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»

Керівник курсової роботи
к.т.н, стар.викладач
Черкасов Д. І.

_____

(підпис)
"____" _____ 2024 р.

Виконав студент 3-го курсу
Фетісов Я. Д.
"____" _____ 2024 р.

Київ 2024

Зміст

Календарний план виконання роботи

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень 2023 р.	
2.	Аналіз предметної області	Листопад 2023 р.	
3.	Вибір концепції додатку	лютий 2024 р.	
4.	Написання текстової частини курсової	квітень 2024 р.	
5.	Розробка додатку	травень 2024 р.	
6.	Оформлення презентації для захисту	травень 2024 р.	
7.	Здача роботи для перевірки на плагіат	14 травня 2024 р.	
8.	Захист курсової роботи	20 травня 2024 р.	

Студент: Фетісов Я.Д.

Керівник: Черкасов Д.І.

“_____” _____

Анотація

Ця робота присвячена розробці багаторівневого веб-застосунку на базі хмарної платформи AWS, що демонструється на прикладі створення онлайн-бібліотеки. Було детально розглянуто різноманітні підходи до побудови багаторівневих веб-застосунків та варіанти їх розміщення, а також моделі хмарних послуг, які можуть бути застосовані для реалізації проекту.

Застосунок розроблено з урахуванням високої доступності та надійності. В роботі проведено глибокий аналіз різноманітних технічних рішень та технологій, які можна застосувати для задоволення основних вимог проєкту.

В ході дослідження було розроблено структурну основу системи, що є легкомаштабованою та демонстує основі приципи хмарних-веб-застосувань.

Вступ

У сучасному світі швидкого розвитку технологій, інформаційні системи стають все більш складними і вимогливими до ресурсів. Ця обставина вимагає розробників відмовлятися від традиційних методів розробки коду та все більше звертатися до послуг різноманітних хмарних сервісів.

Серед основних переваг застосунів, що розроблені на хмарній платформі це:

- Можливість легко масштабувати ресурси вгору або вниз відповідно до потреб застосунку, забезпечуючи оптимальне використання ресурсів без необхідності вкладень у фізичну інфраструктуру.
- Швидка адаптація до змін в навантаженні, автоматичне налаштування використання ресурсів для оптимізації продуктивності та вартості.
- Високий рівень надійності з автоматичним резервним копіюванням даних та реплікація через географічно розподілені центри обробки даних, що знижує ризик втрати даних.
- Фізична безпека, захист від кібератак, шифрування даних та багато інших заходів.
- З хмарними платформами компанії можуть уникнути великих капіталовкладень у власну IT-інфраструктуру та зменшити витрати на обслуговування, оскільки більшість управлінських і технічних задач виконує провайдер.
- Хмарні застосунки можуть бути доступні з будь-якої точки світу, де є доступ до інтернету, забезпечуючи зручність для користувачів та співробітників, що працюють віддалено.

- Розробка, запуск та оновлення застосунків у хмарному середовищі зазвичай займає менше часу, завдяки використанню передвстановлених платформ і сервісів, а також автоматизації багатьох процесів.

Метою цієї курсової роботи є аналіз функціональниих можливостей хмарних платформ, розбір методів та підходів побудови відповідних веб-застосувань, розгляд можливої архітектури таких застосунків на хмарній платформі. Крім цього, метою також є порівняння переваг та недоліків традиційних методів розгортки веб-застосунків з використанням хмарних технологій. Практичну частину цієї роботу присвячено побудові багаторівневої онлайн бібліотеки на хмарній платформі AWS.

1. Аналіз наявних рішень

Цей розділ присвячений аналізу наявних рішень у контексті різнорівневих веб-застосунків, розміщення інфраструктури та хмарних технологій. Він розпочинається з детального розбору архітектури різнорівневих веб-застосунків, де основна увага приділяється чотирьом типам архітектур: однорівневій, дворівневій, трирівневій та багаторівневій. Це дозволяє зрозуміти, як різні архітектурні рішення впливають на взаємодію компонентів системи та її загальну ефективність.

Далі йде аналіз варіантів розміщення веб-застосунків, включаючи розміщення на власних ресурсах, орендованому дата-центрі та на хмарних ресурсах. Кожен варіант розглядається з точки зору його переваг та можливих обмежень, що важливо для вибору найоптимальнішого рішення в залежності від потреб бізнесу та вимог до надійності та масштабованості.

Завершується розділ аналізом хмарної піраміди, що включає моделі Saas (програмне забезпечення як сервіс), Paas (платформа як сервіс) та Iaas (інфраструктура як сервіс). Кожна з цих моделей детально розглядається на рівні абстракції, яку вони надають користувачам та їх відмінності.

1.1 Розбір Архітекури різнорівневих веб-застунків

1.1.1 Однорівнева архітектура

Однорівнева архітектура – це моделль, в якій всі частини застосування, включаючи клієнта, сервера та базу даних, розташовані на одному сервері, будь то віртуальному чи фізичному. Це означає, що система управління базами даних (DBMS) розташована на локальній системі і дозволяє користувачам отримати прямий доступ до бази даних. Такий підхід можна побачити в навчанні SQL, коли SQL-сервер і база даних встановлюються на локальному комп'ютері, що дозволяє користувачам безпосередньо взаємодіяти з реляційною базою даних і виконувати операції [1].

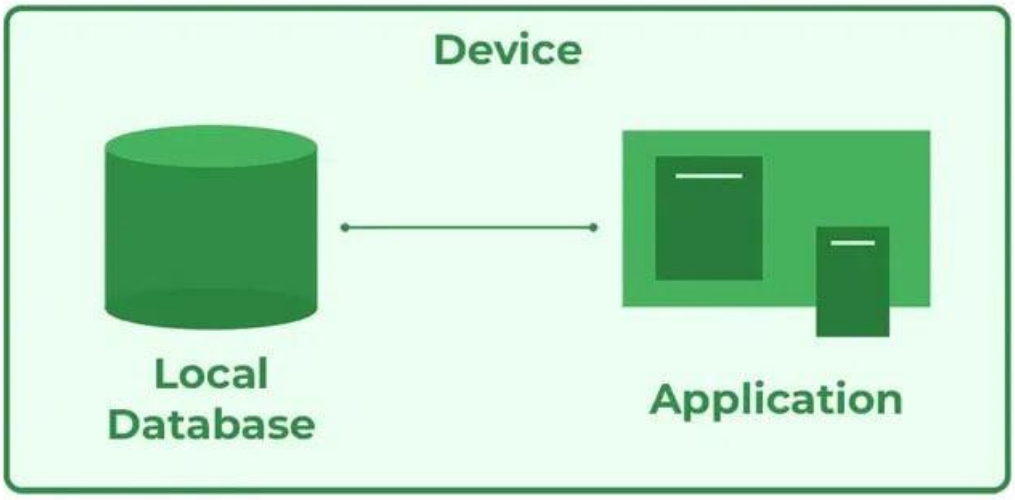


Рисунок 1.1. Схема однорівневої архітектури

Однорівнева архітектура має багато переваг, з яких найважливішою є простота використання. Вона ідеальна для освітніх цілей або невеликих додатків, оскільки вона не вимагає складних конфігурацій і додаткового обладнання. Оскільки в цій архітектурі відсутні зовнішні затримки, пов'язані з мережевими запитами або взаємодією між різними серверами, ця архітектура гарантує швидкий доступ до даних.

Однак одноріvнева архітектура також має багато обмежень. Обмежена масштабованість є основним недоліком. Всі операції виконуються на одному сервері, що може призвести до перевантаження системи, коли кількість користувачів або обсяг даних збільшується. Такий метод також може створити значні загрози для безпеки, оскільки пошкодження одного елементу системи може призвести до несанкціонованого доступу до всіх її компонентів.

І хоча одноріvнева архітектура не знаходить широкого застосування в ринкових застосунках, де перевагу віддають дворівневим та трирівневим архітектурам, вона все ж зберігає свою важливість в освітніх та тестових середовищах, де простота та вартість є ключовими чинниками.

1.1.2 Дворівнева архітектура

Дворівнева архітектура, також відома як модель клієнт-сервер, дозволяє клієнту та серверу ділити функції. У цій архітектурі додаток на стороні клієнта має пряму комунікацію з базою даних на сервері. Завдяки API, таким як ODBC (Open Database Connectivity) і JDBC (Java Database Connectivity), клієнтські додатки можуть запитувати дані та виконувати транзакції через сервер [1].

Рисунок 1.2 Схема дворівненової архітектури

Обробка запитів і управління транзакціями здійснюється на сервері. Така архітектура дозволяє серверу зосередитися на основних елементах управління даними, тоді як клієнт виконує додаткові програми та інтерфейс користувача. Додаток на стороні клієнта створює зв’язок між клієнтом і сервером, що дозволяє ефективно взаємодіяти з СУБД.

Однією з переваг дворівневої архітектури є її відносна простота у технічному обслуговуванні. Ця модель також добре інтегрується з існуючими системами, тому вона популярна для багатьох додатків. Така архітектура особливо корисна для додатків з обмеженою кількістю користувачів, де не вимагається складне масштабування.

1.1.3 Трирівнева архітектура

Трирівнева архітектура — це більш розширена версія моделі клієнт-сервер, яка додає ще один рівень між клієнтом і сервером бази даних. У цій архітектурі клієнтські програми не звертаються до сервера бази даних безпосередньо. Замість цього вони взаємодіють з проміжним програмним сервером, який відповідає за бізнес-логіку та обробку даних до передавання їх до сервера бази даних [1].

Цей додатковий рівень, який часто називають сервером застосунків, виконує ряд важливих завдань. Він не тільки обробляє бізнес-логіку, що полегшує роботу клієнтських систем, але й допомагає обробляти транзакції та запити до бази даних більш ефективно. Така модель ідеально підходить для великих веб-застосунків, які потребують ефективного розподілу навантаження та оптимізації обробки даних.

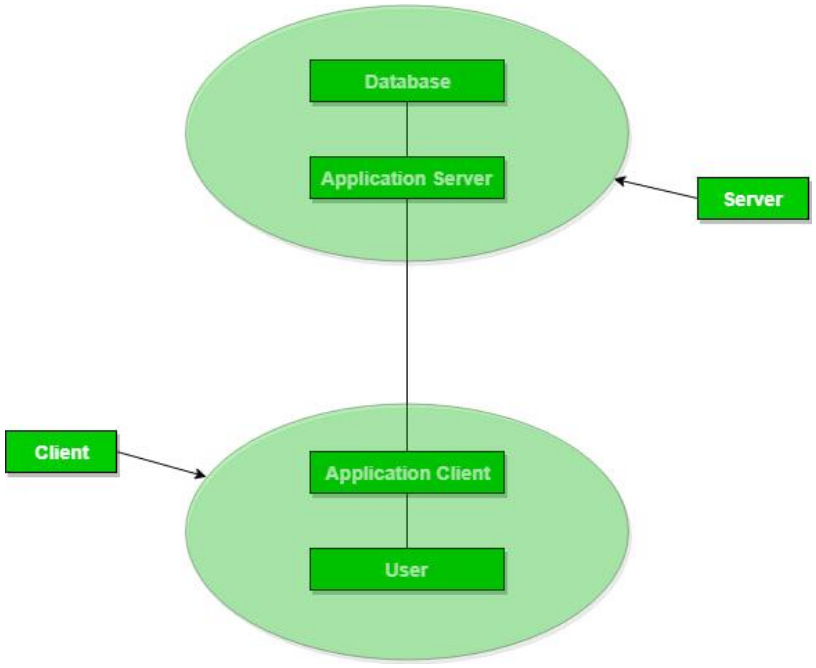


Рисунок 1.3 Схема трьохрівненової архітектури

Однією з основних переваг трирівневої архітектури є її масштабованість. Вона дозволяє збільшувати кількість серверів застосунків для розподілу навантаження без внесення змін до клієнтського інтерфейсу або бази даних, що робить систему більш гнучкою та здатною до розширення. Крім того, ця модель покращує безпеку, оскільки взаємодія між клієнтом і сервером бази даних контролюється та обмежується сервером застосунків, що ускладнює несанкціонований доступ.

Крім того, трирівнева архітектура полегшує технічну підтримку та управління. Розподіл функціоналу між рівнями дозволяє одночасно оновити та оптимізувати окремі частини системи без впливу на решту рівнів. Це важливо для великих компаній, оскільки вимоги до IT-інфраструктури мають швидко змінюватися і потребують гнучкої архітектури, здатної адаптуватися до цих змін.

1.2 Розбір варіантів розміщення

1.2.1 Розміщення на власних ресурсах

Розгортання веб-застосунків на власних ресурсах вимагає використання інфраструктури компанії, щоб керувати серверами, на яких розгортаються ці застосунки. Такий метод дозволяє підприємствам мати повний контроль над своїми даними та системами, що важливо для забезпечення високого рівня безпеки. Власні ресурси також дозволяють гнучко налаштовувати програмне забезпечення та обладнання відповідно до потреб компанії, а стабільність витрат допомагає більш ефективно планувати бюджет і витрати.

Тим не менш, встановлення веб-застосунків на власних ресурсах вимагає значних початкових витрат на серверне обладнання та програмне забезпечення. Обслуговування та оновлення таких інфраструктур також вимагає постійних інвестицій у час, гроші та ресурси. Це включає витрати на електроенергію, охолодження та залучення кваліфікованих працівників. Збільшення потужності серверів часто вимагає додаткових фінансових витрат, що є ще одним недоліком.

В тих випадках, коли для організації важливий високий рівень безпеки та налаштування, використання власних ресурсів може бути оптимальним рішенням, але це вимагає ретельного планування та значних ресурсів для підтримки та управління веб-інфраструктурою.

1.2.2 Розміщення на орендованому дата-центрі

Для компаній, які шукають компроміс між гнучкістю управління ресурсами та контролем над інфраструктурою, розміщення веб-застосунків у орендованому дата-центрі є популярним рішенням. Компанії можуть

використовувати орендовані дата-центри, щоб уникнути значних початкових інвестицій, які зазвичай потрібні для створення власної серверної інфраструктури. Крім того, ці дата-центри гарантують високу надійність і доступність послуг.

Клієнти можуть безперервно використовувати веб-засоби за допомогою обладнання та підтримки, які надаються дата-центром, який вони орендують. Зазвичай такі центри мають сучасні системи безпеки, аварійного живлення та ефективні системи охолодження та резервного копіювання даних. Компанії можуть зосередитися на розробці продуктів, не турбуючись про технічні проблеми утримання серверів.

Однак, при виборі орендованого дата-центру важливо звернути увагу на репутацію провайдера, умови договору та рівень підтримки, який вони пропонують. Компанії повинні також оцінити можливі ризики, пов’язані з залежністю від сторонніх постачальників, особливо в аспектах безпеки даних та їх конфіденційності.

1.2.3 Розміщення на хмарних ресурсах

Розміщення веб-застосунків на хмарних ресурсах стає все більш популярним вибором серед компаній різних масштабів, оскільки цей метод пропонує значну гнучкість, масштабованість та зниження витрат на IT-інфраструктуру. Хмарні платформи, такі як Amazon Web Services, Microsoft Azure та Google Cloud Platform, надають різноманітні сервіси, які можуть бути налаштовані для різних потреб та вимог.

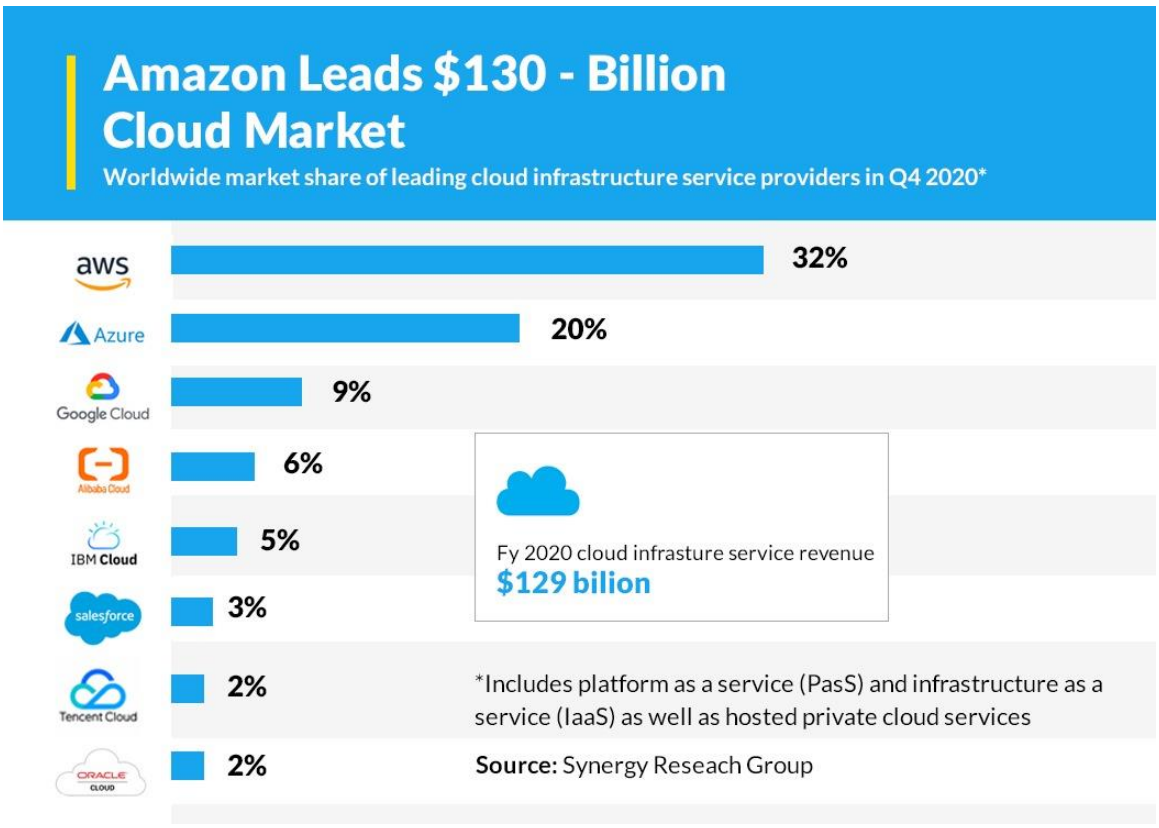


Рисунок 2. Схема розподілення ринку між хмарними платформами

Можливість швидкого масштабування ресурсів є однією з найважливіших переваг хмарного розміщення. Компанії можуть миттєво змінювати або збільшувати кількість ресурсів відповідно до поточного навантаження та потреб, що дозволяє економити гроші та зменшити витрати. Це особливо важливо для компаній, чий трафік сезонний або непередбачуваний.

Крім того, наявність хмарних ресурсів зменшує потребу в значних капіталовкладеннях для створення та підтримки власної IT-інфраструктури. Компанії, зазвичай працюючи за моделлю «pay-as-you-go», платять лише за ресурси, які вони використовують. Малі та середні підприємства, які можуть не мати великого IT-бюджету, можуть отримати доступ до хмарного розміщення завдяки цьому.

Хмарні платформи забезпечують високу надійність і доступність завдяки розподіленим дата-центрам, які гарантують неперервну роботу послуг навіть у

випадку відмови одного з центрів. Це надзвичайно важливо для програм, які вимагають постійного користувача.

Однак хмарне розміщення викликає проблеми, зокрема щодо приватності та безпеки даних, оскільки дані зберігаються та оброблюються на зовнішніх серверах. Крім того, важливо вибрати надійного провайдера хмарних послуг, який може дотримуватися вимог щодо захисту даних і конфіденційності.

1.3 Моделі хмарних послуг

1.3.1 Модель SaaS

Програмне забезпечення як послуга, також відоме як SaaS, є моделлю розподілення програмного забезпечення, в якій додатки хостуються та керуються зовнішнім постачальником на хмарній інфраструктурі, а користувачі отримують доступ до них через Інтернет. Це дозволяє користувачам отримувати доступ до програм і їхніх функцій на основі підписки, уникаючи необхідності інсталяції та технічного обслуговування програм на власних комп'ютерах або серверах [2].

Основною перевагою SaaS є те, що програми можна використовувати з різних пристроїв у будь-якій точці світу, де є інтернет. Платформи SaaS особливо корисні для компаній зі змінним навантаженням, оскільки вони дозволяють швидко змінювати ресурси.

Приклади застосувань SaaS:

- Офісні додатки (Google Workspace, Microsoft Office 365),
- CRM системи (Salesforce),
- ERP системи (SAP Business ByDesign),
- Інструменти автоматизації маркетингу (HubSpot).

1.3.2 Модель PaaS

PaaS, або платформа як послуга, є хмарною моделлю, яка надає розробникам програмного забезпечення широкий набір інструментів і послуг, необхідних для створення, тестування, запуску та управління програмами, без необхідності керувати базовою інфраструктурою. У більшості випадків ці платформи включають інструменти для розробки, бази даних, системи

управління даними, міжпрограмне взаємодія та інші компоненти, які полегшують процес розробки.

PaaS дає розробникам можливість зосередитися на створенні програмного забезпечення, а не витрачати час і ресурси на налаштування та управління серверами, мережами, сховищами даних або іншою інфраструктурою. Це велика перевага PaaS. Через те, що всі необхідні інструменти та сервіси вже інтегровані в платформу, PaaS дозволяє швидко запускати додатки.

1.3.3 Модель IaaS

IaaS, або інфраструктура як послуга, є однією з основних моделей хмарних обчислень, яка дозволяє користувачам орендувати інфраструктуру IT на основі підписки, включаючи сервери, мережі, сховища та інші віртуальні ресурси через інтернет. Це звільняє користувачів від необхідності купувати та управляти фізичним обладнанням, забезпечуючи можливість швидкого нарощування або зменшення використання ресурсів залежно від потреб.

Основними перевагами IaaS є його масштабованість і еластичність, які дозволяють компаніям легко адаптувати свої IT-ресурси до змінних потреб і робочих навантажень. Ця модель покращує ліквідність і гнучкість фінансового планування, перетворюючи капітальні витрати на операційні та зменшуючи необхідність значних початкових інвестицій у IT-інфраструктуру.

Користувачі можуть налаштовувати операційні системи, застосунки та конфігурації безпосередньо на орендованих серверах, що забезпечує високий рівень контролю над інфраструктурою IaaS. IaaS ідеально підходить для розробників і IT-спеціалістів, які потребують ретельної налаштування та управління.

Однак користувачі відповідають за управління операційною системою, базами даних і застосунками, що вимагає більшої технічної грамотності та

управлінських навичок для використання IaaS. Вибір надійного постачальника має вирішальне значення, оскільки від цього залежить стабільність і безпека всієї IT-інфраструктури.

Хостинг веб-сайтів, запуск великих баз даних, розгортання складних додатків і підтримка робочих навантажень з великими обсягами даних — це лише деякі з використання IaaS. Ця модель популярна серед підприємств, які шукають швидку та ефективну роботу, завдяки значній гнучкості та швидкості розгортання.

Найпопулярнішими IaaS платформами є: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), IBM Cloud, Oracle Cloud Infrastructure (OCI) [4].

2. Розробка власного застосунку

Для розробки власного багаторівневого застосунку на хмарній платформія я обрав інтернет-бібліотеку, як варіант функціонального призначення. Хмарною платформою було обрано Amazon Web Services, та зокрема, так сервіси як: Amazon Simple Storage Service (Amazon S3), Identity and Access Management (IAM), AWS Lambda, Amazon CloudFront, Amazon API Gateway, Amazon CloudWatch, Amazon DynamoDB.

Сервер використовує архітектурний підхід REST API, що був реалізований за допомогою бібліотеки Python Boto3 використовуючи AWS Lambda функції. Клієнтська частина додатку була створена за допомогою мови програмування JavaScript та бібліотеки jQuery з використанням HTML і CSS, а також фреймворку Bootstrap. Повідоблення передаються у форматі JSON. Зв’язок між компонентами відбувається через HTTP та внутрішніми налаштуваннями сервісів AWS.

2.1 Структура застосування

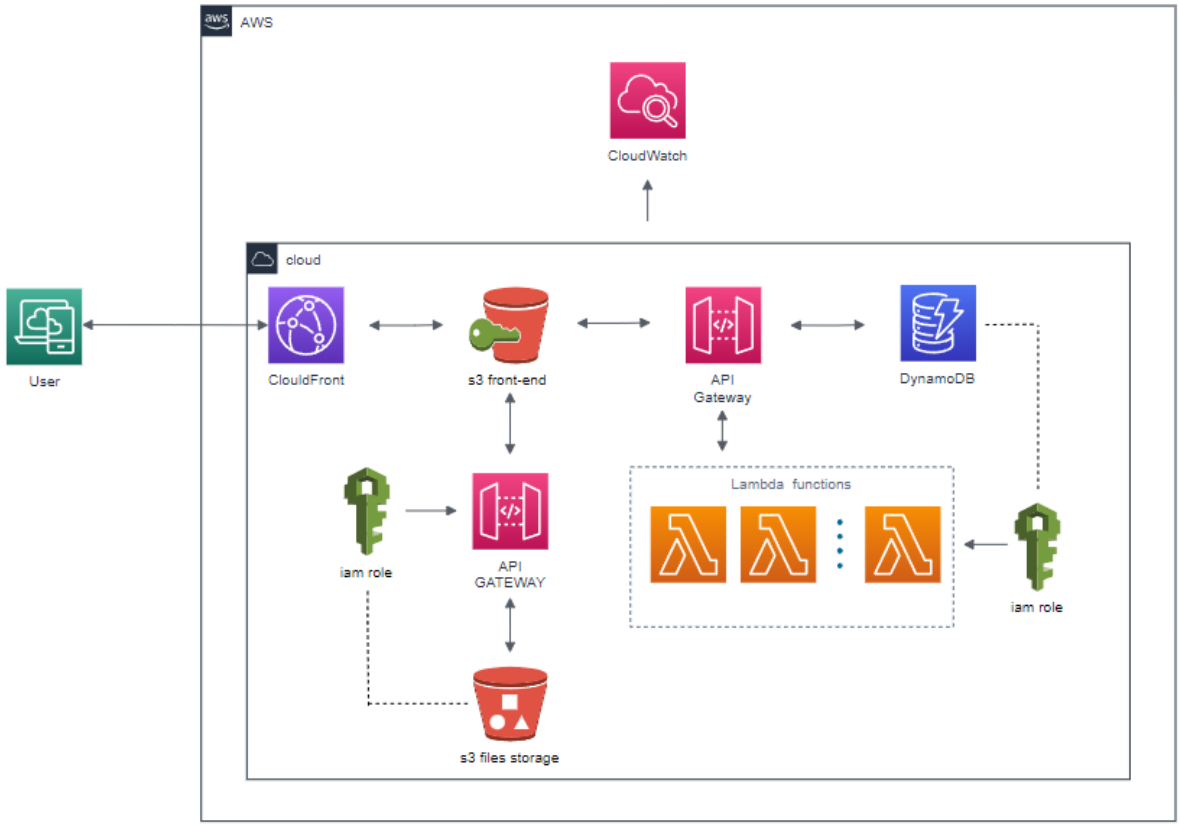


Рисунок 3. Схема структури застосунка

2.2 Опис компонентів

2.2.1 База даних

В рамках розробки хмарного застосунку на платформі Amazon Web Services (AWS) ключовим елементом системи є база даних DynamoDB.Amazon DynamoDB — це повністю керована нереляційна база даних з високою продуктивністю та масштабованістю. Основною перевагою DynamoDB є здатність забезпечувати швидкі та стабільні часи відповіді на запити незалежно від розміру бази даних[5].

У даному застосунку екзепляр DynamoDB LibraryTable використовується для зберігання даних з ключем bookId. Цей ключ унікально ідентифікує кожен запис у базі даних, що спрощує процес пошуку та взаємодії з даними. Використання bookId як ключа дозволяє ефективно реалізувати CRUD-операції (створення, читання, оновлення, видалення) для елементів бази даних.

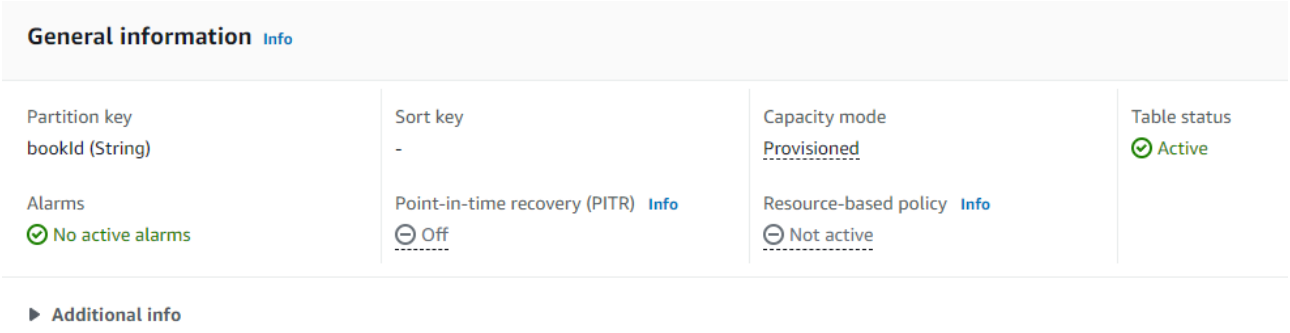


Рисунок 4.1. База даних LibraryTable

2.2.2 Lambda функції

Lambda-функції в архітектурі AWS дозволяють безсерверне виконання коду, що дозволяє виконувати операції CRUD у базі даних DynamoDB, такі як створення, читання, оновлення та видалення. Ці функції відповідають за взаємодію з даними користувачів і є основними обробниками бізнес-логіки

застосунку. Кожна Lambda функція розроблена для виконання конкретних завдань, таких як додавання нових записів, пошук існуючих записів за ключем bookId, оновлення даних або їх видалення. Це полегшує розподіл відповідальності між компонентами застосунку та полегшує управління кодом [6].

Lambda функції мають відповідні IAM ролі, щоб забезпечити безпечний доступ до DynamoDB. Ці ролі дозволяють їм виконувати необхідні операції над базою даних. Принцип найменших привілеїв керує цими ролями IAM, тому кожна функція отримує лише ті дозволи, які їй необхідні для виконання своїх завдань. Це підвищує загальну безпеку застосунку та знижує ймовірність несанкціонованого доступу до даних.

Функції Lambda інтегровані з API Gateway, що спрощує розробку та тестування застосунків і дозволяє ефективно масштабувати обробку запитів і відповідей. У результаті цієї інтеграції коли користувач відправляє запит через фронтенд API Gateway перенаправляє його на відповідну Lambda функцію, яка виконує потрібну операцію в базі даних і повертає результат.

Ця модель взаємодії з базою даних через Lambda функції із відповідними IAM ролями дозволяє застосунку бути ефективним, безпечним та готовим до швидкого масштабування відповідно до потреб користувачів.

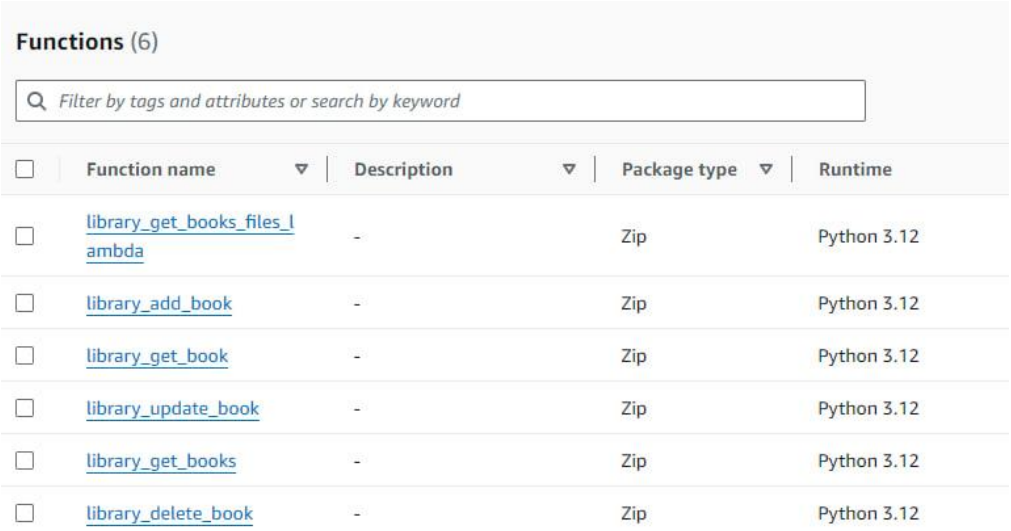


Рисунок 4.2 Lambda функції

2.2.3 Основний API Gateway

В архітектурі AWS API Gateway відіграє важливу роль у визначенні та управлінні REST API для застосування. Ця частина виконує функцію «ворота» і передає всі клієнтські запити до Lambda функцій та інших бекенд-сервісів, зокрема до бази даних DynamoDB. API Gateway полегшує структурування точок доступу (endpoints), що полегшує розробку API, спостереження та управління безпекою [7].

Для сутності Books в API Gateway визначені наступні методи: GET, PATCH, DELETE, та POST. Кожен з цих методів виконує операції з об’єктами book, що зберігається в базі даних BookTable

Формат JSON файлів для цих методів структурований так, щоб забезпечити легке інтегрування з фронтендом і бекендом. Наприклад, для методу POST JSON файл може виглядати так:

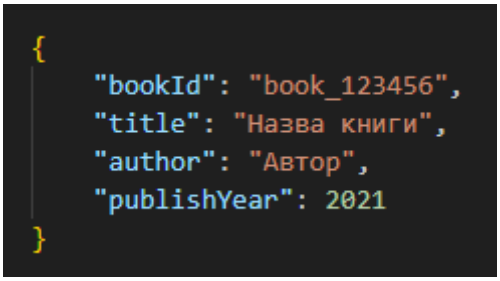


Рисунок 4.3 приклад json-файлу сутності book

2.2.4 S3 фронт-енд

Фронтенд мого хмарного застосунку розміщений як статичний веб-сервіс у S3 бакеті на платформі Amazon Web Services (AWS). Це стандартне рішення

для розгортання статичних веб-сайтів, яке використовує переваги високої доступності та низьких витрат на зберігання, що забезпечує Amazon S3.

Amazon S3 дозволяє розміщувати HTML, CSS, JavaScript та інші компоненти веб-сайту, які не вимагають серверної обробки на боці сервера. Це робить S3 ідеальним для розгортання статичного веб-сайту, де всі необхідні файли завантажуються із бакета S3 напряму в веб-браузер.

Фронтенд застосунку взаємодіє з бекендом через визначений REST API, який управляється за допомогою Amazon API Gateway. Це забезпечує безпечний і ефективний канал для запитів і відповідей між клієнтською частиною, що виконується у веб-браузері користувачів, та серверною логікою, реалізованою через AWS Lambda і DynamoDB для обробки даних.

Коли користувач взаємодіє з веб-інтерфейсом, наприклад, подаючи форму для створення нової книги або запитуючи список книг, фронтенд відправляє HTTP запити до відповідних ендпойнтів API Gateway. Веб-сервіс використовує асинхронні запити, за допомогою AJAX, для взаємодії з REST API, що забезпечує динамічну взаємодію без необхідності перезавантаження сторінки.

2.2.5 S3 files storage

Зберігання файлів S3 для взаємодії з фронтендом налаштоване за допомогою власного API Gateway. Це значно полегшує користувачам завантаження та відвантаження файлів. Ця конфігурація дозволяє безпечно та контролювати доступ до файлів у S3 бакеті.

API Gateway, розроблений для роботи з накопиченням файлів S3, служить основним місцем доступу для всіх операцій з файлами. Він приймає запити від фронтенду та перенаправляє їх до S3, що збільшує безпеку та контроль над доступом до даних

API включає методи для завантаження PUT та відвантаження файлів через посилання до конкретно файлу в бакеті.

Ключова особливість системи полягає в тому, що кожен файл у S3 прив'язаний до конкретного bookId за допомогою унікального ідентифікатора файла, який включає файл у своїй назві. Це дозволяє легко ідентифікувати та асоціювати файл з відповідною книгою. Формат імені файла може бути, наприклад, таким {storage}/{bookId}.pdf. Це спрощує організацію файлів та забезпечує чіткий взаємозв'язок між записами в базі даних та відповідними файлами.

API Gateway використовує відповідну IAM роль для забезпечення можливості надсилання файлу до сховища.

2.2.5 CloudFront

Amazon CloudFront взаємодіє з користувачем і статичним S3 фронтендом, виступаючи як мережа доставки контенту (CDN). Це дозволяє значно підвищити швидкість завантаження веб-сторінок та зменшити затримки для користувача.

CloudFront включає функції безпеки, такі як SSL/TLS для шифрування даних користувачів під час передачі, а також інтеграцію з AWS Shield для захисту від DDoS-атак. Це забезпечує додатковий рівень захисту для вашого статичного [8].

2.2.6 CloudWatch

Amazon CloudWatch — це сервіс моніторингу AWS, який надає повний огляд метрик і логів з усіх компонентів застосунка. Через функції алармів, візуалізації та аналізу логів він дозволяє відстежувати роботу, виявляти

проблеми та оптимізувати витрати. Крім того, CloudWatch допомагає підтримувати стабільність та ефективність застосунку, надаючи автоматичне сповіщення про критичні події [9].

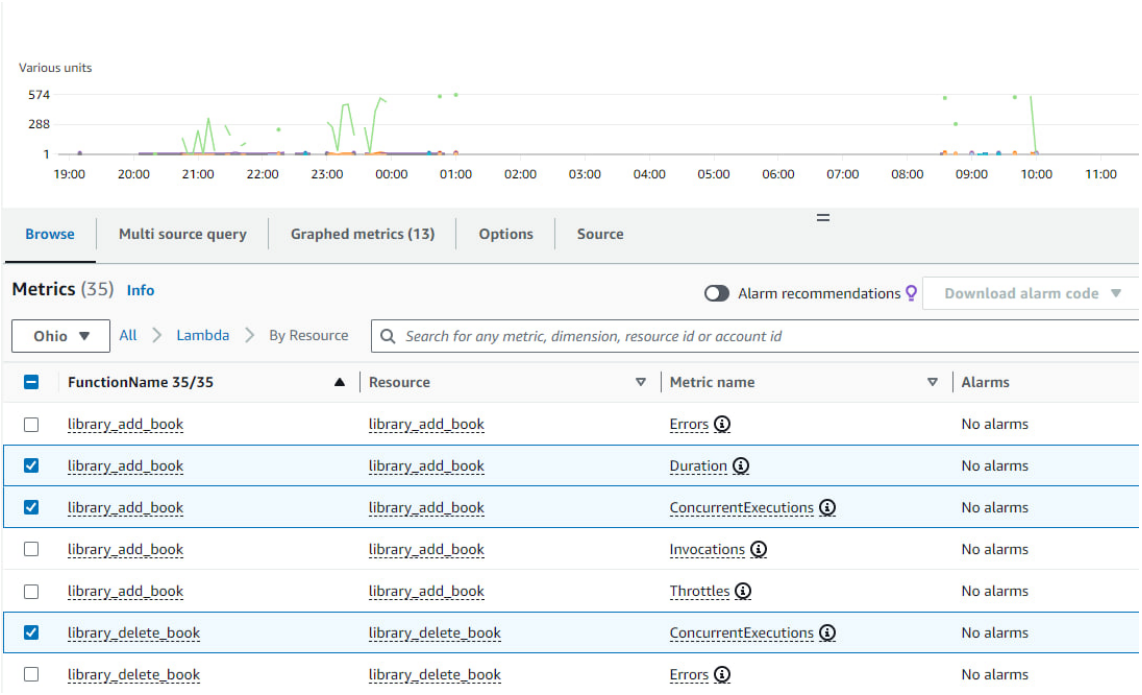


Рисунок 4.4 Приклад віззуалізації даних CloudFront

2.3 Детальна розробка

2.3.1 Детальний розгляд API

Основне API для бази даних

Метод	Ресурс	Параметри	Опис
GET	/books		Повертає повний спик книг
GET	/books	book_id	Повертає обрану книигу із списку
POST	/books	book_id, title, author, publication_year	Публікує книгу на сервері
DELETE	/books	book_id	Видаляє книгу з сервера
PATCH	/books	book_id, інші парамерти за необхідністю	Змінює вміст екземпляра book

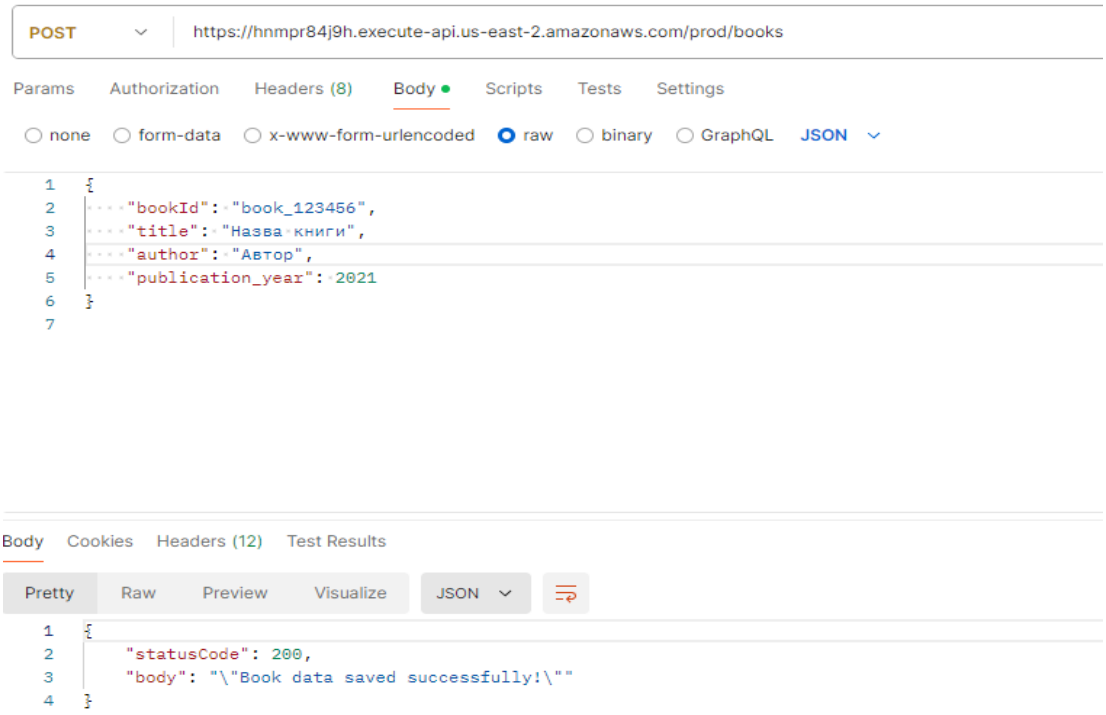


Рисунок 5. приклад використання DB API

S3 Storage book files API

Метод	Ресурс	Параметри	Опис
PUT	/{bucket}/{filename}	Binary file	Завантаження файлу до сховища
GET	/{bucket}/{filename}		Посилання на об’єкт в сховищі

2.3.2 Розбір фронт-енд

Використання статичних HTML-сторінок разом із JavaScript та jQuery для динамічної взаємодії з бекендом через REST API становить основу фронтенд застосунку. Кожна сторінка має JavaScript файл, який використовується для обробки подій і інтерактивних елементів.

```
<!-- Підключення Bootstrap JS та jQuery (необхідно для попередження деяких Bootstrap функцій) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
<!-- Підключення JavaScript скрипту -->
<script src="adminscript.js"></script>
<script src="searchInputscript.js"></script>
```

Рисунок 6. Приклад підключення компонент

1. Основна сторінка

- index.html: Головна сторінка, яка відображає список книг засобами таблиці. Користувачі можуть шукати книги за назвою, а також перейти на сторінку адміністратора.
- indexscript.js: Завантажує список книг із бекенду при завантаженні сторінки та динамічно додає їх до таблиці. Кожна книга має кнопку для перегляду файлу, який зберігається на S3.

2. Сторінка додавання книги

- bookAdd.html: Форма для введення даних нової книги (назва, автор, рік публікації) та завантаження файлу книги.
- bookAddscript.js: Обробляє відправлення форми, створює новий запис книги в базі даних через API та завантажує файл у S3.

3. Сторінка оновлення книги

- updateBook.html: Форма, що дозволяє змінити дані книги (назва, автор, рік публікації) та завантажити новий файл.
- updateBookscript.js: Заповнює поля форми даними, отриманими через URL параметри. Обробляє відправлення форми для оновлення даних книги та файлу через PATCH запити.

4. Сторінка адміністратора

- admin.html: Відображає список книг з можливістю додавання нових, редагування існуючих та видалення книг.
- adminscript.js: Завантажує список книг із сервера при завантаженні сторінки та дозволяє видалення книг за допомогою запитів DELETE.

5. Сценарії обробки пошуку

- searchInputscript.js: Інтегрований у різні сторінки для додавання функціональності пошуку книг у таблиці за назвою.

Візуальна частина являє собою прости та інтуїтивнозрозумілиий інтерфейс для користувача (Додаток А).

Висновки

У даній курсовій роботі було проведено розгляд процесу розробки багаторівневого веб-застосунку для онлайн-бібліотеки на хмарній платформі AWS, що включало аналіз різних архітектурних рішень та моделей розміщення ресурсів.

Значну увагу у роботі було приділено вибору розміщення ресурсів в хмарі, яке забезпечує максимальну еластичність і зниження витрат порівняно з власними ресурсами чи орендою дата-центрів. Розгянуто відміності моделей хмарних послуг, таких як IaaS, PaaS та SaaS та їх залежності від інфраструктури програмних рішень.

Процес розробки власного застосунку з активним використанням AWS сервісів, включно з Lambda функціями, S3 для фронт-енду і зберігання файлів, а також CloudFront для доставки контенту, показав високу продуктивність та безпеку додатку. Детальне вивчення API та оптимізація фронт-енду забезпечили створення привабливого і функціонального інтерфейсу.

Завершуючи, можна стверджувати, що використання хмарних технологій значно покращує розробку та впровадження сучасних веб-застосунків. Приклад онлайн-бібліотеки наочно демонструє, як ефективно можна використовувати різні хмарні сервіси для створення надійних, доступних та масштабованих рішень у веб-розробці.

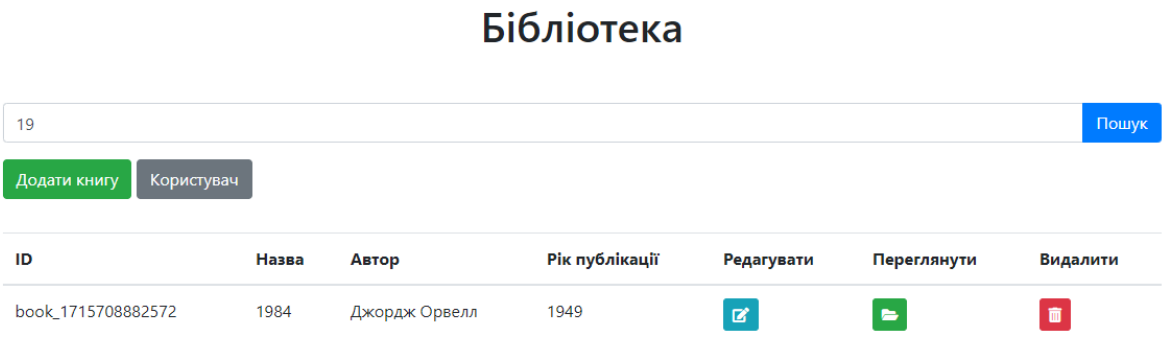
Список використаної літератури

1. DBMS Architecture 1-level, 2-Level, 3-Level [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/dbms-architecture-2-level-3-level/
2. What is SaaS? Everything You Need To Know [Електронний ресурс] – Режим доступу: https://www.softwareadvice.com/resources/saas-faqs-software-service/
3. What is PaaS? [Електронний ресурс] – Режим доступу: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas
4. Best Infrastructure as a Service (IaaS) Providers [Електронний ресурс] – Режим доступу: https://www.g2.com/categories/infrastructure-as-a-service-iaas
5. What isAmazonDynamoDB? [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
6. What is AWS Lambda? [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
7. What is Amazon API Gateway?[Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html
8. What is Amazon CloudFront?[Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html
9. What is Amazon CloudWatch?[Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html
10. jQuery API [Електронний ресурс] – Режим доступу: https://api.jquery.com

Додаток А

(Обов’язковий)

Ілюстрації UI застосунку



Додати книгу

Назва:

Автор:

Рік публікації:

Upload book:

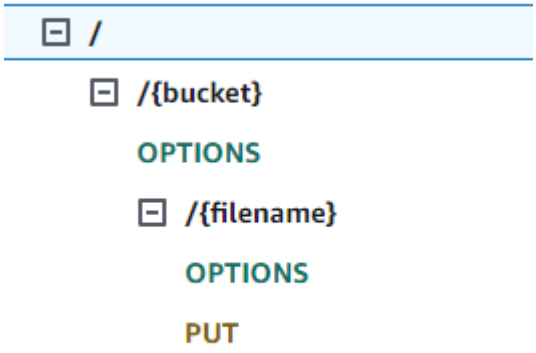
Выберите файл Файл не выбран

Зберегти книгу Назад

Додаток Б

(Обов’язковий)

library_book_files_rest_api



Додаток В

(Обов’язковий)

library_rest_api_to_db