МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Побудова багаторівневого веб-застосування на хмарній платформі

Текстова частина до курсової роботи

За спеціальністю “Комп’ютерні науки” -122

Керівник курсової роботи

к.т.н., ст. викладач

Черкасов Д. І.

_____

(*підпис*)

“____” _____ 2024 р.

Виконав студент 3 курсу

Перекупка А. В.

“____” _____ 2024 р.

Київ 2024

Календарний план

Тема: Побудова багаторівневого веб-застосування на хмарній платформі

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	жовтень 2023р.	
2.	Огляд літератури за темою роботи	грудень 2023р.	
3.	Огляд існуючих рішень	січень 2024р.	
4.	Структурна розробка власного рішення	лютий 2024р.	
5.	Створення практичної частини роботи	березень- квітень 2024р.	
6.	Написання текстової частини роботи	квітень -травень 2024р.	
7.	Корегування роботи згідно з результатами перевірки керівником	травень 2024р.	
8.	Створення та оформлення доповіді	травень 2024р.	
9.	Захист курсової роботи	Травень 2024р.	

Студент _____

Керівник_____

“____ “ _____ ____ р.

Зміст

Contents

Анотація

У цій роботі розглядається побудова багаторівневого веб-застосування, де ключовим критерієм є використання хмарної платформи. При розробці в якості прикладу використовуватиметься хмарна платформа Amazon Web Services(AWS), а функціональним призначенням слугуватиме сайт новин.

Демонструються можливості AWS, як платформи для розгортання різних веб-застосувань, а також порівнюються переваги і недоліки використання хмарних рішень для проєктів. Показується робота різних компонентів через хмарні сервіси. Детально описується розробка одного із компонентів даного багаторівневого веб-застосування.

Вступ

На кожному етапі розвитку мережевих застосунків – від їх планування до фактичного використання – важливо враховувати ряд критичних факторів. Серед них, безумовно, є стійкість до відмов, оптимальне розташування апаратних компонентів, рівномірний розподіл навантаження та гнучкість масштабування. Також важливою є інтеграція з процесами розробки застосунків та підтримка автоматизованого розгортання та оновлення. Для забезпечення такого важливого функціоналу на допомогу приходять хмарні рішення, які є дуже актуальними на сьогоднішній день.

Існують різні хмарні платформи, що надають засоби для вирішення цих та інших завдань, включаючи Microsoft Azure, Google Cloud Platform та Amazon Web Services (AWS). Кожен з них по своєму вартий уваги і має свої переваги та недоліки.

Для розробки цього проєкту, а саме звичайного новинного сайту, було обрано найбільш популярний та розповсюджений варіант AWS через його надійність, гнучкість та широкий спектр послуг. Ці характеристики забезпечують ефективну підтримку та розвиток різноманітних веб-застосувань, допомагаючи забезпечити високу якість та швидкість розробки, розгортання та підтримки сайту.

1. Аналіз існуючих рішень

Популярність хмарних рішень та багаторівневих систем не з’явилась просто так. Багаторівневість вирішує ряд існуючих проблем, які утворилися з ростом різних систем і самостійністю багатьох компонентів, які було б непогано об’єднати в одну систему, яка може розширюватись по мірі створення нових компонентів, без особливих змін в існуючому коді. Використання хмари ж допомагає впоратися з задачами розподілення навантаження на систему, масштабованості серверів у разі потреби. Саме тому такі рішення використовуються здебільшого для комерційної розробки великих систем. Тому інша сторона медалі використання хмарних рішень та розбиття на менші блоки – це непотрібність для деяких проєктів масштабованості взагалі, так як підтримка і розробка може бути важче і коштуватиме в рази дорожче, ніж звичайне монолітне веб-застосування, яке працює на домашньому комп’ютері, яке задовольняє усі потреби. На даний момент існує декілька найбільш популярних рішень побудови систем. Розглянемо їх по-порядку і почнемо з багаторівневості.

Багаторівневість - це архітектурний підхід, який передбачає розділення програмного застосунку на логічні рівні або шари. Кожен рівень відповідає конкретним функціям або обов'язкам застосунку і має свою власну логіку та відповідальності.

1.1 Багаторівневість

1.1.1 Багаторівнева архітектура

Зазвичай багаторівнева архітектура включає наступні шари(але можливі і інші компоненти такі як розбиття на бізнес логіки на мікро сервіси та інші):

1. **Рівень представлення** (Presentation Layer): Цей шар відповідає за представлення інформації користувачу та взаємодію з ним через інтерфейс користувача. Сюди можуть входити веб-сторінки, мобільні додатки, графічні інтерфейси тощо.
2. **Рівень бізнес логіки додатку**: (Application Layer)**:** Цей шар містить бізнес-логіку застосунку, що відповідає за обробку даних, виконання бізнес-правил та логіку взаємодії з базою даних.
3. **Рівень Доступ до даних** (Data Access Layer)**:** Цей шар забезпечує доступ до даних, здебільшого через взаємодію з базою даних. Він відповідає за виконання запитів до бази даних, збереження та витягування даних.
4. **Інфраструктура** (Infrastructure Layer)**:** Цей шар забезпечує базову інфраструктуру застосунку, таку як управління конфігурацією, журналювання, безпеку тощо.

Переваги**:**

Багаторівнева архітектура дозволяє розділити застосунок на окремі компоненти, що спрощує його розробку, тестування та підтримку. Кожен рівень може бути реалізований незалежно від інших, що дозволяє змінювати або модифікувати один шар без впливу на інші.

Недоліки:

Об’ємність розробки через велику кількість відокремлених завдань і налаштувань, які не є необхідними для невеликих систем. А також можливі невиправдано великі витрати для підтримки надлишкового функціоналу.

Узагальнюючи можна сказати, що багаторівневість це дуже гарне рішення для великих, постійно ростучих систем, але є надлишковим для маленьких і самодостатніх.

1.1.2 Трирівнева архітектура

Досить популярним є використання трирівневої архітектури. Зазвичай це представлення без блоку інфраструктури яка включає в себе:

1. Рівень представлення
2. Рівень бізнес-логіки
3. Рівень баз даних

Такий підхід дозволяє так само гарно масштабувати кожен з представлених рівнів незалежно один від одного, але при цьому рівень інфраструктури, який включає в себе налаштування безпеки та конфігурування проєкту.

1.1.3 Дворівнева та монолітна архітектура

Відмінність дворівневої та монолітної(однорівневої) в тому, що в першому випадку це проста реалізація клієнт-серверної архітектури, в якій зазвичай база даних працює на іншому сервері, а частина візуального інтерфейсу та бізнес логіки об’єднані в один проєкт. При монолітній реалізації, рівень бази даних знаходиться на тому ж сервері, що і інша частина проєкту.

Це дозволяє легко комунікувати усій програмі і легко її створювати та запускати, але на відміну від багаторівневих важко масштабувати, через тісну пов’язаність усього коду.

1.2 Використання хмарних та власних систем

1.2.1 Розміщення на хмарній платформі

Розміщення на хмарній платформі передбачає збереження та обробку даних, а також виконання обчислень на віддалених серверах, які належать хмарному провайдеру. При цьому інфраструктура, така як сервери, знаходиться поза межами організації і управляється хмарним провайдером. Однією з головних переваг такого підходу є гнучкість та масштабованість: користувач може легко збільшувати або зменшувати обсяг ресурсів в залежності від потреб застосунку, а також швидко реагувати на зміни в навантаженні. Крім того, хмарні платформи часто пропонують широкий спектр послуг, включаючи резервне копіювання, безпеку даних, моніторинг та інші.

1.2.2 Розміщення на власному сервері.

У випадку використання власного сервера, організація або розробник має повний контроль над інфраструктурою. Купляється або орендується серверне обладнання ,яке розгортається власними силами. Це дозволяє вирішувати специфічні вимоги щодо безпеки, доступності та конфіденційності даних, а також забезпечувати високу швидкодію та надійність роботи. Проте такий підхід потребує значних витрат на придбання та підтримку обладнання, а також на управління та підтримку інфраструктури. Крім того, масштабування такої системи може бути обмеженим, оскільки воно вимагає фізичного розширення серверних парків і відповідної інфраструктури. Проте за відсутності проблеми зі збільшенням клієнтів або потреби покращення заліза, це може бути гарним і вигідним рішенням на дистанції.

Також є варіант з використанням орендованих серверів, де так само проходить самостійне налаштування системи і на відміну від хмари, орендується саме фізичний статичний сервер для своїх потреб, а не контейнер, яким керує провайдер хмарної платформи. При цьому відсутня опція самостійної заміни заліза на сервері за необхідності, на відміну від власних. У противагу можна сказати, що зазвичай орендодавці серверів пропонують можливий перехід на залізо з іншими характеристиками за потреби користувачів.

1.3 Висновок огляду можливих рішень

Кожний з вище розглянутих варіантів має свої переваги і недоліки. Однорівневі застосування легко створювати, просто запустити на одному сервері, але важко масштабувати та змінювати. В той час як використання багаторівневих системи легко масштабуються при додаванні різних компонентів, їх легко підтримувати та модернізувати за наявності необхідних ресурсів, але для невеликих систем такий підхід може бути надлишковим або неприйнятним, якщо розробник не хоче довіряти свої дані іншим системам. Тому можна прийти до висновку, що для маленьких систем, хмарні рішення не є такими значимими, як для багаторівневих веб-застосувань.

2. Огляд хмарних платформ та функціоналу Amazon Web Services

2.1 Вибір платформи для побудови

Розглянувши можливі рішення у попередньому блоці, можна прийти до висновку, що використання хмара дійсно гарне рішення для побудови багаторівневого застосування. Тепер лишається вибрати лише хмарного провайдера для подальшої розробки. Вибір відразу скорочується до 3 найбільших постачальників цієї послуги Amazon Web Services, Google Cloud та Microsoft Azure.

Кожен з цих провайдерів має велику кількість різноманітних сервісів і немає абсолютно точної інформації скільки саме в кожному із них через те, що навіть на сайті виробників немає точної кількості, бо це число постійно змінюється. За однією інформацією переважає AWS, а за іншою Azure.

Якщо оцінювати глобальність, то кожен з них покриває всю планету, тому різниця швидкості чи доступності можна виміряти в декількох мілісекундах.

Розглядаючи переваги цінової доступності, то ціни в кожному регіоні різні, тому важко обрати щось очевидно краще, якщо, наприклад, використовувати декілька серверів по всьому світу.

Тому проаналізувавши ситуацію, можна зрозуміти, що в більшості випадків вибір йде за персональним уподобанням або за дрібними деталями, на які звертають увагу при створенні якогось комерційного проєкту.

Вибір AWS в якості системи для розгляду підкріплюється просто популярністю і володінням більшості ринку порівняно з іншими конкурентами, не враховуючи мінімальні системні переваги.

2.2 Функціонал Amazon Web Services

2.2.1 Amazon EC2

Amazon EC2 (Amazon Elastic Compute Cloud) - це веб-сервіс, що надає масштабовані обчислювальні ресурси у хмарному середовищі. EC2 легко масштабувати і модернізувати. Він надає великий спектр вибору найновіших процесорів, пам'яті, об'єму сховища та багатьох інших характеристик. Окрім вибору заліза є можливість вибору операційних систем, мереж доступних зон. EC2 має багато різних заходів безпеки, такі як VPC, розбиття мережі на декілька підмереж, можливості шифрування та багато іншого. Ну і надзвичайно зручна модель оплати, де користувач платить тільки за ті ресурси, які він використав. Цей сервіс найкраще підходить для розміщення рівня бізнес логіки.

2.2.2 Amazon S3

Amazon S3 (Simple Storage Service) - це служба для зберігання даних у вигляді об'єктів. Тут можна зберігати все що завгодно, в більшості випадків використовується для статичних файлів: зображень, відео, книжок та інших об’єктів, які зазвичай не зберігаються у базах даних. Amazon гарантує доступність і цілісність файлів, навіть при великих збоях системи.

2.2.3 Amazon RDS

Amazon RDS (Relational Database Service) – цей сервіс слугує для розгортання та легкого налаштування баз даних у великих обсягах, використовуючи популярні двигуни для БД, такі як Oracle, PostgreSQL, MySQL та інші

2.2.4 Amazon DynamoDB

DynamoDB - це повністю керований сервіс NoSQL баз даних, який надає масштабованість та високу доступність для додатків з великим обсягом даних. DynamoDB добре підходить для сценаріїв, де потрібен швидкий доступ до даних та масштабованість.

2.2.5 Amazon DocumentDB

Amazon DocumentDB - це керований сервіс баз даних, який сумісний з MongoDB. Він підтримує реляційній модель даних та може бути використаний для застосунків, які потребують гнучкості NoSQL баз даних разом із можливістю використання функціональності MongoDB.

2.2.6 Amazon VPC

Amazon VPC (Virtual Private Cloud) - цей сервіс уже згадувався у блоці EC2, як один з параметрів безпеки . Він дозволяє користувачам створювати власні віртуальні мережі у хмарі, повністю контролюючи їх конфігурацію, включаючи вибір IP-адрес, створення підмереж і налаштування маршрутизації.

2.2.7 Amazon Lambda

Amazon Lambda - це сервіс обчислення без сервера, який дозволяє виконувати код без необхідності керувати інфраструктурою сервера. Він автоматично масштабується, виконуючи код тільки при зверненні та у відповідь на події. Може слугувати аналогом EC2, але без потреби зайвих налаштувань, тільки необхідні підрахунки і сервіси.

2.2.8 Amazon CloudFront

Amazon CloudFront – це служба CDN (Content Delivery Network), яка надає швидку та безпечну доставку вмісту користувачам по всьому світу з використанням мережі серверів, розташованих у різних географічних регіонах. Він кешує контент на своїх серверах, для швидкого доступу

клієнтів. Також допомагає з захистом від DDoS-атак за допомогою розподілу навантаження по різних серверах.

2.2.9 Amazon ECS

Amazon ECS (Elastic Container Service) – цей сервіс слугує для управління Docker-контейнерами, їх легкого запуску, управлінням, масштабуванням різних додатків. Також займається відновленням контейнерів у разі їх відмови.

2.2.10 Узагальнення

Усі вищезазначені сервіси є дуже корисними та використовуваними. Тут перераховано лише найбільш популярні служби для побудови багаторівневого хмарного застосування, якими клієнти найбільше користуються. Також потрібно зазначити, що кожен з цих сервісів прекрасно взаємодіє один з одним, що дає можливість розроблювати безпечні, масштабовані застосування, розміщуючи усі потрібні компоненти у хмарі з мінімальними ризиками втратити дані.

3. Структурна розробка власного рішення

Веб-застосування розподілено на 3 частини: бізнес-логіка, користувацький інтерфейс та база даних. Основна ціль проєкту побудувати сайт новин, які можуть читати різні користувачі з різних регіонів, враховуючи текст, зображення, тощо.

Для створення хмарного застосунку, а саме його обчислювальної частини, використовується сервіс на даний момент Amazon Lambda, який дає можливість виконувати код без створення та налаштування віртуального серверу. Для майбутніх версій розглядається можливість розгортання проєкту на EC2, який забезпечує створити більш налаштовану середу розробки з можливим розширенням.

Для зберігання сторінок та зображень використовується сервіс S3, який забезпечує легку доступність до усіх файлів застосунку і надійність збереження.

Також для зберігання бази даних використовується сервіс RDS, який забезпечує можливість створювати та управляти БД і не хвилюватися про її цілісність.

Також важливо зазначити CDN, який працює через CloudFront сервіс, який допомагає зі швидким доступом статичних файлів, які знаходяться в S3.

3.2 Рівень баз даних

Почнемо з найочевиднішого, це рівня баз даних. Для якого використовується рушій MySQL. Оскільки база новин з самого початку відсутня, нічого не заважає запустити базу даних відразу на RDS без втрати даних. Для створення такої бази достатньо обрати рушій, налаштувати регіон, розмір екземпляра, ім'я бази даних, ім'я користувача та пароль. Також можна обрати параметри мережі та безпеки за необхідністю.

3.2 Рівень бізнес-логіки

Як вже було описано вище, для розгортання коду бізнес логіки було обрано AWS Lambda, як одне із можливих рішень. Воно забезпечує можливість обробки подій без створення віртуального серверу. В якості фреймворку для розробки бізнес-логіки використовується Java Spring Boot. Так як він надає великий спектр можливостей для побудови цільної та багатошарової архітектури. Для розгортання такого проєкту через AWS Lambda, потрібно зібрати Spring Boot застосунок та створити JAR файл. Потім створити новий Lambda Layer та завантажити в нього JAR файл Spring Boot застосунку разом з усіма залежностями. В кінці потрібно налаштувати тригери для виконання проєкту, а також додати звідки будуть братися дані. А це просте посилання з RDS, до розміщена база даних проєкту.

3.3 Рівень користувацького інтерфейсу

Для розробки користувацького інтерфейсу використовувався Vue, для можливості динамічного оновлення сторінки. Тому для забезпечення відображення статичних файлів використовується Amazon S3. Для його використання, потрібно створити новий S3 bucket з ім’ям та регіоном для збереження даних. Далі йде налаштування необхідних дозволів для завантаження файлу. Після усього налаштування усі статичні файли Vue,

JavaScript та зображення завантажуються у Bucket. Потім йде налаштування доступів для використання файлів на сайті. І фінальною стадією є налаштування CDN, для покращення швидкодії доставки файлів за допомогою сервісу CloudFront.

4. Детальна розробка компонента (Бізнес-логіка)

Для показу детально розробки, було обрано рівень бізнес логіки, щоб показати як працює Spring Boot проєкт, що використовується і які запити він має.

Почнемо з налаштувань.

```
dependencies { Edit Starters...
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.mysql:mysql-connector-j'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

Рис. 1

Для базової роботи проєкту додано залежності:

JPA – додавання ORM системи, для того щоб створення сутностей у базі даних відбувалось автоматично на основі певного класу, а об’єкти цього класу вважалися рядками цієї сутності відповідно.

Spring Boot Starter-web допомагає зробити застосунок повноцінним сервером.

Lombok – бібліотека, яка допомагає динамічно створювати конструктори, гетери та сетери. Таким чином при створенні або видаленні, якоїсь змінної, Lombok автоматично врахує це у своїх функціях.

Devtools – просто додають корисний інструментарій для розробки типу автоматичного перевантаження серверу при зміні коду, це дозволяє не робити це руками щоразу.

MySQL connector – назва говорить сама за себе. Ця залежність допомагає зв’язатись з MySQL базою даних.

Starter-test – спрощує налаштування тестування у проєкті

Перейдемо до доступу до бази даних, налаштування досить просте, в якості прикладу ми під'єднуємось до локальної бази даних задаючи параметри користувача і посилання на нього.

```
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/news_site
```

Рис.2

Розробка бізнес-логіки

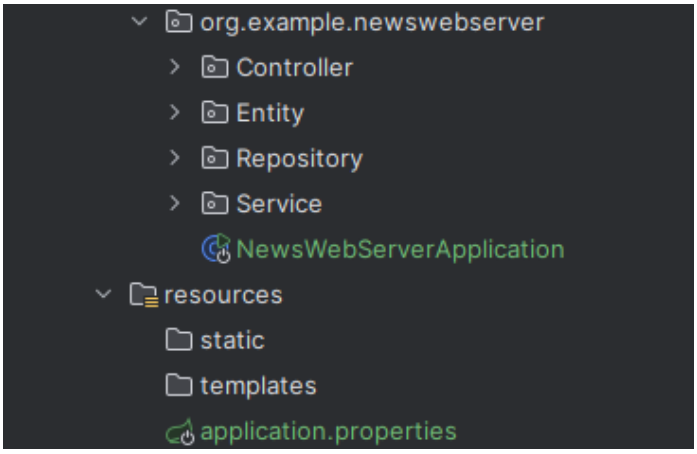

Рис. 3

Весь застосунок розподілений на 4 основні шари:

Repository – там знаходяться усі інтерфейси, які реалізують усі основні операції, такі як збереження, оновлення, видалення та пошук даних, що дозволяє працювати з базою даних з використанням стандартних методів Java, замість писання SQL-запитів вручну

Entity – папка для розміщення класів – сутностей, саме вони конвертуються у дані в базі.

Service – папка для класів сервісів, яка оброблює, налаштовує та реалізує дії з відповідного Repository інтерфейсу.

Controller – папка для збереження класів контролерів, які відповідають за обробку HTTP запитів.

Таким чином при отримані певного запиту ми оброблюємо його в контролері і передаємо сервісу, який вже робить відповідні дії у базі даних через репозиторій.

Для демонстрації ось основні HTTP-запити, які реалізовані в класі NewsController

get:- api/news/all - запит для отримання всіх новин

get:- api/news/{id}- запит для отримання новини за id

get:- api/news/sport – запит для отримання новин за категорією sport

get:- api/news/health - запит для отримання новин за категорією health

post:- api/news/add – запит для додавання новини.

delete:- api/news/delete/{id} – запит для видалення новини за id

Далі всі ці “ендпоінти” використовуються для запитів на клієнті.

Висновки

В результаті курсової роботи був продемонстрований процес побудови багаторівневого веб-застосування на хмарній платформі Amazon Web Services. Було порівняно і досліджено використання різних рівнів побудови застосунку, а також актуальність використання своїх власних серверів та хмарних рішень їм у противагу. Було порівняно найбільш відомі хмарні платформи та розглянуто основні сервіси AWS та їх функціонал, для розуміння та використання у побудові веб-застосунку. Знайдено оптимальні рішення, щодо використання сервісів хмарного провайдера для розміщення окремих компонентів для кожної задачі і прокоментовано процес їх розгортання. Було покращено навички з розробки всіх рівнів веб-застосування, а також отримано нові актуальні знання та вміння взаємодії з хмарною платформою для розміщення бази даних, бізнес-логіки та статичних файлів за допомогою сервісів Amazon Lambda, S3, CloudFront, RDS.

Джерела

1. Amazon Web Services [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Amazon_Web_Services
2. Create your first S3 bucket [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/AmazonS3/latest/userguide/creating-bucket.html
3. AWS Cloud Products [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/products/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc&awsf.re%3AInvent=*all&awsf.Free%20Tier%20Type=*all&awsf.tech-category=*all
4. Getting started with Lambda [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html
5. Creating an Amazon RDS DB instance [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html
6. How to Easily Deploy a Spring Boot Application to AWS EC2 [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=_vOInY6SRVE&ab_channel=TheDevWorld-bySergioLema
7. Setup CloudFront & Amazon S3 to Deliver objects on the Web Apps [Електронний ресурс]. – Режим доступу: https://dev.to/aws-builders/setup-cloudfront-amazon-s3-to-deliver-objects-on-the-web-apps-securely-efficiently-2gnk
8. Overview of Spring Boot Dev Tools [Електронний ресурс]. – Режим доступу: https://www.baeldung.com/spring-boot-devtools