <Table count={data.totalCognito} data={data.allCognito} />
            </div>
        }
    </Query>;

export default Users;
```

Apollo відправляє запит, за допомогою функції Query, на інтерфейс технології GraphQL, отримує дані від нього та передає їх у компонент React, який потім відтворюється у вигляді таблиці користувачів на головній сторінці. Інформація, що надходить, визначена у змінної як:

- totalCognito – загальна кількість учасників;
- allCognito – список всіх програмістів з їх ідентифікатором, ім’ям, логіном, кількістю вирішених задач та потрачених рішень на це.

Все це визначено заздалегідь за допомогою схеми GraphQL. Її код наведений нижче:

```
type Query {
    cognito(username: String!): CognitoUser!
    totalCognito: Int!
    allCognito: [CognitoUser!]!
    fetchDate: Int!
}

type CognitoUser {
    id: ID!
    username: String!
    name: String!
    problems: Int
    solves: Int
    location: Location!
    avatar: String
}

type Location {
    country: String!
    city: String!
}
```

Лише по даній системі можливо запитувати дані у інтерфейсу прикладного програмування, що і дозволяє отримувати саме ті необхідні дані, які точно будуть застосовуватись. Про це було сказано у пункті 2, як і про розвізнавачі. Код останніх до даної схеми наведений нижче:

```
module.exports = {
    Query: {
        cognito: (parent, { username }, { db }) =>
db.collection('users').findOne({ username: username }),
        totalCognito: (parent, args, { db }) =>
db.collection('users').estimatedDocumentCount(),
        allCognito: (parent, args, { db }) =>
db.collection('users').find().sort({ id: 1 }).toArray(),
        fetchDate: async (parent, args, { db }) => {
            const date = await db.collection('dates').findOne();
            return date.fetchDate.minutes;
        }
    }
};
```

Розпізнавачі позначують для системи, що при відправленні запиту до серверу за даними, необхідно зробити в свою чергу вимогу до бази даних та дістати інформацію звідти для того, щоб передати її на клієнтську сторону.

Це все відбувається якщо споживач переглядає веб-додаток вперше. Потім отримана інформація зберігається у кеші браузеру, і при наступному оновлені сайту, Apollo, завдяки функції fetchPolicy, буде передавати дані у компонент звідти, замість ще одного запиту до серверу. Це гарний плюс

бібліотеки, оскільки пришвидшує додаток та зменшує споживання ресурсів і часу. Код для підключення функції кешування наведений нижче:

```
const cache = new InMemoryCache();
persistCache({
  cache,
  storage: localStorage
});
if (localStorage['apollo-cache-persist']) {
  let cacheData = JSON.parse(localStorage['apollo-cache-persist']);
  cache.restore(cacheData);
}

const client = new ApolloClient({
  cache,
  uri: 'http://localhost:4000/graphql'
});
```

Завдяки такому поєднанню технологій та круговороту подій, користувачі веб-додатку можуть переглядати студентів Сумського Державного Університету, їх досягнення та рейтинги на веб-ресурсі E-Olymp без потреби заходити безпосередньо на сам ресурс. Додаткові розроблені особливості у вигляді відфільтрованих потрібних учнів з їх детальною статистикою та графіком надають споживачам системи найкращий досвід користування.

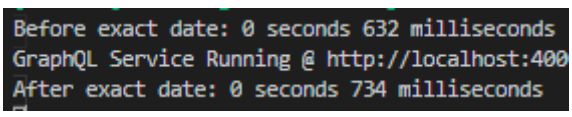
ВИСНОВКИ

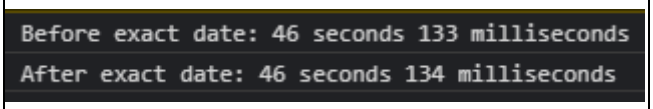
Результатом виконання даної дипломної роботи став розроблений аналітичний веб-додаток для відтворення результатів навчання і практичного застосування програмування студентів Сумського Державного Університету. Завдяки співпраці з веб-ресурсом E-Olymp та їх навчальною програмою, була здійснена реалізація програмного забезпечення для відтворення реальних навичок учнів та їх прагнення до опанування написання коду і виконання різних олімпіад.

Мета завдання була виконана. Серверна частина веб-сайту построєна на основі мови запитів і маніпулювання – GraphQL, із допомогою технології Apollo. Останнє поєднує веб-сервер із клієнтською частиною, що запрограмована на JavaScript-фреймворку React. Завдяки бібліотеці був створений інтерфейс для виведення інформації у вигляді компонентів основної таблиці користувачів, особистої картки учнів, яка зображує їх основні дані та графіку, що демонструє активність на пройденому шляху програміста.

Завдяки такому поєднанню обраних технологій, було розроблено швидкодіючий веб-додаток, який не має вад та працює без перешкод. Нижче наведені результати виконання програмою відправлення запиту до серверу E-Olymp, діставання звідти інформації щодо студентів, їх збереження та виведення на інтерфейс сайту:

Таблиця 1.4 Швидкість додатку

Дії	**Швидкість**	**Скріншот**
Відправлення запиту на сервер E-Olymp для отримання даних та їх збереження до бази	Приблизно 100 мілісекунд або 0.1 секунда	

Відправлення запиту на наш сервер для отримання даних із бази та їх виведення на інтерфейс	Приблизно 1 мілісекунда або 0.01 секунда	

СПИСОК ЛІТЕРАТУРИ

1. Херрон Д. Розробка серверних веб-додатків на JavaScript / Д. Херрон / Packt Publishing, 2020. – 376 с.
2. Ломбарді Е. Початок Spring 5: від початківця до фахівця / Е. Ломбарді, Д. Оттінгер / Apress, 2019. – 379 с.
3. Мецгар Д. .NET Core in Action / Д. Мецгар / Manning Publications, 2018. – 288 с.
4. Дейл К. Візуалізація даних за допомогою Python та JavaScript / К. Дейл / O'Reily, 2016. – 210 с.
5. Кришнамурти Б. Web-протоколи. Теорія та практика / Б. Кришнамурти, Д. Рексфорд / Binom Press, 2010. – 592 с.
6. Порселло Е. GraphQL мова запитів для сучасних веб-додатків / Е. Порселло, А. Бенкс / O'Reily, 2019. – 204с.
7. Порселло Е. React та Redux: функціональна веб-розробка / Е. Порселло, А. Бенкс / O'Reily, 2018. – 336с.
8. Грін Б. AngularJS / Б. Грін, Ш. Сешадрі / O'Reily Media, 2013. – 196 с.
9. Макре К. Vue.js: запуск і робота: створення доступних і продуктивних веб-додатків та програм / К. Макре / O'Reily Media, 2018. – 174 с.
10. Паціанський М. React.js курс для початківців / М. Паціанський – 2018. – 146с.
11. Річардсон Л RESTful Web APIs / Л. Річардсон, М. Амундсен / O'Reily, 2013. – 404с.
12. Чекалов А. Базы данных: от проектування до розробки додатків / А. Чекалов / СПб.: БХВ-Петербург, 2003. – 384с.
13. Маррс Т. JSON на роботі: практична інтеграція даних для інтернету / Т. Маррс / O'Reily Media, 2017. – 376с.
14. Мейер Е. CSS: Візуальна презентація для інтернету / Е. А. Мейер / O'Reily Media, 2011. – 258 с.

ДОДАТОК

server/index.js

```
const { ApolloServer } = require('apollo-server-express');
const express = require('express');
const fs = require('fs');
const typeDefs = fs.readFileSync('./src/schema.graphql', 'utf-8');
const resolvers = require('./src/resolvers');
const expressPlayground = require('graphql-playground-middleware-
express').default;
const fetch = require('node-fetch');
const { MongoClient } = require('mongodb');
const cron = require('node-cron');

require('dotenv').config();

var db;

const query = '
{
  cognito {
    user1: user(id: 1) {
      name
    }
    user2: user(id: 5522) {
      name
    }
    user3: user(id: 5525) {
      name
    }
  }
}
';
const real_users = [
  {
    name: 'Бабій Ігор Володимирович',
    username: 'young_15',
    specialization: 'Кібербезпека',
    group: 'КБ-91'
  },
  {
    name: 'Бова Нікіта Сергійович',
    username: 'nikwin12',
    specialization: 'Електротехнічні системи електроспоживання',
    group: 'ЕТ-91'
  },
  {
    name: 'Бовкун Дмитро Олексійович',
    username: 'Dmitry_Bovkun',
    specialization: 'Кібербезпека',
    group: 'КБ-91'
  },
  {
    name: 'Бугаєц Павло Ігорович',
    username: 'Pavlo_Bugaets',
    specialization: 'Комп'ютеризовані системи управління та робототехніка',
    group: 'СУ-91'
  },
  {
    name: 'Васильченко Микола Анатолійович',
    username: 'Poochie',
```

```
    specialization: 'Наука про дані та моделювання складних систем',
    group: 'ПМ.м-01'
  },
  {
    name: 'Демченко Катерина Романівна',
    username: 'katerina_kd',
    specialization: 'Прикладна математика',
    group: 'ПМ-91'
  },
  {
    name: 'Квітницький Роман Олександрович',
    username: 'roman_kv',
    specialization: 'Кібербезпека',
    group: 'КБ-91'
  },
  {
    name: 'Кіхтенко Дмитро Євгенійович',
    username: 'dima_kihtenko',
    specialization: 'Кібербезпека',
    group: 'КБ-91'
  },
];

async function start() {
  const app = express();

  try {
    const client = await MongoClient.connect(
      process.env.DB_HOST,
      {
        useNewUrlParser: true,
        useUnifiedTopology: true
      }
    );
    db = client.db('gql-app-db');
  } catch (err) {
    console.log('Database connection error thrown: ${err}');
    process.exit(1);
  }

  const context = { db };

  const server = new ApolloServer({
    typeDefs,
    resolvers,
    context
  });

  server.applyMiddleware({ app });
  app.get('/', (req, res) => res.end('Welcome to the GraphQL application'));
  app.get('/playground', expressPlayground({ endpoint: '/graphql' }));
  app.listen({ port: 4000 }, () => console.log('GraphQL Service Running @
http://localhost:4000${server.graphqlPath}'));

  let date = new Date();

  try {
    await db.collection('dates').deleteOne({});
    await db.collection('dates').insertOne({
      fetchDate: {
        hours: date.getHours(),
        minutes: date.getMinutes(),
        seconds: date.getSeconds()
```

```
      }
    });

  } catch (err) {
    console.log(err);
  }

  fetch(process.env.FETCH_TOKEN, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body:
'grant_type=password&username=${process.env.USER}&password=${process.env.PASS
WORD}'
  })
    .then(res => res.json())
    .then((data) => {
      fetch(process.env.FETCH_SITE, {
        method: 'POST',
        headers: {
          'Authorization': 'Bearer ${data.access_token}',
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({ query: query })
      })
        .then(res => res.json())
        .then(await = (res) => {
          db.collection('test').deleteMany({});
          db.collection('test').insertOne(res);
        })
        .catch(err => console.log('Error thrown: ${err}'));
    })
    .catch(err => console.log('Error thrown: ${err}'));

  // await createUsers(8);
}

start();

async function createUsers(num) {
  await db.collection('users').deleteMany({});
  await db.collection('real_users').deleteMany({});

  var { results } = await
fetch('https://randomuser.me/api/?results=${num}').then(res => res.json());
  const users = results.map((user, index) => ({
    id: index,
    // name: '${user.name.first} ${user.name.last}',
    // username: user.login.username,
    problems: user.registered.age,
    solves: user.dob.age,
    location: {
      country: user.location.country,
      city: user.location.city
    },
    avatar: user.picture.large,
    ...real_users[index]
  }));
  await db.collection('real_users').insertMany(users);

  return users;
}
```

client/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import ApolloClient, { InMemoryCache } from 'apollo-boost';
import { ApolloProvider } from 'react-apollo';
import { persistCache } from 'apollo-cache-persist';

import 'bootstrap/dist/css/bootstrap.min.css';
import './index.css';

const cache = new InMemoryCache();
persistCache({
  cache,
  storage: localStorage
});

if (localStorage['apollo-cache-persist']) {
  let cacheData = JSON.parse(localStorage['apollo-cache-persist']);
  cache.restore(cacheData);
}

const client = new ApolloClient({
  cache,
  uri: 'http://localhost:4000/graphql'
});

ReactDOM.render(
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>,
  document.getElementById('root')
);
```

App.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import { gql } from 'apollo-boost';
import { withApollo } from 'react-apollo';
import Menu from './landing/Header';
import ContentUsers from './landing/ContentUsers';
import ContentTable from './landing/ContentTable';
import ContentChart from './landing/ContentChart';
import ContentCTA from './landing/ContentCTA';
import Footer from './landing/Footer';
import User from './user-page/User';
import AboutUs from './about-us/AboutUs';

export const ROOT_QUERY = gql'
  query {
    totalCognito
    allCognito {
      id
      name
      username
      problems
      solves
    }
  }
';

class App extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div className="container main">
        <Router>
          <Menu />
          <Switch>
            <Route path="/user/:username" children={<User />} />
            <Route path="/about-us" children={<AboutUs />} />
            <Route path="/">
              <ContentUsers />
              <ContentTable />
              <ContentChart />
              <ContentCTA />
            </Route>
            <Route component={({ location }) => <h1 className="text-
center">"{location.pathname}" not found</h1>} />
          </Switch>
          <Footer />
        </Router>
      </div>
    );
  }
}

export default withApollo(App);
```

ContentUsers.js

```
import React from 'react';
import ReactPaginate from 'react-paginate';
import { Link } from 'react-router-dom';
import { Query } from 'react-apollo';
import { ROOT_QUERY } from '../App';

class Table extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            currentPage: 0,
            data: [],
            offset: 0,
            pageCount: 0,
            perPage: 10
        }

        this.handlePageClick = this.handlePageClick.bind(this);
    }

    receiveData() {
        let slice = this.props.data.slice(
            this.state.offset,
            this.state.offset + this.state.perPage
        );

        let postData = (
            <table className="table table-stripped table-bordered">
                <thead className="thead-dark table-stripped">
                    <tr>
                        <th>#</th>
                        <th>Хто</th>
                        <th>Ім'я користувача</th>
                        <th>Кількість задач</th>
                        <th>Кількість спроб</th>
                    </tr>
                </thead>
                <tbody>
                    {slice.map(item => (
                        <tr key={item.id}>
                            <td>{Number(item.id) + 1}</td>
                            <td><Link to={{ pathname:
'/user/${item.username}' }}>{item.name}</Link></td>
                            <td>{item.username}</td>
                            <td>{item.problems}</td>
                            <td>{item.solves}</td>
                        </tr>
                    ))}
                </tbody>
            </table>
        );

        this.setState({
            pageCount: Math.ceil(this.props.count / this.state.perPage),
            postData
        });
    }

    handlePageClick = (e) => {
        let selectedPage = e.selected;
        let offset = selectedPage * this.state.perPage;
```

```
        this.setState({
            currentPage: selectedPage,
            offset: offset
        }, () => {
            this.receiveData();
        });
    }

    componentDidMount() {
        this.receiveData();
    }

    render() {
        return (
            <div className="table">
                {this.state.postData}
                <ReactPaginate
                    previousLabel="<<"
                    nextLabel=">>"
                    breakLabel="..."
                    pageCount={this.state.pageCount}
                    onPageChange={this.handlePageClick}
                    containerClassName={"pagination justify-content-center"}
                    pageClassName={"page-item"}
                    pageLinkClassName={"page-link"}
                    previousLinkClassName={"page-link no-dis"}
                    nextLinkClassName={"page-link no-dis"}
                    disabledClassName={"page-item disabled"}
                    activeClassName={"page-item active"}
                />
            </div>
        );
    }
};

const Users = () =>
    <Query query={ROOT_QUERY} fetchPolicy="cache-and-network">
        {({ data, loading }) => loading ?
            <h1 className="text-center">Завантаження користувачів...</h1> :
            <div className="container">
                <h3 className="text-center">Статистика програмістів E-
Olymp</h3>
                <h5 className="text-center">Всі студенти Сумського Державного
Університету в одному місці</h5>
                <Table count={data.totalCognito} data={data.allCognito} />
            </div>
        }
    </Query>;

export default Users;
```

User.js

```
const query = gql'
    query Cognito($username: String!) {
        cognito(username: $username) {
            name
            problems
            solves
            location {
                country
                city
            }
            avatar
            specialization
            group
        }
    }
';
export default User =>
    <Query query={query} variables={{ "username":
window.location.pathname.split('/')[2] }} fetchPolicy="cache-and-network">
        {({ data, loading }) => loading ?
            <h1>Завантаження користувача...</h1> :
            <div className="user-page">
                <div className="container text-center my-5">
                    <h2 className="name py-0">{data.cognito.name}
статистика</h2>
                    <img className="img-fluid py-5" src={data.cognito.avatar}
alt="#" />
                    <div className="dates">
                        <div className="start">
                            <strong>ОНОВЛЕНО</strong>{new
Date().getSeconds()} секунд тому
                            <span></span>
                        </div>
                        <div className="end">

<strong>СПЕЦІАЛІЗАЦІЯ</strong>{data.cognito.specialization}
                        </div>
                    </div>
                    <div className="stats">
                        <div>
                            <strong>ГРУПА</strong> {data.cognito.group}
                        </div>
                        <div>
                            <strong>ПРОБЛЕМ ВИРІШЕНО:</strong>
{data.cognito.problems}
                        </div>
                        <div>
                            <strong>ВСЬОГО ВІДПРАВЛЕНИХ РІШЕНЬ:</strong>
{data.cognito.solves}
                        </div>
                        <div>
                            <strong>РЕЙТИНГ КОРИСТУВАЧА:</strong>
{((data.cognito.problems / data.cognito.solves) * 100).toFixed(2)}%
                        </div>
                    </div>
                </div>
                <div id="chart-two">
                    <Chart />
                </div>
            </div>
        }
    </Query>;
```

Chart.js

```
import React from 'react';
import { Line } from 'react-chartjs-2';

const data = {
    datasets: [
        {
            label: 'Кількість відправлених рішень',
            data: [0, 2, 1, 4],
            fill: false,
            backgroundColor: 'rgb(255, 99, 132)',
            borderColor: 'rgba(255, 99, 132, 0.75)',
        }
    ],
    labels: ['15.05.21', '16.05.21', '17.05.21', '18.05.21']
};

const options = {
    scales: {
        yAxes: {
            ticks: {
                beginAtZero: true,
                precision: 0,
            }
        }
    }
};

export default Chart =>
    <div className="container chart">
        <h3 className="text-center my-5 pt-5">Графік рішень</h3>
        <Line className="my-5" data={data} options={options} />
    </div>;
```