

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ЗАКЛАД
«ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Навчально-науковий інститут математики
та інформаційних технологій

Кафедра інформаційних технологій та систем

Сторчеус Роман Владиславович

ТЕХНОЛОГІЇ РОЗРОБКИ ЗАХИЩЕНИХ REST API ДОДАТКІВ

**кваліфікаційна робота
здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Мультимедійні системи»
за спеціальністю 121 „Інженерія програмного забезпечення”**

Особистий підпис _____ Роман СТОРЧЕУС

Науковий керівник _____ Світлана ПЕРЕЯСЛАВСЬКА,
кандидат педагогічних наук, доцент ка-
федри інформаційних технологій та сис-
тем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук,
доцент кафедри інформаційних техноло-
гій та систем

АНОТАЦІЯ

Сторчеус Р.В.

Тема: Технології розробки захищених REST API додатків.

Спеціальність: 121 „Інженерія програмного забезпечення”

Установа: ДЗ ЛНУ імені Т. Шевченка, 2025 р.

Магістерська робота містить: 79 с., 36 рис., 2 табл., 2 додат., 40 джерел.

Об’єкт дослідження – технології розробки захищених REST API додатків.

Предмет дослідження – технології хмарного API Management.

Мета роботи — дослідження технологій хмарного API Management для розробки захищеного REST API додатку.

Методи дослідження. *Загальнонаукові*: аналіз вразливостей REST API, аналіз та синтез технологій розробки захищених REST API додатків, аналіз технологій хмарного API Management, моделювання REST API додатків; *методи інформаційного пошуку*: бібліографічний пошук та аналіз наукових та спеціалізованих джерел інформації; *експериментальні*: тестування розробленого додатку.

Результат роботи. Досліджено вразливості та технології захисту REST API. Проаналізовано ключові особливості технологій хмарного API Management. Розроблено захищений REST API додаток з використанням хмарного API Management.

Ключові слова: API, REST, безпека, API Management, API Gateway, хмарні технології, AWS.

ABSTRACT

Storcheus R.V.

Subject: Technologies for developing secure REST API applications.

Spetsialty: 121 “Software Engineering”

Institution: Taras Shevchenko National University of Luhansk, 2025.

Qualification work contains: 79 pages, 36 figures, 2 tab., 2 applic., 40 sources.

Object of research technologies for developing secure REST API applications.

Subject of research: cloud API Management technologies.

Purpose of research: researching cloud API Management technologies for developing a secure REST API application.

Methods of reseatch. General scientific: analysis of REST API vulnerabilities, analysis and synthesis of technologies for developing secure REST API applications, analysis of cloud API Management technologies, modeling of REST API applications; methods of information retrieval: bibliographic search and analysis of scientific and specialized sources of information; experimental: testing of the developed application.

Work results. Vulnerabilities and REST API security technologies are investigated. The key features of cloud API Management technologies are analyzed. A secure REST API application using cloud API Management is developed.

Keywords: API, REST, security, API Management, API Gateway, cloud computing, AWS.

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. REST API: ПОНЯТТЯ, ПРИНЦИПИ ТА ВРАЗЛИВОСТІ.....	8
1.1. Визначення та принцип роботи API.....	8
1.1.1. Класифікація API за використанням	10
1.1.2. Переваги API	13
1.2. Визначення та принципи використання REST .. Ошибка! Закладка не определена.	
1.3. API Economy	18
1.4. Вразливості REST API.....	21
1.5. Безпека REST.....	24
РОЗДІЛ 2. ТЕХНОЛОГІЇ ДЛЯ СТВОРЕННЯ БЕЗПЕЧНИХ REST API	
ДОДАТКІВ	28
2.1. API Management.....	28
2.2. API Gateway	30
2.3. Хмарні веб-сервіси від Amazon	32
2.3.1. Amazon API Gateway.....	33
2.3.2. Amazon CloudWatch	36
2.3.3. AWS Identity and Access Management	39
2.3.4. AWS CloudTrail	42
2.3.5. Amazon Inspector	44
2.3.6. Amazon Cognito.....	46
Висновки до розділу 2.....	49
РОЗДІЛ 3. ВИКОРИСТАННЯ ХМАРНИХ ТЕХНОЛОГІЙ ДЛЯ ЗАХИСТУ	
REST API ДОДАТКУ	50
3.1. Розробка REST API додатку.....	50
3.2. Розміщення додатку в хмарному середовищі AWS	57
3.3. Використання хмарних сервісів AWS для безпеки	61
3.4. Тестування додатку на вразливості	66
Висновки до розділу 3.....	73
ВИСНОВКИ	74
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	76

Додатки.....	80
--------------	----

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- API (Application programming interface) - прикладний програмний інтерфейс
- REST (Representational State Transfer) - передача репрезентативного стану
- JDBC (Java Database Connectivity) - з'єднання з базами даних на Java
- SQL (Structured Query Language) - мова структурованих запитів
- ОС – операційна система
- ПЗ – програмне забезпечення
- SaaS (Software as a service) - програма як послуга
- HTTP (Hypertext Transfer Protocol) - протокол передачі даних
- HTTPS (HyperText Transfer Protocol Secure) - протокол передачі даних з шифруванням
- JSON (JavaScript Object Notation) - текстовий формат обміну даними
- XML (EXtensible Markup Language) - мова розмітки
- SSL (Secure Sockets Layer) - рівень захищених сокетів, криптографічний протокол
- TLS (Transport Layer Security) - захист на транспортному рівні, криптографічний протокол
- SOAP (Simple Object Access Protocol) - простий протокол доступу до об'єктів
- Amazon EC2 (Elastic Compute Cloud) - веб-сервіс, котрий надає обчислювальні потужності в хмарі
- Amazon RDS (Relational Database Service) — веб-сервіс реляційних баз даних
- Amazon S3 (Simple Storage Service) — веб-сервісів для зберігання даних
- IaaS (Infrastructure as a service) - Інфраструктура як послуга

ВСТУП

У сучасному цифровому ландшафті архітектурний стиль REST API став найпопулярнішим для розробки масштабованих і надійних додатків. Незалежно від того, чи створюється мобільний додаток, який отримує дані з хмарного сервера, чи платформу корпоративного рівня, яка інтегрується з сторонніми сервісами, REST API слугує критично важливим каналом комунікації.

Однак, з широким розповсюдженням зростає і ризик загроз безпеці. Захист додатку REST API - це не просто рекомендована практика, це абсолютна необхідність. Конфіденційні дані, включаючи особисту інформацію, фінансові дані та бізнес-логіку, часто передаються через кінцеві точки. Порушення в будь-якій з цих кінцевих точок може зробити дані доступними для зловмисників, що призведе до втрати репутації, штрафів з боку регуляторних органів, а також потенційного простою або втрати бізнесу. Отже, надійність продукту або послуги залежить від того, наскільки ефективним є захист REST API додатку.

В той же час для комплексного забезпечення захисту використовуються багато технологій. Основні загрози безпеці API систематизувала онлайн-спільнота OWASP, яка склала аналізує реальні дані та співпрацюючи з фахівцями з безпеки з усього світу. Ця спільна робота гарантує, що список відображає найактуальніші та найсучасніші вразливості. Організації можуть використовувати його як відправну точку для того, щоб залишатися в курсі подій і визначати пріоритети своїх ініціатив з безпеки.

Одним із ефективних рішень в сфері захисту REST API є хмарні сервіси. Такі компанії, як Amazon, Google, Microsoft, IBM та інші пропонують цілісний набір інструментів, процесів і політик, що дозволяють організаціям ефективно розробляти, публікувати, захищати, відстежувати та оптимізувати API.

Такий підхід для комплексного керування API отримав назву API

Management. В основі надійної стратегії API Management є API Gateway - центральний об'єднуючий компонент, який спрощує зв'язок між службами та додатками, призначеними для користувача.

Таким чином, тема дослідження є сучасною та актуальною

Об'єкт дослідження – технології розробки захищених REST API додатків.

Предмет дослідження – технології хмарного API Management.

Мета роботи - дослідження технологій хмарного API Management для розробки захищеного REST API додатку.

Відповідно до мети дослідження було визначено наступні **завдання**:

- Дослідження архітектури REST API.
- Визначення вразливостей REST API.
- Аналіз технологій створення безпечних REST API: API Management, API Gateway.
- Дослідження можливостей сервісів Amazon для створення захищених REST API.
- Розробка захищеного

Методи дослідження. *Загальнонаукові*: аналіз вразливостей REST API, аналіз та синтез технологій розробки захищених REST API додатків, аналіз технологій хмарного API Management, моделювання REST API додатків; методи інформаційного пошуку: бібліографічний пошук та аналіз наукових та спеціалізованих джерел інформації; *експериментальні*: тестування розробленого додатку.

Результат роботи. Досліджено найактуальніші вразливості та технології захисту REST API за допомогою хмарних технологій. Розроблено захищений REST API додаток з використанням хмарного API Management.

РОЗДІЛ 1. REST API: ПОНЯТТЯ, ПРИНЦИПИ ТА ВРАЗЛИВОСТІ

1.1. Визначення та принцип роботи API

API є основним засобом для обміну інформації між програмним забезпеченням. В програмних додатках обмін даними відбувається через API: комп'ютерні програми, застосунки на смартфонах, сервери бази даних, операційні системи, програмний фреймворк і т.д. Незалежно від складності або масштабу додатку - взаємодія між ними реалізується саме через програмні інтерфейси[1].

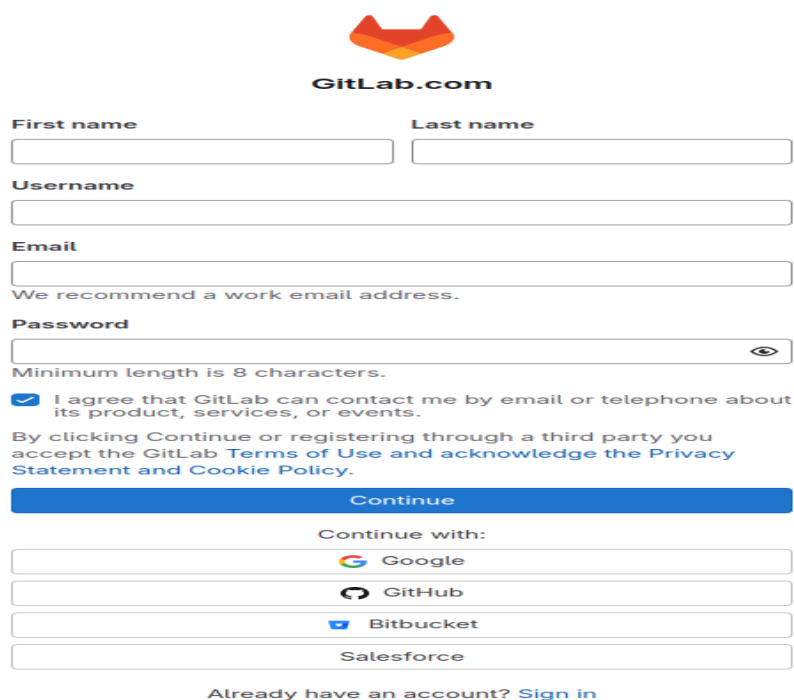
Обмін інформацією через API дозволяє приховати деталі реалізації програмного додатку і надавати конкретну інформацію по запиту клієнта. Клієнт не має необхідності знати специфічну логіку програмного модуля або сервіса, йому необхідно лише відправити запит з параметрами або без, щоб отримати необхідну відповідь від сервера.


Використання API прискорює розробку програмного забезпечення, шляхом використання вже існуючих програмних інтерфейсів. Немає необхідності створювати функціонал спочатку, можна інтегрувати вже готове API. Саме через API можна зробити доступ до необхідних функцій і даних, як в середині компанії для власного використання, так і для партнерів або взагалі зробити API публічним[2].

Згідно стандартного визначення API не є програмним продуктом, а лише інтерфейсом через котрий відбувається передача даних. Тому API не є самостійним або кінцевим результатом, воно лише надає можливість користування програмним забезпеченням більш зрозуміло. В той же час, часто під API мають на увазі програмний продукт разом з його прикладним інтерфейсом. Але з технічної точки зору, це програмний продукт надає API для взаємодії з іншими сервісами. Можна сказати, що API є мостом або посередником між двома або більше програмами.

Взаємодія через API можна описати, як запит від клієнта та відповідь від сервера. Додаток є клієнтом, що створює запит до сервера, в свою чергу сервер надає відповідь згідно запитом. API забезпечує обмін інформацією між клієнтом і сервером за допомогою чітко визначених правил[3].

Для кращого розуміння роботи API розглянемо реєстрацію на сервісі GitLab, що є сервісом для управління репозиторіями програмного коду, популярного серед розробників програмного забезпечення. GitLab надає можливість звичайної реєстрації через електронну пошту для котрої необхідно заповнити 5 полів, створити пароль та підтвердити реєстрацію через лист, котрий клієнт отримає на електронну адресу. Також зробити реєстрацію можна через сторонні сервіси, такі як Google, GitHub, Bitbucket, Salesforce, на рисунку 1.1 зображена форма реєстрації GitLab.




GitLab.com

First name

Last name

Username

Email

We recommend a work email address.

Password


Minimum length is 8 characters.


☒ I agree that GitLab can contact me by email or telephone about its product, services, or events.


By clicking Continue or registering through a third party you accept the [GitLab Terms of Use](#) and acknowledge the [Privacy Statement](#) and [Cookie Policy](#).

Continue

Continue with:







Already have an account? [Sign in](#)

Рис. 1.1. Реєстрація на сайті GitLab

Реєстрація через сторонні сервіси використовує функції API цих сервісів для передачі інформації про користувача в GitLab. Можна вибрати реєстрацію через Google[4], в цьому випадку GitLab виступає клієнтом, а Google – сервером з необхідною інформацією про користувача. GitLab робить

переадресацію на Google, щоб користувач підтвердив надання інформації від Google до GitLab – це є API-запит. Користувач підтверджує, що згоден щоб Google надав його облікову дані GitLab, проілюстровано на рисунок 1.2.

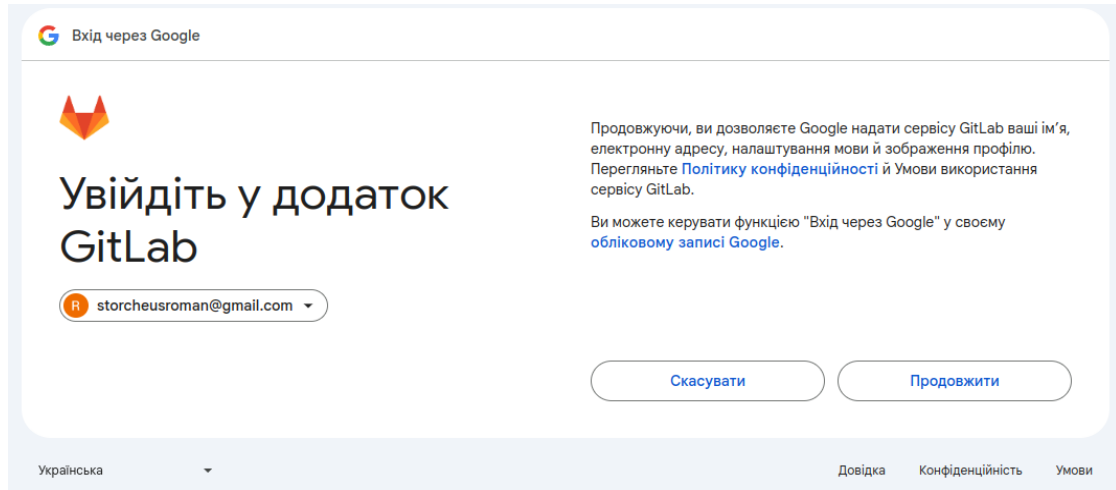


Рис. 1.2. Реєстрація за допомогою Google на сайті GitLab

Після підтвердження користувача переадресовую на GitLab вже зареєстрованого, зі своїм профілем в GitLab – це відповідь від сервера. Тобто вся реєстрація через Google, за допомогою стороннього API, вимагала від користувача два натискання на кнопки. Користувач не вводив жодних даних про себе, він лише надав дозвіл GitLab, використати його облікові дані з Google, вся взаємодія між GitLab та Google відбувалася за допомогою API-запитів та API-відповіді. Це спростило реєстрацію користувачу, за допомогою вже існуючого облікового запису в Google.

1.1.1. Класифікація API за використанням

API поділяються на наступні 4 категорії[5].

API даних (або бази даних). Ці API надають можливість взаємодіями між застосунком та базами даних, забезпечуючи отримання, вставку, оновлення, видалення даних та підтримка транзакцій. Призначення такого API є полегшення зв'язку між додатком і системою управління базами даних. Використовується при розробки додатків, для доступу до систем звітності та

аналітики, управління контентом. Для прикладу можна привести, який дозволяє застосункам виконувати SQL-запити до бази даних. Для не SQL баз даних, такі як MongoDB або Apache CouchDB, надають методи для взаємодії з документами, колекціями або сховищами ключ-значення.

API операційної системи (локальні). Такі API забезпечують інтерфейси для взаємодії з ресурсами та сервісами операційної системи. Дозволяють додаткам використовувати ресурси системи, такі як файли, апаратне забезпечення, пам'ять і процеси. Як приклад можна привести Windows API, що дозволяє програмам Windows керувати файлами, відображати елементи інтерфейсу користувача або взаємодіяти з апаратними пристроями. POSIX API, також відноситься до API операційної системи, що використовується в Unix-подібних системах для управління файловою системою, контролю процесів і міжпроцесної взаємодії. Характерним для цього API є розробка настільного програмного забезпечення або програмного забезпечення на рівні системи. Доступ до апаратного забезпечення (наприклад, принтерів, камер). Управління файловими системами або процесами. При цьому в такому API використовується високорівневі абстракції для системних сервісів, прямий доступ до функцій ОС.

Віддалені API. Віддалені API забезпечують зв'язок між програмними системами, які знаходяться на різних серверах або в мережі Інтернет. Забезпечують віддалений доступ до сервісів або ресурсів, розміщених на іншій системі. Використовуються для взаємодії з хмарними сервісами, доступ до ресурсів у розподіленій системі, а також для реалізації архітектури мікросервісів. Серед особливостей такого API можна виділити мережева комунікація, серіалізація даних та підтримка розподілених систем. Прикладом віддалених API є SOAP з використанням XML та gRPC.

Веб API. Веб API спеціально розроблені для доступу через веб, використовуючи стандартні протоколи, такі як HTTP/HTTPS. Дозволяють взаємодіяти з веб-сервісами та ресурсами через мережу Інтернет. Використовуються для Інтеграція сторонніх сервісів, для доступ до платформ

SaaS або веб-додатків. Особливостями цього API є безстанова комунікація, стандартизація запитів та відповідей від сервера, масштабованість та проста інтеграція. Прикладами даного API є REST - використовують методи HTTP, такі як GET, POST, PUT і DELETE, для взаємодії з ресурсами, ідентифікованими URL

Сьогодні більшість API є веб API. Веб API — це тип віддалених API (тобто API використовують правила та обмеження для роботи із зовнішніми ресурсами), які надають доступ до додатку через мережу інтернет за допомогою протоколу HTTP. В залежності від доступності, є наступні типи веб API, які часто використовуються у веб-додатках[6]:

- Відкриті API - є вільно доступними без обмежень у доступі. Це означає, що публічний API можуть використовувати як розробники в межах компанії, яка випустила API, так і зовнішні розробники, які реєструються для отримання доступу до інтерфейсу.
- Публічні API - схожі на відкриті API, але зазвичай вони доступні за плату. Це також означає, що вони зазвичай включають певну форму автентифікації або ключа дозволу для відстеження використання. Однак не всі публічні API продаються. Деякі з них можуть функціонувати як внутрішні API у межах ширшої програмної системи.
- Внутрішні API - також називаються приватними API. Внутрішні API створені в основному для внутрішнього використання в межах організації. Їх легко створювати, і вони не потребують такої високої надійності, як ті API, які призначені для публічного використання.
- Партнерські API - обмежують доступ до системи, дозволяючи його використовувати лише певним сторонам. Вони можуть бути як безкоштовними, так і платними. Через те, що партнерські API доступні лише для певних партнерів, вони зазвичай мають більш суворі політики авторизації, автентифікації та безпеки. Надання доступу лише кільком партнерам дозволяє організаціям підтримувати

API максимально безпечним. Це є гарним компромісом між приватністю та публічністю.

- Комбіновані API - об'єднують кілька API даних або сервісів. Вони дозволяють розробникам отримувати доступ до кількох кінцевих точок за допомогою одного запиту. Комбіновані API є корисними у мікросервісній архітектурі, наприклад, коли для виконання однієї задачі потрібно отримати інформацію від багатьох сервісів.

1.1.2. Переваги API

API допомагає спростити проектування та розробку нових додатків та сервісів, інтеграція та управління вже існуючих систем теж спрощується. Вони приносять переваги для компаній та розробників в цілому.

- **Краща взаємодія.** API забезпечує кращу інтеграцію, дозволяючи сервісам взаємодіяти незалежно від технології на яких вони створенні. Завдяки цій інтеграції організації можуть автоматизувати робочі процеси та покращити роботу команд.
- **Гнучкість та інновації.** API надають можливість для розробників і компаній створювати нові сервіси, розширюючи власний функціонал інтегрувавши сторонні сервіси. Такий підхід дозволяє бізнесу налагоджувати зв'язки з новими партнерами та залучати більше клієнтів. Ця гнучкість також дає можливість компаніям надавати нові та комплексні рішення. Розроблений API можна використовувати на різних платформах (мобільні, настільні, веб), не переймаючись проблемами синхронізації даних клієнтів між цими платформами.
- **Монетизація даних.** Деякі організації пропонувати API безкоштовно або лімітовану кількість запитів, щоб залучити розробників та компанії створити свої власні проекти з використанням стороннього API. Прикладом такого підходу є проект Опендатабот, що забезпечує доступ до інформації з державних реєстрів за платною підпискою[7].

- Безпека системи. API, при правильному проектуванні та реалізації забезпечує надійну взаємодію між сервісами використовуючи автентифікацію, шифрування, перевірку входних параметрів, обмежена кількість запитів, інформативні повідомлення в разі помилок. Також API забезпечує контроль, яка інформація буде передаватися клієнту.
- Конфіденційність. API забезпечують додатковий рівень захисту для персональних даних користувачів. Наприклад, мобільний застосунок або веб-сайт має механізм дозволів на камеру та місцезнаходження користувача. Клієнт може дозволити або відхилити цей запит на отримання його місцезнаходження.

1.2. Визначення та принципи використання REST

Representational State Transfer (REST) перекладається, як передача репрезентативного стану - є архітектурним стилем для надання доступу к ресурсу. Ресурс це базовий елемент цього архітектурного підходу. Вперше архітектурний стиль REST API запропонував Рой Філдінг у 2000 році в своїй дисертації «Архітектурні стилі і проектування мережових архітектур програмного забезпечення». Мета роботи Філдінга була запропонувати такий підхід при розробці API, щоб він був масштабований, мав можливість незалежного розгортання програмних компонентів, універсальність інтерфейсів зручний для розробників та уніфікований спосіб доступу до ресурсу[8].

Варто ще раз наголосити, що REST є архітектурним підходом, тому не залежить від мови програмування або формату передачі даних. Головна вимога – дотримання принципів REST, відомих як архітектурні обмеження.

- Безстанність. Кожен запит від клієнта до сервера має містити всю необхідну інформацію для його обробки. Сервер не зберігає жодного стану сесії, що забезпечує самодостатність кожної взаємодії. Це спрощує масштабування та підвищує надійність системи.

- Парадигма Клієнт-Сервер. В REST є чітке розділення на клієнта, сторона що робить запит та сервера, що надає доступ до даних. Клієнт і сервер цілковито ізольовані один від одного. Сервер лише надає інформацію клієнту за запитом. Клієнт та сервер повністю незалежні один від одного.

- Уніфікований інтерфейс. REST забезпечує послідовний і уніфікований інтерфейс для взаємодії з ресурсами. Така стандартизація знижує складність, полегшуючи інтеграцію та покращуючи досвід розробників.

- Кешування. Відповіді від сервера явно позначаються як кешовані або некешовані, що оптимізує ефективність мережі. Належні механізми кешування зменшують кількість повторних запитів до сервера, покращуючи продуктивність і масштабованість.

- Шарова система. REST-архітектура підтримує проектування шарової системи, дозволяючи використовувати посередників (наприклад, балансувальники навантаження, проксі-сервери) між клієнтами та серверами. Ці шари підвищують безпеку, масштабованість і керованість без впливу на загальну поведінку API.

- Код на вимогу (опційно): REST дозволяє серверам розширювати функціональність клієнта, надсилаючи виконуваний код, такий як JavaScript. Хоча це і не є обов'язковим, цей принцип додає гнучкості та розширює можливості клієнта.

Також для підтримки уніфікованого інтерфейсу є наступні обмеження:

- Унікальний ідентифікатор ресурсу (URI) - це обов'язкова умова для того, щоб сервер однозначно міг визначити, яку відповідь очікує клієнт. Ці URI забезпечують незалежність ресурсів від конкретних форматів, які використовуються для їх представлення при відповіді для клієнта.

- За запитом клієнта, сервер, котрий має інформацію про ресурс, надає відповідь як представлення. В той же час внутрішній метод зберігання даних на сервері (СУБД або файли) не повинен залежати від формату представлення

- Клієнтські запити мають бути зрозумілими та передавати всю інформацію, що сервер міг правильно обробити запит.
- Клієнт повинен мати можливість отримати всі існуючі ресурси та методи через гіперпосилання.

REST API є основою багатьох додатків, що сприяє безперебійному зв'язку між різноманітними ПЗ. Вони широко застосовуються технологічними гігантами, стартапами та розробниками через їхню простоту, масштабованість і здатність працювати через протоколи HTTP. REST є найпоширенішим архітектурним стилем для побудови API, котрий використовують 93,4% розробників[21], тому його поширеність у цифровій екосистемі незаперечна. Для прикладу, може слугувати компанія Google, котра надає багато REST API для своїх служб, таких як Google Maps, YouTube і Google Drive. Розробники використовують ці API для виконання завдань, починаючи від вбудовування інтерактивних карт до доступу до хмарного сховища. Портал розробників Google пропонує вичерпну документацію та інструменти.

Принципи використання REST

REST використовує стандартні HTTP-методи для маніпуляції над ресурсами:

- GET: Отримання ресурсу.
- POST: Створення нового ресурсу.
- PUT: Оновлення існуючого ресурсу.
- DELETE: Видалення ресурсу.

Представлення ресурсу – це стан ресурсу в конкретний проміжок часу. Така інформація передається клієнту в зручному йому форматі JSON, XML, інший формат.

При запиті клієнт також вказує заголовки та параметри, вони можуть мати інформацію про токен авторизації, формат для відповіді, чи повинен запит бути кешованим та інші дані.

Також в добре спроектованому REST API відповідь від сервера має код стану HTTP, для розуміння, чи успішно відпрацював запит чи має помилки.

Поширені коди стану в протоколі HTTP:

- 200 OK: Запит успішно виконано.
 - 201 Created: Ресурс успішно створено.
 - 400 Bad Request: Сервер не може зрозуміти запит через некоректний ввід.
 - 404 Not Found: Запитуваний ресурс не існує.
- rror: Виникла помилка на сервері.

Приклад використання REST API можна продемонструвати зробивши запит для отримання курсу валют від компанії APILayer.

Ресурс: <https://api.apilayer.com/fixer>.

Метод: GET.

Заголовок: `apiKey` — це ключ, котрий клієнт отримує після реєстрації на сайті APILayer, він необхідний для авторизації клієнта.

Параметри: `date` - дата з якою необхідно отримати курс валют, `symbols=EUR,GBP,USD,AED,AUD,CAD,MXN,PLN,SEK,TRY` — валюті курс, котрих необхідно отримати, `base=USD` - базова валюта відносно якої розраховуються курси валют.

Відповідь від сервера надійшла у форматі JSON, код стану 200, що означає, що запит був успішно виконаний. Результат запиту та відповіді зображено на рисунку 1.3.

За останні десятиліття використання API стало настільки популярним, що згідно звіту за 2024 рік від компанії Cloudflare 57% інтернет трафіку припадає саме на виклики API[22]. А дослідженню компанії Kong свідчить, що вплив API на світову економіку можна оцінити в \$10.9 трлн за 2023 рік[24]. У зв'язку з великим використанням API з'явилося навіть поняття API-економіки

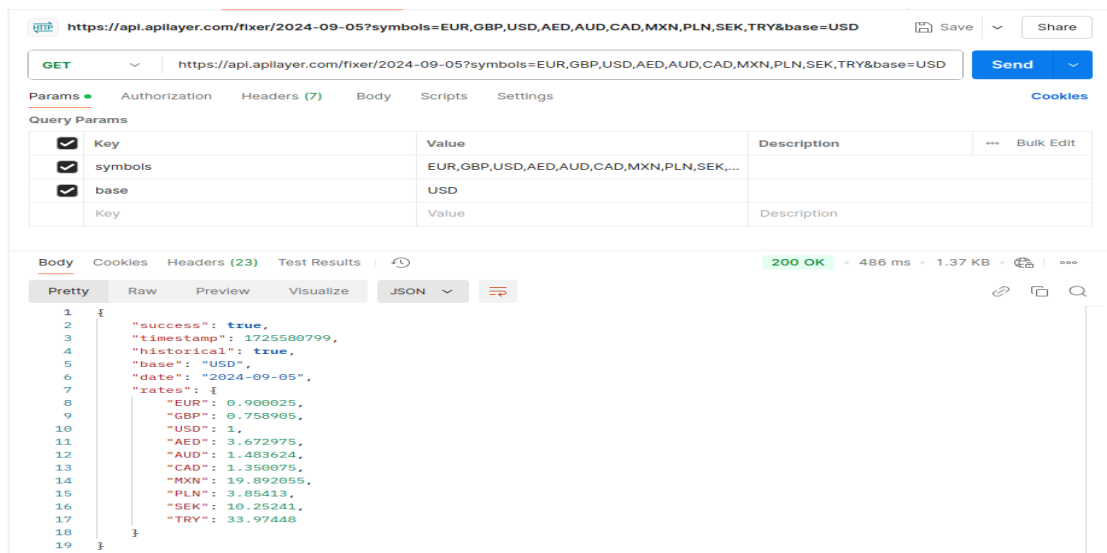


Рис. 1.3. Демонстрація роботи REST API

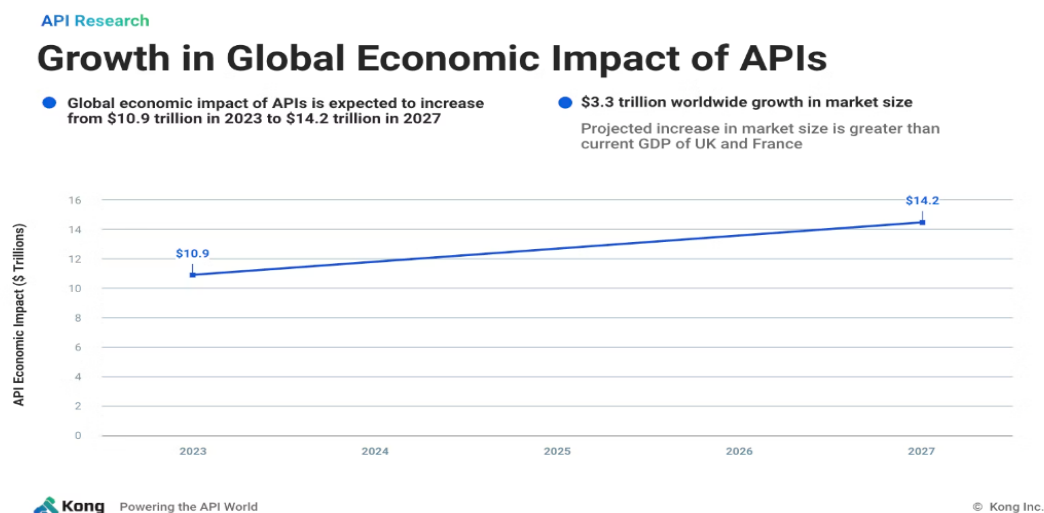


Рис. 1.4. Вплив API на світову економіку

1.2. API Economy

API Economy (API-економіка) можна визначити як екосистему, що формується ринком, у якій організації (як державні, так і приватні) розкривають свої сервіси, дані та можливості через API. Завдяки цим API розробники сторонніх продуктів і партнери можуть інтегрувати нові функції у власні продукти, стимулюючи таким чином інновації та економічне зростання.

Однією з головних переваг API Economy є ліквідація організаційних бар'єрів, що часто блокують вільний обмін інформацією. Стандартизовані й задокументовані інтерфейси дають змогу різним програмним системам обмінюватися даними та командами без складних кастомних інтеграцій. Така оптимізована комунікація сприяє швидким інноваціям, адже розробники можуть оперативно тестувати й впроваджувати нові функції або набори даних від зовнішніх постачальників. Замість того щоб вигадувати колесо заново, команди концентруються на більш масштабних рішеннях, зменшуючи витрати на розробку й скорочуючи час виходу продукту на ринок.

Прикметною рисою API Economy є еволюція від закритих, монолітних систем до відкритих платформ, що заохочують сторонній внесок. Великі технологічні компанії, як-от Google, Amazon, Microsoft та Salesforce, цілеспрямовано розвивають великі ринки API. Менші підприємства також можуть отримувати переваги, публікуючи для загального доступу окремі функціональні можливості. Такий відкритий підхід здатен запустити «мережевий ефект»: що більше розробників використовують і створюють продукти на базі API, то ціннішою стає сама платформа. Ці «цикли зворотного зв'язку» підсилюють видимість продукту, залучення користувачів та загальну присутність на ринку.

Завдяки API з'явилися й нові можливості для монетизації. Деякі організації пропонують доступ до API з розподілом за тарифами: безкоштовно - для базового використання, але з оплатою на високих рівнях навантаження. Інші надають безкоштовний доступ із певними обмеженнями для стимулювання початкового використання, а згодом продають розширені функції, аналітику чи преміум-доступ. Крім того, у певних випадках сама інформація може стати товаром (наприклад, дані про ринок або прогнози погоди). У цих моделях API перетворюються з технічних «конекторів» на рушії доходу, що вигідно і постачальнику, і споживачеві.

Однією з найважливіших переваг стратегії «API-first» є можливість забезпечувати омніканальний клієнтський досвід. Наприклад, банк з

відкритим API може інтегрувати свої сервіси у різноманітні платформи — застосунки для планування бюджету, сайти електронної комерції, а також роздрібні магазини - щоб клієнти могли здійснювати фінансові операції будь-де й будь-коли. Підключення до численних каналів через API дає змогу бізнесам об'єднувати дані користувачів і надавати їм узгоджений досвід, що позитивно впливає на лояльність і задоволеність клієнтів.

Хоча API Economy пропонує значний потенціал, вона передбачає й певні труднощі:

- Безпека. Загальнодоступні API часто стають мішенню для кібератак, витоків даних чи несанкціонованого доступу. Надійні методи безпеки - такі як API-шлюзи, токени та шифрування — мають ключове значення.
- Управління. Коли організація розширює кількість API, які вона надає або споживає, керувати ними стає дедалі складніше. Без належного управління можуть виникати дублювання зусиль і суперечливі стандарти, що знижують ефективність.
- Надійність. Занадто сильна залежність від зовнішніх API створює вразливості у разі змін умов використання або простоїв у роботі постачальників. Щоби забезпечити стабільність, необхідно ретельно планувати й передбачати резервні механізми.
- Масштабованість. Щоб API успішно витримував різкі стрибки навантаження (особливо у пікові періоди), потрібні належні інвестиції в інфраструктуру, а також стратегії кешування й балансування навантаження.

Прикладом однією з найвідоміших історій успіху в API Economy є Stripe - фінтех-компанія, що пропонує платформу для онлайн-обробки платежів в інтернет-бізнесі[23]. Незважаючи на те, що платіжні шлюзи існували і до Stripe, більшість із них були складними у налаштуванні, вимагали тривалих процесів затвердження і заплутаних інтеграцій.

Ключовою перевагою Stripe став дружній для розробників API. Зосередившись на простоті, чіткій документації та зручному середовищі для тестування, Stripe дозволила бізнесам будь-якого масштабу - від стартапів до великих підприємств - швидко впровадити онлайн-платежі. Ця простота стала визначальним фактором відмінності від конкурентів.

Розробники масово почали користуватися Stripe, адже для інтеграції платіжного рішення було достатньо лише кількох рядків коду. Завдяки тому, що їм подобалися зручні інструменти платформи, вони почали ділитися цим досвідом у своїх спільнотах. Це зміцнило репутацію Stripe і забезпечило їй динамічне зростання на ринку.

Завоювавши лідерські позиції у сфері обробки платежів, Stripe почала пропонувати додаткові рішення - регулярне виставлення рахунків, систему виявлення шахрайства і навіть кредитні продукти. Усі ці сервіси функціонують завдяки API, що дозволяє бізнесам обирати саме ті інструменти, які найбільше відповідають їхнім потребам. Нині бренд Stripe майже ототожнюється з простими, потужними та масштабованими платіжними API - це яскравий доказ того, наскільки ефективною може бути добре спланована API-стратегія.

1.3. Вразливості REST API

Вразливість - це слабе місце в програмному забезпеченні, яке виражене в помилках реалізації, поганій архітектурі, порушенні цілісності, неправильному функціонуванню системи. Також під вразливістю мають на увазі отримання інформації стороною, котра не наділена відповідними правами[10].

Класифікація вразливостей API може бути різною в різних компаніях, котрі займаються розробкою. Провідною організацією, котра займається питаннями безпеки в сфері IT є OWASP[11]. OWASP – це некомерційна організація, мета якої є підвищення безпеки в програмних продуктах. В

OWASP є окремий проект, котрий називається API Security Risks (безпекові ризики API). Урядова команда CERT-UA[12], яка створена для реагування на комп'ютерні надзвичайні події в Україні, рекомендує проекти OWASP для захисту веб ресурсів. OWASP згрупував 10 найбільш критичних вразливостей в API[13].

1. Порушена авторизація на рівні об'єкта

Авторизація на рівні об'єкта – це механізм контролю доступу, який зазвичай реалізується на рівні коду, щоб доступ до об'єкта мав лише користувач, котрий наділений відповідним дозволом. Кожна кінцева точка API, яка отримує ідентифікатор об'єкта і виконує дії над об'єктом, повинна реалізовувати перевірки авторизації на рівні об'єкта. Ці перевірки повинні підтверджувати, що авторизований користувач має дозвіл виконувати запитувану дію на обраному об'єкті. Якщо авторизація на рівні об'єкта порушена, то це зазвичай призводить до неавторизованого розкриття, модифікації чи знищення даних.

2. Зламаний механізм автентифікації

Автентифікація - це механізм для перевірки достовірних даних користувачем для входу в систему. Автентифікація реалізована невірно, якщо не має надійного захисту від токенів, термін дії котрих завершився, незашифровані або слабкі паролі та інші методи, які дозволяють зломисникам видавати себе за справжнього користувача. Помилки в автентифікації ставить під загрозу безпеку програмного продукту в цілому.

3. Зламаний дозвіл на рівні властивостей об'єкта

Ця вразливість виникає, коли відсутня або неправильно налаштована перевірка дозволів на рівні властивостей об'єкта. Це призводить до отримання властивостей об'єкта, які вважаються чутливими і повинні бути недоступними користувачу.

4. Необмежене споживання ресурсів

Така вразливість є наслідком, коли API не забезпечують належним чином обмежень на використання ресурсів (наприклад, пам'яті, процесора або

інтегрованих сторонніх сервісів). Зловмисники можуть використовувати це, надсилаючи надмірну кількість запитів, що призводить до відмови в обслуговуванні (DoS) або значного збільшення споживання ресурсів.

5. Необмежений доступ до чутливих бізнес-процесів

Авторизаційні перевірки для функції зазвичай виконуються на рівні конфігурації або коду. Реалізація належних перевірок може бути складним завданням через те, що в застосунках можуть бути багато типів ролей, груп і складних ієрархій користувачів. Вразливості легше виявити в API, оскільки вони більш структуровані, а доступ до функцій більш передбачуваний. Такі уразливості дозволяють зловмисникам отримувати доступ до несанкціонованих функцій. Адміністративні функції є основною метою такого типу атак, що може призвести до розголошення, втрати чи пошкодження даних. У кінцевому підсумку це може спричинити збій у роботі сервісу.

6. Необмежений доступ до чутливих бізнес-процесів

API можуть мати такі вразливості, бо відкриває бізнес-процеси. Це не обов'язково є помилкою в реалізації застосунку. Прикладом даної вразливості є бронювання житла або квитків на різні заходи, якщо не враховувати цього, то надмірне автоматичне використання такого функціоналу може суттєво зашкодити компанії. Якщо додаток інтегрований з іншим API за платною підпискою, то це також призводить до незапланованих фінансових витрат.

7. Серверна фальсифікація запитів

Програмний продукт може отримувати дані через віддалений ресурс, котрий надає користувач без перевірки URI. Приклади випадків, коли користувач надав URI є обробка web-hook, отримання файлів по URL, попередній перегляд URL-адрес. Наслідками такої вразливості може бути розкриття внутрішньої інформації сервера, витік конфіденційних даних, DoS-атаки.

8. Неправильна конфігурація безпеки

Зловмисники намагаються знайти помилки в сервісі, який працює з незахищеними конфігураціями, файлами, каталоги для отримання несанкціонованого доступу до системи або інформації про неї.

Прикладами такої інформації є:

- Повідомлення про помилки з трасування стеку, що можуть розкрити структуру таблиць.
- Відсутнє використання транспортного рівня безпеки (TLS).
- Невідповідності в обробці вхідних запитів серверами.
- Неправильне налаштування прав доступу до хмарних сервісів.

Неправильна конфігурація сервера не тільки відкриває доступ до конфіденційних даних користувачів, але й надає інформацію про систему, що може призвести до порушення роботи сервера.

9. Неналежне управління інвентаризацією

Якщо API має декілька версій, то це збільшує вразливість, якщо не забезпечити належне управління кожною версією. Зловмисники можуть мати доступ до чутливих даних на сервері через застарілі версії API. Інколи старі версії API можуть бути підключені до бази даних з реальними даними. Зловмисники можуть використати застарілі версії для доступу до адміністративних функцій.

10. Небезпечне використання API

Інтеграція сторонніх API без належної перевірки, не захищена передача даних може зробити систему вразливою. Інколи при розробці ігнорують фільтрацію даних отриманих від сторонніх API, що призвести до SQL-ін'єкцій, перевантаження сервера або відмова в обслуговуванні.

1.4. Безпека REST

З огляду на вразливості зазначені в OWASP API Security Risks, безпека REST API є одним з найважливіших критеріїв при створенні додатку. Автентифікація для визначення, чи має клієнт право на доступ до системи і

можливість комунікувати з нею. Шифрування при передачі даних між сервером і клієнтом є важливим механізмом захисту інформації[14].

У своїй основі безпека REST API забезпечує, що лише авторизовані користувачі можуть виконувати дозволені дії. Це досягається насамперед через надійні процеси автентифікації та авторизації API. Про витоки даних швидко дізнаються медіа, і шкода для репутації компанії може бути катастрофічною. І споживачі, і бізнес-партнери прагнуть співпрацювати лише з надійними організаціями. Один єдиний прокол у безпеці може звести нанівець багаторічну роботу над позитивним іміджем і вдарити по лояльності клієнтів. Тож інвестиції в заходи з безпеки API також є інвестицією в довгострокову стійкість бренду. Ці процеси є ключовими для перевірки особи користувачів і належного надання дозволів.

Як приклад можна привести групу компаній T-Mobile, котрі забезпечує мобільним зв'язком клієнтів в Європі та США. В 2021 T-Mobile була атакована, що призвело до витоку персональних даних 79 мільйонів клієнтів у всьому світі, включаючи номери соціального страхування[16]. В свою чергу Федеральна комісія зі зв'язку США заявила, що згідно її розслідування кібератаки, які зазнала T-Mobile, призвели до витоку даних, «які вплинули на мільйони клієнтів мобільних телефонів, були різними за своїм характером, використанням і очевидними методами атаки». У результаті T-Mobile повинна буде сплатити 15,75 мільйонів доларів штрафу до Міністерства фінансів США[17].

REST API може бути об'єктом атак типу відмови в обслуговуванні. У такому разі зловмисники надсилають величезну кількість запитів на кінцеві точки, внаслідок чого сервіси стають недоступними. Якщо організація не впроваджує обмеження частоти запитів чи інші механізми для стримування атак, то ризикує опинитися офлайн, втрачаючи клієнтів і дохід. Належні заходи безпеки допоможуть підтримувати безперебійну роботу сервісів і зменшити шкоду від скоординованих атак.

Однак сфера безпеки REST API виходить за межі лише контролю доступу. Вона включає моніторинг та ведення журналів активності REST API для виявлення та усунення потенційних загроз, впровадження обмеження швидкості для запобігання зловживанням і управління життєвим циклом REST API для зменшення вразливостей, які можуть використовувати злоумисники[15].

Залежно від сфери діяльності, компанії можуть підпадати під дію жорстких норм, які зобов'язують їх захищати дані користувачів. Наприклад, у Європейському Союзі за порушення Загального регламенту із захисту даних (GDPR) передбачено штрафи в розмірі 4% від обігу за попередній фінансовий рік або до 20 млн євро[18]. Порушення GDPR призвело, що 5 вересня 2022 року Комісія із захисту даних Ірландії наклала на Meta Ireland штраф у розмірі 405 мільйонів євро щодо законності обробки персональних даних дітей відповідно до правових підстав виконання контракту та законного інтересу[19]. Надійний захист API допомагає компаніям дотримуватися вимог законодавства, уникати штрафних санкцій та юридичних ускладнень.

Висновки до розділу 1

З аналізу літератури, статей по REST, досвіду організацій та розробників було визначено, що саме архітектурного стиль REST має найбільше використання в побудові API додатків. Його поширеність зумовлена простотою, масштабованістю та гнучкістю, що робить його кращим вибором для організацій та розробників, які прагнуть безперешкодної взаємодії між клієнтом та сервером. Однією з головних переваг REST є його безстанність, що означає, що кожен запит від клієнта до сервера повинен містити всю інформацію, необхідну для обробки цього запиту.

Попри простоту та переваги архітектурного стилю REST було розглянуто, що він має суттєві прогалини в безпеці. Найбільшими вразливостями є шифрування при передачі даних, авторизація, автентифікація, логування та обмеження швидкості (rate limit).

З огляду на такі суттєві вразливості було встановлено, що саме безпека відіграє ключову роль в проектування, розробці та використанні REST додатків. Безпека REST API - це не лише технічні питання, а фундаментальна необхідність та важливий аспект для бізнесу, котрий вимагає комплексного підходу, щоб надавати користувачам безпечний і надійний сервіс.

РОЗДІЛ 2. ТЕХНОЛОГІЇ ДЛЯ СТВОРЕННЯ БЕЗПЕЧНИХ REST API ДОДАТКІВ

2.1. API Management

API Management включає створення, розповсюдження, контроль і аналіз API у безпечному та масштабованому середовищі. Це охоплює широкий спектр інструментів і стратегій, які допомагають організаціям надавати свої послуги внутрішнім командам, зовнішнім партнерам або публіці, зберігаючи контроль над використанням і продуктивністю. Платформи управління API зазвичай включають такі функції, як шлюзи API, портали для розробників, аналітичні панелі та механізми безпеки[20].

Основна мета API Management - забезпечити максимальну цінність API для бізнесу, покращуючи їхню зручність, надійність і безпеку. Оскільки компанії продовжують впроваджувати хмарні технології, мікросервіси, важливість надійного управління API зростає.

Посилення безпеки. API є шлюзами до конфіденційних даних і послуг, що робить їх основною цілью для кіберзлочинців. Ефективний API Management допомагає захистити API за допомогою функцій, таких як автентифікація, авторизація, шифрування та обмеження швидкості. Політики безпеки можуть бути застосовані для запобігання несанкціонованому доступу та пом'якшення загроз, таких як DoS-атаки або витоки даних. Інтегруючи ці механізми, організації можуть захистити свої цифрові активи та зміцнити довіру користувачів і партнерів.

Сприяння масштабованості. Зі зростанням бізнесу використання API часто масштабуються експоненційно. API Management забезпечує балансування навантаження та механізми контролю трафіку, щоб гарантувати, що API можуть справлятися з підвищеним попитом без зниження продуктивності. Ця масштабованість особливо критична для бізнесів, які працюють на динамічних ринках, де можливе швидке зростання.

Покращення досвіду розробників. Добре організована екосистема API покращує досвід розробників, які є ключовими зацікавленими сторонами у використанні API. Функції, такі як інтуїтивна документація, портали для розробників і тестові середовища, спрощують для розробників розуміння, тестування та інтеграцію API у свої додатки. Це оптимізує процеси розробки, прискорює вихід на ринок і стимулює інновації.

Забезпечення бізнес-аналітики. API Management надає інструменти аналітики, які пропонують цінну інформацію про шаблони використання API, метрики продуктивності та поведінку користувачів. Ці дані дозволяють організаціям приймати обґрунтовані рішення, оптимізувати продуктивність API та визначати нові можливості для отримання доходів. Наприклад, розуміння, які API є найпопулярнішими, може спрямовувати стратегічні інвестиції у розвиток продуктів.

Спрощення управління та дотримання вимог. API часто мають відповідати галузевим нормам і організаційним політикам. Рішення API Management спрощують управління завдяки централізованому контролю над політиками API, правилами доступу та журналами аудиту. Це забезпечує відповідність API вимогам і знижує адміністративне навантаження.

Підтримка інновацій і співпраці. API дозволяють компаніям створювати екосистеми, де внутрішні команди та зовнішні партнери можуть безперешкодно співпрацювати. Завдяки ефективному API Management організації можуть стимулювати інновації, створювати нові джерела доходів і розширювати свій ринковий вплив. Наприклад, відкриті API дозволяють стороннім розробникам створювати додаткові застосунки, які розширюють основні пропозиції компанії.

Реальні застосування API Management.

Платформи електронної комерції. Онлайн-маркетплейси значною мірою залежать від API для інтеграції платіжних систем, систем управління запасами та сторонніх логістичних сервісів. API Management забезпечує безперебійну роботу цих інтеграцій, надаючи клієнтам зручний досвід покупок.

Сфера охорони здоров'я. API сприяють обміну даними між медичними установами, страховиками та пацієнтами. API Management забезпечує конфіденційність і безпеку медичної інформації, а також взаємодію між системами.

Фінансові послуги. Банки та фінансові установи використовують API для надання послуг, таких як мобільний банкінг, обробка платежів і виявлення шахрайства. API Management гарантує надійність, безпеку та масштабованість цих послуг.

2.2. API Gateway

API Gateway (API-шлюз) - це ключовий компонент у стратегії API Management, проте його роль вужча й більш технічна[25]. Він діє як єдина централізована точка входу, через яку зовнішні клієнти (зокрема, мобільні додатки, вебсистема або партнерські системи) взаємодіють із бекенд-сервісами та мікросервісами. Знаходячись перед бекенд-сервісами, шлюз керує потоком запитів і може застосовувати функції, такі як маршрутизація, автентифікація та обмеження швидкості.

Основні аспекти API Gateway включають:

- Маршрутизацію запитів: Перенаправлення вхідних запитів до відповідного мікросервісу чи бекенд-системи.
- Перетворення протоколів: Переклад запитів з одного протоколу в інший (наприклад, з HTTP в AMQP або gRPC).
- Безпека: Керує перевіркою токенів, завершенням SSL/TLS, ключами API, OAuth2.0, JWT та іншими механізмами автентифікації.
- Оптимізація продуктивності: Передбачає кешування, балансування навантаження, обмеження швидкості та керування піковими навантаженнями (throttling).

- Реалізація політик: Виконує спеціальні політики чи правила, визначені на рівні шлюзу (наприклад, фільтрація IP, обмеження використання або налаштування заголовків).

API Gateway надає численні переваги для організацій, розробникам кінцевим користувачам. Ось деякі істотні переваги:

API Gateway діють як захисний шар між клієнтами та серверними службами. Вони можуть керувати автентифікацією (наприклад, ключі OAuth), шифрувати передачу даних і блокувати зловмисні запити, тим самим зменшуючи ризики безпеки.

Оптимізація продуктивності. Впроваджуючи кешування, балансування навантаження та агрегацію запитів, API Gateway забезпечують швидший час відповіді та оптимізовану продуктивність сервера. Це особливо корисно для програм із високим трафіком.

Централізований моніторинг та аналітика. API Gateway надають централізовані інструменти для відстеження використання API, помилок і продуктивності. Завдяки даним і аналітиці в реальному часі команди можуть контролювати стан системи, оптимізувати ресурси та завчасно виявляти проблеми.

Гнучкість і масштабованість. API Gateway спрощують масштабування додатків, ефективно розподіляючи запити та керуючи примірниками серверної служби. Нові послуги можна додавати або оновлювати, не впливаючи на клієнтів.

Економія коштів. Завдяки оптимізації обробки запитів, кешування відповідей і зменшенню споживання ресурсів, API Gateway знижують витрати на інфраструктуру та підвищують ефективність серверних служб.

Таблиця 1. Порівняння основних відмінностей між API Management та API Gateway

Аспект	API Management	API Gateway
Призначення	Повний життєвий цикл (проектування, документація, аналітика використання, портал для розробників, управління)	Етап впровадження (маршрутизація, погодження протоколів, безпека)
Фокус	Орієнтація на бізнес (управління, досвід розробників, монетизація)	Технічний/операційний (управління трафіком, автентифікація, моніторинг)
Компоненти	Портал для розробників, аналітичний модуль, управління політиками, життєвим циклом	Балансування навантаження, маршрутизація, кешування, автентифікація, обмеження швидкості
Сценарії використання	Комплексна стратегія API, монетизація, корпоративне управління	Точка входу для мікросервісів, підвищення продуктивності
Приклади готових рішень	Apigee, Azure API Management, MuleSoft, Kong Enterprise, IBM API Connect	Kong Gateway, NGINX, Istio, AWS API Gateway

2.3. Хмарні веб-сервіси від Amazon

AWS є провідною платформою хмарних обчислень, яку пропонує Amazon[26]. Він надає широкий спектр послуг на вимогу та API, які обслуговують окремих осіб, компанії та уряди. Використовуючи AWS, організації можуть отримати доступ до обчислювальних ресурсів без значних

попередніх інвестицій в апаратне забезпечення, що дозволяє їм плавно масштабувати свої операції відповідно до попиту.

Платформа включає в себе послуги для обчислень, зберігання даних, мереж, баз даних, штучного інтелекту, машинного навчання, безпеки, аналітики тощо, розроблені для задоволення різноманітних потреб бізнесу. AWS дає змогу розробникам швидко створювати, тестувати та розгортати програми, забезпечуючи при цьому економічну ефективність за допомогою моделі оплати за використання. Він відомий своєю глобальною інфраструктурою, яка забезпечує високу доступність і низьку затримку продуктивності незалежно від місця розташування.

Основна мета AWS - забезпечити масштабовану обчислювальну потужність і гнучке, безпечне та інноваційне середовище для бізнесу. Допмагаючи організаціям модернізувати їх IT-інфраструктуру, AWS підтримує цифрову трансформацію та сприяє інноваціям у різних галузях, що робить його кращим вибором для підприємств, стартапів та окремих розробників у всьому світі.

Кампанія Amazon надає близько 230 веб-сервісів для обслуговування додатків різних масштабів, під назвою Amazon Web Services (AWS). Зробимо огляд деяких важливих веб-сервісів для безпеки API.

2.3.1. Amazon API Gateway

Amazon API Gateway - це повністю керована служба, яка надає розробникам можливість створювати, публікувати, обслуговувати, відстежувати й захищати REST API. Вона виступає як «вхідні двері» для додатків, що отримують доступ до даних, бізнес-логіки чи функціональності з бекенд-сервісів, які працюють з інші служби AWS, а також із зовнішніми HTTP ресурсами[27].

Завдяки абстрагуванню більшості складнощів під час розробки й керування API, API Gateway бере на себе такі завдання, як маршрутизація

трафіку, фільтрація запитів, обмеження пропускної здатності (throttling), кешування, аналітика використання, авторизація та контроль доступу. Це дає змогу розробникам зосередитися на розробці саме на функціональності додатку. Принцип роботи Amazon API Gateway зображено на рисунку 2.1.



Рис. 2.1. Схема роботи Amazon API Gateway[28]

Деякі ключові функції Amazon API Gateway.

Відмовостійкість. API Gateway допомагає керувати трафіком до ваших внутрішніх систем, дозволяючи встановлювати правила дроселювання на основі кількості запитів в секунду для кожного HTTP-методу в API. Також API Gateway обробляє будь-який рівень трафіку, що надходить до API, тому можна зосередитися на бізнес-логіці, а не на підтримці інфраструктури. При використанні REST API можна налаштувати кеш з ключами і часом життя в секундах для даних API, щоб уникнути навантаження на внутрішні сервіси при кожному запиті. Підвищує продуктивність API й зменшує навантаження на бекенд, кешуючи відповіді на визначений час (TTL).

Масштабування та висока доступність. API Gateway автоматично масштабується горизонтально, обробляючи раптові сплески трафіку без необхідності вручну керувати інфраструктурою. Вбудована відмовостійкість забезпечує високу доступність і стабільну продуктивність. Встановлення

лімітів швидкості запитів та пікових навантажень, щоб запобігти надмірним обсягам трафіку й пом'якшити DDoS-подібні атаки.

Просте створення, розгортання та інтеграція API. За допомогою API Gateway можна швидко і легко створити власний API для додатку, що працює в AWS Lambda, а потім викликати код Lambda з вашого API. API Gateway може виконувати код AWS Lambda у вашому обліковому записі, запускати машини станів AWS Step Functions або здійснювати виклики до AWS Elastic Beanstalk, Amazon EC2 або веб-сервісів поза AWS з загальнодоступними кінцевими точками HTTP. За допомогою консолі API Gateway ви можете визначити свій REST API і пов'язані з ним ресурси та методи, керувати життєвим циклом API, генерувати клієнтські SDK і переглядати метрики API.

Моніторинг операцій API. Після розгортання та використання API, API Gateway надає інформаційну панель для візуального моніторингу викликів до сервісів. Консоль API Gateway інтегрована з Amazon CloudWatch, тому можна отримувати показники продуктивності додатку, такі як виклики API, затримки та кількість помилок. Оскільки API Gateway використовує CloudWatch для запису інформації моніторингу є можливість налаштувати власні правила для API. API Gateway також може реєструвати помилки виконання API в журналах CloudWatch, щоб полегшити налагодження.

Ключі API для сторонніх розробників. Якщо ви використовуєте REST API, API Gateway допоможе вам керувати екосистемою сторонніх розробників, які отримують доступ до ваших API. Ви можете створювати ключі API на API Gateway, встановлювати детальні дозволи доступу для кожного ключа API та розповсюджувати їх серед сторонніх розробників для доступу до ваших API. Ви також можете визначати плани, які встановлюють обмеження на запити та квоти для кожного окремого ключа API. Використання ключів API є повністю необов'язковим і має бути ввімкнена на рівні кожного методу.

Управління життєвим циклом API. При використанні REST API, API Gateway дозволяє запускати кілька версій одного API одночасно, щоб

програми могли продовжувати викликати попередні версії API навіть після публікації останніх версій. API Gateway також допомагає вам керувати кількома етапами випуску для кожної версії API, такими як dev-, test- та prod-версіями. Кожну стадію API можна налаштувати на взаємодію з різними кінцевими точками бекенда на основі налаштувань API. Конкретні етапи та версії API можна пов'язати з власним доменним ім'ям і керувати ними через API Gateway. Управління етапами та версіями дозволяє легко тестувати нові версії API, які покращують або додають нову функціональність до попередніх випусків API, а також забезпечує зворотну сумісність, коли спільноти користувачів переходять на використання останньої версії.

Авторизація. Для авторизації та перевірки запитів API до сервісів AWS, API Gateway може допомогти вам використовувати підпис версії 4 для REST API. Використовуючи автентифікацію підпису версії 4, ви можете використовувати AWS Identity and Access Management (IAM) і політики доступу для авторизації доступу до API і всіх інших ресурсів AWS. Ви також можете використовувати лямбда-функції AWS для перевірки та авторизації токенів на пред'явника, таких як токени JWT або SAML-твердження.

Робота зі спадковими застосунками та монолітами. API Gateway може слугувати фасадом перед локальними системами або традиційними застосунками на AWS. Що дає змогу поступово переносити застарілі ендпоінти, не порушуючи контрактів із клієнтами.

2.3.2. Amazon CloudWatch

Amazon CloudWatch - це надійний сервіс від Amazon, який зосереджений на моніторингу та спостереженні. Він дозволяє користувачам контролювати свої хмарні ресурси та додатки, збираючи, отримуючи доступ та аналізуючи дані про продуктивність і роботу в режимі реального часу. Ця послуга надає практичну інформацію для забезпечення ефективної роботи додатків,

дотримання стандартів продуктивності, оптимізації використання ресурсів і мінімізації часу простою[30].

Amazon CloudWatch перевершує інші сервіси в кількох сферах, починаючи з можливості збирати та зберігати різноманітні дані, такі як метрики, журнали та події. Показники - це впорядковані за часом точки даних, пов'язані з ресурсами та додатками AWS, які фіксують такі деталі, як використання процесора, споживання пам'яті та активність на диску. Журнали дозволяють користувачам збирати, відстежувати та аналізувати файли журналів служб AWS та користувацьких додатків, а події забезпечують потік змін у системі майже в реальному часі[31].

Візуалізація є ще одним важливим аспектом, оскільки CloudWatch надає інформаційні панелі, які можна налаштувати для відображення метрик і журналів. Ці панелі консолідують дані з декількох облікових записів AWS і регіонів, пропонуючи уніфікований досвід моніторингу. Крім того, тривоги в CloudWatch дозволяють здійснювати проактивний моніторинг, повідомляючи користувачів або запускаючи дії при перевищенні певних порогових значень. Ці тривоги можуть виконувати такі завдання, як масштабування екземплярів EC2, зупинка служб або надсилання сповіщень.

Сервіс також підтримує автоматичні дії, інтегруючись з іншими пропозиціями AWS, що дозволяє йому налаштовувати ресурси на основі попиту. Використовуючи машинне навчання, CloudWatch включає функцію виявлення аномалій для виявлення незвичайних патернів у метриках, що сприяє вирішенню проблем.

Завдяки безшовній інтеграції з широким спектром сервісів AWS, таких як EC2, S3, Lambda та RDS, CloudWatch також може відстежувати сторонні та користувацькі додатки. Його аналітичні можливості очевидні завдяки таким функціям, як CloudWatch Logs Insights, яка використовує підхід на основі запитів для вилучення інформації з даних журналів. ServiceLens ще більше покращує спостережливість, надаючи наскрізні уявлення про продуктивність і залежності додатків.

Amazon CloudWatch надає значні переваги для бізнесу, починаючи з покращеної спостережливості. Пропонуючи комплексний огляд операційних даних в усіх облікових записах і службах AWS, він спрощує моніторинг і усунення несправностей. Можливості моніторингу підвищують надійність, зменшуючи час простою та забезпечуючи швидке реагування на проблеми. Масштабованість CloudWatch гарантує, що він може працювати як з невеликими додатками, так і з інфраструктурою корпоративного рівня, адаптуючись до мінливих потреб бізнесу. Детальна інформація, яку надає сервіс, дозволяє користувачам оптимізувати використання ресурсів, що сприяє підвищенню економічної ефективності. Крім того, безшовна інтеграція з AWS і підтримка користувацьких додатків роблять його надзвичайно універсальним.

Amazon CloudWatch виявляється незамінним у різних сценаріях. Він допомагає підтримувати продуктивність додатків, відстежуючи ключові показники та виявляючи вузькі місця в режимі реального часу. Для управління інфраструктурою він забезпечує стабільність серверів, сховищ і мережевих ресурсів. Коли виникають проблеми, CloudWatch спрощує усунення несправностей завдяки можливостям аналізу журналів, що дозволяє користувачам ефективно діагностувати та вирішувати проблеми. Відстежуючи використання ресурсів, організації можуть динамічно адаптуватися для досягнення економії витрат. CloudWatch також допомагає в дотриманні нормативних вимог та аудиті, зберігаючи та аналізуючи дані журналів для цілей безпеки та регулювання.

Сервіс працює шляхом збору даних з ресурсів AWS і користувацьких додатків, які надійно зберігаються для аналізу та візуалізації. Користувачі можуть вивчати ці дані за допомогою інформаційних панелей, запитів до журналів або інших інструментів, наданих CloudWatch. При порушенні встановлених порогових значень тривоги запускають сповіщення або дії, що дозволяє вчасно реагувати на критичні події.

CloudWatch використовує модель ціноутворення, де вартість залежить від використовуваних функцій, таких як моніторинг показників, зберігання та пошук журналів, а також використання спеціальних інформаційних панелей та тривог. Безкоштовний рівень AWS включає базовий моніторинг для EC2 та інших ресурсів AWS, що робить його доступним для нових користувачів.

Amazon CloudWatch виділяється як важливий сервіс для моніторингу та управління середовищами AWS і користувацькими додатками. Його комплексні можливості, адаптивність та безперешкодна інтеграція дозволяють компаніям підвищити продуктивність, надійність та економічну ефективність. Незалежно від того, чи контролюєте ви один додаток, чи складну інфраструктуру, CloudWatch надає інструменти, необхідні для досягнення операційної досконалості в хмарі.

2.3.3. AWS Identity and Access Management

AWS Identity and Access Management (IAM) - це фундаментальна служба, яка забезпечує безпечний контроль доступу до ресурсів AWS. Вона дозволяє організаціям керувати дозволами для користувачів, груп і додатків, забезпечуючи детальний контроль над тим, хто може виконувати певні дії в середовищі AWS. Будучи важливим компонентом безпеки AWS, IAM допомагає захистити ресурси, визначаючи і забезпечуючи контроль доступу на детальному рівні[32].

IAM дозволяє керувати ідентичностями, включаючи користувачів, групи, ролі та об'єднані ідентичності. Користувачі представляють індивідуальні облікові записи для людей або додатків, яким потрібен доступ до служб AWS, тоді як групи дозволяють колективно призначати дозволи декільком користувачам, спрощуючи контроль доступу. Ролі полегшують отримання тимчасових облікових даних для доступу до ресурсів AWS, що особливо корисно для додатків або зовнішніх користувачів, які виконують певні завдання. Крім того, IAM підтримує федерацію ідентичностей, що

дозволяє зовнішнім об'єктам, таким як корпоративні каталоги або сторонні постачальники ідентичностей, отримувати доступ до ресурсів AWS без необхідності в індивідуальних облікових записах користувачів IAM.

З IAM управління доступом стає структурованим і безпечним процесом. Дозволи визначаються за допомогою політик, які є JSON-документами, що визначають дії, ресурси та умови. Ці політики мають вирішальне значення для визначення того, які дії дозволені на яких ресурсах за певних обставин. IAM також надає розширені функції, такі як межі дозволів для встановлення максимальних рівнів дозволів та політики на основі ресурсів, які приєднуються безпосередньо до ресурсів AWS. На організаційному рівні політики контролю служб (Service Control Policies, SCP) забезпечують загальне управління декількома обліковими записами, забезпечуючи узгодженість і відповідність вимогам.

Безпека додатково посилюється за допомогою багатофакторної автентифікації (MFA), яка підвищує захист, вимагаючи додаткового рівня автентифікації. Це може бути щось відоме користувачеві, наприклад, пароль, у поєднанні з чимось, чим він володіє, наприклад, пристроєм безпеки або додатком. Увімкнення MFA є життєво важливим кроком у зменшенні ризиків, пов'язаних зі скомпрометованими обліковими даними.

Надання тимчасових облікових даних IAM є ще одним рівнем безпеки, особливо цінним для додатків або служб, які динамічно отримують доступ до ресурсів AWS. Ці тимчасові облікові дані мінімізують вплив довгострокових ключів, таким чином зменшуючи потенційні вразливості.

Для забезпечення підзвітності та аудиту IAM легко інтегрується з AWS CloudTrail. Ця інтеграція реєструє діяльність, пов'язану з IAM, надаючи детальні записи про те, хто зробив запит, коли він був зроблений і які дії були виконані. Ці журнали незамінні для аудиту безпеки та реагування на інциденти, пропонуючи прозорість та можливість відстежувати операції з управління доступом.

Політики відіграють ключову роль у функціоналі IAM, визначаючи дозволи, які регулюють доступ. Керовані політики, заздалегідь визначені AWS або створені користувачем, можна використовувати багаторазово, що сприяє ефективному контролю доступу. Вбудовані політики, з іншого боку, вбудовуються безпосередньо в конкретні об'єкти IAM, що робить їх унікальними для користувача, групи або ролі, до якої вони прив'язані. Клієнтські політики дозволяють організаціям адаптувати дозволи до своїх конкретних потреб, забезпечуючи гнучкість і точність у визначенні доступу.

Ролі відіграють важливу роль у наданні доступу до ресурсів AWS, особливо для додатків, що працюють на сервісах AWS. Приймаючи роль, додатки або сервіси можуть тимчасово отримати дозволи, необхідні для виконання завдань, наприклад, екземпляр EC2 може отримати доступ до S3-бакета. Ролі також підтримують доступ між обліковими записами, що забезпечує безпечну співпрацю між різними обліковими записами AWS.

При розробці IAM пріоритетами є безпека та ефективність роботи. Межі дозволів пропонують додатковий захист, визначаючи верхню межу дозволів, які може отримати організація, гарантуючи, що жодні надмірні дозволи не будуть надані ненавмисно. Ця функція особливо корисна в середовищах, де кілька адміністраторів керують контролем доступу.

Впровадження IAM має відповідати найкращим практикам, щоб максимізувати безпеку та операційну ефективність. Організаціям рекомендується захищати конфіденційні облікові записи за допомогою багатофакторної автентифікації, надавати дозволи на основі принципу найменших привілеїв, а також регулярно переглядати та оновлювати політики, щоб вони відповідали потребам безпеки, що змінюються. Використання ролей для доступу до додатків дозволяє уникнути пасток жорстко закодованих облікових даних, а групи спрощують управління дозволами, застосовуючи колективні дозволи до користувачів зі схожими вимогами до доступу. Централізований контроль за допомогою AWS Organizations і SCPs ще більше

покращує управління, забезпечуючи узгоджений контроль доступу в різних облікових записах.

IAM також відіграє вирішальну роль у дотриманні нормативних вимог, підтримуючи обмеження доступу до даних, автентифікацію користувачів і можливість аудиту. Забезпечуючи детальний контроль доступу та ведення надійних аудиторських журналів, IAM допомагає організаціям дотримуватися нормативних стандартів, таких як GDPR та HIPA.

Таким чином, AWS Identity and Access Management - це комплексний і універсальний інструмент для управління безпечним доступом до ресурсів AWS. Його надійні функції, включаючи деталізовані дозволи, тимчасові облікові дані та інтеграцію з інструментами аудиту, дозволяють організаціям зменшити ризики безпеки та досягти відповідності нормативним вимогам. Ефективно використовуючи IAM, компанії можуть захистити свої хмарні середовища, спростити управління доступом і забезпечити цілісність своїх операцій.

2.3.4. AWS CloudTrail

AWS CloudTrail - це потужний сервіс, розроблений для підтримки управління, відповідності та аудиту в середовищі AWS. Він забезпечує повний запис активності в інфраструктурі AWS, фіксуючи виклики API та пов'язані з ними події, що виконуються користувачами, ролями та службами. Ці записи зберігаються безпечно і централізовано, що дозволяє проводити детальний аналіз, пошук і усунення несправностей та аудит[33].

CloudTrail автоматично відстежує дії, виконані через різні інтерфейси, включаючи консоль управління AWS, CLI, SDK і API. Він розрізняє події управління, такі як створення або модифікація ресурсів, і події даних, які фіксують більш деталізовані дії, такі як доступ до певних об'єктів в Amazon S3 або виконання лямбда-функції.

Ключовою перевагою CloudTrail є вбудована функція історії подій, яка дозволяє користувачам переглядати активність облікового запису за останні 90 днів без необхідності додаткового налаштування. Для безперервного моніторингу ви можете налаштувати трєї, які реєструють події в різних регіонах і доставляють їх в Amazon S3. Це забезпечує централізований нагляд за всіма діями, незалежно від того, де вони відбуваються у вашій інфраструктурі AWS.

CloudTrail також легко інтегрується з іншими сервісами AWS, такими як CloudWatch, що дозволяє отримувати сповіщення в режимі реального часу і автоматично реагувати на певні дії. Журнали, що зберігаються в S3, можна аналізувати за допомогою таких інструментів, як Amazon Athena або сторонніх рішень для більш глибокого розуміння операційних моделей і потенційних проблем.

Безпека та відповідність вимогам є центральним елементом дизайну CloudTrail. Журнали шифруються і зберігаються надійно, з ретельним контролем доступу, щоб відповідати нормативним стандартам, таким як GDPR, HIPAA. Виявлення аномалій покращується за допомогою CloudTrail Insights, який виявляє незвичайні шаблони активності, такі як сплески викликів API, і допомагає організаціям реагувати на потенційні ризики.

На додаток до підвищення безпеки, CloudTrail допомагає діагностувати операційні проблеми, пропонуючи видимість змін, зроблених в середовищі. Для дотримання нормативних вимог він слугує безцінним інструментом для створення детальних аудиторських слідів, які часто вимагаються зовнішніми аудиторами або внутрішніми політиками.

AWS CloudTrail є важливим компонентом для управління сучасними хмарними операціями, забезпечуючи видимість діяльності, підвищуючи безпеку та відповідність вимогам відповідності. Збираючи та аналізуючи журнали активності, він дає змогу організаціям ефективно контролювати хмарні середовища.

2.3.5. Amazon Inspector

Amazon Inspector - це повністю керована служба безпеки, яка підвищує безпеку додатків, розгорнутих у хмарі AWS. Він автоматизує оцінку безпеки, виявляючи вразливості, ненавмисні вразливості мережі та відхилення від найкращих практик. Ці знання дозволяють організаціям захистити свої робочі навантаження в AWS та ефективніше відповідати стандартам безпеки.

Amazon Inspector пропонує автоматизовану оцінку безпеки, усуваючи ручну роботу, необхідну для перевірки вразливостей. Він легко інтегрується з сервісами AWS, включаючи Amazon EC2, гарантуючи, що оцінка охоплює різноманітні робочі навантаження в декількох облікових записах. Сервіс використовує заздалегідь створені пакети правил, що підтримуються AWS, які охоплюють широкий спектр вразливостей і проблем з конфігурацією. На відміну від традиційних інструментів, Amazon Inspector забезпечує безперервне сканування, виявляючи вразливості та ризики в режимі, близькому до реального часу, що дозволяє швидко їх усунути. Кожній виявленій проблемі присвоюється оцінка серйозності, що дозволяє організаціям ефективно визначати пріоритети своїх зусиль.

Amazon Inspector починає роботу з виявлення підтримуваних ресурсів у середовищі AWS, таких як екземпляри Amazon EC2 і зображення контейнерів, що зберігаються в ECR. Він оцінює ці ресурси відповідно до кураторських пакетів правил, перевіряючи операційні системи, встановлене програмне забезпечення, конфігурації та залежності. Сервіс генерує висновки, які включають детальний опис вразливостей, уражених ресурсів, рекомендовані кроки з усунення та оцінки серйозності. Ці результати надсилаються до таких інструментів, як AWS Security Hub або Amazon EventBridge, для подальшого аналізу та вжиття заходів, оптимізуючи робочий процес безпеки.

Організації використовують Amazon Inspector для проактивного виявлення та усунення вразливостей у своїх робочих навантаженнях. Безперервно скануючи ресурси, сервіс гарантує, що жодна критична проблема

не залишиться без уваги. Він також є цінним інструментом для дотримання нормативних вимог, оскільки надає автоматизовані та задокументовані оцінки, узгоджені з такими стандартами, як GDPR. Для розгортання на основі контейнерів Amazon Inspector інтегрується з Amazon ECR, щоб оцінити образи контейнерів на наявність вразливостей перед їх розгортанням. Великі підприємства отримують вигоду від його здатності оцінювати безпеку в декількох облікових записах AWS, забезпечуючи централізовану видимість потенційних ризиків.

Надаючи дієві ідеї та безперервний моніторинг, Amazon Inspector допомагає організаціям підтримувати надійний рівень безпеки. Він зменшує ручну роботу, необхідну для оцінки вразливостей, дозволяючи командам безпеки зосередитися на інших критично важливих сферах. Сервіс масштабується, автоматично оцінюючи нові ресурси в міру їх додавання в середовище, а його модель ціноутворення «оплата за фактом використання» робить його економічно вигідним рішенням у порівнянні з традиційними інструментами.

Хоча Amazon Inspector є потужним інструментом, він в першу чергу фокусується на екземплярах EC2 і сховищах ECR, які можуть не охоплювати всі ресурси в середовищі AWS. Тісна інтеграція з сервісами AWS може обмежити його використання в гібридних або мультихмарних середовищах. Крім того, хоча готові пакети правил є всеосяжними, деяким організаціям можуть знадобитися додаткові інструменти для задоволення конкретних вимог безпеки.

Amazon Inspector є критично важливим інструментом для організацій, які прагнуть підвищити безпеку своїх робочих навантажень AWS. Його можливості автоматизації, інтеграції та безперервного сканування роблять його безцінним активом для виявлення та усунення вразливостей. Використовуючи Amazon Inspector, компанії можуть підвищити рівень безпеки, забезпечити відповідність нормативним стандартам і зміцнити довіру між зацікавленими сторонами.

2.3.6. Amazon Cognito

Amazon Cognito - це потужна та масштабована служба автентифікації та управління користувачами, яку надає Amazon Web Services (AWS). Вона призначена для задоволення потреб додатків в автентифікації, авторизації та управлінні користувачами. Amazon Cognito дозволяє розробникам швидко та безпечно додавати функціонал реєстрації, входу та управління доступом до веб- і мобільних додатків, забезпечуючи при цьому безперешкодний користувацький досвід та високі стандарти безпеки.

Сервіс поділений на два основні компоненти: пул користувачів (user pools) та пул ідентифікацій (identity pools). Пул користувачів — це каталоги користувачів, які забезпечують функціонал реєстрації та входу для додатків. Вони фактично є керованими каталогами користувачів, що масштабуються для підтримки мільйонів користувачів. Розробники можуть використовувати пул користувачів для автентифікації за допомогою комбінації імені користувача та пароля або через сторонніх постачальників ідентифікації, таких як Facebook, Google, Apple або Amazon. Пули користувачів Cognito пропонують функції, такі як багатофакторна автентифікація (MFA), відновлення облікового запису, перевірка електронної пошти та номера телефону, а також налаштовані робочі процеси за допомогою тригерів AWS Lambda.

Пул ідентифікацій, з іншого боку, використовується для надання користувачам тимчасового обмеженого доступу до ресурсів AWS. Цей компонент дозволяє розробникам автентифікувати користувачів через різні постачальники ідентифікації, включаючи пули користувачів Amazon Cognito, зовнішні постачальники, такі як Google або Facebook, або навіть власні кастомні постачальники. Пули ідентифікацій дозволяють розробникам призначати автентифікованим чи неавтентифікованим користувачам певні ролі IAM, забезпечуючи точний контроль над тим, до яких ресурсів користувачі мають доступ.

Amazon Cognito підтримує інтеграцію з OpenID Connect (OIDC), OAuth 2.0 та SAML, що робить його надзвичайно гнучким сервісом, який вписується в різні екосистеми додатків. Ця гнучкість забезпечує сумісність із численними сторонніми сервісами та корпоративними системами, надаючи розробникам широкий спектр можливостей для впровадження автентифікації та авторизації.

Однією з визначних особливостей Amazon Cognito є його здатність забезпечувати детальне управління користувачами. Розробники можуть визначати групи користувачів, призначати спеціальні атрибути та встановлювати детальні політики для контролю взаємодії користувачів із додатками. Наприклад, адміністратори можуть створювати кастомні робочі процеси автентифікації, використовуючи функції AWS Lambda, які запускаються за певними подіями, такими як реєстрація чи спроба входу користувача. Ці робочі процеси можуть впроваджувати додаткові заходи безпеки, перевіряти спеціальні атрибути або інтегруватися з сторонніми сервісами.

Безпека є головним пріоритетом для Amazon Cognito, який використовує шифрування за галузевими стандартами для зберігання та передачі конфіденційних даних. Він також підтримує багатофакторну автентифікацію (MFA), додаючи додатковий рівень безпеки для облікових записів користувачів. Окрім вбудованих функцій безпеки, Amazon Cognito відповідає різним нормативним вимогам та стандартам, таким як GDPR, що робить його придатним для додатків, які потребують високого рівня відповідності.

Amazon Cognito безперешкодно інтегрується з іншими сервісами AWS, підвищуючи його корисність у розробці додатків. Наприклад, він добре працює з Amazon API Gateway для автентифікації запитів API, AWS AppSync для GraphQL API та AWS Lambda для виконання серверної логіки. Крім того, розробники можуть використовувати Amazon CloudWatch для моніторингу метрик і логів, пов'язаних із активністю користувачів, що допомагає виявляти та усувати потенційні проблеми.

Налаштування є ще однією ключовою перевагою Amazon Cognito. Розробники можуть адаптувати вигляд і функціональність інтерфейсів реєстрації та входу, щоб відповідати бренду їхніх додатків. Це включає кастомізацію шаблонів електронної пошти та SMS для сповіщень користувачів, а також додавання спеціальних полів до профілю користувача. Такі опції забезпечують узгоджений і впізнаваний користувацький досвід.

Таким чином, Amazon Cognito - це комплексний, безпечний і гнучкий сервіс для управління автентифікацією та авторизацією користувачів у сучасних додатках. Підтримка пулів користувачів і пулів ідентифікацій, інтеграція з різними постачальниками ідентифікації, надійні функції безпеки та безшовна інтеграція з іншими сервісами AWS роблять його ідеальним вибором для розробників, які створюють масштабовані та безпечні додатки. Відмовляючись від складнощів автентифікації та управління користувачами, Amazon Cognito дозволяє розробникам зосередитися на основній функціональності додатка, що в кінцевому результаті прискорює процес розробки та покращує досвід кінцевих користувачів.

Висновки до розділу 2

Було досліджено хмарні сервіси від компанії Amazon - лідера в сфері хмарних технологій. Amazon надає якісну інфраструктуру IaaS для побудови надійних REST API за допомогою веб-сервісів AWS. Було встановлено, що сервіси API Gateway, CloudWatch, AWS IAM, Cognito, Inspector, CloudTrail надають ефективні рішення для захисту від найбільш актуальних вразливостей. Завдяки цим веб-сервісам покриваються основні складові безпеки REST: авторизація, логування та його аудит, автентифікація, шифрування та обмежені кількості запитів [34]. В комплексі ці сервіси надають багаторівневий підхід до безпеки, який допомагає захистити REST API від загроз, забезпечуючи масштабовані, безпечні та прості в обслуговуванні сервіси, стимулюючи інновації та ефективність сучасних програмних екосистем

РОЗДІЛ 3. ВИКОРИСТАННЯ ХМАРНИХ ТЕХНОЛОГІЙ ДЛЯ ЗАХИСТУ REST API ДОДАТКУ

3.1. Розробка REST API додатку

Розробимо REST API додаток бухгалтерського обліку, котрий буде називатись “Bookkeeping”. Для розробки додатку обрані наступні технології: операційна система — Ubuntu 24.04, сервер - Apache2 мова програмування - PHP 8.3, фрейворк - Laravel 11, СУБД — MySQL, репозиторій — GitHub.

Додаток буде мати наступні сутності, таблиця 3.1.

Таблиця 3.1. - Сутності додатку

Назва сутності	Призначення
Invoices	Документи, видані клієнтам, які деталізують надані товари чи послуги разом із сумою до оплати
Customers	Особи або підприємства, які купують товари чи послуги у компанії.
Invoice payments	Оплати, отримані від клієнтів для погашення неоплачених рахунків-фактур
Accounts	Структурований список усіх фінансових рахунків у головній книзі компанії, який використовується для категоризації транзакцій
Vendors	Постачальники або сервісні компанії, у яких компанія купує товари чи послуги.
Bills	Записи витрат або сум, які необхідно сплатити постачальникам за отримані товари чи послуги.

Назва сутності	Призначення
Bill payments	Платежі, здійснені постачальникам для погашення неоплачених рахунків.

На рисунку 3.1 представлена логічна модель даних додатку, що описує зв'язок між його сутностями.

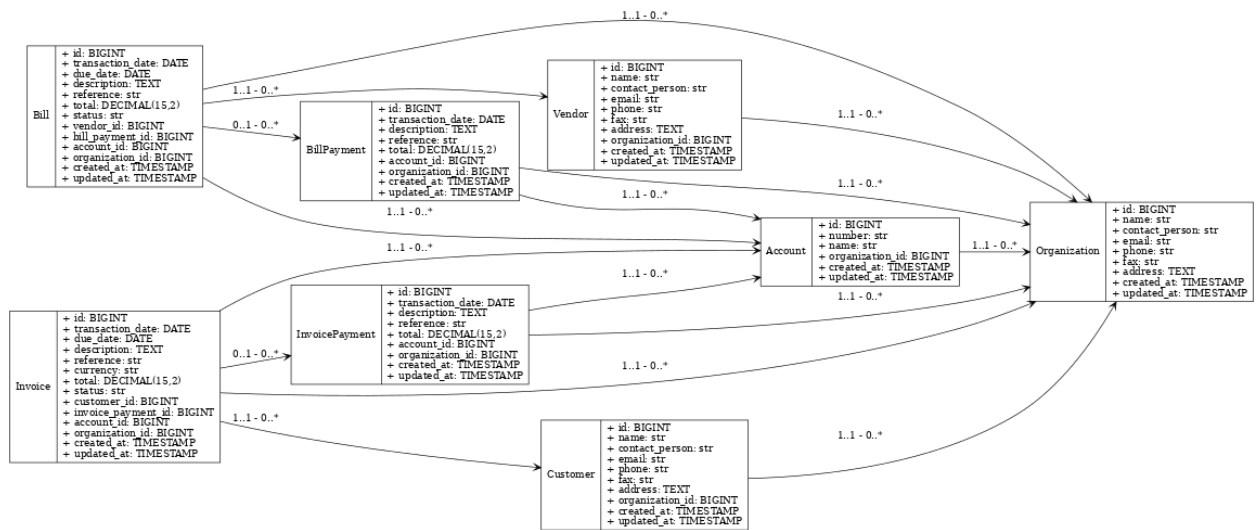


Рис. 3.1. Логічна модель даних.

Фреймворк Laravel надає можливість за допомогою однієї команди, створити модель, контролер та міграцію для сутності [40]. Нижче наведені команди для створення моделей, контролерів та міграцій для кожної сутності.

```
php artisan make:model Organization -mcr
```

```
php artisan make:model Account -mcr
```

```
php artisan make:model Customer -mcr
```

```
php artisan make:model Invoice -mcr
```

```
php artisan make:model InvoicePayment -mcr
```

```
php artisan make:model Vendor -mcr
```

```
php artisan make:model Bill -mcr
```

```
php artisan make:model BillPayment -mcr
```

Після запуску міграцій, скористаємось програмою MySQL Workbench для побудови фізичної схеми бази даних додатку “Bookkeeping”. Схеми бази даних “Bookkeeping” зображена на рис 3.2.

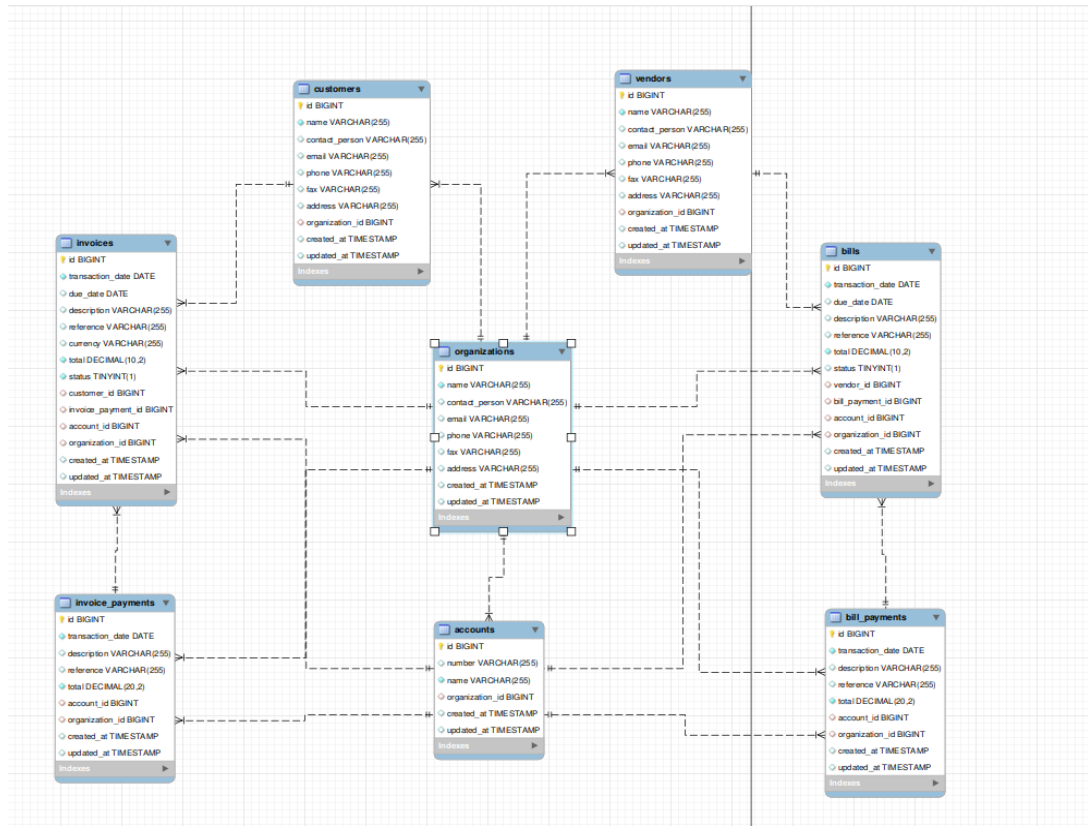


Рис. 3.3. Фізична модель даних додатку «Bookkeeping»

Створемо для кожної сутності унікальний ідентифікатор (URI). Це можна зробити в файлі `routes/api.php`, призначити назву сутності до назви контролера та вибрати HTTP метод, за яким буде доступний ресурс. Список усіх URI для додатку зображено на рисунк 3.3.

```

php api.php x
23
24 // CustomerController
25 Route::post( uri: 'customers', [CustomerController::class, 'store']);
26 Route::get( uri: '/customers', [CustomerController::class, 'index']);
27 Route::put( uri: '/customers/{customerId}', [CustomerController::class, 'update']);
28 Route::delete( uri: '/customers/{customerId}', [CustomerController::class, 'destroy']);
29 Route::get( uri: '/customers/{customerId}', [CustomerController::class, 'get']);
30 // VendorController
31 Route::post( uri: '/vendors', [VendorController::class, 'store']);
32 Route::get( uri: '/vendors', [VendorController::class, 'index']);
33 Route::put( uri: '/vendors/{vendorId}', [VendorController::class, 'update']);
34 Route::delete( uri: '/vendors/{vendorId}', [VendorController::class, 'destroy']);
35 Route::get( uri: '/vendors/{vendorId}', [VendorController::class, 'get']);
36 // AccountController
37 Route::post( uri: '/accounts', [AccountController::class, 'store']);
38 Route::get( uri: '/accounts', [AccountController::class, 'index']);
39 Route::put( uri: '/accounts/{accountId}', [AccountController::class, 'update']);
40 Route::delete( uri: '/accounts/{accountId}', [AccountController::class, 'destroy']);
41 Route::get( uri: '/accounts/{accountId}', [AccountController::class, 'get']);
42 // BillController
43 Route::post( uri: '/bills', [BillController::class, 'store']);
44 Route::get( uri: '/bills', [BillController::class, 'index']);
45 Route::put( uri: '/bills/{billId}', [BillController::class, 'update']);
46 Route::delete( uri: '/bills/{billId}', [BillController::class, 'destroy']);
47 Route::get( uri: '/bills/{billId}', [BillController::class, 'get']);
48 // InvoiceController
49 Route::post( uri: '/invoices', [InvoiceController::class, 'store']);
50 Route::get( uri: '/invoices', [InvoiceController::class, 'index']);
51 Route::put( uri: '/invoices/{invoiceId}', [InvoiceController::class, 'update']);
52 Route::delete( uri: '/invoices/{invoiceId}', [InvoiceController::class, 'destroy']);
53 Route::get( uri: '/invoices/{invoiceId}', [InvoiceController::class, 'get']);
54 // OrganizationController
55 Route::post( uri: '/organizations', [OrganizationController::class, 'store']);
56 Route::get( uri: '/organizations', [OrganizationController::class, 'index']);
57 Route::put( uri: '/organizations/{organizationId}', [OrganizationController::class, 'update']);
58 Route::delete( uri: '/organizations/{organizationId}', [OrganizationController::class, 'destroy']);
59 Route::get( uri: '/organizations/{organizationId}', [OrganizationController::class, 'get']);
60

```

Рис. 3.3. Список URI для кожного ресурсу «Bookkeeping»

Зробимо документація для додатку, з описом кожного ресурсу та можливістю роботи API запити через графічний інтерфейс, використаємо для цього OpenAPI [39]. OpenAPI - це інструмент для використання та візуалізації ресурсів REST API, раніше був відомий як Swagger [38]. Таку документацію можна згенерувати з анотацій до кожного метода додатку. Приклад анотації, котру необхідно створити та розмістити перед методом в контролері сутності зображено на рис. 3.4.

```

1  k?php
2
3  namespace App\Http\Controllers;
4
5  > use ...
6
7
8  /**
9   * @OA\Tag(
10   *     name="Vendors",
11   *     description="API Vendors for managing accounts"
12   * )
13   */
14  6 usages  ± Roman Storcheus *
15  class VendorController extends Controller
16  {
17      /**
18       * @OA\Get(
19       *     path="/vendors",
20       *     summary="Retrieve a list of vendors",
21       *     tags={"Vendors"},
22       *     @OA\Response(
23       *         response=200,
24       *         description="List of vendors",
25       *         @OA\JsonContent(
26       *             type="array",
27       *             @OA\Items(
28       *                 type="object",
29       *                 @OA\Property(property="id", type="integer", example=1),
30       *                 @OA\Property(property="name", type="string", example="John Doe"),
31       *                 @OA\Property(property="contact_person", type="string", example="Jane Smith"),
32       *                 @OA\Property(property="email", type="string", format="email", example="john.doe@example.com"),
33       *                 @OA\Property(property="phone", type="string", example="123-456-7890"),
34       *                 @OA\Property(property="fax", type="string", example="123-456-7891"),
35       *                 @OA\Property(property="address", type="string", example="123 Elm Street, Springfield, USA"),
36       *                 @OA\Property(property="created_at", type="string", format="date-time", example="2023-12-20T12:34:56Z"),
37       *                 @OA\Property(property="updated_at", type="string", format="date-time", example="2023-12-20T12:34:56Z")
38       *             )
39       *         )
40     )
41 }

```

Рис. 3.4. Анотація для генерації документації OpenAPI

Результ згенерованої документації через OpenAPI можна побачити на рис. 3.6.

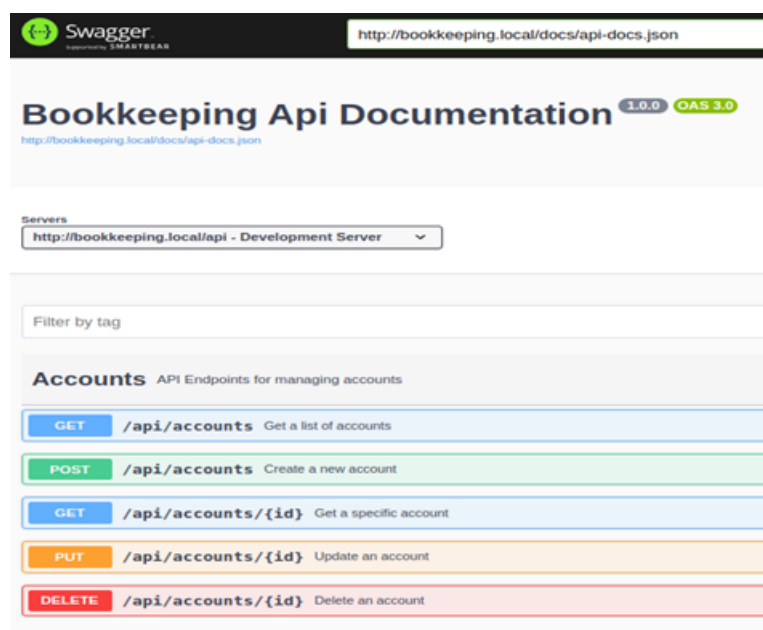


Рис. 3.5. Згенерована документація OpenAPI

Даний додаток відповідає архітектурі REST. Додаток є сервером, хто робить запит до застосунку є клієнтом. Не має збереження станів, кожен запит є автономним і незалежним. Всі ресурси доступні за стандартними методами HTTP: GET, POST, PUT, DELETE. Продемонструвати запити та відповіді від сервера можна за допомогою OpenAPI та програми Postman [37]. Зробимо запит на отримання списку Vendors за допомогою документації OpenAPI для цього необхідно натиснути кнопку “Execute”, рисунок 3.6. Відповідь на даний запит зображено на рисунок 3.7, запит через Postman – рисунок 3.8.

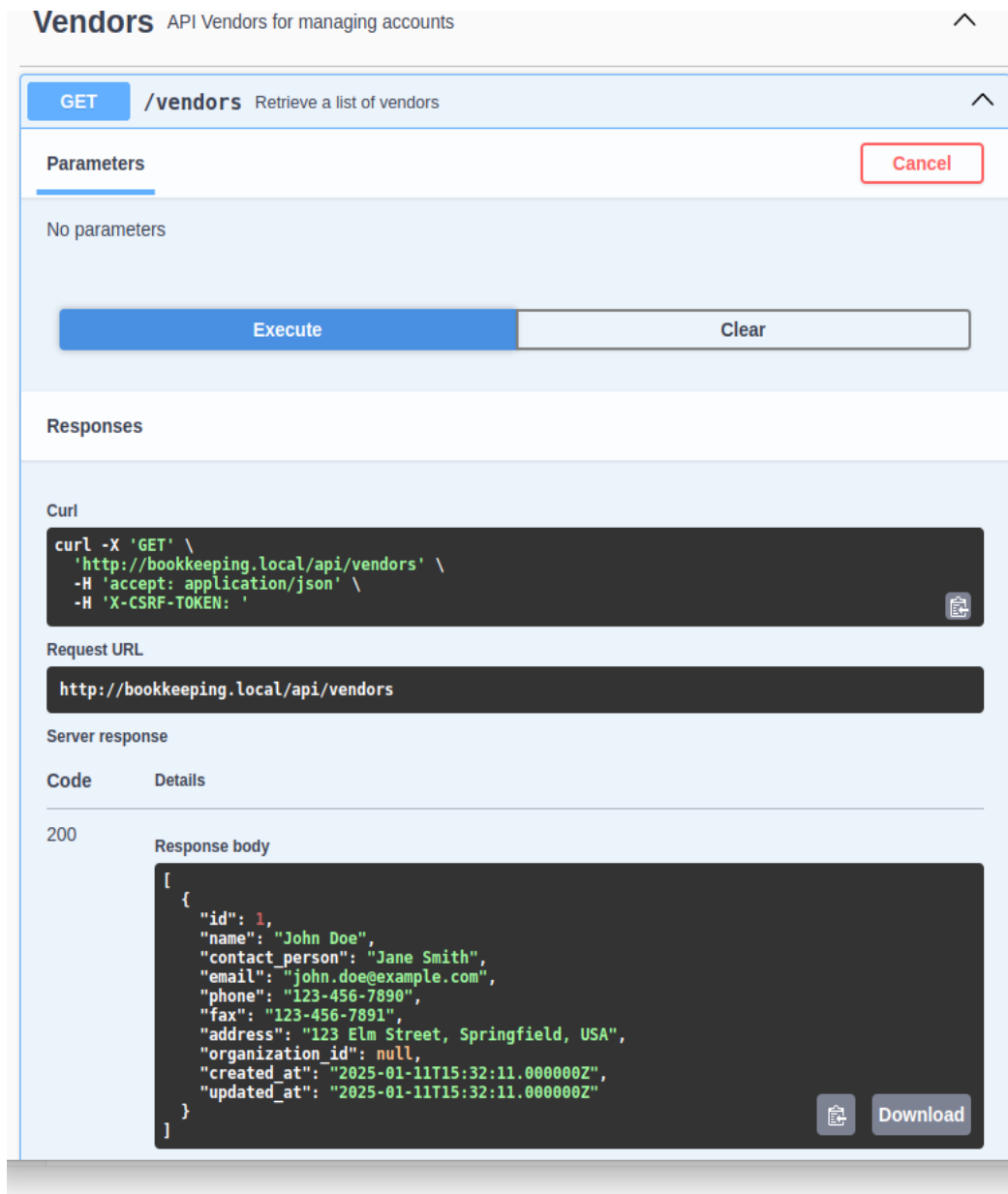


Рис. 3.6. Запит до ресурса Vendors

200

Response body

```
[
  {
    "id": 1,
    "name": "John Doe",
    "contact_person": "Jane Smith",
    "email": "john.doe@example.com",
    "phone": "123-456-7890",
    "fax": "123-456-7891",
    "address": "123 Elm Street, Springfield, USA",
    "organization_id": null,
    "created_at": "2025-01-11T15:32:11.000000Z",
    "updated_at": "2025-01-11T15:32:11.000000Z"
  }
]
```

Response headers

```
access-control-allow-origin: *
cache-control: no-cache,private
connection: Keep-Alive
content-type: application/json
date: Thu, 16 Jan 2025 19:11:08 GMT
keep-alive: timeout=5,max=100
server: Apache/2.4.58 (Ubuntu)
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	List of vendors	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
[
  {
    "id": 1,
    "name": "John Doe",
    "contact_person": "Jane Smith",
    "email": "john.doe@example.com",
    "phone": "123-456-7890",
    "fax": "123-456-7891",
    "address": "123 Elm Street, Springfield, USA",
    "created_at": "2023-12-20T12:34:56Z",
    "updated_at": "2023-12-20T12:34:56Z"
  }
]
```

Рис. 3.7. Відповідь на запит до ресурса Vendors

Даний додаток Bookkeeping повністю відповідає функціональній частині, а саме створення, оновлення, отримання та видалення записів, але в безпеці він має явні недоліки. Не має авторизації, звісно фреймворк Laravel має пакет для авторизації за протоколом OAuth2.0 або JWT-токен. Стосовно автентифікації також можна використати додаткові пакети. Для моніторингу та логування необхідно підключити окремі пакети. У випадку обмеження швидкості також можна використати пакети для відслідковування та обмеження швидкості запитів до API. Як наслідок необхідно використовувати все більшу кількість пакетів, котрі також можуть мати вразливості та збільшують залежність від сторонніх компонентів. Ще одним недоліком є фрагментарність, тому що кожен пакет є окремим – не має комплексного рішення для захисту додатку.

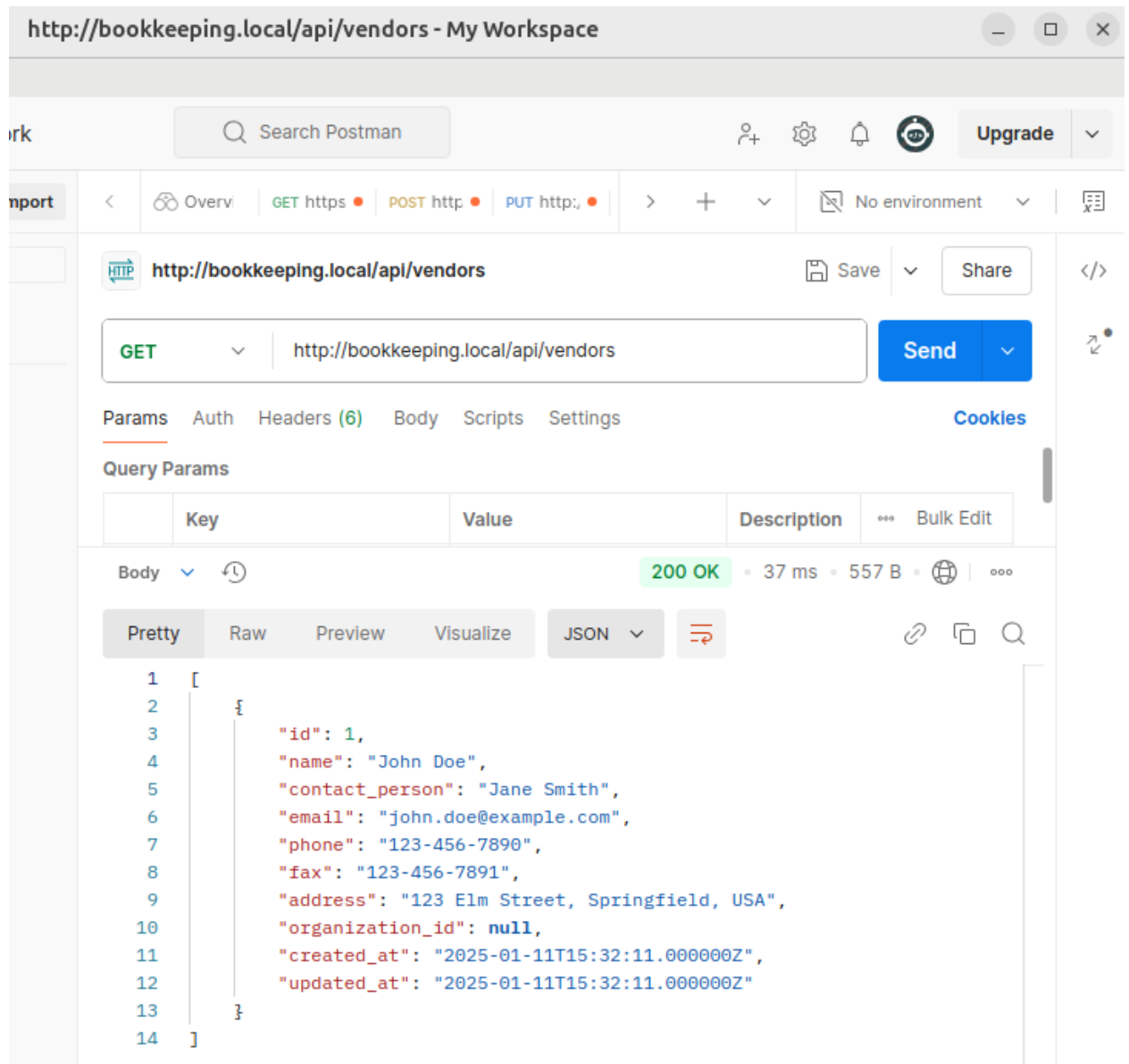


Рис. 3.8. Запит та відповідь до ресурса через Postman

3.2. Розміщення додатку в хмарному середовищі AWS

Після створення аккаунту в системі AWS, отримуємо доступ до консолі хмарних сервісів, котра зображена на рисунок 3.9.

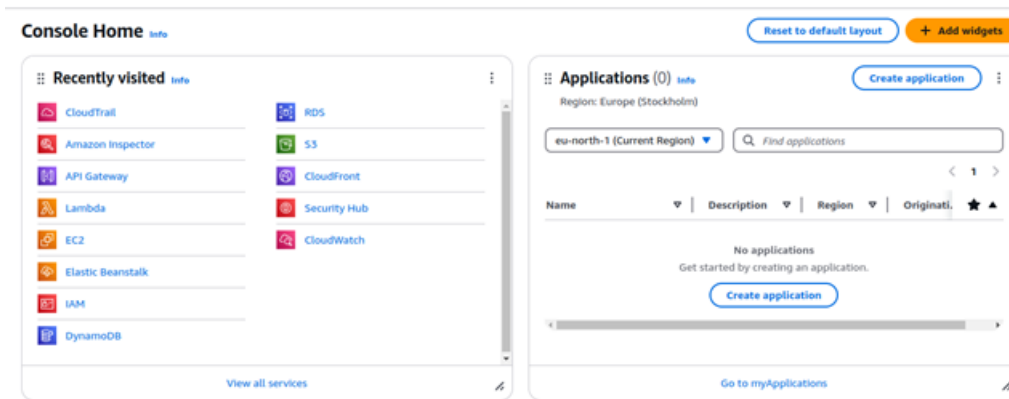


Рис. 3.9. Консоль AWS

Для розміщення файлів в хмарному провайдері треба вибрати сервіс EC2. Наступним кроком є створення інстансу EC2, де фізично будуть знаходитися файли додатку, натиснувши кнопку «Launch instances». Відкриється сторінка створення самого інстансу, де необхідно вести назву проект «bookkeeping_api». Головний параметр при налаштуванні інстансу є операційна система. В якості операційну систему виберемо Ubuntu 24.04 LTS, рисунок 3.10.

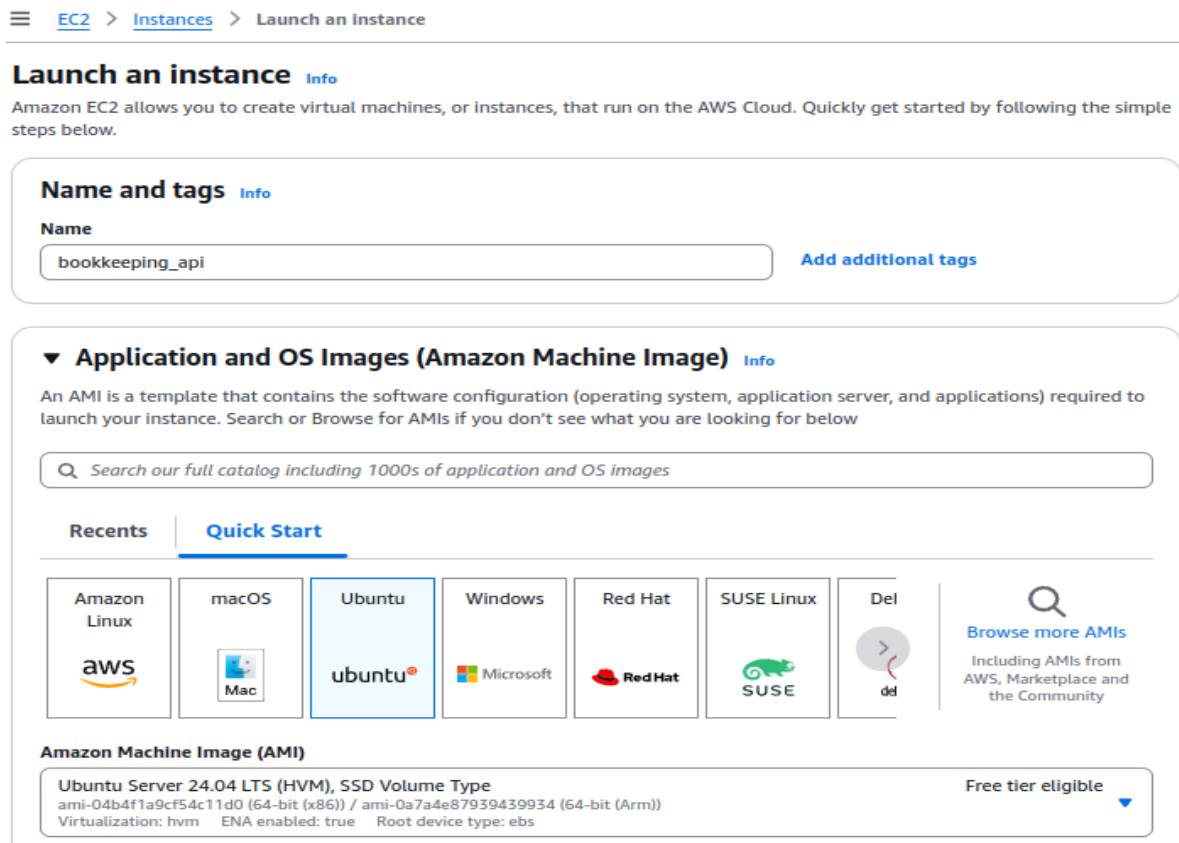
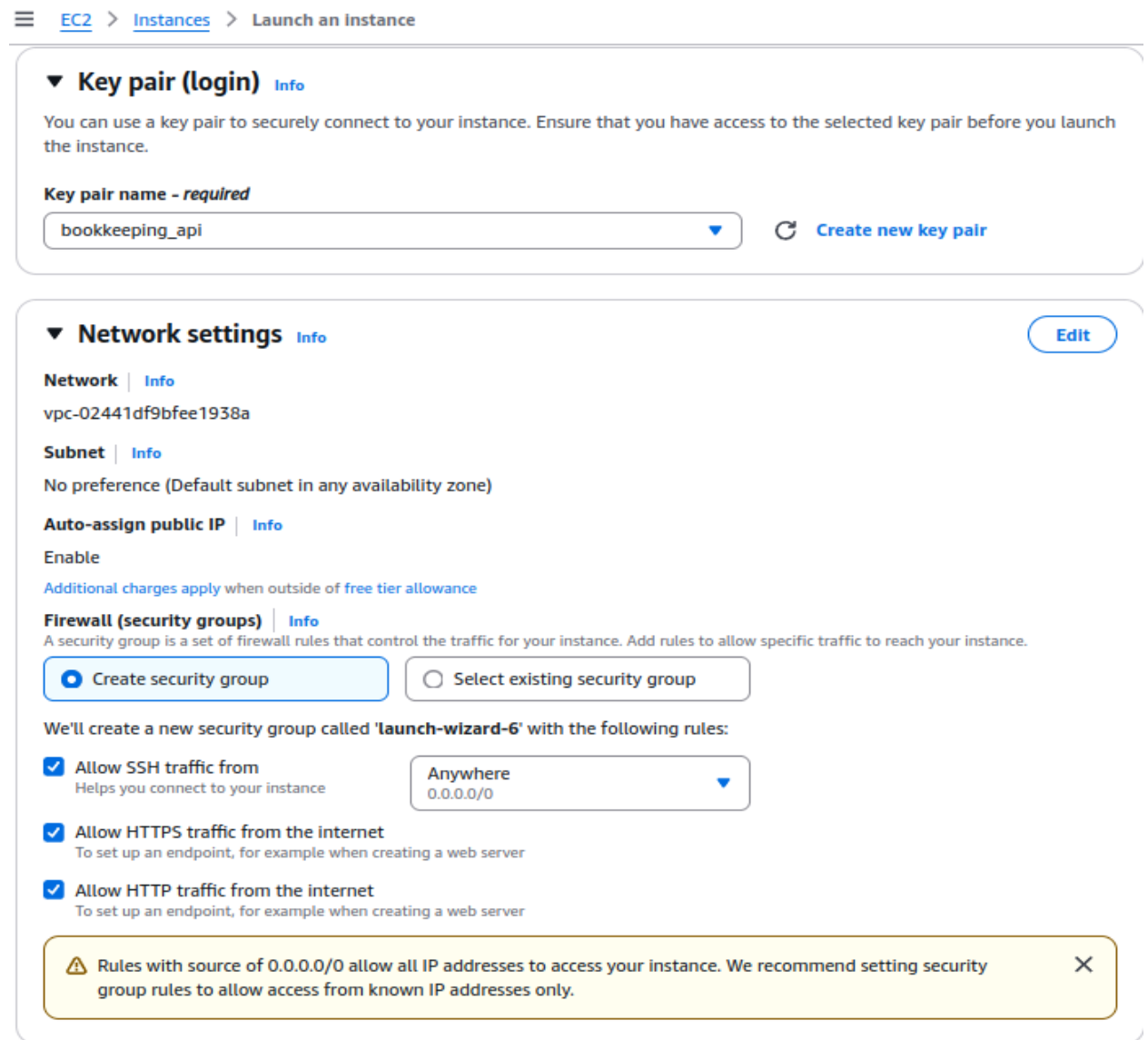


Рис. 3.10. Налаштування інстансу EC2.

Також необхідно вибрати Key pair для безпечного завантаження файлів на інстанс EC2 через SSH протокол, в подальшому цей ключ буде використаний для підключення до інстансу. Останнє налаштування стосується «Network settings» - мережі, щоб інстанс міг отримувати трафік через протоколи HTTP/HTTPS виберемо їх в налаштуваннях, рисунок 3.11. За замовчуванням інстанс отримує трафік лише через протокол SSH.



☰ [EC2](#) > [Instances](#) > Launch an Instance

▼ **Key pair (login)** [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

bookkeeping_api ▼ [Create new key pair](#)

▼ **Network settings** [Info](#) [Edit](#)

Network | [Info](#)

vpc-02441df9bfee1938a

Subnet | [Info](#)

No preference (Default subnet in any availability zone)

Auto-assign public IP | [Info](#)

Enable

[Additional charges apply when outside of free tier allowance](#)

Firewall (security groups) | [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group ☐ Select existing security group

We'll create a new security group called 'launch-wizard-6' with the following rules:

- ☒ Allow SSH traffic from Anywhere
0.0.0.0/0 ▼
Helps you connect to your instance
- ☒ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server
- ☒ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only. ✕

Рис. 3.11. Налаштування входного трафіку для інстансу EC2

Після створення інстанса йому призначається унікальний ідентифікатор, рисунок 3.12. Також для інстанса є можливість його моніторингу, скільки ресурсів і яких саме він споживає, рисунок 3.13. Додатково можна

налаштувати alarms для інстанса, у випадку якщо треба автоматично зробити сповіщення, наприклад підвищення навантаження на сервер.

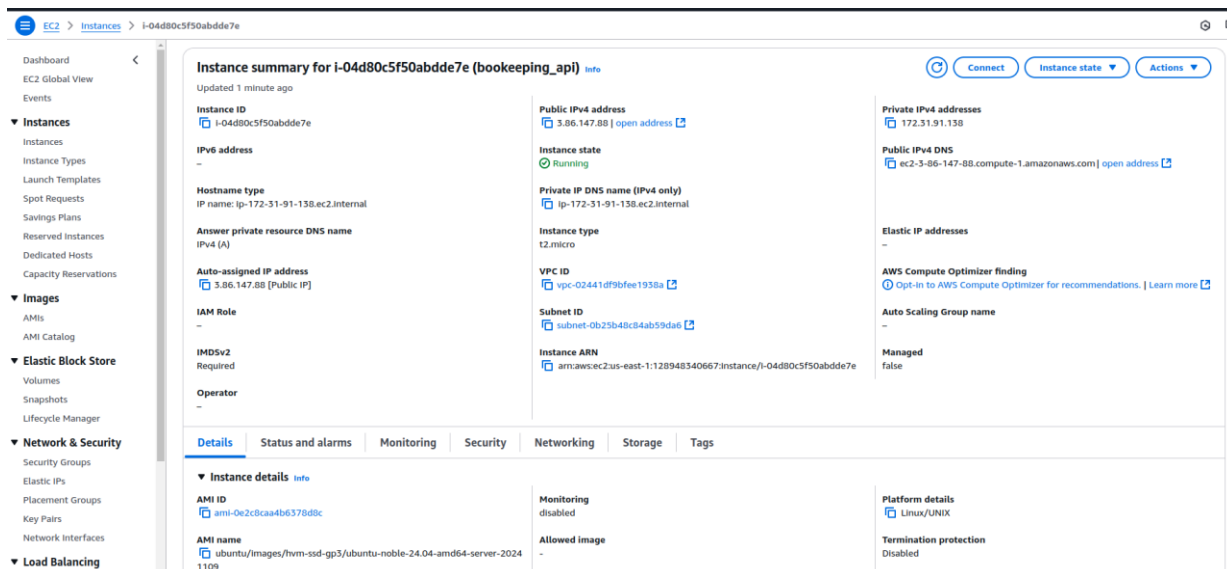


Рис. 3.12. Створений інстанс в EC2

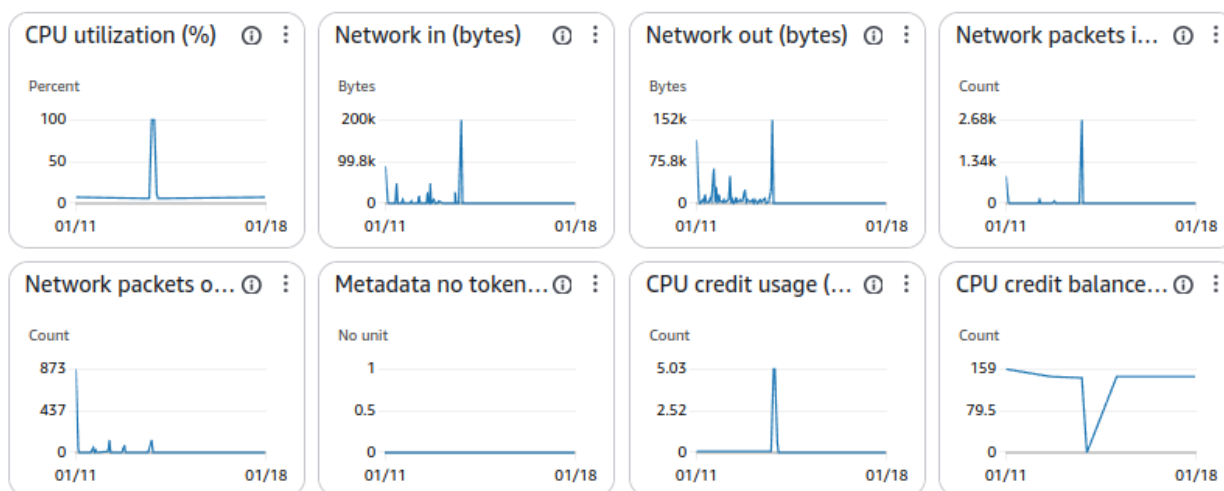


Рис. 3.13. Споживання ресурсів інстансом EC2.

Після створення інстансу EC2, можемо завантажити додаток підключившись до інстанса через протокол SSH. Публічною IP-адресою для додатку є 3.88.147.88, за цією адресою перевіримо список ресурсів додатку, рисунок 3.14.

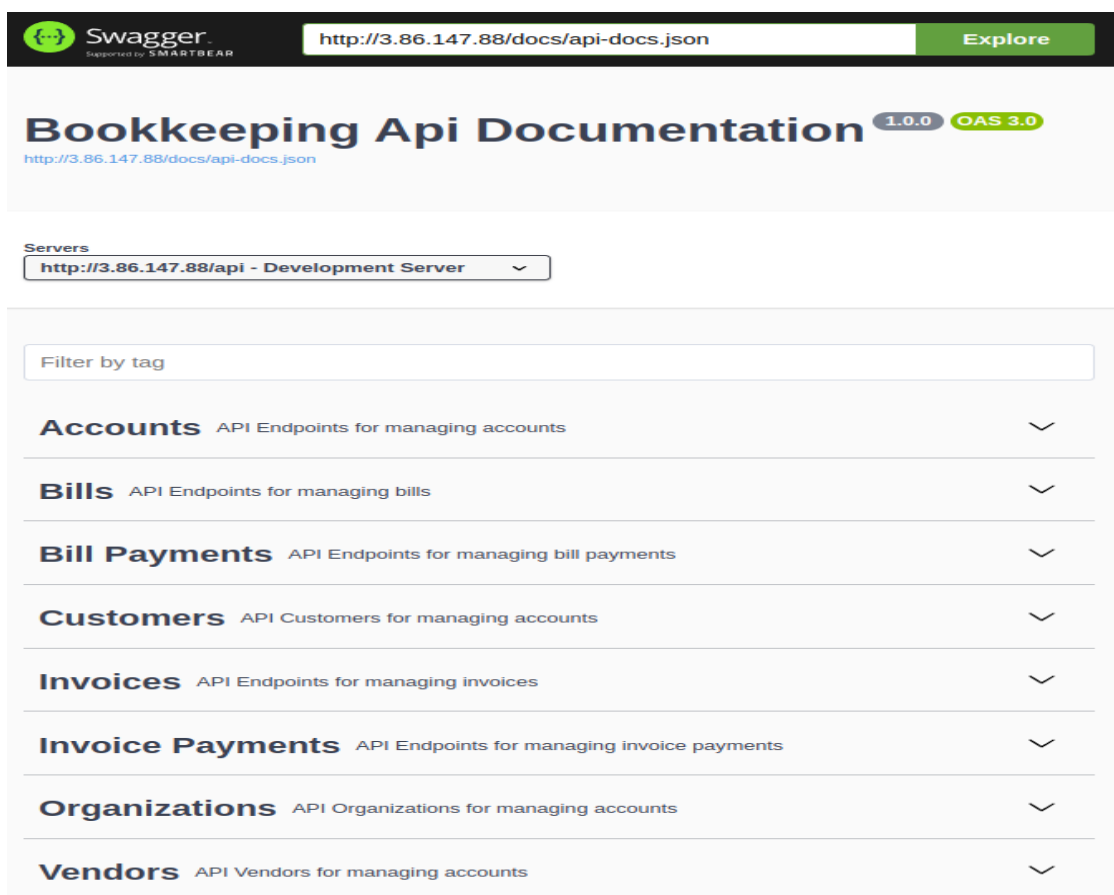


Рис. 3.14. Ресурси додатку на інстансі EC2.

3.3. Використання хмарних сервісів AWS для безпеки

Наступний етап, буде створення Amazon API Gateway для додатку, щоб покращити його безпеку. API Gateway пропонує вибрати тип API, котрий необхідно створити. Необхідно вибрати тип – REST API, рисунок 3.15.

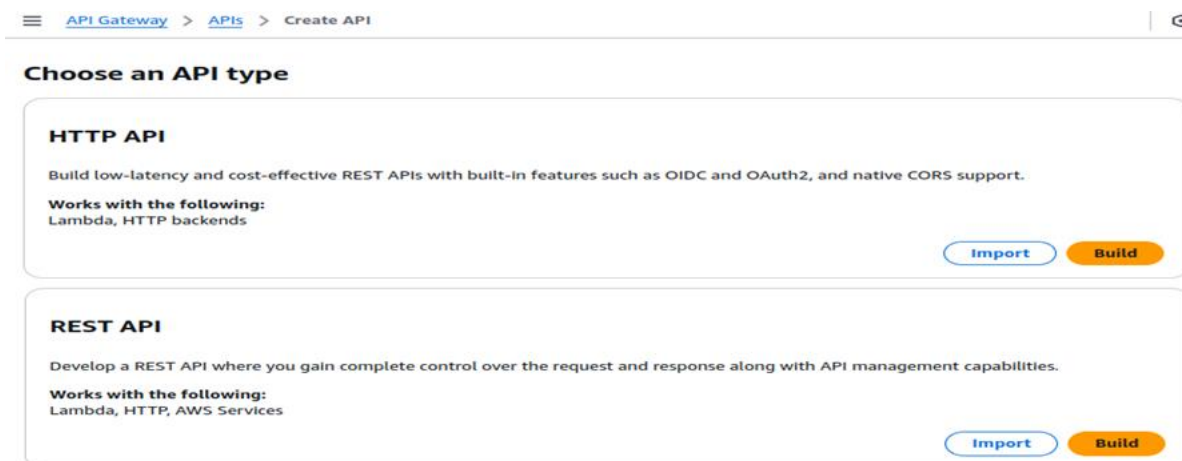


Рис. 3.15. Створення REST API в API Gateway.

В подальшому налаштування API Gateway інтуїтивно зрозуміле, необхідно ввести ім'я «Bookkeeping» та вибрати тип API – new API та натиснути на кнопку “Create API”, як зображено на рисуюнок 3.16.

API Gateway > APIs > Create API > Create REST API

Create REST API

API details

☒ **New API**
Create a new REST API.

☐ **Clone existing API**
Create a copy of an API in this AWS account.

☐ **Import API**
Import an API from an OpenAPI definition.

☐ **Example API**
Learn about API Gateway with an example API.

API name
Bookkeeping

Description - optional

API endpoint type
Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.
Regional

Cancel Create API

Рис. 3.16. Налаштування REST API в API Gateway

Після створення екземпляру API Gateway, можете перейти безпосередньо для створення ресурсів для додатку Bookkeeping. Створимо для початку ресурси для сутностей Vendors та Invoices. Для цього необхідно натиснути на кнопку «Create resource» та ввести ім'я «Resource name» - Invoices, як зображено на рисунку 3.17.

API Gateway > APIs > Resources - Bookkeeping (t4qrbislr9) > Create resource

Create resource

Resource details

☒ **Proxy resource** [Info](#)
Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

Resource path
/

Resource name
invoices

☐ **CORS (Cross Origin Resource Sharing)** [Info](#)
Create an OPTIONS method that allows all origins, all methods, and several common headers.

Cancel Create resource

Рис. 3.17. Створення ресурсу Invoices

В свою чергу після створення ресурсу, для кожного ресурсу необхідно створити метод для запиту на ресурс, натиснувши на кнопку «Create method», рисунок 3.18. Зробимо базові запити для створення, оновлення, отримання та видалення. Відповідно до цих операцій використаємо HTTP методи: POST, PUT, GET, DELETE, рисунок 3.19.

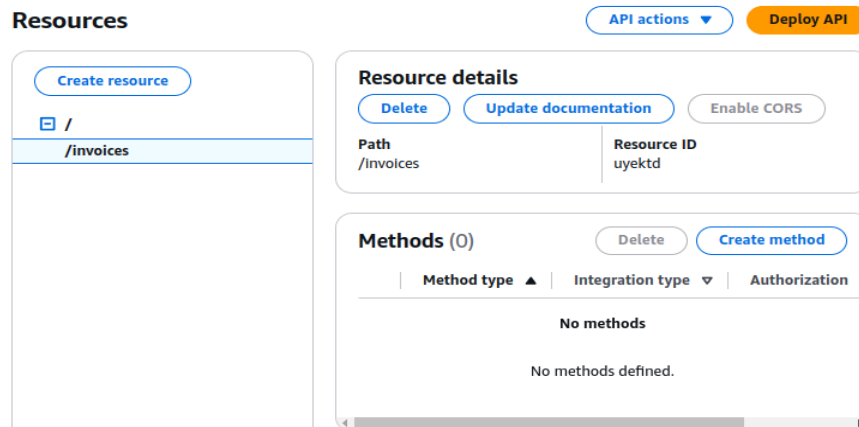


Рис. 3.18. Створення методу для ресурсу Invoices

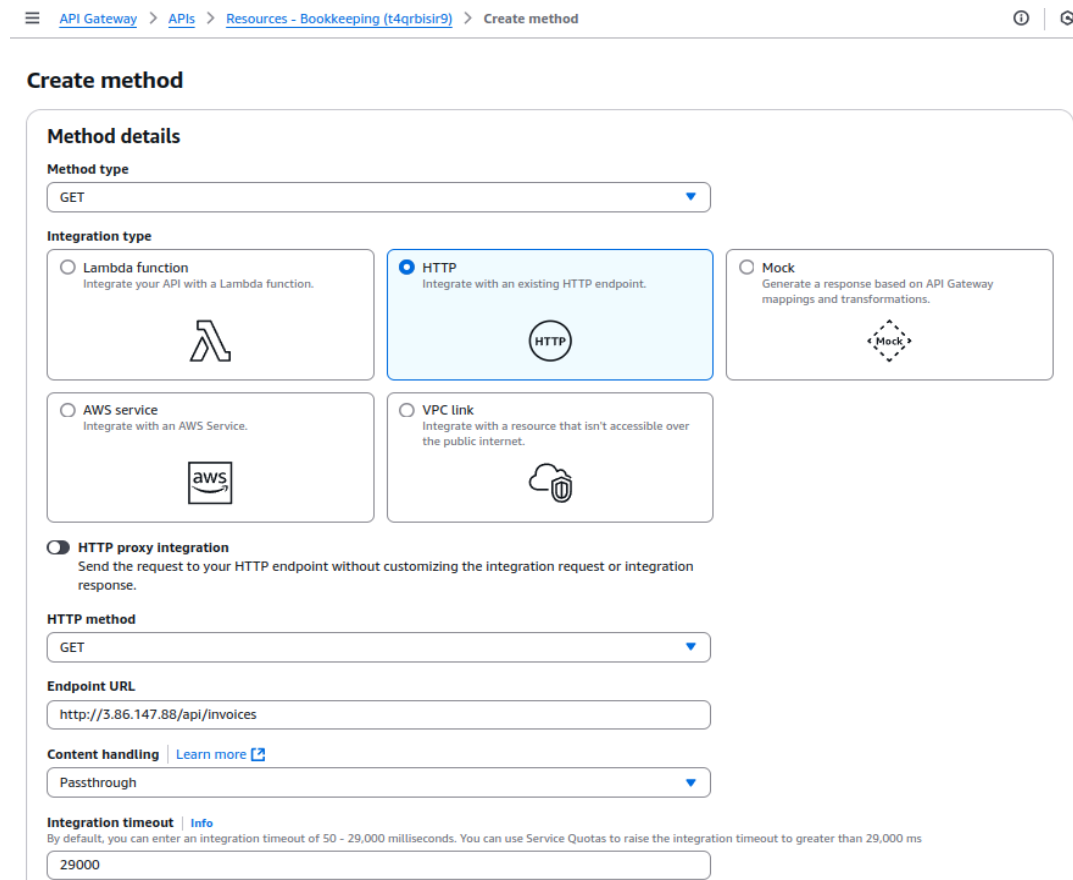


Рис. 3.19. Налаштування методу для ресурсу Invoices

Зручною функцією в API Gateway, що кожен метод ресурсу можна тестувати одразу в панелі ресурсів і також отримати детальний лог, стосовно запиту та відповіді до ресурсу, рисунок 3.20.

<div> <div> <div></div> <div>/vendors - GET method test results</div> </div> <div> <div>Request</div> <div>/vendors</div> </div> <div> <div>Latency ms</div> <div>18</div> </div> <div> <div>Status</div> <div>200</div> </div> </div>		
<div>Response body</div> <pre>[{"id":1,"name":"John Smith","contact_person":"Jane Smith","email":"john.smith@example.com","phone":"555-456-7890","fax":"777-456-7891","address":"65 Jordan Street, Texas, USA","organization_id":null,"created_at":"2025-01-11T15:41:24.000000Z","updated_at":"2025-01-11T15:41:24.000000Z"}, {"id":2,"name":"John Doe","contact_person":"Jane Smith","email":"john.doe@example.com","phone":"123-456-7890","fax":"123-456-7891","address":"123 Elm Street, Springfield, USA","organization_id":null,"created_at":"2025-01-11T15:42:37.000000Z","updated_at":"2025-01-11T15:42:37.000000Z"}]</pre>		
<div>Response headers</div> <pre>{ "Content-Type": "application/json", "X-Amzn-Trace-Id": "Root=1-6784e6db-989d24d4d90c4bc15337d29a" }</pre>		
<div>Logs</div> <pre>Execution log for request a8988e9d-c571-499a-a4ec-f53030c2ec4a Mon Jan 13 10:11:39 UTC 2025 : Starting execution for request: a8988e9d-c571-499a-a4ec-f53030c2ec4a Mon Jan 13 10:11:39 UTC 2025 : HTTP Method: GET, Resource Path: /vendors Mon Jan 13 10:11:39 UTC 2025 : Method request path: {} Mon Jan 13 10:11:39 UTC 2025 : Method request query string: {} Mon Jan 13 10:11:39 UTC 2025 : Method request headers: {} Mon Jan 13 10:11:39 UTC 2025 : Method request body before transformations: Mon Jan 13 10:11:39 UTC 2025 : Endpoint request URI: http://3.86.147.88/api/vendors Mon Jan 13 10:11:39 UTC 2025 : Endpoint request headers: {x-amzn-apigateway-api-id=ruawcb00pc, Accept=application/json, User-Agent=AmazonAPIGateway_ruawcb00pc, X-Amzn-Trace-Id=Root=1-6784e6db-989d24d4d90c4bc15337d29a} Mon Jan 13 10:11:39 UTC 2025 : Endpoint request body after transformations: Mon Jan 13 10:11:39 UTC 2025 : Sending request to http://3.86.147.88/api/vendors Mon Jan 13 10:11:39 UTC 2025 : Received response. Status: 200, Integration latency: 15 ms Mon Jan 13 10:11:39 UTC 2025 : Endpoint response headers: {Date=Mon, 13 Jan 2025 10:11:39 GMT, Server=Apache/2.4.58 (Ubuntu), Cache-Control=no-cache, private, Access-Control-Allow-Origin=*, Keep-Alive=timeout=5, max=100, Connection=Keep-Alive, Transfer-Encoding=chunked, Content-Type=application/json} Mon Jan 13 10:11:39 UTC 2025 : Endpoint response body before transformations: [{"id":1,"name":"John Smith","contact_person":"Jane Smith","email":"john.smith@example.com","phone":"555-456-7890","fax":"777-456-7891","address":"65 Jordan Street, Texas, USA","organization_id":null,"created_at":"2025-01-11T15:41:24.000000Z","updated_at":"2025-01-11T15:41:24.000000Z"}, {"id":2,"name":"John Doe","contact_person":"Jane Smith","email":"john.doe@example.com","phone":"123-456-7890","fax":"123-456-7891","address":"123 Elm Street, Springfield, USA","organization_id":null,"created_at":"2025-01-11T15:42:37.000000Z","updated_at":"2025-01-11T15:42:37.000000Z"}] Mon Jan 13 10:11:39 UTC 2025 : Method response body after transformations: [{"id":1,"name":"John Smith","contact_person":"Jane Smith","email":"john.smith@example.com","phone":"555-456-7890","fax":"777-456-7891","address":"65 Jordan Street, Texas, USA","organization_id":null,"created_at":"2025-01-11T15:41:24.000000Z","updated_at":"2025-01-11T15:41:24.000000Z"}, {"id":2,"name":"John Doe","contact_person":"Jane Smith","email":"john.doe@example.com","phone":"123-456-7890","fax":"123-456-7891","address":"123 Elm Street, Springfield, USA","organization_id":null,"created_at":"2025-01-11T15:42:37.000000Z","updated_at":"2025-01-11T15:42:37.000000Z"}] Mon Jan 13 10:11:39 UTC 2025 : Method response headers: {X-Amzn-Trace-Id=Root=1-6784e6db-989d24d4d90c4bc15337d29a, Content-Type=application/json} Mon Jan 13 10:11:39 UTC 2025 : Successfully completed execution Mon Jan 13 10:11:39 UTC 2025 : Method completed with status: 200</pre>		

Рис. 3.20. Логи при запиті до ресурсу

Для того щоб покращити захист для кожного методу, використаємо сервіс AWS IAM для авторизації, щоб доступ до ресурсу мав лише авторизований користувач. Це можна зробити зайшовши в панель редагування методу ресурсу. Для прикладу зробимо авторизацію для ресурсу Invoices для кожного методу, рисунок 3.21.

Edit method request

Method request settings

Authorization
AWS IAM

Request validator
Validate body, query string parameters, and headers

☒ API key required

Operation name - optional
GetInvoices

Рис. 3.21. Додамо авторизація для метода Get Invoices

Список методів для ресурсів Vendors та Invoices зображено на рисунку 3,22.

Resources

API actions ▼ Deploy API

Create resource

/

/invoices

DELETE

GET

POST

PUT

/vendors

DELETE

GET

POST

PUT

Resource details Delete Update documentation Enable CORS

Path: /invoices Resource ID: 3suwwc

Methods (4) Delete Create method

	Method type ▲	Integration type ▼	Authorization ▼	API key ▼
<input type="radio"/>	DELETE	HTTP	IAM	Required
<input type="radio"/>	GET	HTTP	IAM	Required
<input type="radio"/>	POST	HTTP	IAM	Required
<input type="radio"/>	PUT	HTTP	IAM	Required

Рис. 3.21. Ресурси Vendors та Invoices

Настпнім важливоим сервісом для моніторингу та логування є CloudWatch. Моніторинг більності показників автоматично налаштований і можна їм користуватись через панель сервсу CloudWatch. Для логування необхідно налаштувати роль (role) та дозволи(permission). Призначемо назву ролі RoleApiGatewayCloudWatch та виберемо дозвіл AmazonApiGatewayPushToCloudWatchLogs, рисунок 3.22.

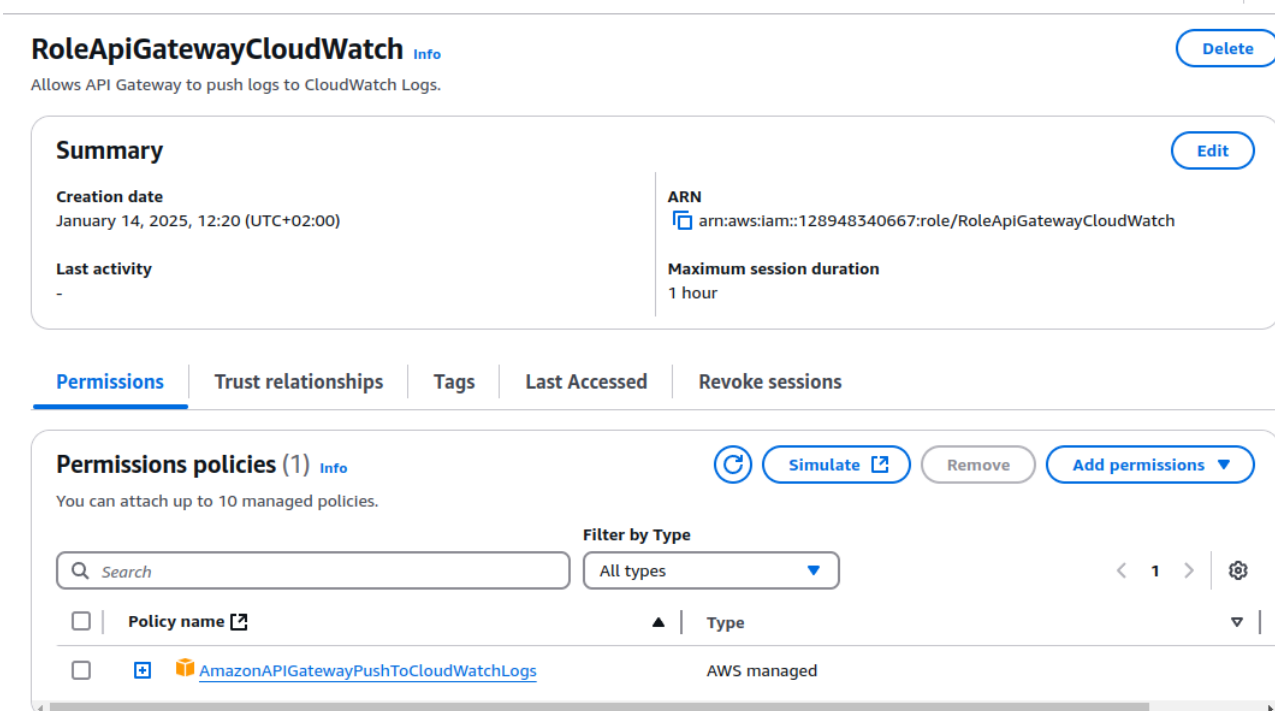


Рис. 3.22. Налаштування логів в CloudWatch

3.4. Тестування додатку на вразливості

Для перевірки додатку на вразливість стосовно обмеження кількості запитів (Rate Limiting) використаємо програму JMeter Apache. Функція обмеження кількості запитів захищає від атак типу «відмова в обслуговуванні», спроб входу в систему методом перебору паролів, стрибків трафіку та інших типів шкідливих дій, націлених на REST API. Для тестування ресурсу vendors створемо HTTP Request в програмі Jmeter з параметрами, як кількість запитів — 12000 та необхідно зазначити період часу — 1 секунда, рисунок 3.23.

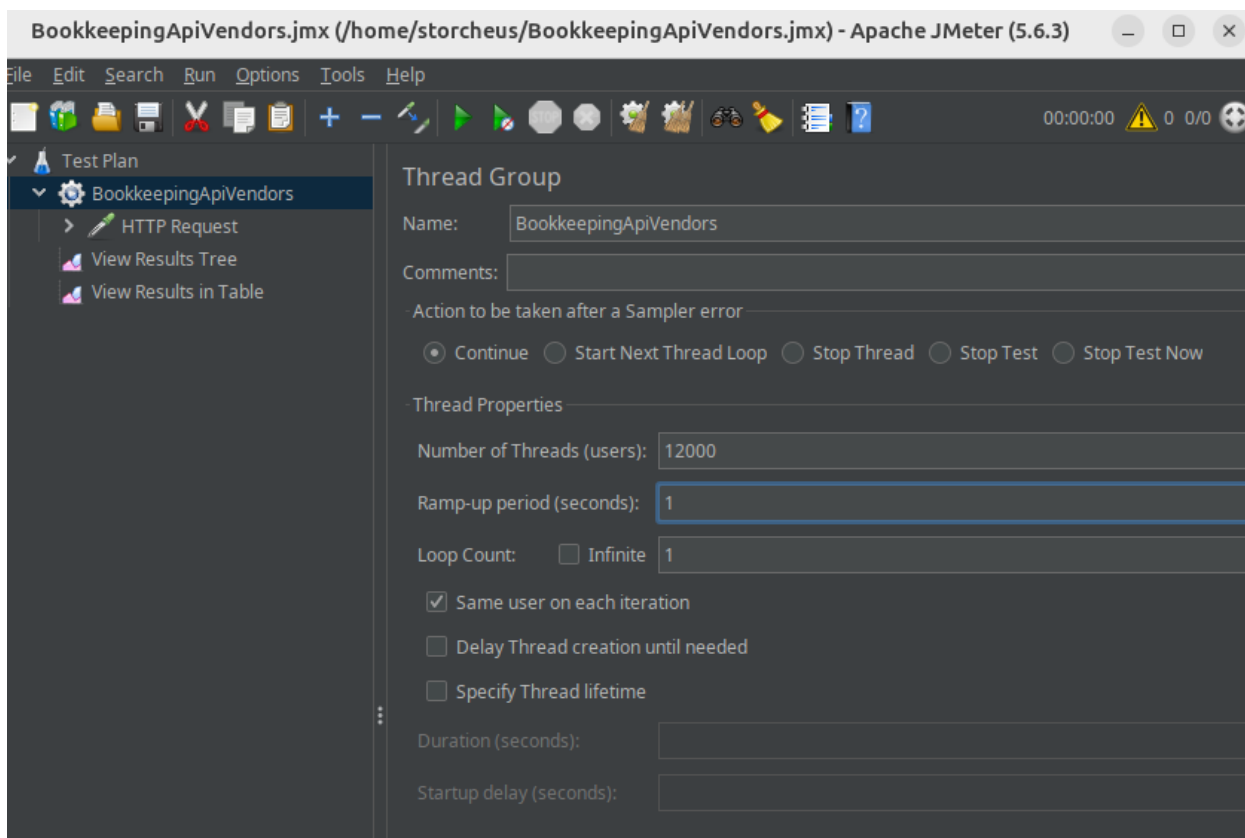


Рис. 3.23. Налаштування запиту в JMeter

Наступним кроком необхідно вказати ресурс до якого буде запит. Обов'язково треба зазначити протокол — HTTPS, захищений протокол передачі даних, метод - GET, назву домена - ruawcb00pc.execute-api.us-east-1.amazonaws.com та саму назву ресурсу - /bookkeeping2/vendors, рисунок 3.24.

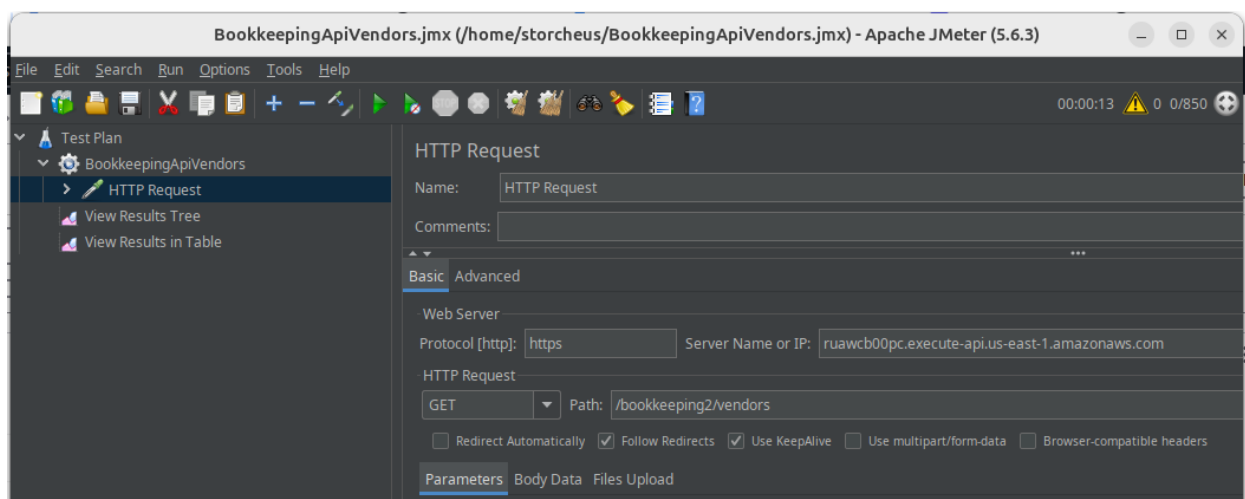


Рис. 3.24. Налаштування запиту до ресурсу в JMeter

Після запуску запитів в програмі Jmeter подивимось результат моніторингу в CloudWatch для ресурсу vendors, рисунок 3.25.

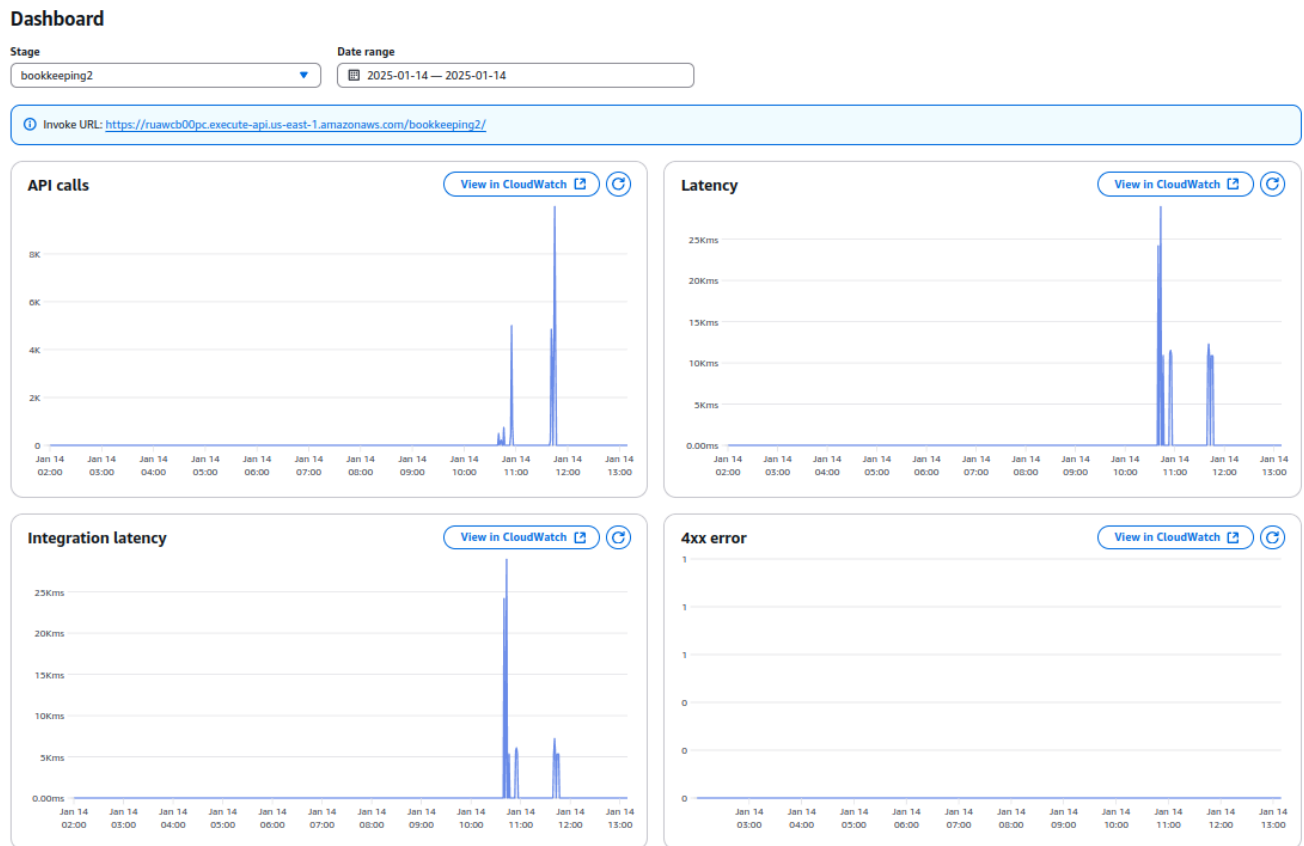


Рис. 3.25. Моніторинг додатку за допомогою CloudWatch

З моніторингу видно, що всі запити були оброблені без помилок, не має помилок 5xx, тобто відмови в обслуговуванні (DoS). Також видно кількість звернень до ресурсу vendors в секції API calls.

Ще одним важливим сервісом для перевірки додатку на вразливості є Amazon Inspector. Inspector проводить автоматизовану оцінку безпеки, щоб виявити такі проблеми, як неправильні конфігурації, вразливі залежності в проекті, програмне забезпечення та потенційні поверхні атаки. Відносно додатку Bookkeeping було виявлено 4 вразливості, 2 з котрих низького пріоритету, 2 інші — середньої, рисунок 3.26.

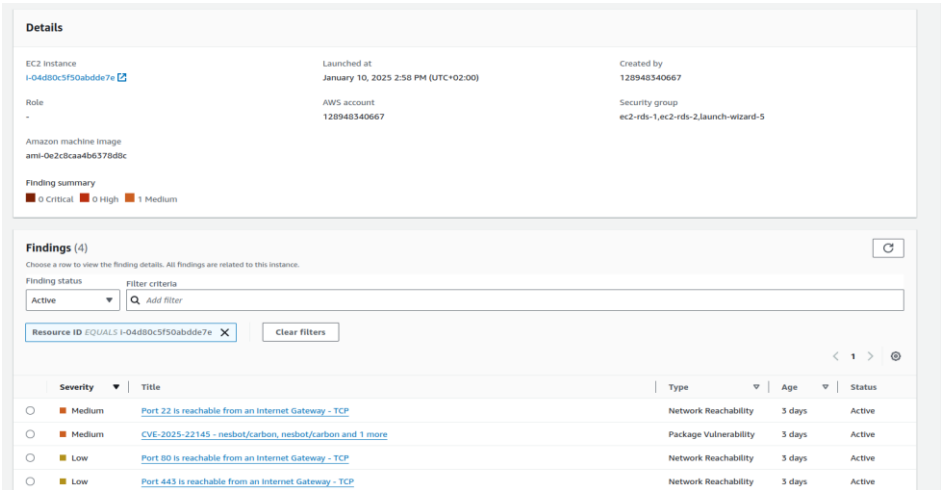


Рис. 3.26. Діагностика додатку на вразливості

Загалом Inspector зазначив, що в додатку три вразливості пов’язані з вразливостями «Доступність мережі», а четверта вразливість пов’язана безпосередньо з пакетом для обробки дати в фреймворк Laravel.

Наступним корисним сервісом для відстежування подій є CloudTrail. Саме за допомогою CloudTrail можна відстежувати всі події, котрі були виконані стосовно додатку, ким, в який час, з яким статусом завершилась конкретна дія, тип сервісу котрий виконав цю дію. Цей механізм має назву «Event History», рисунок 3.27.

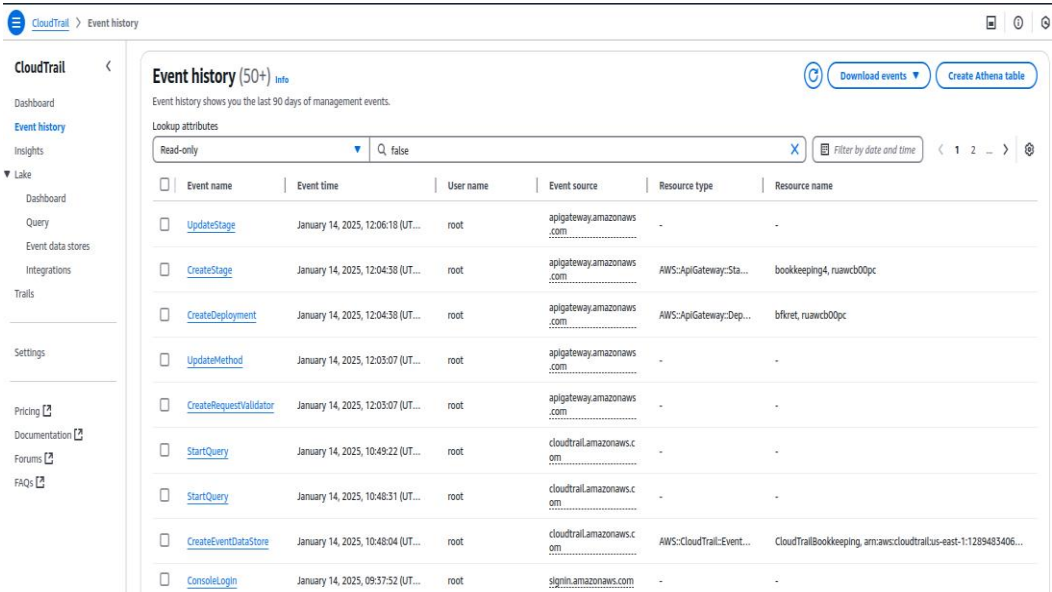


Рисунок 3.27. Історія подій в CloudTrail

Наступним етапом створемо автентифікація та авторизація для доступу до ресурсів REST API додатку. AWS Cognito спрощує ці завдання знадзвичайною ефективністю. Необхідно створити User pool в Cognito для подальшого створення авторизації в API Gateway. Для того щоб створити User pool необхідно створити додаток в Cognito, рисунок 3.28. Виберемо тип додатку «Traditional web application», призначемо ім'я — «Bookkeeping App».

Define your application
Choose an application type and give it a name.

Application type | Info
Choose the type of application that you're developing. Your choice determines the example application code that Cognito will display.

- ☒ **Traditional web application**
An application hosted on a webserver. Uses redirects and separate pages to display information. Examples are Java, Python, nodeJS.
- ☐ **Single-page application (SPA)**
Modifies the current web page based on user interaction. Examples are JavaScript, Angular, React.
- ☐ **Mobile app**
An app built with a mobile SDK. Examples are Android, iOS.
- ☐ **Machine-to-machine application**
Platform-independent server-to-server communications without user interaction. Authorizes API access with OAuth 2.0 scopes.

Name your application | Info
BookkeepingApp
Names are limited to 128 characters or fewer. Names may only contain alphanumeric characters, spaces, and the following special characters: + = , . @ -

Configure options
You must make a few initial choices about the user pool that supports your application. To change these settings later, you must create a new user pool.

Options for sign-in identifiers | Info
Choose sign-in attributes. Usernames can be an email address, phone number, or a user-selected username. When you select only email and phone, users must select either email or phone as their username type. When username is an option, users can sign in with any options you select if they have provided a value for that option.

- ☒ **Email**
- ☐ **Phone number**
- ☐ ..

Рис. 3.28. Створення додатку в Cognito

Після створення додатку в Cognito отримаємо ключі доступу, котрі в подальшому будуть використовуватись для авторизованого доступу до ресурсу, рисунок 3.29.

App client: BookkeepingApp Info Delete

App client information		Created
App client name BookkeepingApp	Authentication flow session duration 3 minutes	January
Client ID 7c5upmvaeovdlt7v4hv08s1m5	Refresh token expiration 5 day(s)	Last up January
Client secret ***** <input type="checkbox"/> Show client secret	Access token expiration 60 minutes	
Authentication flows Choice-based sign-in Secure remote password (SRP) Get user tokens from existing authenticated sessions	ID token expiration 60 minutes	
	Advanced authentication settings Enable token revocation Enable prevent user existence errors	

Рис. 3.29. Створений додаток в Cognito.

Авторизатор для API Gateway, тому створемо User pool в Cognito, рисунок 3.30.

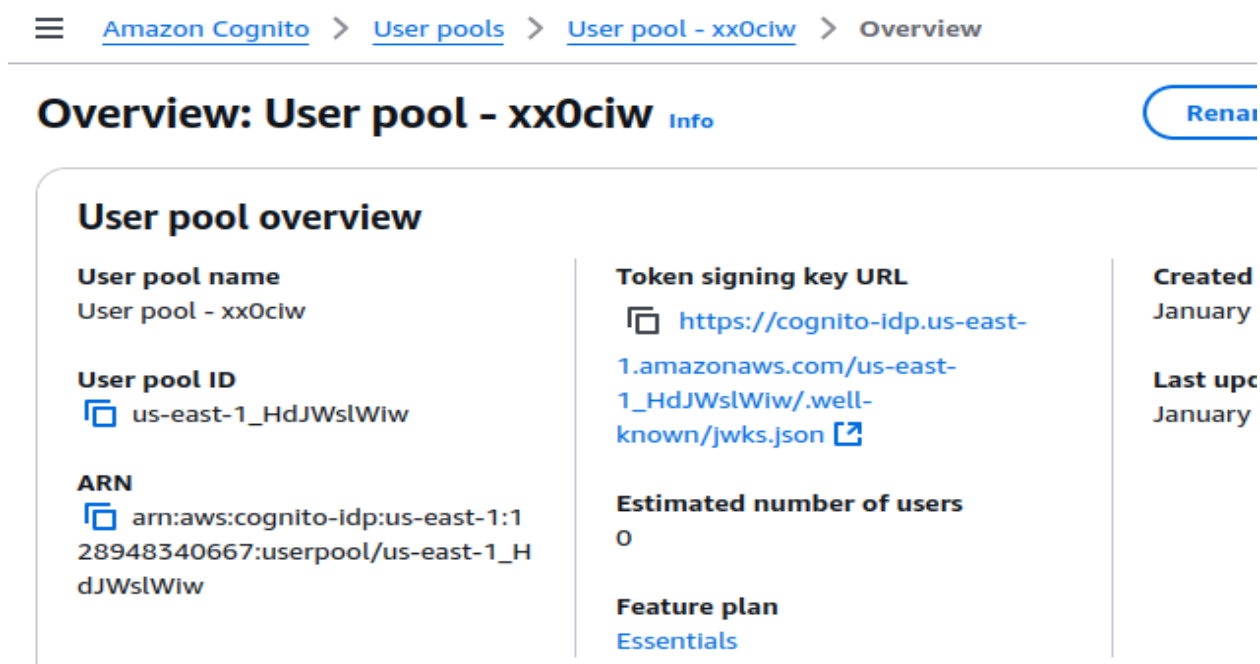


Рис. 3.30. Створений додаток в User pool в Cognito

Тепер можна створити авторизацію в API Gateway — BookkeepingAuth, вибравши User pool – xx0ciw, рисунок 3.31.

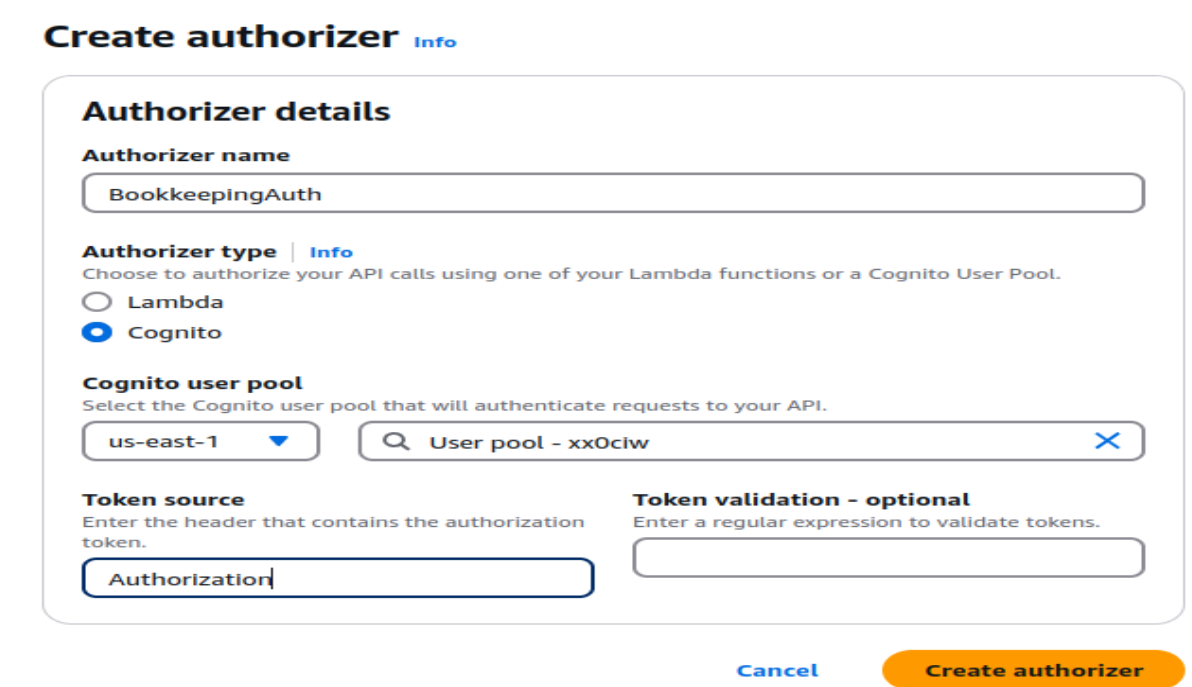


Рис. 3.31. Авторизація для API Gateway

Після створення авторизації для API Gateway, зробимо запит до ресурсу vendors з заголовком Authorization та токеном авторизації, рисунок 3.32.

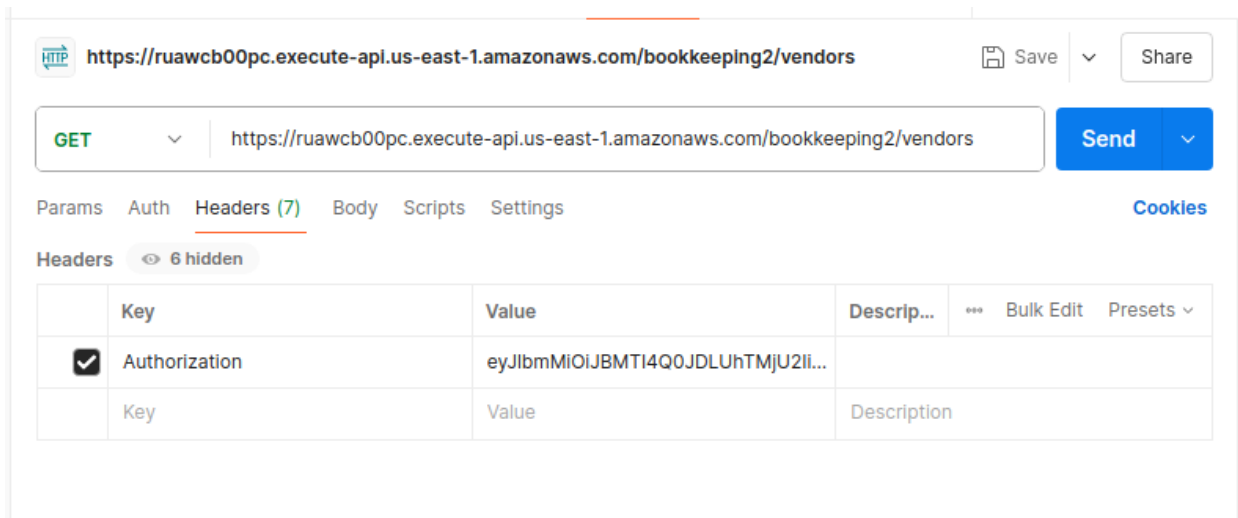


Рис. 3.32. Запит до ресурсу з авторизацією

Таким чином, тестування на вразливості додатку дозволило підтвердити захищеність розробленого REST API додатку.

Висновки до розділу 3

Був розроблений REST API додаток, для відображення та зручного тестування ресурсів був використаний OpenAPI. В розробленому додатку, було виявлено розповсюдженні вразливості REST API. Завдяки зручній архітектурі AWS було швидко розгорнуто додаток на хмарному сервісі Amazon EC2.

Експериментальним шляхом було доведено, що API Gateway забезпечує захист при передачі даних завдяки використанню протоколу HTTPS. В якості автентифікації для додатку був використаний хмарний сервіс AWS IAM, а для авторизації - Cognito, котрий дозволив надавати дозволи для використання ресурсів додатку. Моніторинг забезпечив сервіс CloudWatch, котрий надав детальну статистику запитів додатку в реальному часі. Amazon Inspector робить автоматичне відстеження вразливостей інстанса Amazon EC2, що дозволяє усувати ці небезпеки ще на етапі розгортання додатку в хмарному провайдері. CloudTrail забезпечив повною інформацією про усі події стосовно REST API додаток, його розгортання, використання, виклики ресурсів, відповіді на запити, помилки.

Саме завдяки хмарним сервісам, можна забезпечити надійний захист REST API додатку з урахуванням сучасних викликів в сфері безпеки.

ВИСНОВКИ

В даній дипломній роботі було дослідження хмарні технології для побудови захищених REST API додатків.

В першій частині розглянуті теоретичні основи REST API, його переваги та поширеність застосування в сучасній розробці. Однією з головних причин широкого поширення архітектурного стилю REST є безстанність. Кожен запит до API містить всю інформацію, необхідну для обробки, що гарантує, що серверу не потрібно зберігати інформацію про сеанс між запитами. Такий принцип побудови не тільки спрощує реалізацію, але й робить REST API по суті масштабованими, оскільки сервери можуть обробляти кілька запитів незалежно один від одного.

Також були проаналізовані найпоширеніші вразливості API.

В другій частині був зроблений аналіз хмарних технологій для покращення безпеки REST API додатків.

API Gateway займає центральне місце в управлінні REST API. Він виступає в ролі «вхідних дверей» для внутрішніх служб, забезпечуючи безперебійний інтерфейс між клієнтами та додатком. Однак його цінність виходить за рамки простого підключення. API Gateway надає розробникам можливість здійснювати ретельний контроль над вхідним трафіком, включаючи обмеження швидкості та перевірку запитів. Це гарантує, що API залишається продуктивним і стійким до різних навантажень, а також запобігає несанкціонованим або неправильно сформованим запитами. Інтеграція шифрування TLS додатково гарантує, що дані залишаються безпечними, захищаючи конфіденційну інформацію.

Надійна система управління доступом має вирішальне значення для захисту чутливих ресурсів, і AWS IAM відіграє в цьому відношенні життєво важливу роль. Вона дозволяє реалізувати принцип найменших привілеїв, гарантуючи, що суб'єкти, які взаємодіють з API, мають лише ті дозволи, які їм

абсолютно необхідні. За допомогою ролей і політик IAM розробники можуть визначати детальний контроль доступу для користувачів, груп і сервісів.

Автентифікація та авторизація користувачів є фундаментальними аспектами захисту REST API, і AWS Cognito спрощує ці завдання з надзвичайною ефективністю. Cognito забезпечує безперешкодну реєстрацію, вхід і контроль доступу користувачів. Він також підтримує багатофакторну автентифікацію, що є критично важливою функцією для підвищення безпеки в сучасному середовищі загроз. Крім того, Cognito легко інтегрується з іншими сервісами AWS.

Прозорість і відстежуваність мають важливе значення для підтримки довіри та підзвітності в будь-якому додатку. AWS CloudTrail досягає успіху в цій сфері, надаючи детальний запис усіх викликів API, зроблених в обліковому записі AWS. Крім того, можливість аналізувати журнали CloudTrail на предмет виявлення шаблонів може стати основою для вдосконалення політик безпеки, забезпечуючи постійне поліпшення стану безпеки додатків.

В третій частині був розроблений REST API додаток. Додаток був розміщений в хмарному середовищі Amazon, на прикладі якого було продемонстровано саме хмарні технології AWS для забезпечення захисту від найпоширеніших загроз.

Використовуючи в комплексі API Gateway, CloudWatch, AWS IAM, Cognito, Inspector та CloudTrail, компанії та розробники можуть створювати REST API додатки, які є не лише функціональними, але й безпечними та відмовостійкими. Ці сервіси спільно вирішують критичні аспекти безпеки додатків, від автентифікації користувачів і контролю доступу до моніторингу та виявлення загроз. Результатом є екосистема, яка не тільки відповідає вимогам сьогоденного складного цифрового ландшафту, але й адаптується до нових викликів в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Arnaud Lauret. The Design of Web APIs. Manning . 2019 - 400 p.
2. API integration. URL: <https://www.ibm.com/topics/api-integration> (дата звернення: 12.12.2024)
3. How do APIs work?. URL: <https://www.cloudflare.com/learning/security/api/how-do-apis-work/> (дата звернення: 08.12.2024)
4. Sign in with Google button UX. URL: <https://developers.google.com/identity/gsi/web/guides/personalized-button> (дата звернення: 13.10.2024)
5. Michael Goodwin. What is an API (application programming interface)?. URL: <https://www.ibm.com/think/topics/api> (дата звернення: 12.11.2024)
6. J Simpson. Types of APIs. URL: <https://nordicapis.com/6-types-of-apis-open-public-partner-private-composite-unified/> (дата звернення: 12.12.2024)
7. Опендатабота: веб сайт. URL: <https://opendatabot.ua/> (дата звернення: 22.12.2024)
8. Roy Thomas Fielding. Dissertation: Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine. 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (дата звернення: 04.11.2024)
9. APILayer: Fixer API. URL: <https://apilayer.com/marketplace/fixer-api> (дата звернення: 12.11.2024)
10. What Are API Vulnerabilities?. URL: <https://www.akamai.com/glossary/what-are-api-vulnerabilities> (дата звернення: 12.11.2024)
11. Open Worldwide Application Security Project (OWASP). URL: <https://owasp.org/about> (дата звернення: 02.12.2024)
12. Рекомендації CERT-UA з безпеки вебресурсів.. URL: <https://cert.gov.ua/recommendation/19> (дата звернення: 22.12.2024)

13. OWASP Top 10 API Security Risks . [URL: <https://owasp.org/API-Security/editions/2023/en/0x11-t10/>] (дата звернення: 12.12.2024)
14. Abigail Ojeda. REST API Security Best Practices. URL: <https://www.akamai.com/blog/security/rest-api-security-best-practices> (дата звернення: 11.11.2024)
15. Alexa Sevilla. REST API Security:Best Practices Guide. URL: <https://www.stackhawk.com/blog/rest-api-security-best-practices/> (дата звернення: 07.10.2024)
16. US state sues T-Mobile over 2021 data breach which leaked data of millions. URL: <https://www.techradar.com/pro/us-state-sues-t-mobile-over-2021-data-breach-that-leaked-data-of-millions> (дата звернення: 06.11.2024)
17. T-Mobile will pay FCC millions in settlement over multiple data breaches. URL: <https://www.techradar.com/pro/us-state-sues-t-mobile-over-2021-data-breach-that-leaked-data-of-millions> (дата звернення: 12.11.2024)
18. GDPR: Fines / Penalties. URL: <https://gdpr-info.eu/issues/fines-penalties/> (дата звернення: 12.11.2024)
19. What is API management?. URL: <https://cloud.google.com/learn/what-is-api-management> (дата звернення: 24.12.2024)
20. API Management. URL: <https://aws.amazon.com/api-gateway/api-management/> (дата звернення: 12.12.2024)
21. Impressive API Economy Statistics. URL: <https://nordicapis.com/20-impressive-api-economy-statistics/> (дата звернення: 10.11.2024)
22. 2024 API Security and Management Report. URL: <https://www.cloudflare.com/lp/api-security-report/> (дата звернення: 12.11.2024)
23. Stripe Revenue and Growth Statistics. URL: <https://backlinko.com/stripe-users> (дата звернення: 11.12.2024)
24. The Economic Impact of APIs: API Monetization, AI, Web3, and Beyond. URL: <https://konghq.com/blog/enterprise/the-economic-impact-of-apis> (дата звернення: 07.12.2024)

25. What is an API gateway?. URL: <https://www.ibm.com/think/topics/api-gateway>
(дата звернення: 12.12.2024)
26. Cloud Market Growth Surge Continues in Q3 – Growth Rate Increases for the Fourth Consecutive Quarter. URL: <https://www.srgresearch.com/articles/cloud-market-growth-surge-continues-in-q3-growth-rate-increases-for-the-fourth-consecutive-quarter> (дата звернення: 16.12.2024)
27. Amazon API Gateway. URL: <https://aws.amazon.com/api-gateway/> (дата звернення: 13.11.2024)
28. Architecture of API Gateway. URL: <https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html#api-gateway-overview-aws-backbone> (дата звернення: 12.11.2024)
29. Amazon CloudWatch. URL: <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html> (дата звернення: 12.12.2024)
30. Amazon CloudWatch concepts. URL: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_concepts.html (дата звернення: 02.11.2024)
31. AWS CloudTrail. URL: <https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html> (дата звернення: 12.12.2024)
32. AWS Identity and Access Management (IAM). URL: <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html> (дата звернення: 22.12.2024)
33. Amazon Inspector. URL: <https://docs.aws.amazon.com/inspector/latest/user/what-is-inspector.html> (дата звернення: 07.11.2024)
34. Neil Madden. API Security in Action. Manning. 2020 – 576p.
35. Amazon Cognito. URL: <https://docs.aws.amazon.com/cognito/> (дата звернення: 24.12.2024)
36. Testing Web APIs. Mark Winteringham. Manning. 2022 - 264 pages

- 37. Postman. URL: <https://www.postman.com/> (дата звернення: 11.01.2025)
- 38. Swagger. URL: <https://swagger.io/> (дата звернення: 11.01.2025)
- 39. OpenAPI. URL: <https://www.openapis.org/> (дата звернення: 11.01.2025)
- 40. Database: Migrations. URL: <https://laravel.com/docs/11.x/migrations> (дата звернення: 11.01.2025)

Додаток А

Лістинг коду розроблених програмних компонентів

Клас сутності Vendor

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Vendor extends Model
{
    protected $fillable = [
        'name',
        'contact_person',
        'email',
        'phone',
        'fax',
        'address',
        'organization_id'
    ];
}
```

Клас VendorController

```
<?php

namespace App\Http\Controllers;

use App\Models\Vendor;
use Illuminate\Http\Request;

/**
 * @OA\Tag(
 *     name="Vendors",
 *     description="API Vendors for managing accounts"
 * )
 */
class VendorController extends Controller
{
    /**
     * @OA\Get(
     *     path="/vendors",
     *     summary="Retrieve a list of vendors",
     *     tags={"Vendors"},
     *     @OA\Response(
     *         response=200,
     *         description="List of vendors",
     *         @OA\JsonContent(
     *             type="array",
     *             @OA\Items(
     *                 type="object",
     *                 @OA\Property(property="id", type="integer", example=1),
     *                 @OA\Property(property="name", type="string", example="John Doe"),

```



```

*         @OA\Property(property="contact_person", type="string", example="Jane Smith"),
*         @OA\Property(property="email", type="string", format="email",
example="john.doe@example.com"),
*         @OA\Property(property="phone", type="string", example="123-456-7890"),
*         @OA\Property(property="fax", type="string", example="123-456-7891"),
*         @OA\Property(property="address", type="string", example="123 Elm Street, Springfield,
USA"),
*         @OA\Property(property="created_at", type="string", format="date-time", example="2023-
12-20T12:34:56Z"),
*         @OA\Property(property="updated_at", type="string", format="date-time", example="2023-
12-20T12:34:56Z")
*     )
* )
* )
* )
*/

```

```
public function index()
```

```

{
    $vendors = Vendor::all();
    return response()->json($vendors);
}

```

```

/**
 * @OA\Post(
 *     path="/vendors",
 *     summary="Create a new vendor",
 *     tags={"Vendors"},
 *     @OA\RequestBody(
 *         required=true,
 *         @OA\JsonContent(
 *             required={"name"},
 *             @OA\Property(property="name", type="string", example="John Doe"),
 *             @OA\Property(property="contact_person", type="string", example="Jane Smith"),
 *             @OA\Property(property="email", type="string", format="email",
example="john.doe@example.com"),
 *             @OA\Property(property="phone", type="string", example="123-456-7890"),
 *             @OA\Property(property="fax", type="string", example="123-456-7891"),
 *             @OA\Property(property="address", type="string", example="123 Elm Street, Springfield,
USA")
 *         )
 *     ),
 *     @OA\Response(
 *         response=201,
 *         description="Vendor created successfully",
 *         @OA\JsonContent(
 *             @OA\Property(property="message", type="string", example="Vendor created successfully!"),
 *             @OA\Property(property="vendor", type="object",
 *                 @OA\Property(property="id", type="integer", example=1),
 *                 @OA\Property(property="name", type="string", example="John Doe"),
 *                 @OA\Property(property="contact_person", type="string", example="Jane Smith"),
 *                 @OA\Property(property="email", type="string", format="email",
example="john.doe@example.com"),
 *                 @OA\Property(property="phone", type="string", example="123-456-7890"),
 *                 @OA\Property(property="fax", type="string", example="123-456-7891"),
 *                 @OA\Property(property="address", type="string", example="123 Elm Street, Springfield,
USA"),
 *                 @OA\Property(property="created_at", type="string", format="date-time", example="2023-
12-20T12:34:56Z"),
 *                 @OA\Property(property="updated_at", type="string", format="date-time", example="2023-
12-20T12:34:56Z")
 *             )
 *         )
 *     ),

```

```

* @OA\Response(
*     response=422,
*     description="Validation error",
*     @OA\JsonContent(
*         @OA\Property(property="message", type="string", example="The given data was invalid."),
*         @OA\Property(property="errors", type="object",
*             additionalProperties=@OA\Property(type="array",
*                 @OA\Items(type="string"))
*         )
*     )
* )
* )
* )
*/
public function store(Request $request)
{
    $validatedData = $request->validate([
        'name' => 'required|string|max:255',
        'contact_person' => 'nullable|string|max:255',
        'email' => 'nullable|email|max:255',
        'phone' => 'nullable|string|max:20',
        'fax' => 'nullable|string|max:20',
        'address' => 'nullable|string',
    ]);

    $vendor = Vendor::create($validatedData);
    return response()->json(['message' => 'Vendor created successfully!', 'vendor' => $vendor], 201);
}

/**
 * @OA\Get(
 *     path="/vendors/{id}",
 *     tags={"Vendors"},
 *     summary="Get a specific vendor",
 *     description="Retrieve a specific vendor by its ID.",
 *     @OA\Parameter(
 *         name="id",
 *         in="path",
 *         required=true,
 *         description="The ID of the vendor.",
 *         @OA\Schema(type="integer")
 *     ),
 *     @OA\Response(
 *         response=200,
 *         description="Vendor details",
 *         @OA\JsonContent(
 *             type="object",
 *             @OA\Property(property="id", type="integer", example=1),
 *             @OA\Property(property="name", type="string", example="John Doe"),
 *             @OA\Property(property="contact_person", type="string", example="Jane Smith"),
 *             @OA\Property(property="email", type="string", format="email",
 * example="john.doe@example.com"),
 *             @OA\Property(property="phone", type="string", example="123-456-7890"),
 *             @OA\Property(property="fax", type="string", example="123-456-7891"),
 *             @OA\Property(property="address", type="string", example="123 Elm Street, Springfield, USA"),
 *             @OA\Property(property="created_at", type="string", format="date-time", example="2023-12-
20T12:34:56Z"),
 *             @OA\Property(property="updated_at", type="string", format="date-time", example="2023-12-
20T12:34:56Z")
 *         )
 *     ),
 *     @OA\Response(
 *         response=404,

```

```

*      description="Vendor not found",
*      @OA\JsonContent(
*          @OA\Property(property="message", type="string", example="Vendor not found")
*      )
*  )
*  )
*/
public function get($id)
{
    $vendor = Vendor::find($id);

    if (!$vendor) {
        return response()->json(['message' => 'Vendor not found'], 404);
    }
    return response()->json($vendor);
}

/**
 * @OA\Put(
 *     path="/vendors/{vendorId}",
 *     summary="Update a specific vendor",
 *     tags={"Vendors"},
 *     @OA\Parameter(
 *         name="vendorId",
 *         in="path",
 *         required=true,
 *         description="ID of the vendor to update",
 *         @OA\Schema(type="integer", example=1)
 *     ),
 *     @OA\RequestBody(
 *         required=true,
 *         @OA\JsonContent(
 *             required={"name"},
 *             @OA\Property(property="name", type="string", example="John Doe"),
 *             @OA\Property(property="contact_person", type="string", example="Jane Smith"),
 *             @OA\Property(property="email", type="string", format="email",
example="john.doe@example.com"),
 *             @OA\Property(property="phone", type="string", example="123-456-7890"),
 *             @OA\Property(property="fax", type="string", example="123-456-7891"),
 *             @OA\Property(property="address", type="string", example="123 Elm Street, Springfield,
USA"),
 *         ),
 *     ),
 *     @OA\Response(
 *         response=200,
 *         description="Vendor updated successfully",
 *         @OA\JsonContent(
 *             @OA\Property(property="message", type="string", example="Vendor updated successfully!"),
 *             @OA\Property(property="vendor", type="object",
 *                 @OA\Property(property="id", type="integer", example=1),
 *                 @OA\Property(property="name", type="string", example="John Doe"),
 *                 @OA\Property(property="contact_person", type="string", example="Jane Smith"),
 *                 @OA\Property(property="email", type="string", format="email",
example="john.doe@example.com"),
 *                 @OA\Property(property="phone", type="string", example="123-456-7890"),
 *                 @OA\Property(property="fax", type="string", example="123-456-7891"),
 *                 @OA\Property(property="address", type="string", example="123 Elm Street, Springfield,
USA"),
 *                 @OA\Property(property="created_at", type="string", format="date-time", example="2023-
12-20T12:34:56Z"),
 *                 @OA\Property(property="updated_at", type="string", format="date-time", example="2023-
12-20T12:34:56Z")
 *             )
 *         )
 *     )

```

```

* )
* ),
* @OA\Response(
*   response=404,
*   description="Vendor not found",
*   @OA\JsonContent(
*     @OA\Property(property="message", type="string", example="Vendor not found")
*   )
* ),
* @OA\Response(
*   response=422,
*   description="Validation error",
*   @OA\JsonContent(
*     @OA\Property(property="message", type="string", example="The given data was invalid."),
*     @OA\Property(property="errors", type="object",
*       additionalProperties=@OA\Property(type="array",
*         @OA\Items(type="string")
*       )
*     )
*   )
* )
* )
* )
* )
*/
public function update(Request $request, $vendorId)
{
    $vendor = Vendor::find($vendorId);

    if (!$vendor) {
        return response()->json(['message' => 'Vendor not found']);
    }

    $validatedData = $request->validate([
        'name' => 'required|string|max:255',
        'contact_person' => 'nullable|string|max:255',
        'email' => 'nullable|email|max:255',
        'phone' => 'nullable|string|max:20',
        'fax' => 'nullable|string|max:20',
        'address' => 'nullable|string',
    ]);

    $vendor->update($validatedData);
    return response()->json(['message' => 'Vendor updated successfully!', 'vendor' => $vendor]);
}

/**
 * @OA\Delete(
 *   path="/vendors/{vendorId}",
 *   summary="Delete a specific vendor",
 *   tags={"Vendors"},
 *   @OA\Parameter(
 *     name="vendorId",
 *     in="path",
 *     required=true,
 *     description="ID of the vendor to delete",
 *     @OA\Schema(type="integer", example=1)
 *   ),
 *   @OA\Response(
 *     response=200,
 *     description="Vendor deleted successfully",
 *     @OA\JsonContent(
 *       @OA\Property(property="message", type="string", example="Vendor deleted successfully!")
 *     )
 *   ),
 */

```

```

*      @OA\Response(
*          response=404,
*          description="Vendor not found",
*          @OA\JsonContent(
*              @OA\Property(property="message", type="string", example="Vendor not found")
*          )
*      )
*  )
*/
public function destroy($vendorId)
{
    $vendor = Vendor::find($vendorId);

    if (!$vendor) {
        return response()->json(['message' => 'Vendor not found']);
    }

    $vendor->delete();
    return response()->json(['message' => 'Vendor deleted successfully!']);
}
}

```

Клас міграції для таблиці Vendor

```
<?php
```

```

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('vendors', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('contact_person')->nullable();
            $table->string('email')->nullable();
            $table->string('phone')->nullable();
            $table->string('fax')->nullable();
            $table->string('address')->nullable();
            $table->unsignedBigInteger('organization_id')->nullable();
            $table->foreign('organization_id')->references('id')->on('organizations')->onUpdate('SET NULL')->onDelete('SET NULL');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('vendors');
    }
};

```

Додаток В

Конфігурація політики(policy) для AmazonAPIGatewayPushToCloudWatchLogs

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:GetLogEvents",
        "logs:FilterLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```