1.2. СТВОРЕННЯ WEB-ДОДАТКІВ ІЗ ВИКОРИСТАННЯМ ХМАРНИХ ОБЧИСЛЕНЬ. ТЕХНОЛОГІЯ MERN

1.2.1. Теоретичний модуль

Хмарні обчислення

Під терміном "***Хмара***" розуміється образ складної інфраструктури за якою приховуються всі технічні деталі сучасних методів обробки та обміну інформаційними даними. Безпосередньо "***хмарна обробка даних***" - це парадигма IEEE-2008 (англ. Institute of Electrical and Electronics Engineers - міжнародна організація інженерів у галузі електротехніки, радіоелектроніки та радіоелектронної промисловості, світовий лідер в галузі розроблення стандартів), в рамках якої дані постійно зберігається на серверах Інтернет, тимчасово кешуються на комп'ютері (ПК, ноутбуки, планшети тощо).

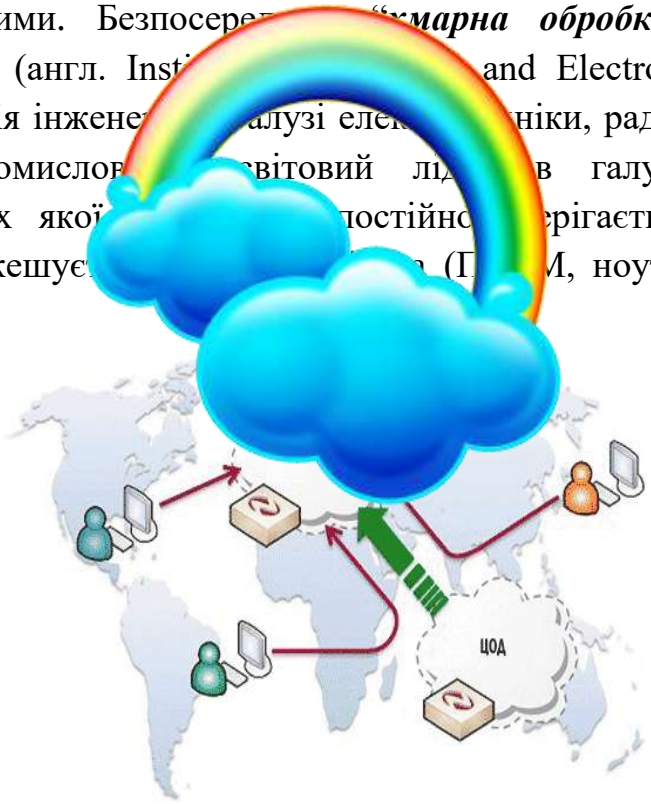

Рис. 1.4. Хмарна обробка даних

Хмарна обробка даних зорієнтована на доступі до глобальних сховищ даних у режимі on-line, їх обчислення та подальшої синхронізації для збереження на віддалених серверах, подання цих даних за викликом користувача.

Хмарні обчислення (англ. cloud computing) – це модель забезпечення повсюдного та зручного мережевого доступу до даних та до обчислювальних ресурсів згідно з конфігурацією користувацького програмного додатку, які можуть з мінімальними експлуатаційними витратами, оперативно надаватися сервісами хмарних обчислень. В цілому, сервіси хмарних обчислень є додатками, доступ до яких забезпечується через Інтернет за допомогою браузерів та інших мережевих застосувань доступу до серверів додатків, серверів баз даних та FTP серверів.

Сучасна ***розробка web-додатків*** орієнтована на застосуванні під час їх складання, так званих, ***ліній розроблення програм***, які ***об'єднуються у фабрику програм*** (рис. 1.5). Одним з найважливіших технічних ресурсів фабрик програм є алгоритми та розроблені на їх базі програми та ***компоненти повторюваного використання*** (КПВ) [10].

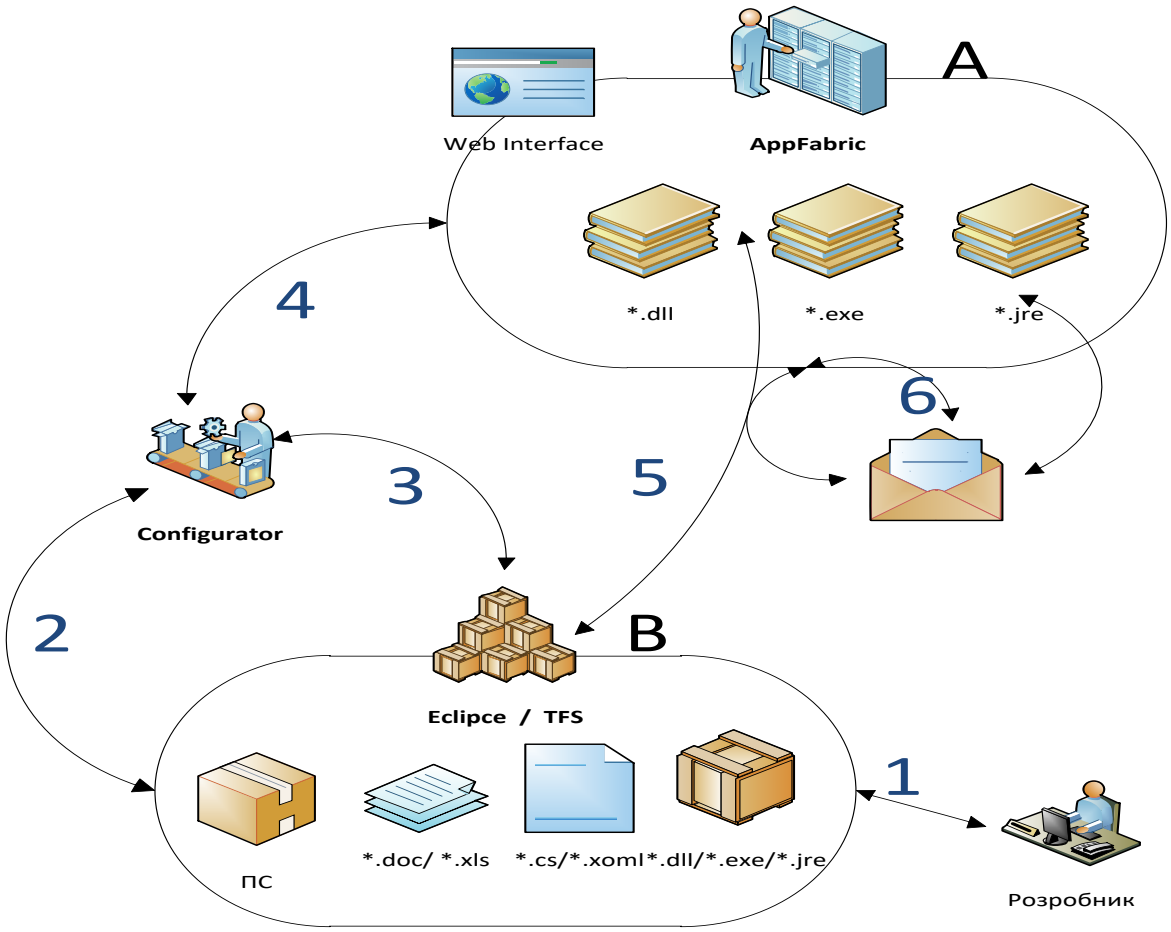

Рис. 1.5. Загальна структура фабрики програм [11]

Під час web-програмування організація обчислень потребує визначення таких методів складання та об'єднання ресурсів, які полягають у координації, кооперації та взаємодії різних сервісів і інших готових ресурсів через ***конфігураційний файл*** для виконання обчислень відповідних задач [11]. Тому, ***до компетентностей розробників web-додатків*** відноситься знання щодо

використання готових програмних ресурсів та вміння їх поєднання, вибудовування бізнес логіки за результатами об'єктного аналізу прикладної галузі та функціонального наповнення розроблюваної web-системи, вибору компонентного методу реалізації цих об'єктів, завдання інтерфейсів, необхідних під час складання у середині різних фабрик програм.

У сучасному web програмуванні широке застосування знаходить стек ресурсів MEAN/MERN, що є комбінацією трьох фреймворків JavaScript і технології баз даних на основі NoSQL (рис. 1.6). Стек MEAN/MERN ґрунтується на переважному застосуванні однієї мови програмування JavaScript впродовж розробки web-додатку.

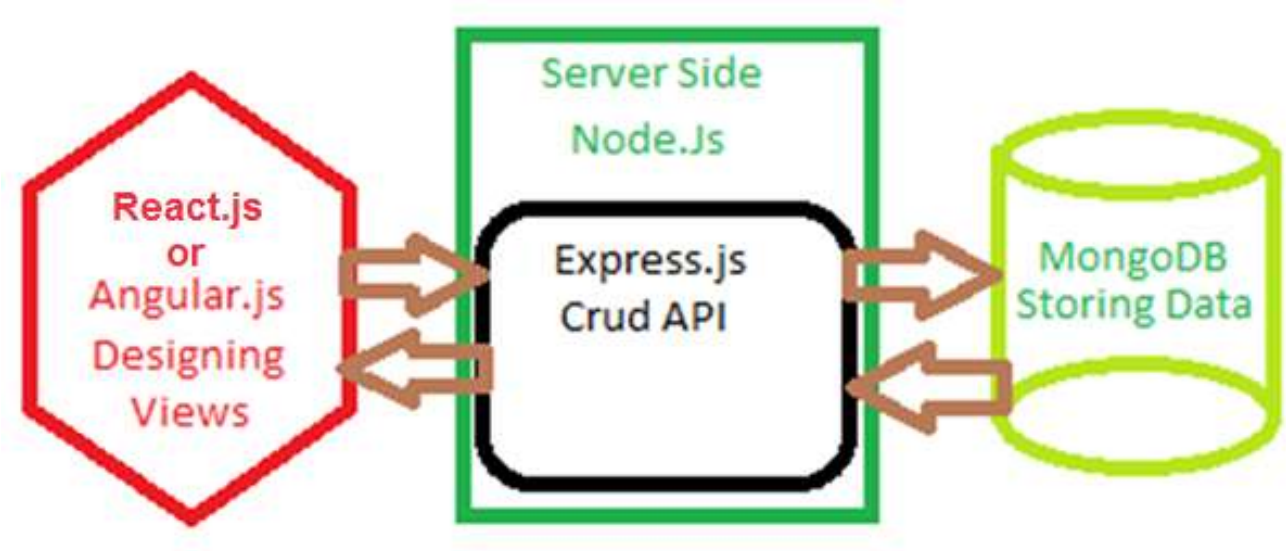

Рис. 1.6. Загальна структура MEAN/MERN [12]

MEAN (абревіатура від MongoDB, Express.js, Angular.js, Node.js) – стек (набір, комплекс) серверного програмного забезпечення, що використовується для web-розробки.

У стеці MERN Angular.js замінений на React.js – JavaScript бібліотеку для створення користувацького інтерфейсу (на рівні front-end).

MongoDB - це документно-орієнтована база даних з відкритим вихідним кодом, розроблена з урахуванням масштабованості та швидкості розробника. Замість того, щоб зберігати дані в таблицях і рядках, як у реляційній базі даних, у MongoDB документи зберігаються у колекціях (collections) у форматі даних, що є з динамічними схемами, подібних до JSON.

Express.js - це фреймворк сервера web-додатків Node.js, розроблений для створення односторінкових, багатосторінкових і гібридних web-додатків. Це - стандартна серверна платформа для Node.js (рівень back-end).

Angular.js/React.js - це бібліотечна структура для динамічних web-програм (на рівні front-end). Вони дозволяють використовувати HTML як мову шаблонів і дозволяють розширювати синтаксис HTML, щоб чітко і лаконічно утворювати компоненти web-додатків.

Node.js - це кросплатформне середовище виконання з відкритим вихідним кодом для розробки web-додатків на стороні сервера. Програми Node.js написані на JavaScript і можуть запускатися в середовищі виконання Node.js в OS X, Microsoft Windows, Linux, FreeBSD, NonStop, IBM AIX, IBM System.

Важлива ***особливість стеку MEAN/MERN***:

- перехід від генерації web-сторінок на стороні сервера до створення переважно односторінкових додатків;
- перенесення ядра реалізації архітектурного шаблону MVC зі сторони сервера на сторону клієнта, що забезпечується включенням в склад стеку Angular.js/React.js;
- включений в склад фреймворк Express.js забезпечує і традиційну маршрутизацію, і генерацію сторінок на стороні сервера.

Програмна платформа Node.js.

Node.js - платформа з відкритим кодом для виконання високопродуктивних мережевих додатків, написаних мовою JavaScript. Платформу розробив у 2009 р. Раян Дал (англ. Ryan Dahl). Якщо раніше Javascript застосовувався для обробки даних в браузері користувача, то node.js надав можливість виконувати ***JavaScript-скрипти*** на сервері та відправляти користувачеві результат їх виконання. Платформа Node.js перетворила JavaScript на мову широкого загального використання з великою спільнотою розробників на рівнях back-end та front-end.

Раніше розробники могли запускати JavaScript тільки в браузері, але від тепер, коли вони розширили його, ви можете запускати JS на своєму комп'ютері в якості окремого (на ПК віртуального) серверного додатка. Цей додаток і є Node.js.

Node.js - це платформа, а не середовище розробки JavaScript-додатків. Платформа окрім роботи із серверними скриптами для web-запитів, також використовується для створення клієнтських та серверних програм.

Із застосуванням Node.js розробляються ***DIRTy-додатки***. Абревіатура DIRT (data-intensive real-time) визначає додатки, що працюють у реальному часі та спроможні обробляти великі обсяги даних.

Платформою Node.js використовується розроблений компанією Google ***рушій V8***. V8 - рушій з відкритим вихідним кодом, написаний на C ++ (рис.1.7).

JavaScript-рушій – це програма, або, іншими словами, інтерпретатор, що виконує код, написаний на JavaScript. Рушій може бути реалізований з використанням різних підходів: у вигляді звичайного інтерпретатора, у вигляді динамічного компілятора (або JIT-компілятора), який перед виконанням програми перетворює вихідний код на JS в байт-код якогось формату.

Рушій JavaScript виконує скрипти JavaScript, переважно, в браузерах, а з появою Node.js також і у серверній реалізації.

V8 використовує метод - приховані класи. Приховані класи схожі на звичайні класи в типовій об'єктно-орієнтованій мові програмування, на зразок Java, крім того, що створюються вони під час виконання програми.

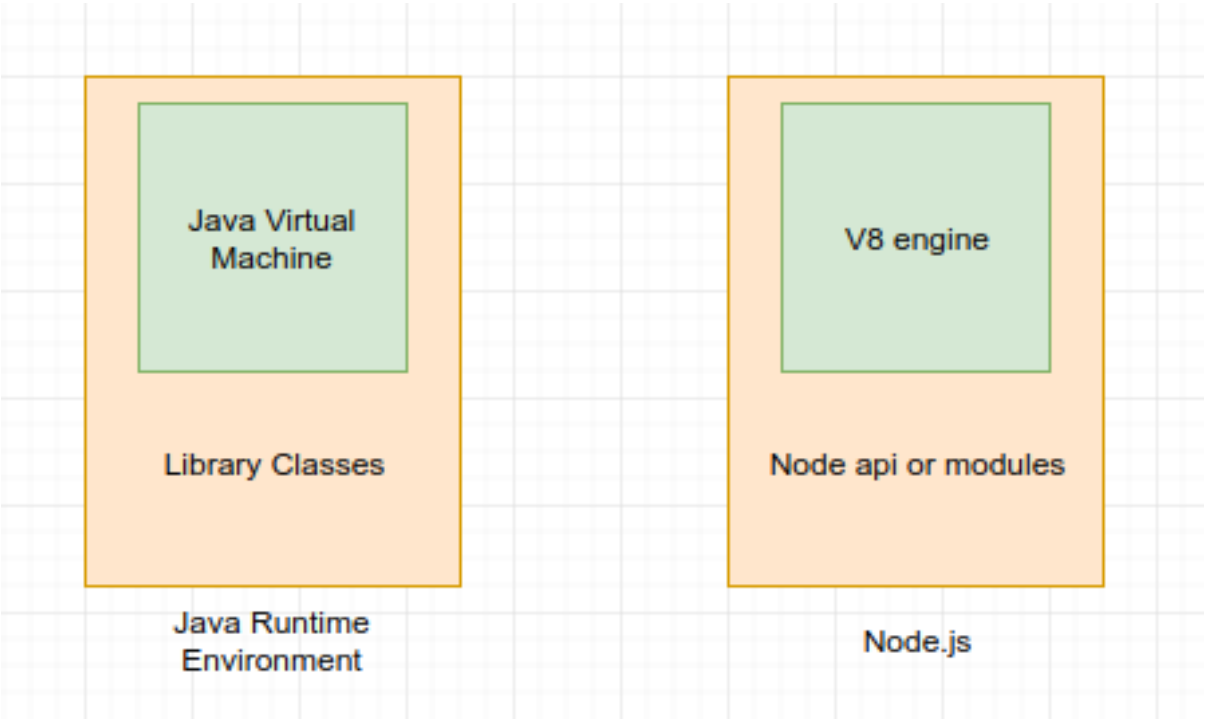

Рис. 1.7. Аналог JVM - рушій V8 [15]

Рушій V8 має такі **особливості** [21]:

- компіляція вихідного коду JavaScript безпосередньо у власний машинний код, минаючи стадію проміжного байт-коду;
- ефективна система керування пам'яттю, яка дозволяє швидко резервувати місця для об'єкту та зменшити очікування на прибирання сміття;
- V8 призупиняє виконання коду під час виконання прибирання сміття;
- V8 регулює використання пам’яті та точно визначає, де містяться в пам'яті об'єкти й вказівники/посилання, що дозволяє запобігати витоку інформації з пам'яті при помилковій ідентифікації об'єктів як посилань;
- використовує приховані класи й вбудований кеш, що значно прискорює доступ до властивостей та виклики функцій.

Node.js має наступні **властивості**:

- асинхронна одно-ланцюгова модель виконання запитів;
- неблокуюче введення/виведення;

Якщо додаток працює синхронно, то користувач не зможе взаємодіяти зі сторінкою, поки не прийде результат. Обробка може тривати досить довго.

Для забезпечення обробки великої кількості паралельних запитів у Node.js використовується **асинхронна модель запуску коду**, заснована на обробці подій в неблокуючому режимі та визначенні обробників зворотніх викликів (callback). В цьому випадку головний потік виконання поділяється на дві гілки. Одна з них продовжує займатися інтерфейсом, а інша виконує запит.

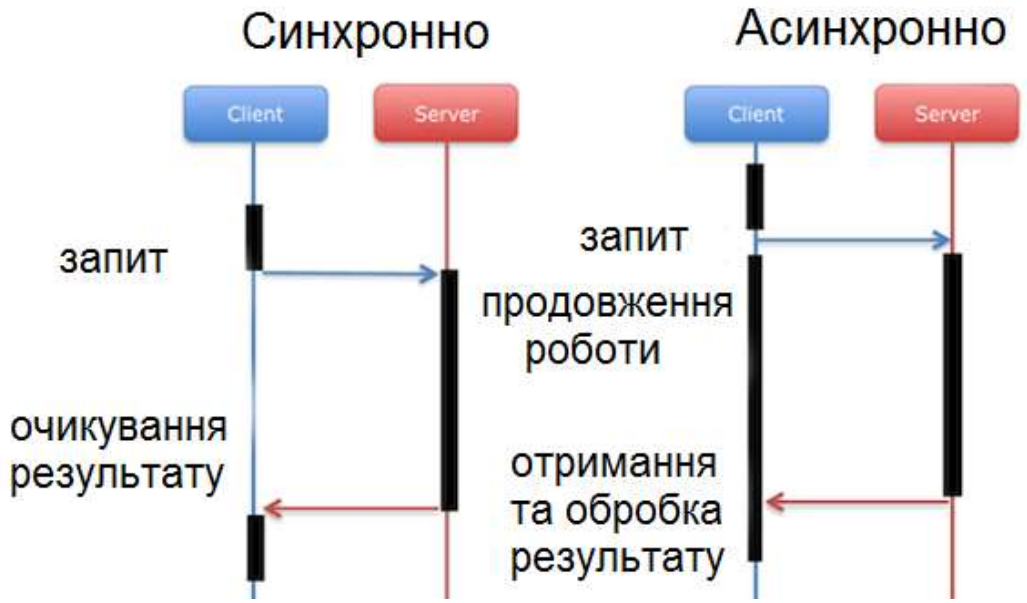

Рис. 1.8. Відмінність синхронної та асинхронної моделі виконання запитів [22]

Асинхронність

Ідея асинхронного виконання полягає в тому, що початок і кінець однієї операції відбуваються в різний час в різних частинах коду. Щоб отримати результат, необхідно почекати, причому час очікування непередбачувано.

Коли мова заходить щодо асинхронності, використовують ще три близькі поняття. Це конкурентність (concurrency), паралелізм (parallel execution) і багатопотоковість (multithreading) [23]. Всі вони пов'язані з одночасним виконанням завдань, проте це не одне і те ж (рис.1.9).

Рис. 1.9. Ілюстрація виконання послідовного, паралельного та конкурентного виконання

Паралелізм

Паралельне виконання використовується для поділу одного завдання на частини для прискорення обчислень. Наприклад, потрібно зробити кольорове зображення чорно-білим. Обробка верхньої половини не відрізняється від обробки нижньої. Отже, можна розділити цю задачу на дві частини і роздати їх різним потокам, щоб прискорити виконання в два рази.

Наявність двох фізичних потоків тут принципово важливо, так як на комп'ютері з одним обчислювальним пристроєм (процесорним ядром) такий прийом провести неможливо.

Багатопотоковість

Тут потік є абстракцією, під якою може ховатися і окреме ядро процесора, і тред ОС. Деякі мови навіть мають власні об'єкти потоків. Таким чином, ця концепція може мати принципово різну реалізацію.

Може здатися очевидним, що багатопотокове середовище зможе обробляти більше запитів за секунду, ніж однопотокове середовище. Це, як правило, справедливо для запитів із інтенсивними обчисленнями, які інтенсивно використовують виділені ресурси.

Однак у випадках, коли запити включають багато введення/виведення, наприклад виклики бази даних або веб-служб, кожен запит повинен чекати, поки зовнішній механізм відповість на зроблені виклики, і, отже, виділені ресурси CPU і пам’яті не використовуються під час цього очікування. Тому, у програмах обробки даних, де сервер повинен обробляти велику кількість одночасних запитів, кожен з яких включає час очікування введення/виведення, однопотокова модель асинхронного вводу-виводу NodeJS забезпечує значну перевагу над багатопоточною моделлю синхронного введення-виводу.

Конкурентність

Поняття конкурентного виконання саме загальне. Воно буквально означає, що безліч завдань вирішуються в один час. Можна сказати, що в програмі є кілька логічних потоків - по одному на кожну задачу.

При цьому потоки можуть фізично виконуватися одночасно, але це не обов'язково. Завдання при цьому не пов'язані один з одним. Отже, не має значення, яка з них завершиться раніше, а яка пізніше.

Node модулі

Node модулі це - багаторазово використовувані блоки коду (рівень КПВ), існування яких не впливає на інший код.

Ви можете написати свої власні модулі та їх використовувати в різних додатках. Node.js має набір вбудованих модулів, які можна використовувати без спеціальної установки.

Система модулів CommonJS - проект, метою якого є визначення екосистеми мови програмування JavaScript поза межами браузера (наприклад, серверний JavaScript або рідні додатки).

Обидва - браузерний JavaScript та Node.js запускаються в середовищі виконання Google рушій V8. Цей движок використовує ваш JS код і перетворює його в більш швидкий машинний код. Машинний – це низькорівневий код, який комп'ютер може запускати без необхідності спочатку його інтерпретувати.

Для керування модулями Node.js використовується пакетний менеджер **npm** (node package manager).

Як способи мультиплексування з'єднань підтримується epoll, kqueue, /dev/poll і select. Для мультиплексування з'єднань використовується бібліотека libuv, для створення пулу нитей (thread pool) задіяна бібліотека libeio, для виконання DNS-запитів у неблокуючому режимі інтегрований c-ares.

Всі системні виклики, що спричиняють блокування, виконуються всередині пулу ланцюгів і потім, як і обробники сигналів, передають результат своєї роботи назад через неіменовані канали (pipe).

JavaScript Object Notation (JSON) - це стандартний текстовий формат для представлення структурованих даних на основі синтаксису об’єкта JavaScript.

JSON - це текстовий формат даних, що відповідає синтаксису об’єктів JavaScript, який був популяризований Дугласом Крокфордом.

Незважаючи на те, що він дуже нагадує синтаксис об'єктів JavaScript у літералі, його можна використовувати незалежно від JavaScript, а багато середовищ програмування мають можливість читати (аналізувати) та генерувати JSON.

Властивості JSON:

1. JSON існує у вигляді рядка - корисний, коли ви хочете передавати дані через мережу. Його потрібно перетворити на власний об’єкт JavaScript, коли ви хочете отримати доступ до даних. Це не велика проблема - JavaScript надає глобальний об'єкт JSON, який має доступні методи для перетворення між ними.

2. Рядок JSON може зберігатися в окремому файлі, який в основному просто текстовий файл з розширенням **.json**.

3. JSON - це суто рядок із заданим форматом даних - він містить лише властивості, а не методи.

4. JSON вимагає використання подвійних лапок навколо рядків та назв властивостей. Поодинокі лапки не є дійсними, крім того, що оточують весь рядок JSON.

5. Навіть одна неправильна кома або двокрапка може призвести до того, що файл JSON пішов неправильно і не працював. Ви повинні бути обережними, щоб перевірити будь-які дані, які ви намагаєтесь використовувати (хоча створений комп'ютером JSON рідше включає помилки, якщо програма генератора працює належним чином). Ви можете перевірити JSON за допомогою такої програми, як JSONLint .

6. JSON насправді може приймати форму будь-якого типу даних, який допустимий для включення всередину JSON, а не лише масивів або об'єктів. Так, наприклад, один рядок або число буде дійсним JSON.

7. На відміну від коду JavaScript, в якому властивості об'єкта можуть не цитуватися, у JSON як властивості можуть використовуватися лише рядки, що цитуються.

Структура JSON

Формат JSON - це рядок, структура якого дуже нагадує літеральний формат об'єкта JavaScript. Ви можете включити ті самі основні типи даних всередині JSON, що і стандартний об’єкт JavaScript - рядки, числа, масиви, логічні значення та інші об’єкти. Це дозволяє побудувати ієрархію даних приблизно так [13]:

```
'{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
```

```
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}'
```

Якщо завантажити цей об’єкт до програми JavaScript, створивши змінну під назвою superHeroes, то можна потім отримувати доступ до даних всередині неї, використовуючи нотації “крапка” та “квадратна дужка” як для об’єкту JavaScript. Наприклад:

```
superHeroes.homeTown
superHeroes ['active']
```

Щоб отримати доступ до даних нижче за ієрархією, потрібно об’єднати необхідні імена властивостей та індекси масивів разом. Наприклад, щоб отримати доступ до третьої надможливості другого героя, перерахованого у списку, слід зробити таке:

```
superHeroes['members'][1]['powers'][2]
```

Розкриємо зміст такого запису коду JavaScript:

1. Спочатку задаємо ім'я змінної - superHeroes.

2. Всередині для того, щоб отримати доступ до властивості members - використовуємо ["members"].

3. members вміщує масив, заповнений об’єктами. Для того щоб отримати доступ до другого об’єкта всередині масиву, тому використовуємо данні з індексом 1.

4. Усередині цього об’єкта отримуємо доступ до властивості powers із застосуванням запису ["powers"].

5. Безпосередньо усередині властивості powers розміщується масив, що містить надможливості обраного супергероя. Для отримання третьої надможливості використовуємо данні з індексом 2.

NPM (менеджер пакетів Node)

Пакет (може також називатися модулем або бібліотекою) - є набором функцій або класів, які виконують визначене для них завдання.

NPM пакет - проект, який знаходиться у відкритому доступі для використання сторонніми програмістами у реєстрі NPM.

npm - найбільший у світі реєстр програмного забезпечення.

Розробники з відкритим кодом з усіх континентів використовують npm для спільного використання та запозичення пакетів, а багато організацій використовують npm для управління власними розробками.

Це бібліотеки, побудовані співтовариством. Вони вирішують більшість проблем, що часто зустрічаються.

npm містить пакети, які ви можете використовувати в своїх додатках, щоб зробити вашу розробку більш швидкою та ефективною.

npm складається з трьох різних компонентів:

Перш за все, це Інтернет-сховище для публікації проектів з відкритим кодом Node.js.

По-друге, це інструмент CLI, який допомагає встановлювати ці пакети та керувати їх версіями та залежностями. Існує сотні тисяч бібліотек і програм Node.js на npm, і багато інших додаються щодня.

По-третє – реєстр є великою громадської базою даних програмного забезпечення JavaScript і мета-інформації навколо нього.

npm сам по собі не запускає жодних пакунків-модулів.

Якщо ви хочете запустити пакет за допомогою npm, ви повинні вказати цей пакет у своєму package.json файлі.

Коли виконувані файли встановлюються через пакети npm, npm створює посилання на них:

- **локальні установки** мають посилання, створені в ./node_modules/.bin/каталозі;

- **глобальні встановлення** мають посилання, створені з глобального bin/каталогу (наприклад: /usr/local/biny Linux або в /AppData/npmWindows).

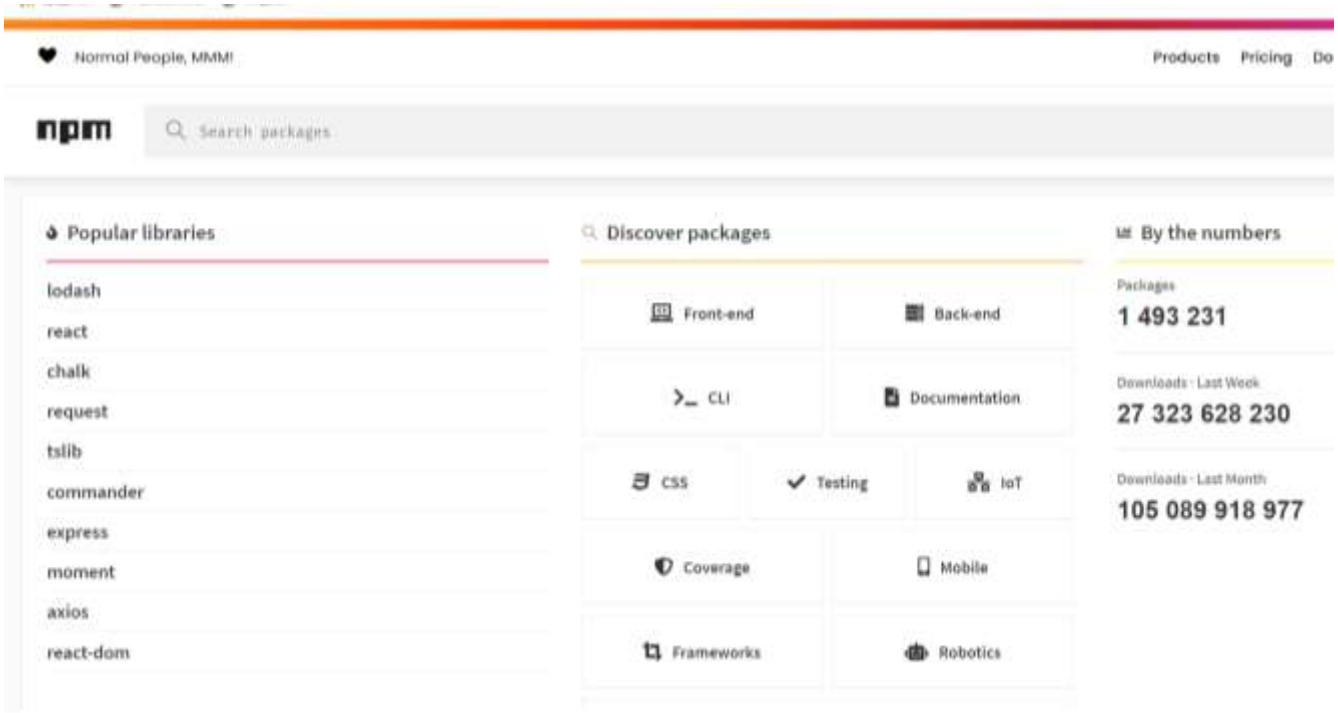

Рис. 1.10. Вигляд ресурсу[20]

Принцип роботи Node.js

Розглянемо принцип роботи **Node.js** на прикладі програми, що запускає web-сервер, виводить в консоль повідомлення, та на кожен HTTP запит відповідає повідомленням «Hello World» [15]:

```
var http = require('http');      // Завантажуємо модуль http
// Створюємо web-сервер і вказуємо функцію обробки запиту
var server = http.createServer(function (req, res) {
    console.log ('Початок обробки запиту');
// Передаємо код відповіді і заголовки
    res.writeHead (200, {
        'Content-Type': 'text/plain; charset=UTF-8'
    });
    res.end ('Hello world!');
});
// Запускаємо web-сервер
server.listen (1991, "127.0.0.1", function () {
    console.log ('Сервер запущено за адресою http://127.0.0.1:1991/');
});
```

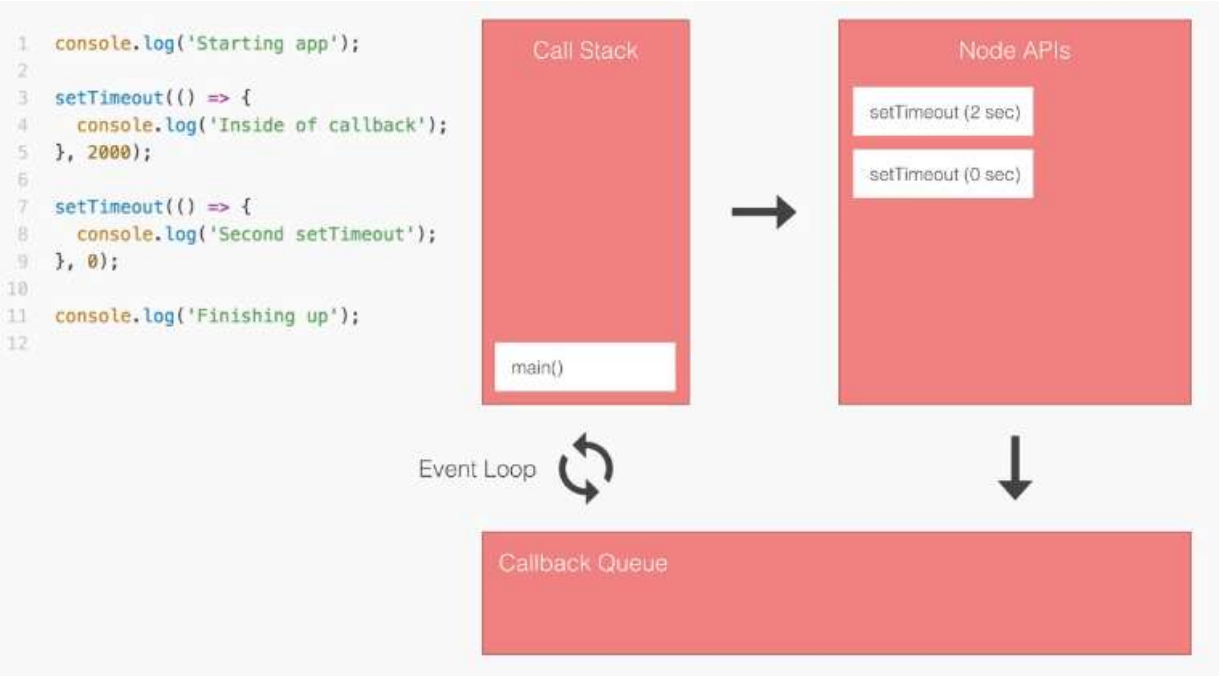

Рис. 1.11. Ілюстрація того, як працює цикл подій JavaScript [15]

Код виконує такі дії [15]:

1. Посилаєте main() в стек викликів.
2. Посилаєте console.log() в стек викликів. Він запускається відразу і з'являється.
3. Посилаєте setTimeout(2000) до стеку. setTimeout(2000) - це Node API. Коли ми викликаємо його, ми реєструємо пару подія-коллбек. Подія буде чекати 2000 мілісекунд, а потім викличе коллбек.
4. Після реєстрації, setTimeout(2000) з'являється в стеку викликів.
5. Тепер другий setTimeout(0) реєструється таким же чином. Тепер у нас є два API-інтерфейси Node, які чекають виконання.
6. Після очікування 0 секунд setTimeout(0) переміщується в чергу виконання коллбеків (callback queue), і те ж саме відбувається з setTimeout(2000).
7. У черзі виконання коллбеків функції чекають, коли стек викликів буде порожнім, тому що тільки одна функція може виконуватись одночасно. Це забезпечує event loop.
8. Остання викликається console.log(), а main() викликається з стека викликів.
9. Цикл подій бачить, що стек викликів порожній, а черги зворотного виклику - немає. Таким чином, він переміщує зворотні запити (по порядку) у стек викликів для виконання.

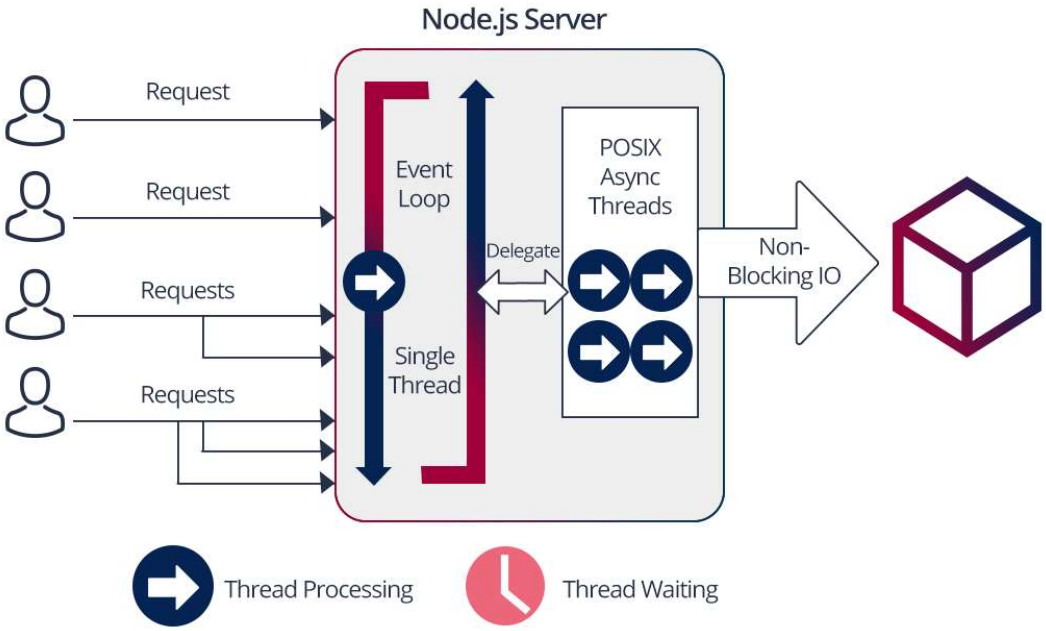

Рис. 1.12. Приклад використання node.js для виконання запиту POST request функції Google (запит до певного користувача, підключеного до кількох пристроїв) [16]

Require виконує три функції [16]:
1. Завантажує модулі, що поставляються в комплексі з Node.js наприклад файлової системи або HTTP з API Node.js.
2. Завантажує сторонні бібліотеки, такі як Express і Mongoose, які ви встановлюєте з npm.
3. Дозволяє створювати власні файли і ділити проект на модулі.
Require - це функція, і вона приймає параметр «шлях» і повертає module.exports.

Сервер на основі Node.js. Фреймворк Express.js

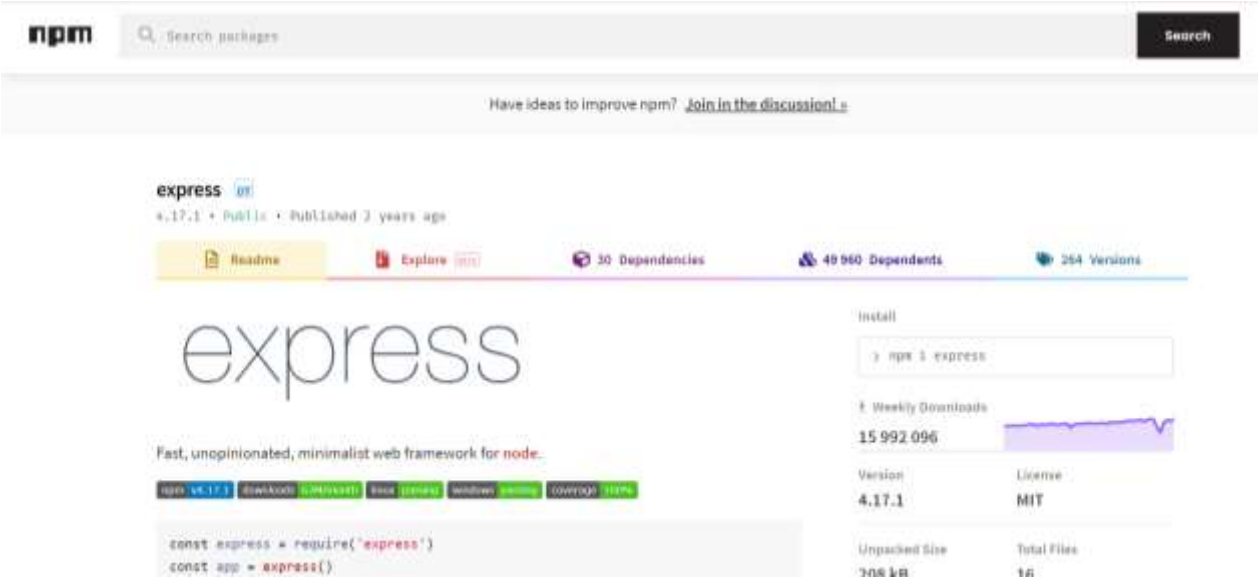

Рис. 1.13. Знаходження Express.js на [20]

Комбінація розробки Node.js та Express.js. Перша платформа займається побудовою додатків на сервері, а другий фреймворк публікує їх як web-сайти.

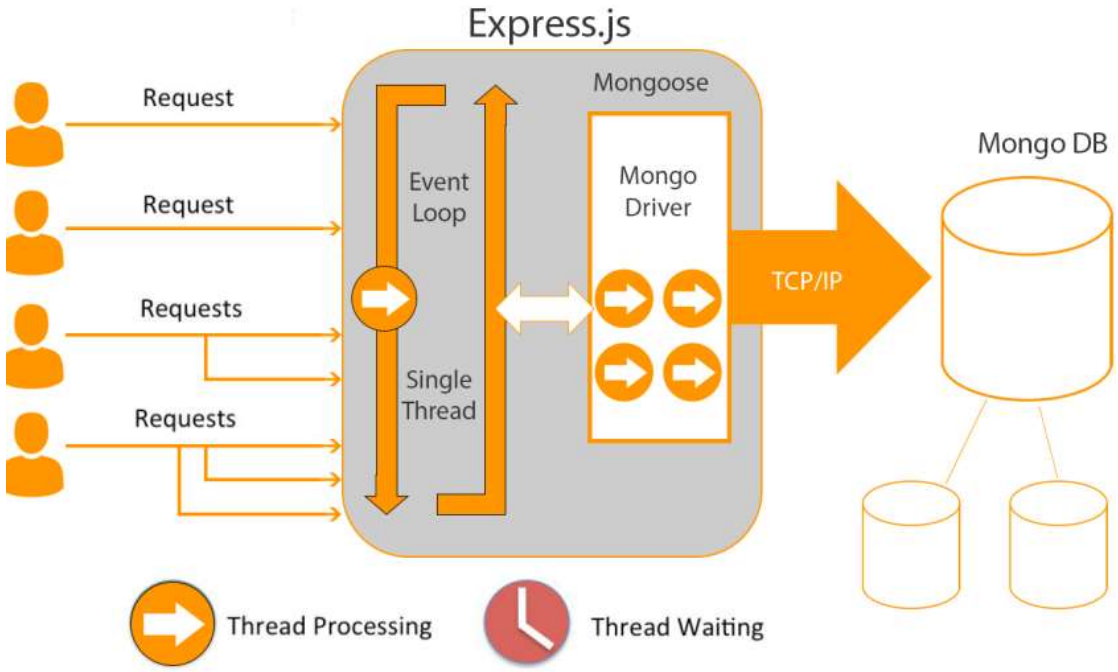

Рис. 1.14. Використання Express як сервера web-додатку [17]

Express - це мінімальна та гнучка рамка web-додатків Node.js, яка забезпечує надійний набір функцій для розробки web-та мобільних додатків. Це сприяє швидкому розвитку web-додатків на основі вузла.

Нижче наведено деякі основні риси Express Framework:

1. Дозволяє налаштовувати середні програми для відповіді на запити HTTP.

2. Визначає таблицю маршрутизації, яка використовується для виконання різних дій на основі методу та URL-адреси HTTP.

3. Дозволяє динамічно відтворювати HTML-сторінки на основі передачі аргументів у шаблони.

ExpressJS має наступні переваги:

- Якщо ви використовуєте Node.js для виробництва додатків, цей процес можна пришвидшити та вдосконалити за допомогою Express.
- На основі методів URL та HTTP можна визначити маршрути існуючої програми.
- Розробники Express.js можуть використовувати універсальні модулі програмного забезпечення для виконання додаткових завдань.

- Доступна інтеграція з різними шаблонів- препроцесорів Jade, Vash та інші.
- Ресурси та статичні файли програми легко обслуговувати.
- Створення сервера API REST - одна з найпривабливіших особливостей розробки додатків на Express.js.
- Ви можете скористатися з'єднанням з такими базами даних, як MySQL, Redis та MongoDB.

Продуктивність Express.js

Кількість запитів, що обробляються в секунду, є найбільш підходящим показником для визначення продуктивності в рамках. Оскільки час затримки в мілісекунді менш важливий, ніж абсолютна продуктивність, показник справедливий для багатьох web-додатків.

Ви можете знайти численні тести, зроблені на універсальних системах та за певних умов. Кількість запитів, зроблених протягом секунди, змінюється залежно від типу перевірених операцій. Проте вони показують базові показники.

Express.js дозволяє оптимізувати ефективність за допомогою певних прийомів як у коді, так і в середовищі.

Інструменти та засоби розробки

Додатково до інструментів та засобів, що запропоновані в попередніх підрозділах, для продовження розробки навчального проекту на базі стеку MEAN/MERN використовуємо такі інструменти та засоби, що мають безкоштовний доступ:

- MongoDB - документоорієнтована система управління базами даних, яка не потребує опису схем таблиць [19]. Є NoSQL-системою, використовує JSON-подібні документи та схему бази даних. Широко застосовується у веб-розробці для роботи з JavaScript-орієнтованим стеком MEAN/MERN для створення web-додатків;

- npm - менеджер пакетів, який входить до складу Node.js, що використовується як репозиторій для публікації проектів з відкритим вихідним кодом, та як інструмент командного рядка, який допомагає взаємодіяти з браузерами та серверами, допомагає в установці та видаленні пакетів, керуванні версіями та залежностями, необхідними для роботи проекту [20].

- nodemon - модуль для перезавантаження сервера після зміни коду в проекті. Це інструмент для налагодження програм на основі node.js, який може виявляти зміни у файлах проекту та автоматично перезапускає процес.

1.2.2. Практичний модуль

У практичній частині продовжуємо проектну розробку web-додатку Tours City (екскурсії містом) у якому можна рекламувати подорожі будь-яким містом у галузі культурного міського туризму.

Етап 1.3. Створення проекту web-додатку та налагодження зв'язку з ресурсними модулями Node.js. Тестовий запуск

Дія 1.3.1. Встановлення Node.js

- для Windows завантажуємо з сайту https://nodejs.org/download/release/latest/ останню версію з розширенням .msi або можете оновити за допомогою команди у командному рядку:

npm install npm -g

- після завершення установки Node.js і NPM будуть оновлені до останньої версії, а потім ви зможете очистити пакет виконавши команди:

npm cache clean -f

npm update -g

Дія 1.3.2. Встановлення Express.js.

Необхідно встановити глобально генератор додатків Express.

npm install -g express-generator

Дія 1.3.3. Створення папки проекту та перехід по неї

- на диску операційної системи Windows створюємо папку з ім'ям tours_city:

mkdir tours_city

- переходимо до цієї папки:

cd tours_city

Дія 1.3.4. Ініціалізація проекту.

- ініціалізуємо проект шляхом створення файлу конфігурації web-додатку package.json:

npm init

- далі вводимо в консолі:

package name: (tours_city) // погоджуємось, натискаємо Enter
version: (1.0.0) // пропускаємо, натискаємо Enter
description: *Mern stack* // може бути будь що
entry point (index.js) : *app.js* // буде роутером і основним файлом проекту
// за замовчуванням пропонується index.js замінюємо на app.js
test command: // пропускаємо
git repositoty: // пропускаємо
keywords: *mern, react* // може бути будь що
author: *Yurii Tulashvili < y.tulashvili@lutsk-ntu.com.ua>*
license: // пропускаємо
->OK

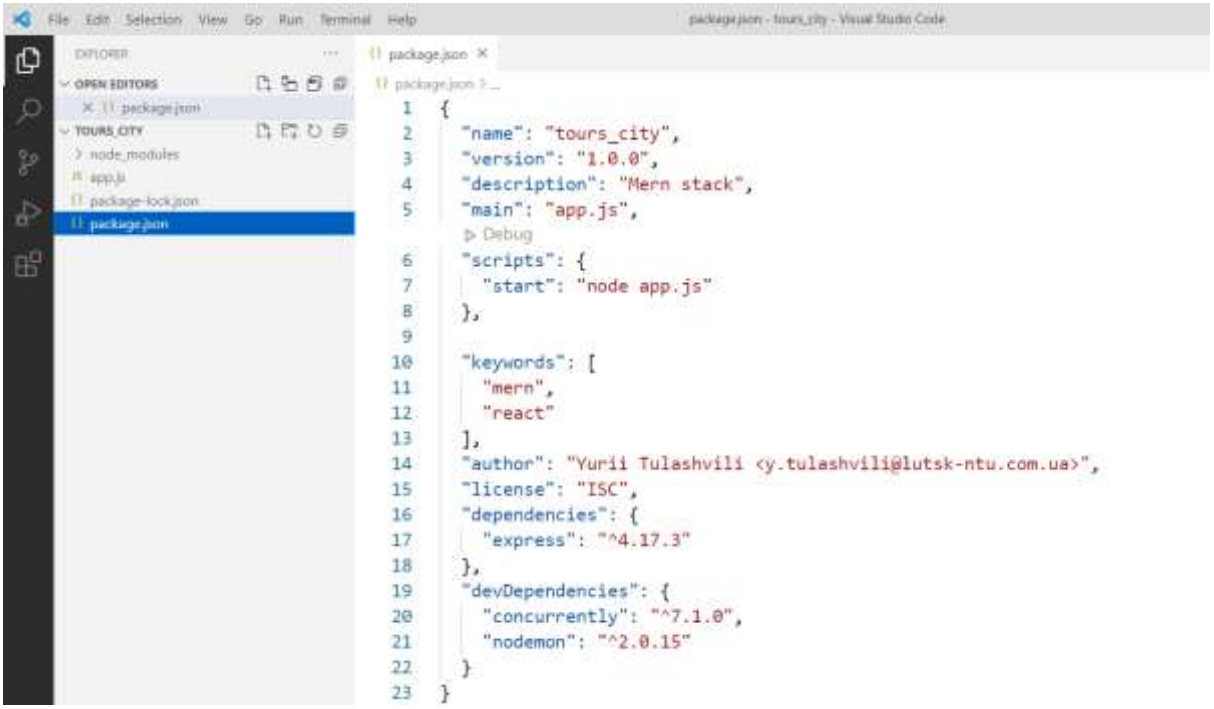

Рис. 1.15. Відкритий файл package.json у Visual Studio Code

Дія 1.3.5. Підключення конфігурації ресурсів проекту

- додаємо до проекту Express.js:

npm install express

- встановлюємо утиліти для роботи з проектом, що будуть потрібні під час розробки:

npm install -D nodemon concurrently

де -

nodemon – пакет, який дозволяє перезавантажувати сервер;

concurrently - буде потрібен пізніше, щоб запускати одночасно back-end та front-end;

npm install -D - щоб записалися залежності package.json у розділ devDependencies { }

Дія 1.3.6. Доопрацьовуємо файл package.json

- допрацьовуємо файл package.json для запуску з консолі. Переписуємо розділ "scripts":

```
"scripts": {
    "start": "node app.js"
  },
```

Файл package.json набува вигляду - рис. 1.15.

Дія 1.3.7. Програмуємо перший тестовий запуск проекту

- створюємо в корні проекту за допомогою Visual Studio Code основний файл для back-end під ім'ям, як було вказано в package.json - app.js. Пишемо в середині файлу код:

```
const express = require('express')
const app = express()            // створюємо сервер під ім'ям app

app.get ('/', (req, res) => {     // відкриття web-сторінки
                res.send ('This is my project');
            });
app.listen (5000)  // запускаємо прослуховувач проекту
```

Дія 1.3.8. Виконуємо тестовий запуск проекту (сервера)

- у консолі вводимо:
  npm start
- у браузері вводимо:
  localhost:5000
- зупиняємо проект:
  Ctrl + C.

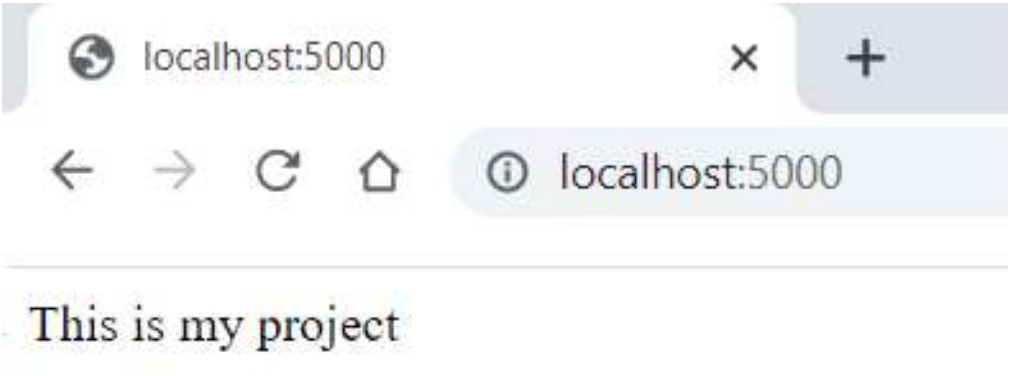

Рис. 1.16. Тестовий запуск проекту на http://localhost:5000/

1.3. СТВОРЕННЯ БАЗИ ДАНИХ WEB-ДОДАТКУ. MONGODB

1.3.1. Теоретичний модуль

Документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом - MongoDB

Зазвичай дані для web-додатку Express зберігають у який-небудь базі даних. Сьогодні є технології баз даних на самі різні випадки - традиційні сховища на основі SQL, документо-орієнтовані бази даних без використання SQL, прості сховища ключів і значень або web-служби запитів типу YQL.

Найпоширеною базою даних для платформи Node є документо-орієнтована база даних MongoDB. Основні можливості MongoDB:

MongoDB - відкрите програмне забезпечення, яке є розширюваною, високопродуктивною, вільною від схем, документо-орієнтованою базою даних, яка написана на C++. Розробляється з жовтня 2007 року компанією 10gen. Зберігає всі ваші дані у форматі бінарного JSON (BSON). Вже має широке застосування у реальних проектах.

Основні можливості MongoDB:

- Документо-орієнтоване сховище (проста та потужна JSON-подібна схема даних)
- Повна підтримка індексів
- Підтримка відмовостійкості і масштабованості
- Гнучка мова для формування запитів
- Динамічні запити
- Профілювання запитів
- Швидкі оновлення
- Журналювання операцій, що модифікують дані в БД
- Підтримка MapReduce

Документоорієнтована СУБД MongoDB призначена для зберігання даних в документах колекцій в форматі JSON (як звичайний текст), які не мають чіткої структури та призначені для зберігання будь-якого типу даних. Такий формат і організація зберігання даних в СУБД MongoDB забезпечує оперативну обробку даних, їх запис і виведення результатів. СУБД MongoDB призначена для роботи з нереляціоннимі даними, наприклад, для роботи з базою даних Website, яка може включати в себе наступні колекції (сутності): статті, новини, фото, архів тощо.

У MongoDB застосовується безсхемна організація неструктурованих або слабоструктурованих даних, тобто організація, яка не потребує опису схеми

бази даних. Цей спосіб організації даних не призначений для того, щоб пов'язувати документи з даними однієї колекції з документами, що містять дані, інший колекції через ключові поля. У MongoDB немає власних ресурсів створення відносин між колекціями і документами.

Документоорієнтована СУБД MongoDB призначена для зберігання даних в документах колекцій в форматі JSON (як звичайний текст), які не мають чіткої структури та призначені для зберігання будь-якого типу даних. Такий формат і організація зберігання даних в СУБД MongoDB забезпечує оперативну обробку даних, їх запис і виведення результатів. СУБД MongoDB призначена для роботи з нереляціоннимі даними, наприклад, для роботи з базою даних Website, яка може включати в себе наступні колекції (сутності): статті, новини, фото, архів тощо.

СУБД MongoDB пропонує документоорієнтовану модель даних. Якщо в традиційних реляційних БД основними компонентами є двовимірні таблиці (кожної сутності або базового типу інформації призначається таблиця) з записами (рядками) і полями (стовпцями), то в MongoDB - тематичні колекції (сутності), які складаються з документів, що включають в себе набір текстових полів в форматі JSON, тобто пар: ключ-значення ({ "key": "value"}).

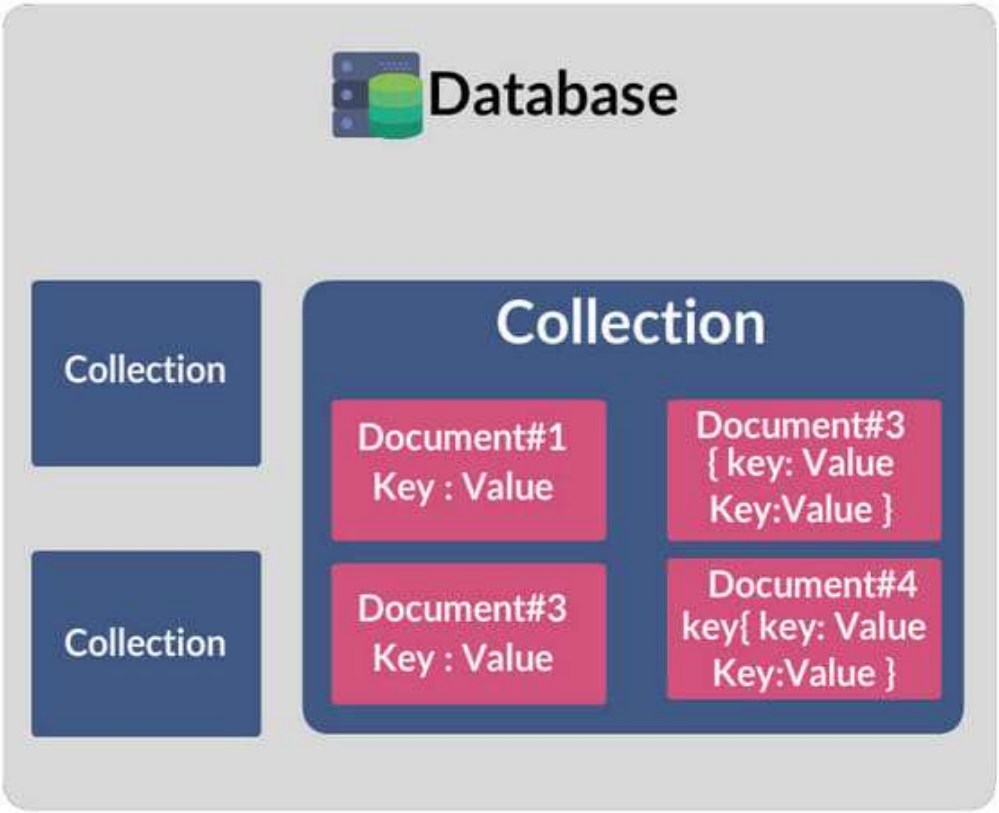

Рис. 1.17. Структура бази даних MongoDB [26]

Основною одиницею зберігання, доступу та обробки в базі MongoDB є документ. Документ має набір властивостей - атрибутів. Кожен атрибут в документі задається парою

<ім'я (ключ) атрибута>: <значення атрибута>.

Звідси випливає, що MongoDB добре працює з наборами даних (колекціями і документами), які не пов'язані між собою, і її можна використовувати для створення Web додатків, що забезпечують високу продуктивність і необмежену горизонтальне масштабування колекцій даних [27].

Кожна **колекція** має своє унікальне ім'я - довільний ідентифікатор, що складається з не більше ніж 128 різних алфавітно-цифрових символів і знака підкреслення.

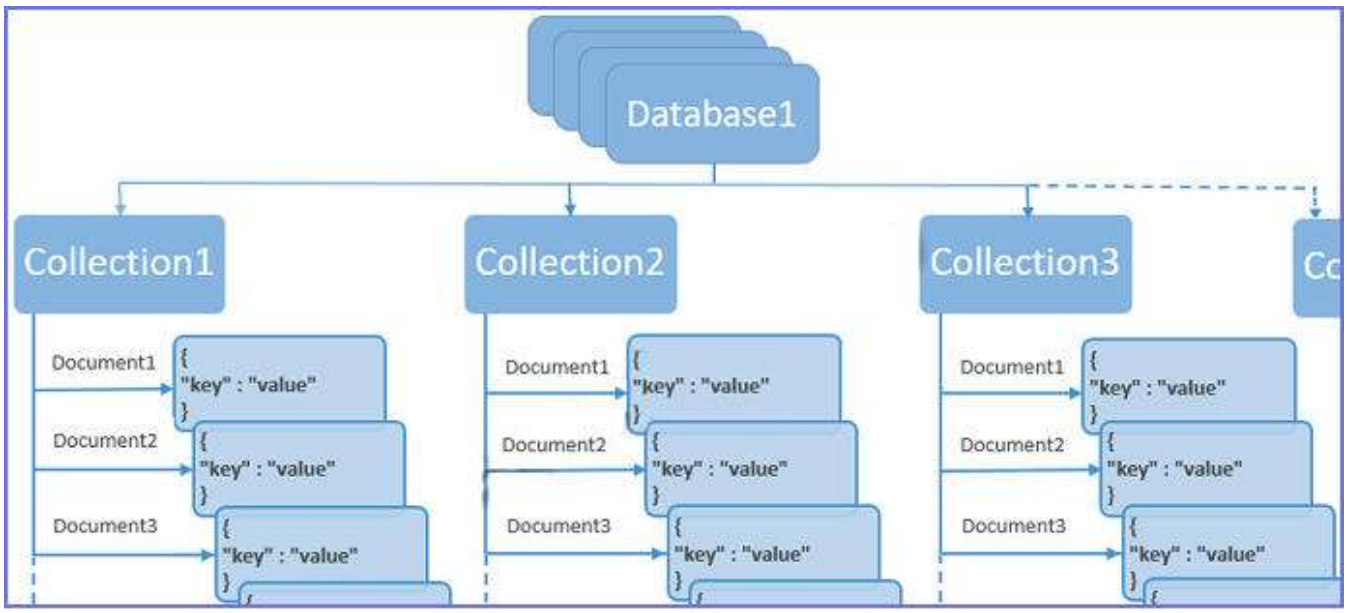

Рис. 1.18. Структура бази даних MongoDB [28]

На відміну від реляційних баз даних MongoDB не використовує табличний пристрій з чітко заданим кількістю стовпців і типів даних. MongoDB є документо-орієнтованої системою, в якій центральним поняттям є документ .

При використанні стандартної реляційної БД структура таблиць була б, приблизно, такою:

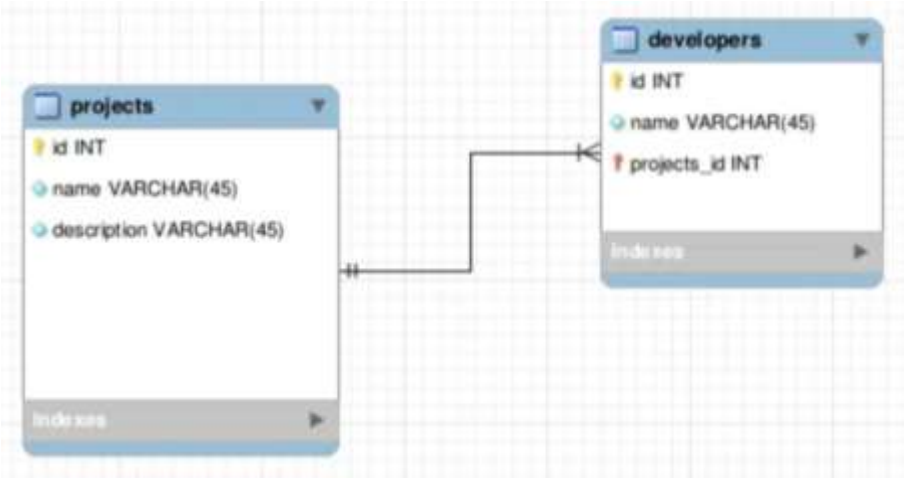

Рис. 1.19. Структура реляційної бази даних

У MongoDB у нас буде одна колекція проектів з наступною структурою:

```
{
  _id: PROJECT_ID
  name: NAME_OF_PROJECT,
  developers: [
    {
      name:'DEVELOPER_NAME',
    },
    {
      name: 'DEVELOPER_NAME'
    }
                ]
}
```

Для зв'язку MongoDB із додатком застосовується **mongoose** - інтерфейс між Node. Mongoose - одна з лідируючих «nosql» СУБД (nosql означає, що вона не базується на мові SQL). Mongoose - це бібліотека JavaScript, часто використовувана в додатках Node.js з базою даних MongoDB. В описі говориться, що це «масштабована, високопродуктивна, документо-орієнтована СУБД з відкритим вихідним кодом». Вона дозволяє зберігати документи в форматі, близькому до JSON, без чітко визначеної схеми, і має цілу низку передових можливостей.

За докладнішою інформацією та документацію можна знайти на сайті проекту http://www.mongodb.org/.- один з декількох модулів для доступу до MongoDB, що представляє собою засіб об'єктного моделювання, тобто ваша програма визначає об'єкти Schema, що описують дані, а mongoose бере на себе турботу про їх збереженні в MongoDB. Це надзвичайно потужний інструмент,

що володіє такими засобами, як вбудовані документи, гнучка система типізації полів, контроль введення полів, віртуальні поля тощо.

На відміну від реляційних баз даних MongoDB не використовує табличний пристрій з чітко заданим кількістю стовпців і типів даних. MongoDB є документо-орієнтованої системою, в якій центральним поняттям є документ .

Документ можна уявити як об'єкт, який зберігає деяку інформацію. У певному сенсі він подібний до рядкам в реляційних СУБД, де рядки зберігають інформацію про окремий елемент. Наприклад, типовий документ:

```
{
   "name": "Bill",
   "surname": "Gates",
   "age": "48",
   "company": {
      "name" : "microsoft",
      "year" : "1974",
      "price" : "300000" }
}
```

Документ являє набір пар **ключ-значення**. Наприклад, в вираженні

"name": "Bill"

name представляє ключ, а Bill - значення.

Mongoose - це ODM (Object Document Mapper - объектно-документний образувач). Це означає, що Mongoose дозволяє вам визначати об’єкти за суворою схемою, відповідним документом MongoDB.

Ключі подаються як рядковий тип даних. Значення ж можуть відрізнятися за типом даних. В даному випадку у нас майже всі значення також представляють строковий тип, і лише один ключ (company) посилається на окремий об'єкт.

Mongoose надає величезний набір функціональних можливостей для створення та роботи зі схемами. На даний момент Mongoose містить список SchemaTypes (типи даних схем), які можуть мати властивість, що зберігається в MongoDB.

Типи значень:

String : рядковий тип даних, як в наведеному вище прикладі (для рядків використовується кодування UTF-8);

Binary data (двійкові дані) : тип для зберігання даних в бінарному форматі

Boolean : булевий тип даних, який зберігає логічні значення TRUE або FALSE, наприклад,{"married": FALSE};

Date : зберігає дату в форматі часу Unix ISODate, наприклад, ("1831-02-16");

Double : числовий тип даних для зберігання чисел з плаваючою точкою;

Integer : використовується для зберігання цілочисельних значень, наприклад,{"age": 29};

JavaScript : тип даних для зберігання коду JavaScript;

Min key / Max key : використовуються для порівняння значень з найменшим / найбільшим елементів BSON;

Null : тип даних для зберігання значення Null;

Object : строковий тип даних, як в наведеному вище прикладі;

ObjectID : тип даних для зберігання id документа, наприклад,

_id: ObjectId ("559a8c8d250c6e372a67abbd");

Regular expression : застосовується для зберігання регулярних виразів;

Symbol : тип даних, ідентичний строковому. Використовується переважно для тих мов, в яких є спеціальні символи;

Timestamp : застосовується для зберігання часу;

Приклад документа, що містить відомості про автора Джеймса Олдріджа і заданих в масиві Books двох його книгах:

```
{
“au_id”: "000–11–0001",
 “au_lname”: "Олдридж, Джеймс",
“years”: [1918, 2015],
“contract”: false,
“books”: [
{“title_id”: "bb1111",
 “title”: "Морській орел"},
 {“title_id”: "bb2222",
“title”: "Останній дюйм"}
            ]
}
```

На відміну від рядків документи можуть містити різнорідну інформацію. Так, поруч з документом, описаним вище, в одній колекції може знаходитися інший об'єкт, наприклад:

```
{
   "name": "Tom",
   "birthday": "1985.06.28",
   "place" : "Berlin",
   "languages" :[
      "english",
      "german",
      "spanish"
   ]  }
```

Здавалося б різні об'єкти за винятком окремих властивостей, але всі вони можуть перебувати в одній колекції.

Ще пара важливих зауважень: в MongoDB запити мають чутливі до регістру і строгою типізацією. Тобто такі два документу не будуть ідентичні:

{"age" : "28"}

{"age" : 28}

Якщо в першому випадку для ключа age визначена в якості значення рядок, то в другому випадку значенням є число.

Ідентифікатор документа

У процесі створення кожного документа (ObjectID) призначається "_id" унікальний ідентифікатор документа, який призначений для автоматичного сортування документів.

При додаванні документа в колекцію даний ідентифікатор створюється автоматично. Однак розробник може сам явно задати ідентифікатор, а не покладатися на автоматично генеруються, вказавши відповідний ключ і його значення в документі.

Дане поле повинно мати унікальне значення в рамках колекції. І якщо ми спробуємо додати в колекцію два документа з однаковим ідентифікатором, то додасться лише один з них, а при додаванні другого ми отримаємо помилку.

Якщо ідентифікатор не заданий явно, то MongoDB створює спеціальне бінарне значення розміром 12 байт. Це значення складається з декількох сегментів: значення типу timestamp розміром 4 байта, ідентифікатор машини з 3 байт, ідентифікатор процесу з 2 байт і лічильник з 3 байт. Таким чином, перші 9 байт гарантують унікальність серед інших машин, на яких можуть бути репліки бази даних. А наступні 3 байта гарантують унікальність протягом однієї секунди для одного процесу. Така модель побудови ідентифікатора гарантує з високою часткою ймовірності, що він буде мати унікальне значення.

Структура таблиць в MongoDB

Якщо створення relational DB починається зі створення структури таблиць і установки зв'язків між таблицями, а потім заповнення таблиць і формування SQL запитів або інструкцій SQL, то в MongoDB спочатку створюють нову базу даних і задають її ім'я, потім при додаванні в неї даних вона автоматично створює колекцію і об'єкт (документ) з описом, що додається об'єкта в форматі JSON. Для зберігання даних в MongoDB застосовується формат, який називається BSON, скорочення від binary JSON.

В СУБД MongoDB для створення, читання, оновлення, видалення об'єктів застосовуються операції CRUD (create, read, update, delete).

Нижче наведено таблицю відповідності між основним термінами MySQL та MongoDB [29]:

Таблиця 1.2. Взаємозв'язок термінології СУБД із MongoDB

MySQL term	Mongo term/concept
database	database
table	collection
index	index
row	BSON document
column	BSON field
join	embedding and linking
primary key	_id field
group by	aggregation

Якщо у реляційних БД таблиця має вигляд:

Last Name	First Name	Date of Birth
DUMONT	Jean	01-22-1963
PELLERIN	Franck	09-19-1983
GANNON	Dustin	11-12-1982

тоді колекція документів у MongoDB виглядає абсолютно інакше:

```
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc9"),
   "Last Name": "DUMONT",
   "First Name": "Jean",
   "Date of Birth": "01-22-1963"
},
```

```
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
   "Last Name": "PELLERIN",
   "First Name": "Franck",
   "Date of Birth": "09-19-1983",
   "Address": "1 chemin des Loges",
   "City": "VERSAILLES"
}
```

Кожне поле документа може містити складну підструктуру довільної глибини вкладеності:

```
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
   "Last Name": "PELLERIN",
   "First Name": "Franck",
   "Date of Birth": "09-19-1983",
   "Address": {
      "Street": "1 chemin des Loges",
      "City": "VERSAILLES"
   }
}
```

За рахунок такої структури запити є простішим та логічнішими. Виконання операцій на великих об'ємах даних займає не більше (а, зазвичай, значно менше) часу.

Інструменти та засоби розробки

Додатково до інструментів та засобів, що запропоновані в попередніх підрозділах, для продовження розробки навчального проекту на базі стеку MEAN/MERN використовуємо такі інструменти та засоби, що мають безкоштовний доступ:

- MongoDB - документоорієнтована система управління базами даних, яка не потребує опису схем таблиць [19]. Є NoSQL-системою, використовує JSON-подібні документи та схему бази даних. Широко застосовується у веб-розробці для роботи з JavaScript-орієнтованим стеком MEAN/MERN для створення web-додатків;

- mongoose - це бібліотека JavaScript, часто використовувана в додатках Node.js з базою даних MongoDB [30];

- config – модуль Node.js, що організовує ієрархічні конфігурації для розгортання програмного додатку. Дає змогу задавати набір параметрів за замовчуванням та розширити їх для різних середовищ розгортання. Конфігурації зберігаються у файлах конфігурації [31].

1.3.2. Практичний модуль

У практичній частині продовжуємо проектну розробку web-додатку Tours City (екскурсії містом) у якому можна рекламувати подорожі будь-яким містом у галузі культурного міського туризму.

Етап 1.4. Підключення до проекту web-додатку MongoDB

Дія 1.4.1. Встановлюємо пакет **config**

- організовуємо ієрархічну конфігурацію для зручності роботи з портами та з константами конфігурації програми, встановтвши пакет config. Він дає змогу визначати набір параметрів за замовчуванням та розширити їх для різних середовищ у середині проекту. У консолі вводимо:

npm i config

Дія 1.4.2. Встановлюємо пакет **mongoose**

- для підключення до бази даних MongoDB за допомогою пакету mongoose. У консолі вводимо:

npm i mongoose

Дія 1.4.3. Створення БД на mongodb.com

- для створення бази даних MongoDB виконуємо декілька послідовних дій на сайті ресурсу mongodb.com (рис. 1.20 – рис. 1.33) після реєстрування на ньому.

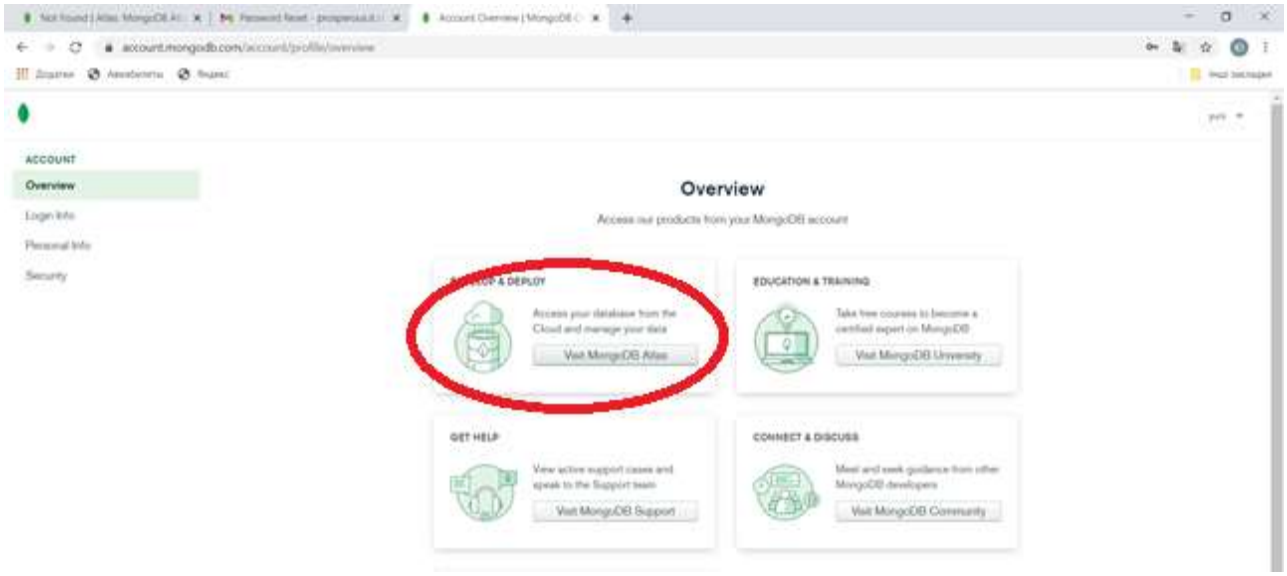

Рис. 1.20. Визначаємо прозначення бази даних

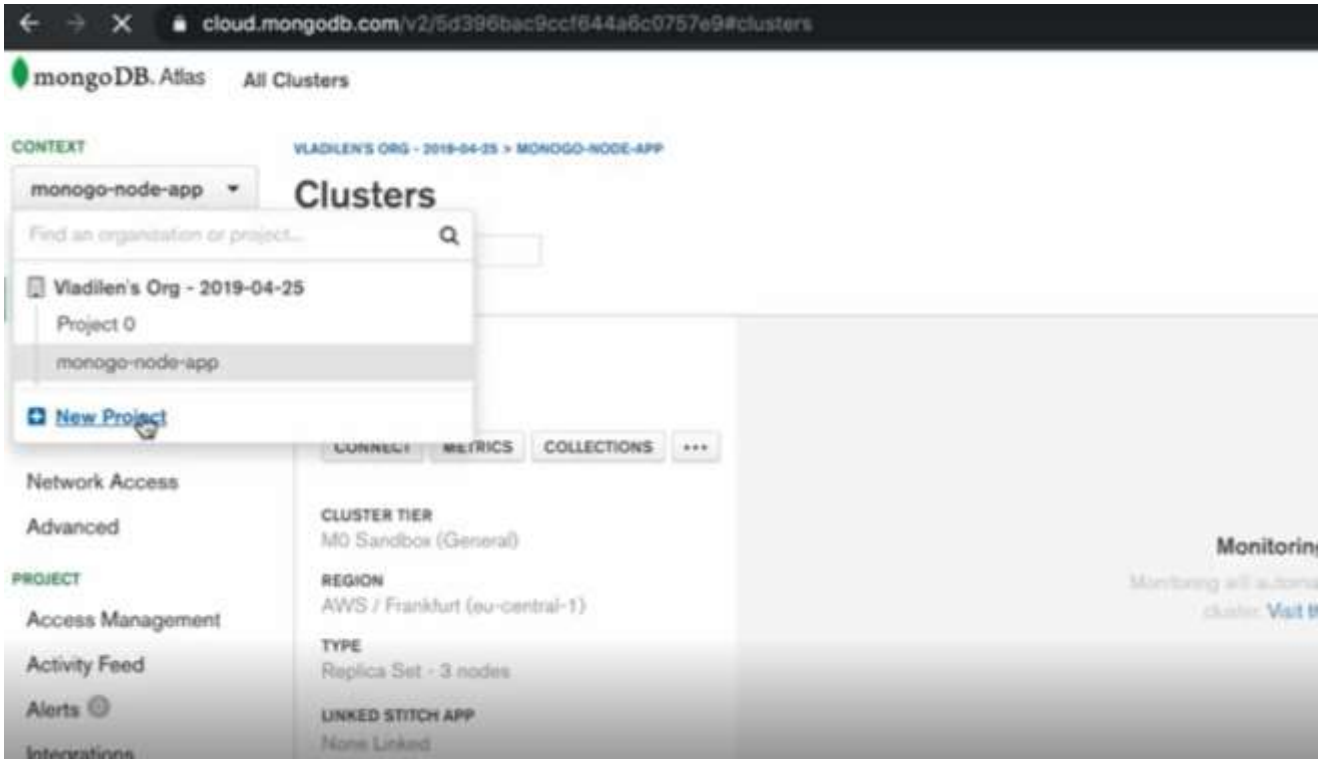

Рис. 1.21. Ініціюємо створення нового проекту

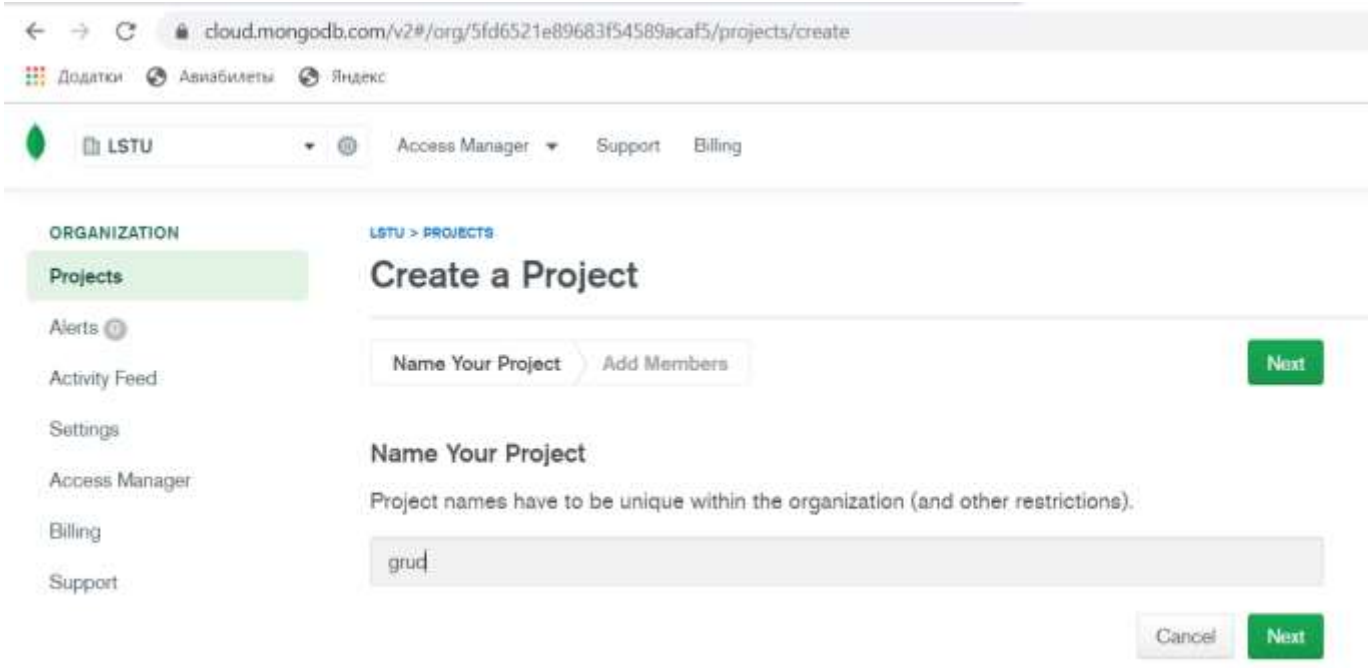

Рис. 1.22. Називаємо проект бази даних

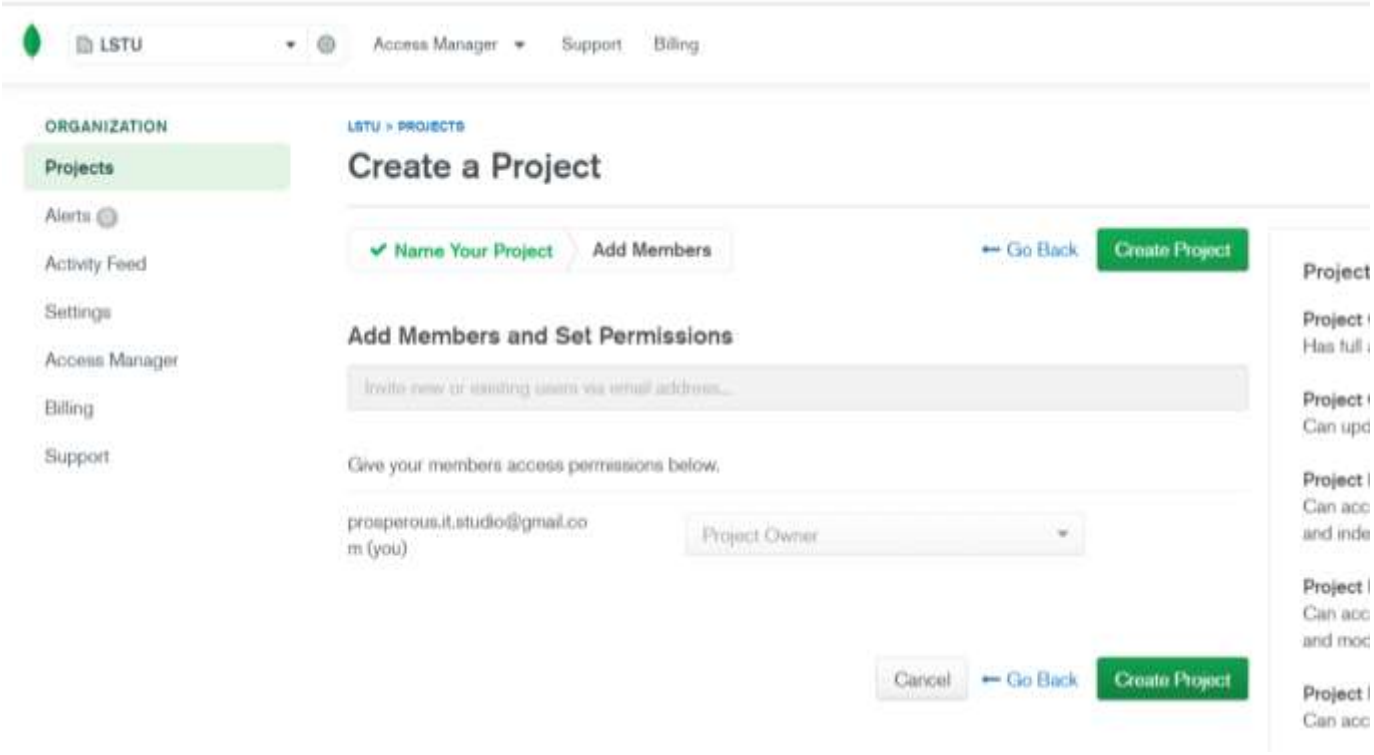

Рис. 1.23. Натискаємо Create Project

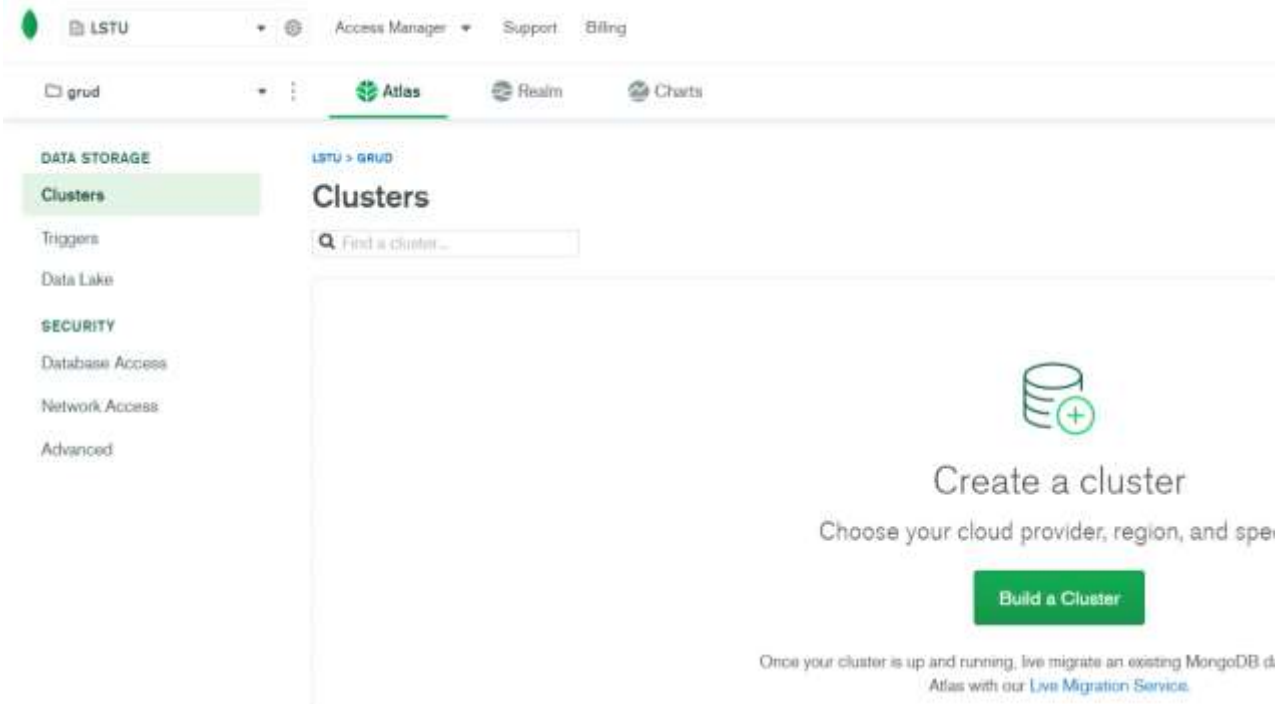

Рис. 1.24. Ініціюємо побудову нового кластеру. Далі обираємо Free

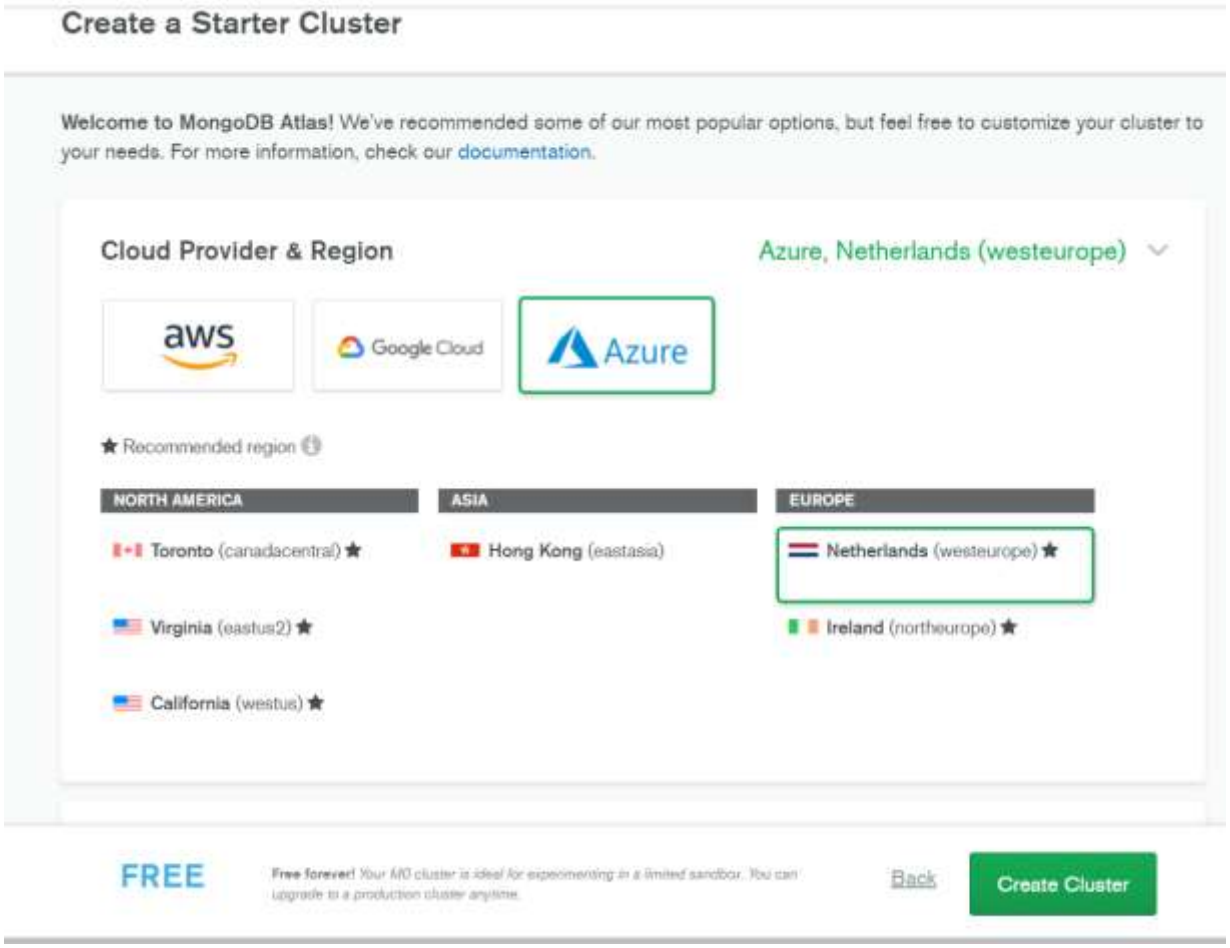

Рис. 1.25. Обраємо найближчий ресурс за географічним положенням.

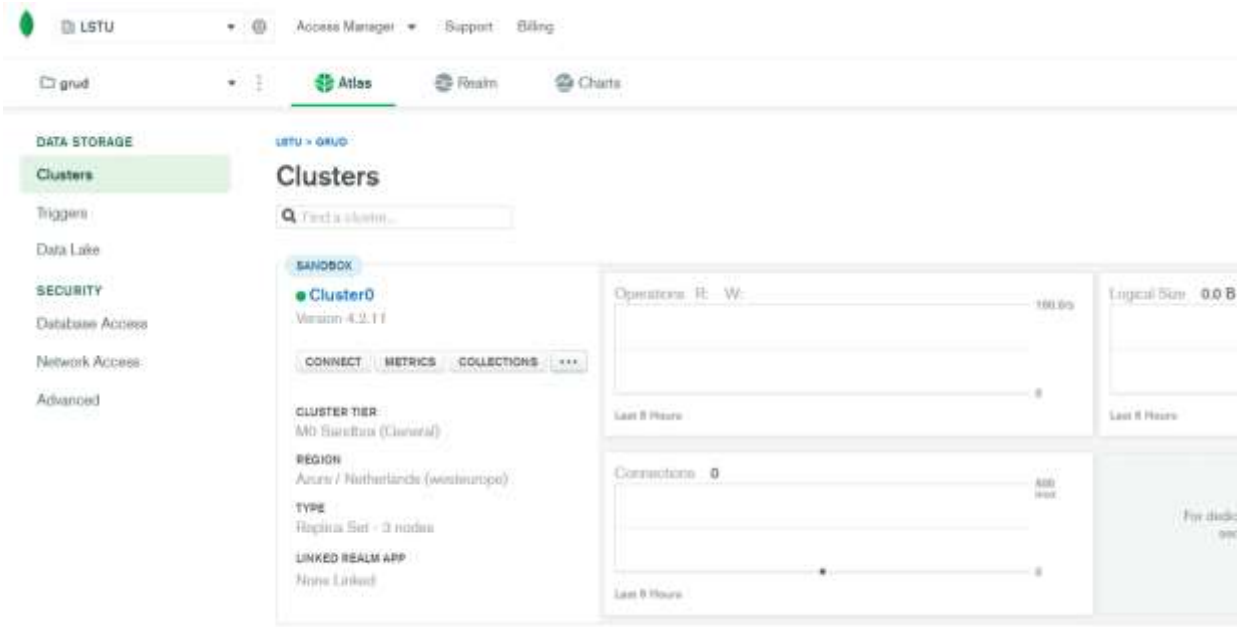

Рис. 1.26. Створюємо командою CONNECT

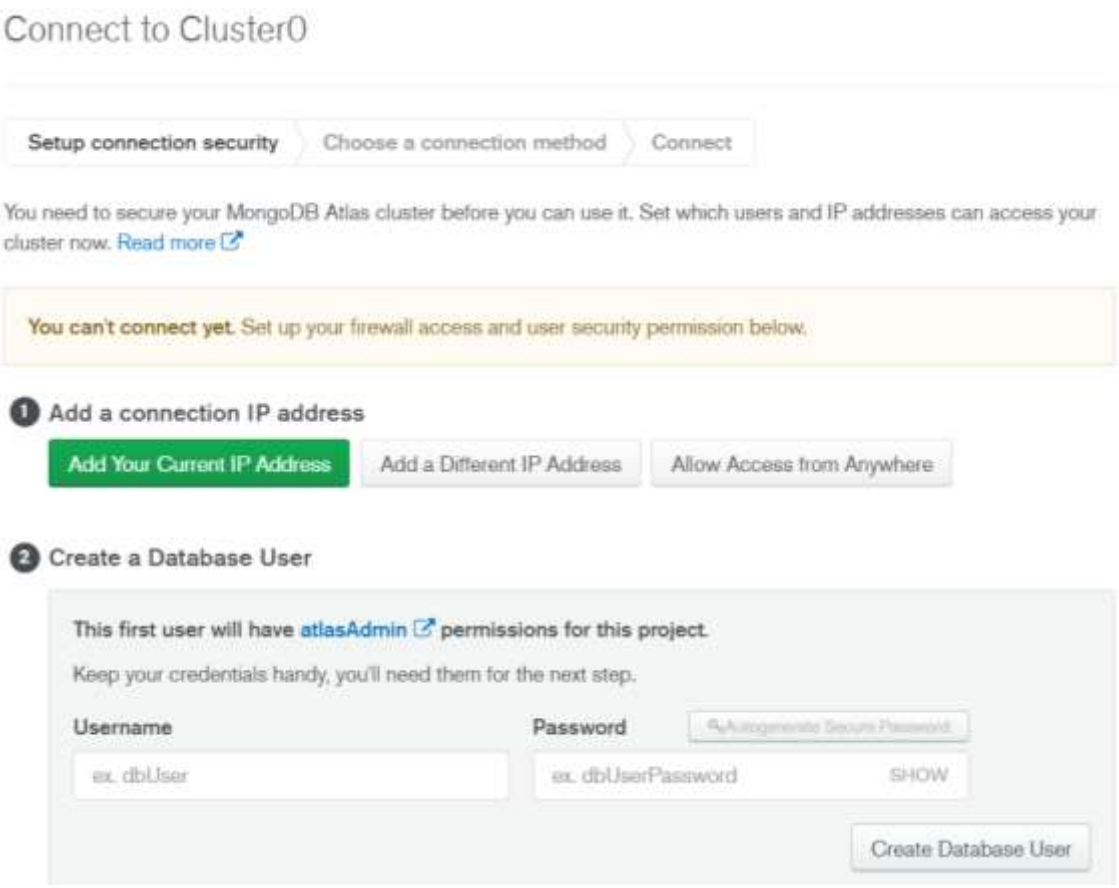

Рис. 1.27. Розпізнаємо наш IP

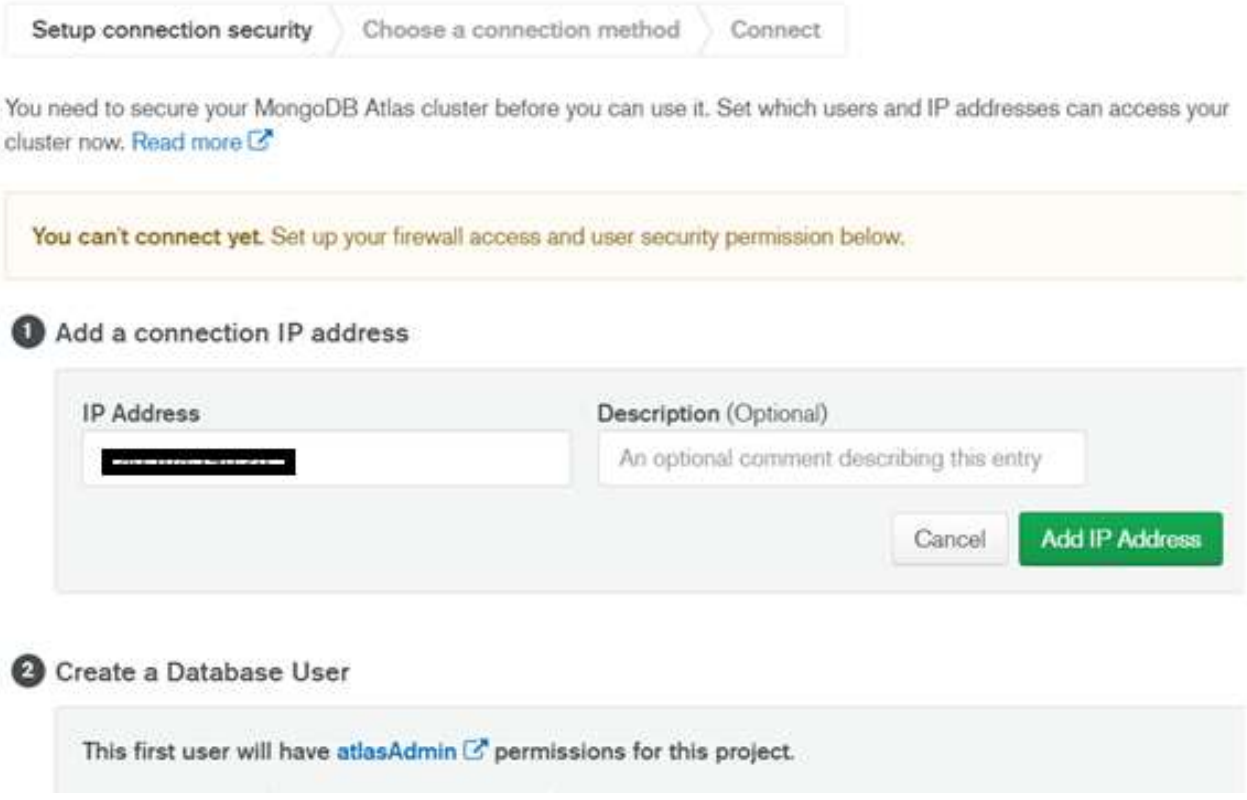

Рис. 1.28. Додаємо наш IP

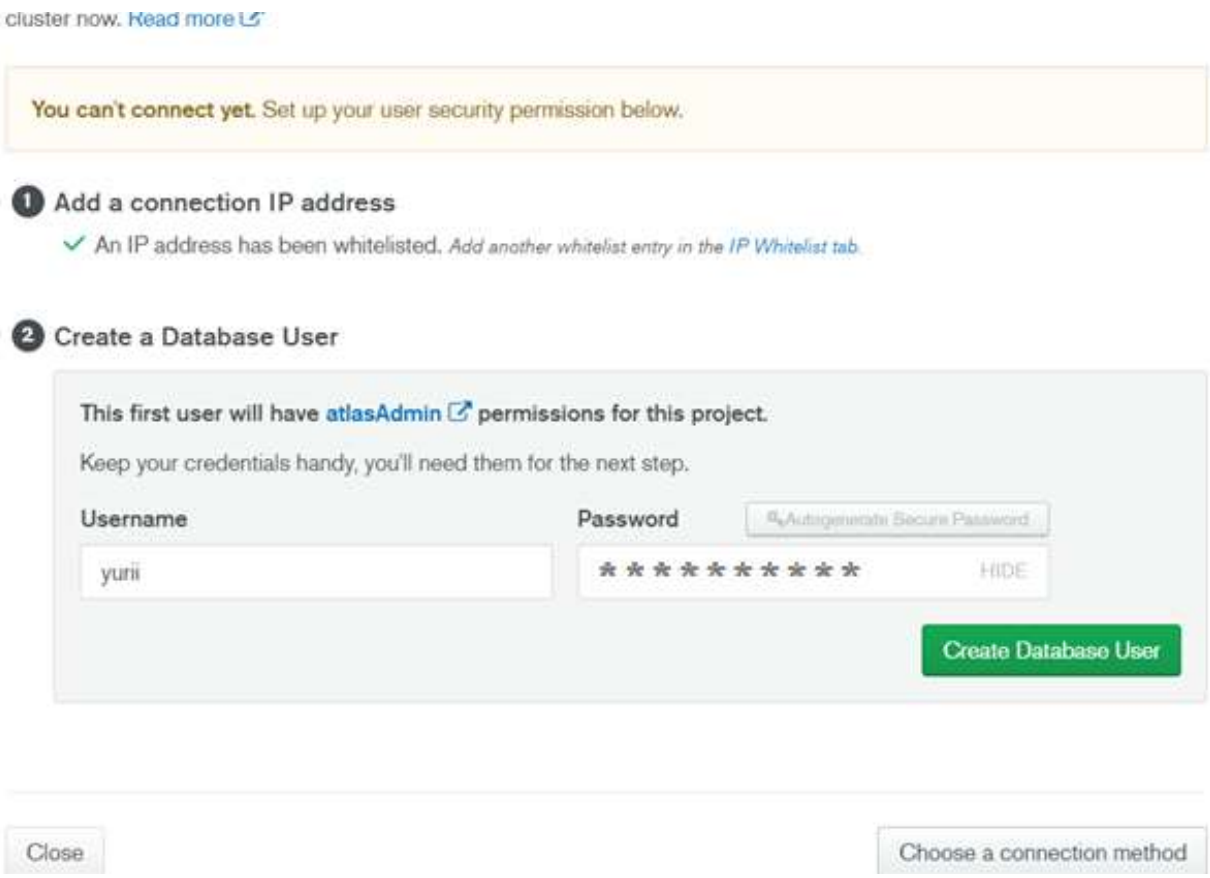

Рис. 1.29. Задаємо користувача та пароль

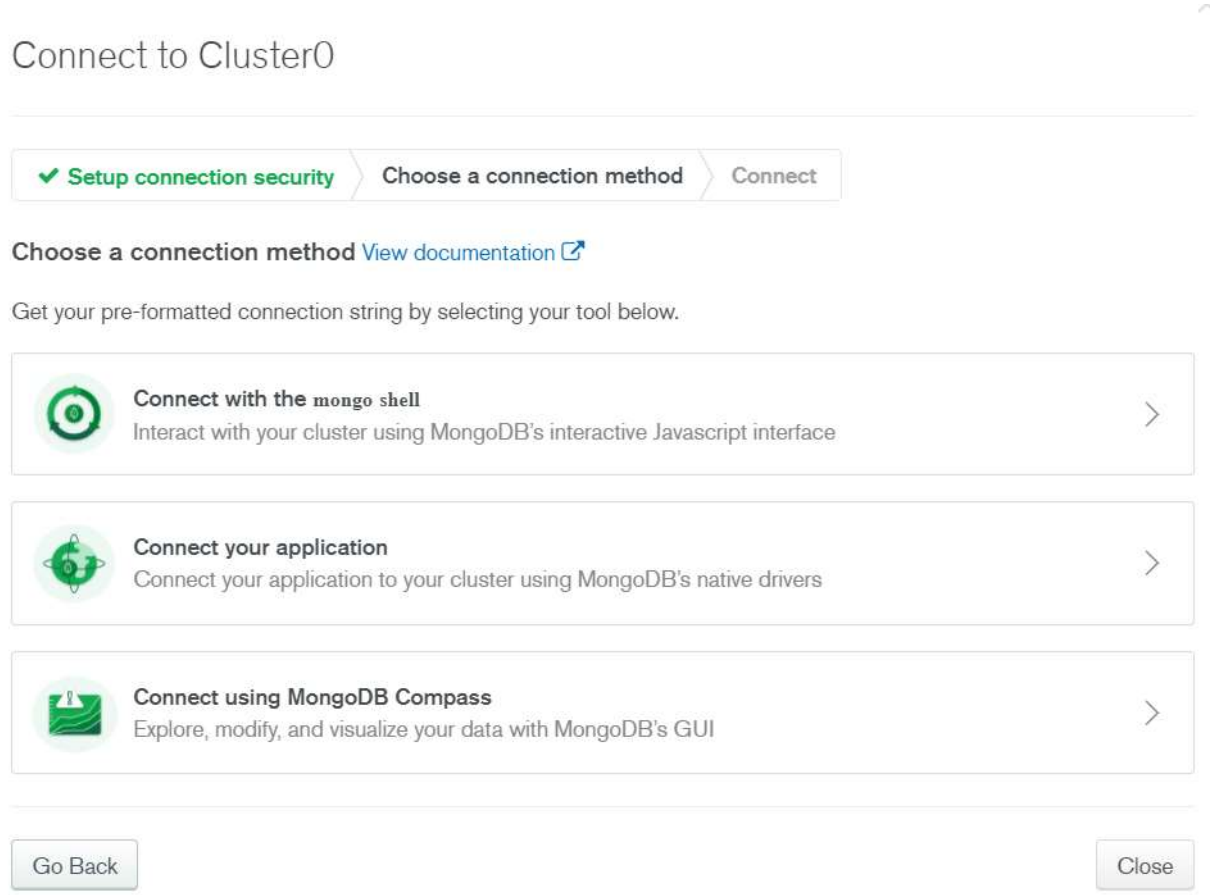

Рис. 1.30. Обираємо метод Connect your application

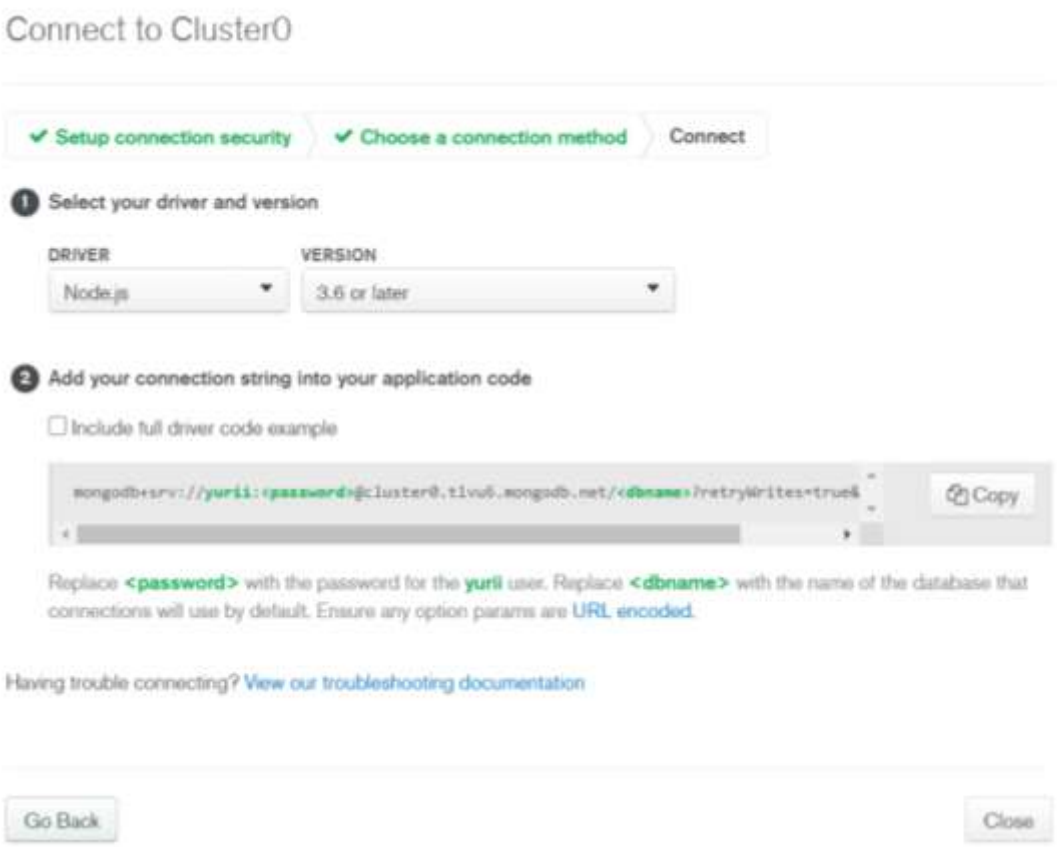

Рис. 1.31. Копіюємо код для зв’язку з базою даних тура

- замінюємо написи password та dbname на пародь та назву бази даних.

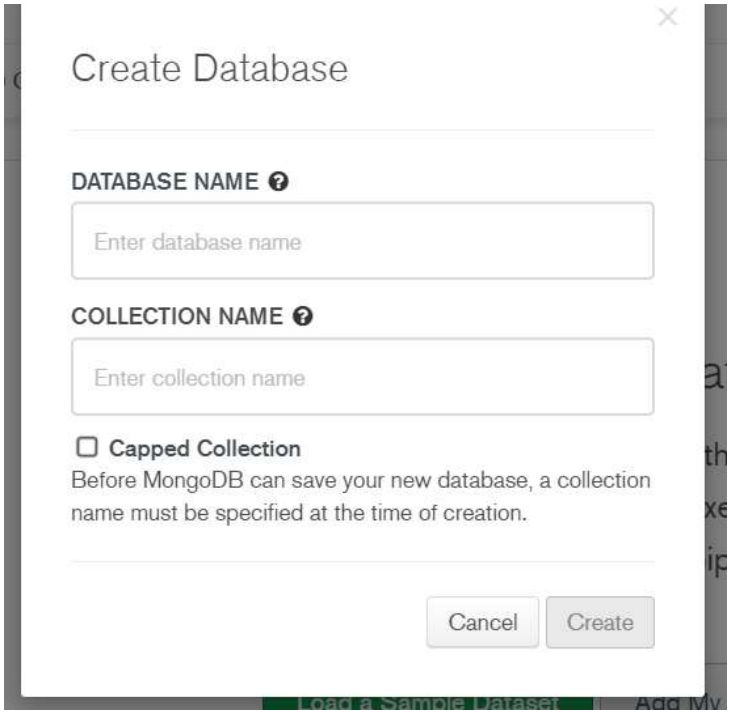

Рис. 1.32. Створюємо назви DB та Collection

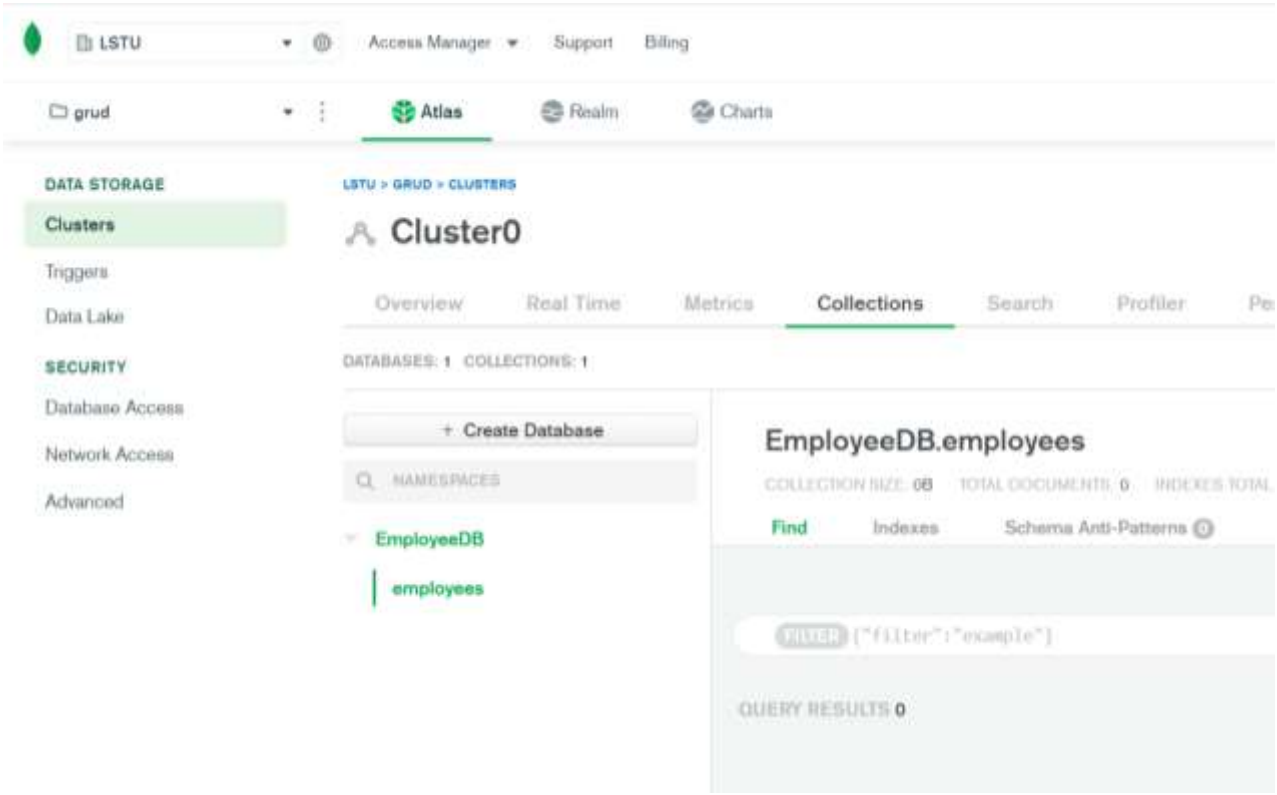

Рис. 1.33. Отримуємо готові базу даних та колекцію

Дія 1.4.4. Підключення до MongoDB у файлі app.js

- для підключення у файлі app.js:

```
const express = require('express')
const config = require('config')
const mongoose = require('mongoose')
const app = express()
// сервер повертає promis
const PORT = config.get('port') || 5000
// створюємо асинхронну функцію
async function start() {
  try {
    await mongoose.connect(config.get('mongoUri'), {
      useNewUrlParser: true,
      useUnifiedTopology: true
    })
    app.listen(PORT, () => console.log(`Server App has been started on port
${PORT}...`))
  } catch (e) {
    console.log('Server Error', e.message)
    process.exit(1)
```

```
  }
}
start()
```

Дія 1.4.5. У config/default.json створюємо константи

```
{
   "port": 5000,
   "mongoUri": "mongodb+srv://yurii: …  "
}
```

Дія 1.4.6. У **package.json** замінюємо скрипти на

```
"start": "node app.js",
"server": "nodemon app.js" // запускає бекент
```

Дія 1.4.7. Тестуємо, у консолі

npm run server

У консолі VS Code отримуємо відповідь:

Server App has been started on port 5000...