Міністерство освіти і науки України
Чернівецький національний університет імені Юрія Федьковича
Кафедра математичних проблем управління і кібернетики

Кириченко О.О., Кириченко О.Л.

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ:

ТЕХНОЛОГІЇ BACKEND РОЗРОБКИ

НАВЧАЛЬНИЙ ПОСІБНИК
(матеріали лекцій)

Чернівці
2023

УДК 004.04

Друкується за ухвалою Вченої ради Навчально-наукового інституту фізико-технічних та комп'ютерних наук Чернівецького національного університету імені Юрія Федьковича (протокол № 1 від 31.08.2022)

Рецензенти:
Луцюк В. - Solutions Architect в IT- компанії з консалтингу і розробки ПЗ «SoftServe»;
Кнопов П. С. – член-кор. НАН України, доктор фіз.-мат наук, професор, завідувач відділу математичних методів дослідження операцій Інституту кібернетики ім. В.М. Глушкова НАН України

Спеціалізовані мови програмування: технології backend розробки (матеріали лекцій): Навчальний посібник. / укл.: Кириченко О.О., Кириченко О.Л. Чернівці : Чернів. нац. ун-т ім. Ю.Федьковича, 2023, 80 с. (електронне видання)

Запропонований посібник містить теоретичний матеріал з технології Backend розробки. Подано основні поняття та принципи функціонування платформи NodeJs, способи її використання разом з фреймворками ExpressJs і Mongoose, викладено теоретичні відомості про NoSQL сховища даних на прикладі Mongodb, наведено приклади побудови веб-додатків з використанням MEAN стек.

Навчальний посібник призначено для студентів технічних спеціальностей, які володіють базовими знаннями з Веб-програмування, JavaScript та основами баз даних.

Для студентів першого (бакалаврського) рівня вищої освіти спеціальності «Комп'ютерні науки» (ОП «Алгоритмічне та програмне забезпечення комп'ютерних систем») навчально-наукового інституту фізико-технічних і комп'ютерних наук ЧНУ.

Укладачі: Кириченко Олександр Олексійович, асистент кафедри математичних проблем управління і кібернетики ННІФТКН (відповідальний за випуск),

Кириченко Оксана Леонідівна, асистент кафедри математичних проблем управління і кібернетики ННІФТКН.

Зміст

ЛЕКЦІЯ 1.

РОЗРОБКА З JS НА BACKEND

1.1. Клієнт-серверна архітектура

Клієнт-серверна архітектура визначає загальні принципи організації взаємодії у мережі, де є сервери, вузли-постачальники деяких специфічних функцій (сервісів) та клієнти, споживачі цих функцій. Сервером може бути програма, розташована на якомусь з вузлів в мережі або в хмарі, яка здатна обслуговувати запити, що входять. Клієнтом може бути програма на десктопній системі або мобільній платформі, яка звертається за інформацією до сервера. Модель взаємодії на основі подібної архітектури дозволяє розділяти функціонал та навантаження між клієнтськими програмами (замовниками послуг) та серверними програмами (постачальниками послуг).

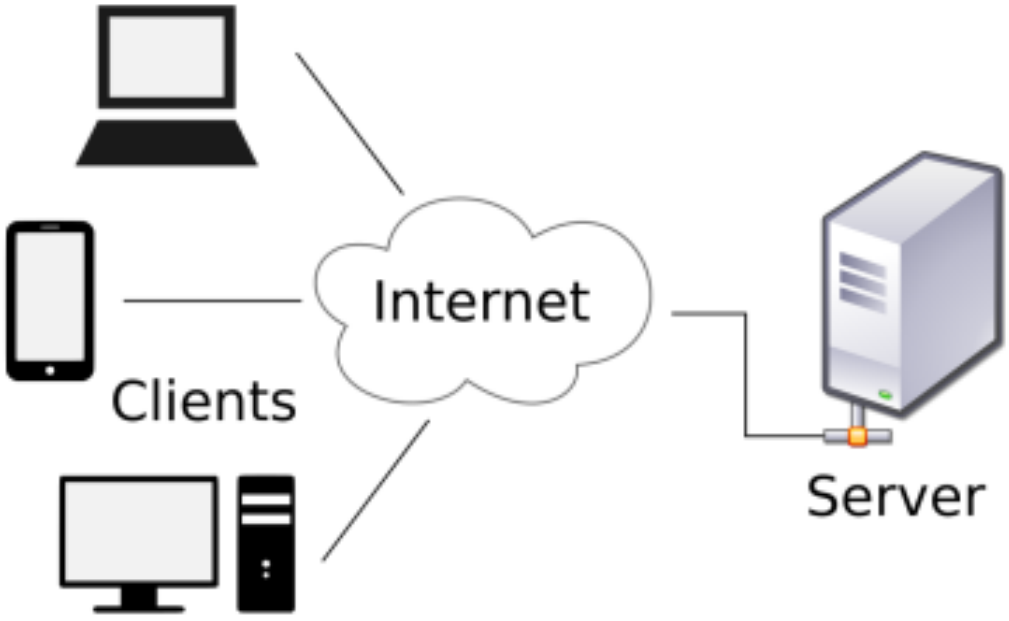

Рис.1.1. Приклад клієнт-серверної архітектури у веб

Зазвичай сервер може обслуговувати кількох клієнтів одночасно. У цьому випадку говорять про розрахований на багато користувачів режим. У такому режимі сервер обробляє запити у кількох потоках. Наприклад, сервер Apache

має Multi-Processing Modules (MPMs), які відповідають за розподіл та виконання запитів у різних потоках.

1.2. Обмеження багатопоточної моделі

Багатопоточність – властивість платформи (наприклад, операційної системи, віртуальної машини і т. д.) або програми, яка полягає в тому, що процес, породжений в операційній системі, може складатися з декількох потоків, які виконуються «паралельно», тобто без передбаченого порядку в часі.

Коли за короткий проміжок часу надходить велика кількість запитів, запити встановлюються в чергу. В даному випадку черга – це перелік невиконаних клієнтських запитів.

Іноді запити можуть мати пріоритети. Пріоритет – це рівень "важливості" виконання запиту. Запити з вищими пріоритетами мають виконуватися раніше. Зазвичай пріоритет для запитів визначається розробниками системи, базуючись на інформації в запиті та на джерелі запиту.

При запланованому навантаженні, тобто очікуваній кількості запитів за певний проміжок часу, багатопоточність дає приріст у продуктивності.

Але при пікових навантаженнях продуктивність системи різко падає, що виражається у часі очікування. "Час очікування" – це час, через який користувач, надіславши запит серверу, отримає від нього відповідь.

Вся справа в тому, програмування та підтримка багатопотокового додатка досить складна справа. У ході обробки запитів користувачів потоки можуть взаємодіяти між собою, конкурувати при використанні тих самих ресурсів. У ході взаємодії та конкуренції виникають конфлікти доступу до ресурсів та взаємні блокування.

Взаємне блокування це одна з найнеприємніших проблем при програмуванні багатопоточності. Найчастіше вона трапляється, коли потік вже заблокував ресурс А, після чого намагається провести блокування Б, інший потік заблокував ресурс Б, після чого намагається заблокувати ресурс А. Така

ситуація, як правило, відбувається дуже рідко і зловити її під час тестування практично неможливо.

Для запобігання подібним конфліктам у багатопотоковому програмуванні використовується багато різних механізмів. Але так чи інакше часто виникають ситуації, коли потік просто простоює, очікуючи доступу до ресурсу або результату виконання іншого потоку. І саме тому багатопотокова модель недостатньо ефективна для розрахованих на багато користувачів додатків з великою кількістю операцій введення виведення.

Для ефективного використання потоків було запропоновано асинхронне програмування. У традиційній практиці програмування більшість операцій введення-виведення відбувається синхронно, тобто послідовно в одному потоці. Цей потік буде заблокований доки подібні операції не будуть завершені. В асинхронному програмуванні виконання операцій введення виводу дозволяє продовжити обробку інших завдань, не чекаючи завершення операцій введення виведення.

Саме тому, зараз набирають популярності технології, які підтримують асинхронну модель програмування.

1.3. Технології MEAN стек та еволюція веб до односторінкових додатків

У світі IT стало модним говорити про стек, тобто про набір технологій, які працюють разом. Не є винятком і веб-технології. Одним з ранніх стеків веб-технологій з відкритим вихідним кодом є стек LAMP: Linux® для операційної системи, Apache для веб-сервера, MySQL для бази даних і Perl (або Python або PHP) як мова програмування для створення веб-сторінок на основі HTML.

Ці технології спочатку не були написані для спільної роботи, але за обставин застосовувалися разом. З того часу ми стали свідками вибуху веб-стеків. Здається, що у кожної сучасної мови програмування є відповідний веб-

фреймворк (або два), що включає строкатий набір технологій, щоб зробити швидкою і легкою розробку нового веб-сайту.

Тому таку велику увагу привертає MEAN стек, що з'явився, на основі технологій: MongoDB (базаданих), Express (бекенд фреймворк), AngularJS (фронтенд фреймфорк), Node.js (платформа для виконання JavaScript).

Цей стек є повністю сучасним підходом до веб-розробки, в якому одна мова програмування (JavaScript) працює на кожному рівні вашої програми, від клієнта до сервера та бази даних.

MEAN стек – це більше, ніж просто абревіатура та набір технологій.

Спроба перенести базову платформу з ОС (Linux) у середовище виконання JavaScript (Node.js) призводить до незалежності від ОС: Node.js працює у Windows® і OS X, як у Linux. Node.js також замінює веб-сервер Apache у стеку LAMP. Але Node.js – це набагато більше, ніж простий веб-сервер. Фактично, ви не розвертаєте готову програму на автономному веб-сервері натомість веб-сервер включений у вашу програму і автоматично встановлюється в стеку MEAN. В результаті процес розгортання стає значно простішим, тому що необхідна версія веб-сервера явно визначена разом з рештою часу виконання.

Перехід від традиційної реляційної бази даних, такої як MySQL до NoSQL, schemaless, document-oriented сховища даних (всі ці поняття будуть розглянуті пізніше більш детально), наприклад, MongoDB, є фундаментальною зміною в стратегії зберігання даних. Ви витратите менше часу на створення SQL і більше часу на написання map/reduce функцій JavaScript.

Ви також уникаєте великої кількості логіки для перетворення даних, оскільки MongoDB підтримує об'єктну нотацію JavaScript (JSON). Отже, писати RESTful веб-сервіси простіше, ніж будь-коли.

Але найбільша зміна в MEAN порівняно з LAMP – це перехід від традиційної генерації сторінок на боці сервера до односторінкового додатку (SPA) на боці клієнта. Використовуючи Express, ви все ще можете обробляти створення сторінок і маршрутизацію на боці сервера, але тепер акцент робиться

на представленнях на боці клієнта за допомогою Angular. Ці зміни пов'язані не тільки з перенесенням ядра реалізації Model-View-Controller (MVC) з боку сервера на бік клієнта (притому фреймворк Express.js, що входить до складу стека, забезпечує і традиційну маршрутизацію, і генерацію сторінок на стороні сервера). Насамперед вам доведеться перейти від синхронного менталітету роботи з даними до асинхронного. Також, і це найголовніше, ви перейдете від сторінково-орієнтованого представлення вашої програми до компонентно-орієнтованого.

Таким чином основна перевага MEAN стек це простота та загальна структура:

- MongoDB пропонує більш гнучкий, зручний рівень зберігання даних.
- Node.js забезпечує середовище виконання та по суті є вашим сервером, а Express допомагає стандартизувати процес створення веб-сайтів.
- з боку клієнта Angular забезпечує прозорий спосіб додавання інтерактивних функцій та компонентів на основі AJAX.

Об'єднуючи їх разом, ви можете створити простий, прозорий та узгоджений механізм для переміщення даних від користувача до сховища даних та у зворотному напрямку.

Основні переваги MEAN стек:

- MongoDB пристосовано для використання у хмарі, що дозволяє легко створити, адмініструвати та масштабувати кластер MongoDB.
- Node.js має високу продуктивність і спрощує рівень сервера.
- JavaScript використовується на всіх рівнях, що значно спрощує програмування.
- JSON використовується як формат даних на всіх рівнях.
- Angular повністю відокремлює рівень уявлення з інших рівнів у додатку.

ЛЕКЦІЯ 2.

ФІЛОСОФІЯ NODEJS. ОСОБЛИВОСТІ АРХІТЕКТУРИ

2.1. NodeJs – серверна платформа для виконання JavaScript

2.1.1. Філософія NodeJs. Основні принципи (small core, small modules, small surface area, simplicity and pragmatism)

Кожна платформа має свою власну філософію – набір принципів, які прийняті спільнотою розробників, і які впливають на розвиток платформи та те, як розробляються програми. Деякі з цих принципів випливають із самої технології, деякі – лише тенденції в співтоваристві розробників, а інші – еволюції різних ідеологій.

У NodeJs частина принципів виходять безпосередньо від творця, Райана Дала, від усіх людей, які зробили свій внесок у розробку ядра, а якісь із принципів успадковані від культури JavaScript або під впливом філософії Unix.

Жоден з цих принципів не є обов'язковим, і їх завжди слід застосовувати відповідно до здорового глузду, проте вони можуть виявитися надзвичайно корисними при прийнятті рішень щодо архітектури та дизайну додатків.

Small core

Або принцип компактного ядра. Суть цього принципу полягає в тому, що ядро має містити мінімально необхідний набір функціоналу, весь решта функціонал повинен перебувати в модулях користувача. Цей принцип дуже впливає на культуру Node.js, оскільки він дає свободу спільноті, щоб експериментувати і швидко додавати необхідний функціонал до ядра замість того, щоб чекати, коли відповідний функціонал з'явиться безпосередньо в ядрі.

Збереження основного набору функціоналу максимально компактним стає зручним не тільки з точки зору підтримки, але і з точки зору позитивного впливу, що призводить до еволюції всієї екосистеми.

Наприклад, в ядрі знаходиться низько рівнева функціональність, яка потрібна майже всім мережевим програмам: TCP, HTTP, DNS, файлова

Додамо критерій для фільтрації даних перед етапом агрегації.

```
rs-ds115533:PRIMARY> db.temp_collection.aggregate([{$match: {size: "M"}}, {$group: {_id: "$brand", totalPrice: {$sum: "$price"}}}, {$sort: {
totalPrice: -1}}, {$project:{_id: 0, brandGroup: "$_id", totalPrice: 1}}])
{ "totalPrice" : 3470, "brandGroup" : "Adidas" }
{ "totalPrice" : 1150, "brandGroup" : "Nike" }
rs-ds115533:PRIMARY>
```

Наш фінальний запит буде виглядати наступним чином:

```
db.temp_collection.aggregate(
    [
        {
            $match: {size: "M"}
        },
        {
            $group: {
                _id: "$brand",
                totalPrice: {$sum: "$price"}
            }
        },
        {
            $sort: {"totalPrice": -1}
        },
        {
            $project:{
                _id: 0,
                brandGroup: "$_id",
                totalPrice: 1
            }
        }
    ])
```

ЛІТЕРАТУРА

1. Holmes S. Getting MEAN with Mongo, Express, Angular, and Node. Second edition. Manning, 2019. 504 p.
2. Bojinov V. RESTful Web API Design with Node.js. Second edition. Packt Publishing Ltd., 2016. 148 p.
3. Mardan A. Expert Practical Node.js: Building Real-World Scalable Web Apps. Apress, 2014. 300 p.
4. Mardan A. Pro Express.js: Master Express.js: The Node.js Framework For Your Web Development. Apress, 2014. 394 p.
5. Gackenheimer G. Node.js Recipes: A Problem-Solution Approach (Expert's Voice in Web Development). Apress, 2013. 399 p/
6. Lim G. Beginning Node.js, Express & MongoDB Development. Independently published, 2011. 155 p.
7. Nodejs документація. https://nodejs.org/en/docs/
8. ExpressJs документація. https://expressjs.com
9. Mongoose документація. https://expressjs.com
10. MongoDB документація. https://www.mongodb.com
11. MongoDB Atlas. https://www.mongodb.com/cloud/atlas/efficiency