DOI 10.31891/2307-5732-2024-339-4-19
УДК 004.42

КОТЕНКО НАТАЛІЯ
Державний торговельно-економічний університет
https://orcid.org/0000-0002-2675-6514
e-mail: kotenkono@knute.edu.ua
ЖИРОВА ТЕТЯНА
Державний торговельно-економічний університет
https://orcid.org/0000-0001-8321-6939
e-mail: zhyrova@knute.edu.ua
АЛЄКСАНДРОВ АНДРІЙ
Державний торговельно-економічний університет
https://orcid.org/0009-0007-5950-4886
e-mail: a.aleksandrov_fit_2m_22_m_d@knute.edu.ua
РУДЕНКО ОЛЬГА
Державний торговельно-економічний університет
https://orcid.org/0009-0004-6171-8953
e-mail: o.rudenko_fit_2m_23_m_z@knute.edu.ua

ДОСЛІДЖЕННЯ ПЕРЕДУМОВ РОЗРОБКИ СЕРВІСУ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ API

У статті проаналізовано наявну документацію, якою забезпечуються основні компоненти запропонованого до розробки програмного забезпечення та загалом його концепцію. Визначено стандарти та підходи, які забезпечать роботу сервісу зі створення програмних інтерфейсів. Визначено, що основою роботи програмного забезпечення є мережева взаємодія систем, для якої передбачено використання стеку протоколів TCP/IP як бази для передачі даних у мережі. Зазначено, що сутністю програмних інтерфейсів є протокол HTTP, який забезпечує передачу даних до та від програмних інтерфейсів. REST API, як архітектурний стиль клієнт-серверної взаємодії, є фундаментальною вимогою до реалізації програмних інтерфейсів з точки зору сервісу. Проведено порівняльний аналіз віртуалізації та контейнеризації, що дозволяє виконувати користувацькі інструкції бізнес-логіки без впливу на працююче програмне та апаратне забезпечення.

Проведено порівняльний аналіз з аналогічними програмними засобами та визначено переваги й недоліки кожного з них, що дозволяє краще зрозуміти очікування користувачів та покращити їхній досвід, виявивши ще нереалізовані можливості. Особлива увага приділяється аналізу таких сервісів, як AWS API Gateway та Directus, які є конкурентними рішеннями у даній сфері.

Висновки дослідження підкреслюють необхідність дотримання встановлених стандартів та використання переваг віртуалізації та контейнеризації для створення надійної та зручної платформи для розробки та управління програмними інтерфейсами. Майбутні дослідження мають бути спрямовані на усунення виявлених ризиків та подальше вдосконалення сервісу відповідно до потреб користувачів та технологічних досягнень. Також необхідно продовжувати аналіз аналогів для визначення можливих проблем, що можуть виникнути під час розробки, впровадження чи функціонування програмного забезпечення, з метою їх мінімізації.

Ключові слова: API, REST, програмне забезпечення.

KOTENKO NATALIYA, ZHYROVA TETYANA, ALEKSANDROV ANDRII, RUDENCO OLGA
State University of Trade and Economics

RESEARCH ON THE PREREQUISITES FOR DEVELOPING A SERVICE TO ENHANCE API EFFICIENCY

The article analyzes the existing documentation that provides the main components of the proposed software and its overall concept. Standards and approaches are identified that will ensure the functioning of the service for creating software interfaces. It is determined that the foundation of the software's operation is the network interaction of systems, for which the use of the TCP/IP protocol stack is envisaged as the basis for data transmission in the network. It is noted that the essence of software interfaces is the HTTP protocol, which ensures data transmission to and from software interfaces. REST API, as an architectural style of client-server interaction, is a fundamental requirement for the implementation of software interfaces from the service's perspective. A comparative analysis of virtualization and containerization has been conducted, enabling the execution of user instructions for business logic without impacting the functioning software and hardware.

A comparative analysis with similar software tools has been carried out, identifying the advantages and disadvantages of each, which allows for a better understanding of user expectations and improves their experience by identifying unrealized opportunities. Special attention is paid to analyzing services such as AWS API Gateway and Directus, which are competitive solutions in this area.

The research conclusions emphasize the necessity of adhering to established standards and leveraging the advantages of virtualization and containerization to create a reliable and convenient platform for developing and managing software interfaces. Future research should focus on mitigating identified risks and improving the service according to user needs and technological advancements. Additionally, it is necessary to continue analyzing similar services to identify potential issues that may arise during the development, implementation, or operation of the software, in order to minimize them.

Keywords: API, REST, software.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

У сучасному світі інформаційних технологій потреба у створенні ефективних та надійних програмних інтерфейсів (API) є надзвичайно актуальною, оскільки вони забезпечують мережеву взаємодію

між різними системами та додатками. Наразі існує необхідність у розробці програмного забезпечення, яке має на меті забезпечити функціонування таких інтерфейсів, використовуючи стек протоколів TCP/IP та протокол HTTP як основу для передачі даних.

Є потреба у проведенні дослідження, що спрямоване на аналіз існуючої документації та визначення стандартів і підходів, які сприятимуть розробці високоефективних API. Вибір архітектурного стилю REST для клієнт-серверної взаємодії є фундаментальним для забезпечення відповідності сервісів сучасним вимогам до програмних інтерфейсів.

Завдання полягає у вивченні та порівнянні методів віртуалізації та контейнеризації для забезпечення ізоляції бізнес-логіки та користувацьких інструкцій від працюючого програмного та апаратного забезпечення. Це дозволить мінімізувати ризики, пов'язані з впливом на існуючі системи, і забезпечити гнучкість у розгортанні нових рішень. Проведення порівняльного аналізу з аналогічними програмними засобами дасть змогу виявити їх переваги та недоліки, що сприятиме покращенню користувацького досвіду та відкриттю нових можливостей для інновацій. Таким чином, це дослідження має важливе значення як для наукового прогресу у сфері розробки програмного забезпечення, так і для практичного застосування в індустрії, що потребує ефективних рішень для мережевої взаємодії систем.

Аналіз досліджень та публікацій

Дисертація Роя Томаса Філдінга “Architectural Styles and the Design of Network-based Software Architectures” [1] досліджує архітектурні стилі для проектування мережевих програмних архітектур. Основний акцент зроблено на REST (Representational State Transfer), що стало основою для сучасного веб-дизайну. Філдінг детально аналізує проблеми та підходи до створення веб-архітектури, що призвели до розробки REST. Він описує принципи REST, зокрема клієнт-серверну взаємодію.

Формулювання цілей статті

Метою роботи є піддати науковому огляду взаємодію сучасних програмних засобів з метою подальшої розробки сервісу для підвищення ефективності роботи API як способу взаємодії на основі отриманої інформації.

Виклад основного матеріалу

Взаємодія різних систем є ключовим аспектом у роботі програмного забезпечення. Системи можуть виробляти, обробляти та зберігати різні типи даних, і взаємодія між ними дозволяє обмінюватися цими даними для досягнення певних цілей. Кожна система має свої особливі можливості та обмеження, і взаємодія з іншими системами дозволяє розширити її функціональність, інтеграція з зовнішніми сервісами може доповнити існуючу систему новими можливостями.

Залежно від конкретного контексту та потреб системи використовуються різні підходи до взаємодії. Одним із найпоширеніших підходів є використання клієнт-серверної моделі. З таким підходом передбачається дві ролі, які отримують системи, - клієнт та сервер.

Клієнт-серверна модель дозволяє забезпечити розподілену обробку завдань, де різні сервери можуть виконувати окремі функції та обробляти запити від багатьох клієнтів. Це сприяє масштабованості та ефективності системи, спрощує розробку, тестування та підтримку системи.

У контексті клієнт-серверної взаємодії заплановане нами програмне забезпечення покликане повністю або, у випадку розподілених системи – частково, заповнити роль сервера, обробляючи користувацькі запити на основі заздалегідь визначених структур програмних інтерфейсів.

Програмні інтерфейси – API (Application Programming Interface) – є важливим аспектом взаємодії між різними системами. API визначає набір правил та протоколів, які використовуються для взаємодії між програмними компонентами. Використання API дозволяє системам обмінюватися даними та виконувати операції через стандартизовані інтерфейси. Серед ряду наявних опцій – RPC, gRPC, GraphQL, SOAP та REST.

REST API (Representational State Transfer) є одним з найпоширеніших видів API для взаємодії між системами. REST API – архітектурний стиль, заснований на ряді обмежень, які й визначають вимоги та можливості, які будуть надаватися користувачам при побудові власних програмних інтерфейсів (рис.1).

Рис. 1. Модель REST API застосунку

Серед обмежень, визначених архітектурним стилем: клієнт-сервер, або принцип розподілення відповідальності, відсутність стану, кешування, уніфікований інтерфейс, багатошарова система [1].

Для правильного отримання, обробки та повернення даних потрібно визначити ряд стандартів та

умов, дотримання яких забезпечує правильне, очікуване функціонування програмних інтерфейсів. Передача даних у мережі та загалом концепція функціонування мережі описується моделлю OSI (The Open Systems Interconnection model), а конкретна імплементація функціонуючої мережі забезпечується стеком протоколів TCP/IP.

На найвищому рівні абстракції програмні інтерфейси оперують з прикладним рівнем моделі TCP/IP та OSI – рівнем взаємодії людини з комп'ютером, надають інтерфейс до даних для взаємодії з іншими серверами, іншим програмним забезпеченням або користувачами. На прикладному рівні моделі TCP/IP доступ до інтерфейсу забезпечується протоколом HTTP, описаним в стандарті RFC 9110.

HTTP – протокол прикладного рівня без збереження стану для розподілених спільних гіпертекстових інформаційних систем. HTTP приховує деталі того, як реалізовується служба, надаючи клієнтам єдиний інтерфейс, який не залежить від типів наданих ресурсів. Так само серверам не потрібно знати мету кожного клієнта: запит можна розглядати ізольовано, а не пов'язувати з конкретним типом клієнта чи попередньо визначеною послідовністю кроків програми. Це дозволяє ефективно використовувати реалізації загального призначення в багатьох різних контекстах, зменшує складність взаємодії та забезпечує незалежну еволюцію з часом [2].

Будь-яке повідомлення в рамках протоколу HTTP є або запитом або відповіддю – клієнт створює запит з необхідними даними та направляє його на потрібний шлях, сервер зчитує запит, формує та повертає відповідь клієнту.

Запит складається з таких основних частин: метод, ціль запиту, набір заголовків, тіло запиту.

Відповідь складається з таких основних частин: статус, набір заголовків, тіло відповіді.

Протокол HTTP покладається на нижчі рівні стеку TCP/IP і потребує їх для правильного функціонування. TCP – для транспортування та IP для ідентифікації серверу та клієнту.

За виключенням примітивних програмних систем, функціональність яких полягає в здійсненні запитів до бази даних, складніші системи потребують виконання певної бізнес-логіки, яка визначає способи створення, представлення та зміни даних.

В умовах трирівневної архітектури шар бізнес-логіки є проміжним між шаром представлення та шаром доступу до даних у базі даних, на цьому рівні виконуються розрахунки, здійснюються логічні рішення. Шар визначає, яким чином використовуються дані з бази даних та як вони потрапляють у неї.

Можливість виконання нестандартної бізнес-логіки потребує збору, збереження та виконання користувацьких інструкцій. Це створює проблему запуску ненадійного коду на використовуваному апаратному забезпеченні, що потенційно може призвести до компрометації даних чи будь-яких інших, руйнівних дій щодо працюючого апаратного та програмного забезпечення.

Вирішення такої проблеми потребує достатнього рівня ізоляції на різних рівнях, – по-перше, ізоляція від працюючого застосунку, по-друге, ізоляція запущених інструкцій між собою. Вирішення проблеми такого вигляду може реалізовуватися віртуалізацією чи контейнеризацією.

Контейнер – це стандартна одиниця програмного забезпечення, яка пакує код і всі його залежності, щоб програма швидко й надійно запускалася з одного обчислювального середовища в інше [3]. Віртуалізація сервера може забезпечити деякі переваги безпеки. Запуск сервера в гіпервізорі забезпечує пісочницю, яка може обмежити вплив компрометації, гіпервізор може забезпечити меншу поверхню для атаки, ніж хост-операційна система, зменшуючи можливість розширення успішної компрометації за межі гостьової ОС [4].

Завдяки повній відокремленості віртуальних машин від операційної системи основної машини вони надають значний рівень ізоляції, при цьому кожна окремо запущена віртуальна машина потребує власної операційної системи, файлової системи та програмного забезпечення. Контейнеризація, з іншого боку, більш легка та забезпечує менший рівень ізоляції, використовуючи основну операційну систему. Порівняння віртуальних машин та контейнерів представлено в таблиці 1.

Проведення порівняльного аналізу існуючих аналогічних програмних засобів дозволяє набути кращого розуміння про очікування потенційних користувачів, визначити поточні стандарти та практики у відповідній області – технічних вимог, щодо вибору мов програмування, побудови архітектури системи чи інших аспектів, а також бізнес-вимог. Окрім цього, аналіз аналогів дозволяє визначити можливі ризики та проблеми, які можуть виникнути під час розробки, впровадження чи функціонування програмного забезпечення, та мінімізувати їх.

Серед аналогічних, та таких, що можуть вважатися потенційно конкурентними розроблюваному програмному забезпеченню проєктів можна виділити AWS API Gateway та Directus.

AWS (Amazon Web Services) API Gateway – сервіс для створення, публікації, підтримки, моніторингу і захисту REST, HTTP і WebSocket API будь-якого масштабу [5]. Важливою особливістю є орієнтованість на роботу з іншими AWS-сервісами – базами даних (DynamoDB), безсерверними функціями (Lambda) та безліччю інших. Така особливість може вважатися перевагою, якщо брати до уваги різносторонність сервісів та, відповідно, можливість створення застосунків будь-яких розмірів в одному місці – AWS. Проте пов'язування сервісів між собою та з власне API Gateway, а також простота налаштування не є очевидними. Налаштування сервісів AWS не є елементарним процесом, вимагає визначеного набору знань та призначене для спеціалістів певного профілю.

Таблиця 1

Порівняння віртуальних машин та контейнерів

Особливість	Віртуальні машини	Контейнери
Розмір	Значно більший	Менший
Операційна система	Окрема	З основної системи
Швидкість запуску	Менша	Більша
Рівень ізоляції	Значний	Ізоляція присутня, але менша
Використання пам'яті та інших ресурсів	Вище, кожна віртуальна машина потребує ядро операційної системи та усе необхідне програмне забезпечення	Нижче, використовується базова операційна система
Збереження даних	Дані зберігаються між перезавантаженнями	Зазвичай без збереження стану
Мережева взаємодія	Окремі мережеві інтерфейси	Мережеві інтерфейси базової системи

Directus – сервіс для створення програмних інтерфейсів до SQL (structured query language) бази даних без застосування навичок програмування. Перевагою сервісу є простота налаштування, яка забезпечується примітивністю виконуваних задач, – створення REST API для записування, читання, оновлення, а також видалення даних з існуючої бази даних, – що, при цьому, є найбільш поширеним варіантом використання програмних інтерфейсів, і є достатнім для більшості виконуваних кінцевих завдань – створення вебсайтів, мобільних застосунків тощо. Для більш складних систем є можливість здійснення пошуку, фільтрування, агрегування даних, зміна схеми таблиць, налаштування автентифікації, шифрування.

Бачимо необхідність у розробці програмного забезпечення яке б відрізнялося простотою використання та налаштування програмних інтерфейсів, наявністю вбудованої бази даних, інших ресурсів та засобів, складність розуміння, налаштування та впровадження яких зростає поступово. Низку інших перевіг у порівнянні з іншими сервісами наведено у таблиці 2.

Таблиця 2

Порівняння програмних засобів

Особливість	AWS API Gateway	Directus	Запропонований до розробки сервіс
Джерело даних	Зовнішнє, з ряду доступних в AWS баз даних	зовнішня SQL база даних	Вбудоване внутрішнє
Складність налаштування	Висока	Середня	Низька
Додаткова валідація/кастомізація даних	Так, використовуючи інші сервіси	Так	Так
Контроль доступу до даних	Так, використовуючи інші сервіси	Так	Ні
Автоматична документація	Ні	Ні	Так
Підтримувані типи API	REST, Websocket	REST, GraphQL, Websocket	REST

Висновки з даного дослідження
і перспективи подальших розвідок у даному напрямі

Дослідження надало всебічне розуміння передумов для розробки ефективного сервісу API. Дотримуючись встановлених стандартів та використовуючи переваги віртуалізації та контейнеризації, запропонований сервіс спрямований на забезпечення надійної, зручної платформи для створення та управління програмними інтерфейсами. Майбутні дослідження мають зосередитися на усуненні виявлених потенційних ризиків та подальшому вдосконаленні сервісу відповідно до потреб користувачів і технологічних досягнень.

Література

1 Fielding R. T. Architectural styles and the design of network-based software architectures. Home - UC Irvine Donald Bren School of Information & Computer Sciences. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 09.06.2024).

2 Fielding R., Nottingham M., Reschke J. HTTP Semantics. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc9110.html

3 What is a container? | docker. Docker. URL: https://www.docker.com/resources/what-container/.

4 Scarfone K. A., Souppaya M. P., Hoffman P. Guide to security for full virtualization technologies. Gaithersburg, MD: National Institute of Standards and Technology, 2011. URL: https://doi.org/10.6028/nist.sp.800-125.

5 What is amazon API gateway? – amazon API gateway. URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html