УДК 004.75:004.455

DOI: 10.31673/2412-9070.2024.012629

Д. С. МІРОНОВ, студент;
А. М. ТУШИЧ, доктор філософії (PhD), доцент;
Д. Є. КОЗЛОВ, аспірант,
Державний університет інформаційно-комунікаційних технологій, Київ

ДОСЛІДЖЕННЯ АРХІТЕКТУРИ API ДЛЯ ЗВ'ЯЗКУ МІЖ КЛІЄНТОМ ТА СЕРВЕРОМ

Архітектури API є важливим компонентом сучасних систем. Вони забезпечують взаємодію між різними програмами та сервісами, а також надають доступ до даних та функціональності іншим системам. Також не менш важливим є влив API на бізнес. Є кілька тенденцій API, які впливають на бізнес: розширення використання API, спрощення застосування API, безпека API. Ці тенденції дають змогу компаніям збільшити взаємодію з клієнтами, співпрацювати з партнерами, оптимізувати внутрішні процеси. Однак вибір правильної архітектури API є складним завданням. Існує безліч різних архітектур API, кожна з яких має свої переваги та недоліки. Крім того, різні типи систем мають різні вимоги до архітектури API. Можливими проблемами можуть стати відсутність достатньої документації, що додає труднощів розробникам під час вибору правильного вирішення, також несумісність між різними протоколами, форматами даних, методів запитів і відповідей, ускладнюючи взаємодію між різними системами, та значна вартість реалізації архітектури API, що є важливим фактором для розробників. Як майбутні перспективи можуть бути створення нових архітектур, розширення використання API, автоматизація API.

Запропоновану статтю присвячено вивченню архітектур API для зв'язку між клієнтом та сервером. Мета дослідження — визначити, яка архітектура API є найкращою для конкретного типу системи з певними вимогами. Для досягнення цієї мети було проведено аналіз наявних архітектур API, який охоплював вивчення переваг та недоліків різних архітектур, а також їх відповідності вимогам різних типів систем. Було розглянуто такі архітектури API: SOAP, REST, GraphQL, gRPC, WebSocket, WebHook, MQTT, що мало на меті допомогти розробникам у виборі оптимальної архітектури API для відповідності специфічним потребам їхніх проєктів.

Ключові слова: API-архітектура; SOAP; REST; GraphQL; gRPC; WebSocket; WebHook; MQTT; клієнт-сервер.

ВСТУП

Постановка проблеми. API — це програмний інтерфейс, що з'єднує сучасні технології з бізнес-екосистемами. Компанії, які використовують API, можуть монетизувати дані, створювати вигідні партнерства та відкривати нові шляхи для інновацій і зростання.

Постійні зміни на ринку та в технологічному ландшафті сприяють зростанню інтересу до архітектур API. Компанії відшукують нові цінні способи використання раніше ізольованих джерел даних, оскільки все більше споживачів і підприємств використовують веб- та мобільні застосунки у своїй повсякденній діяльності. Сьогодні інтерфейси дають змогу використовувати дані, надихаючи розробників на створення нових бізнес-можливостей та вдосконалення наявних продуктів, систем та операцій.

Багато інноваційних компаній мають розроблювати та впроваджувати ефективну архітектуру в цьому динамічному середовищі. Нині власники бізнесу, які намагаються підібрати найкращі практики для розроблення API, стикаються з дилемами щодо вибору цифрових каналів для максимізації прибутку. Отже, потрібно з'ясувати, яка архітектура може бути найбільш прийнятною для інформаційної системи, котра взаємодіє з IoT-пристроями на базі Arduino.

ОСНОВНА ЧАСТИНА

З моменту свого зародження наявна екосистема API стала набагато складнішою та досконалішою, щоб відповідати дедалі вищим вимогам сучасного цифрового світу. На початку свого існування API були відносно простими і часто становили прості кінцеві точки, які клієнти могли використовувати для отримання даних, взаємодії один з одним або виконання певних дій. Однак зі зростанням використання API нарощувалась і складність екосистеми API.

Сьогодні системи API охоплюють чимало компонентів і рівнів, кожен з яких має вирішальне значення для забезпечення належної роботи API. Наприклад, система API може містити різні API, різні протоколи та формати передавання даних, а також безліч інструментів для створення, тестування та керування API.

Ці інструменти варіюються від шлюзів API для керування трафіком і доступом до API, від платформи керування API, яка надає утиліти для розроблення, публікації та керування API, до різних інших інструментів для конкретних завдань, зокрема автентифікації, авторизації та аналітики.

Задля створення сервісу чи програми використовують програмний інтерфейс API. Проблема полягає в тому, що є кілька видів архітектури API, які відповідають різним вимогам сервісів чи про-

грам. Тому далі детальніше розглянемо типи API-архітектур, а саме: SOAP, REST, GraphQL, gRPC, WebSocket, WebHook, MQTT.

1. SOAP — простий протокол доступу до об'єктів, є офіційним протоколом API. Формат SOAP описує API за допомогою мови опису вебслужб (WSDL). Його написано мовою розширюваної розмітки (XML). Цей формат накладає внутрішні стандарти відповідності, які підвищують безпеку, стабільність, ізоляцію та довговічність. Ці функції забезпечують надійні транзакції бази даних, що робить SOAP кращим для розвитку підприємства. Коли користувач запитує вміст через SOAP API, він проходить через стандартні протоколи рівня. Відповідь надається у форматі XML, який читається людиною та машиною. Як і REST API, SOAP API не кешують/зберігають інформацію. Якщо вам знадобляться дані пізніше, ви маєте подати інший запит. SOAP підтримує обмін даними стану та без нього. Прикладом використанням є банківська система, яка використовує SOAP для обміну фінансовою інформацією між різними банківськими системами для безпечного і надійного передавання фінансових транзакцій.

2. REST — сучасна та найпопулярніша архітектура API, якою послуговуються розробники. Цю архітектуру використовують для розроблення програм клієнт-сервер. Це не протокол чи стандарт, тому ви можете реалізувати його в різні способи. Цей аспект підвищує вашу гнучкість як розробника. REST надає доступ до запитуваних даних, що зберігаються в базі даних. Ви можете виконувати основні функції CRUD за допомогою REST API. Коли клієнти запитують вміст через RESTful API, вони мають використовувати відповідні заголовки та параметри. Заголовки містять корисні метадані для ідентифікації ресурсу, зокрема статус і коди авторизації. Інформація, що передається через HTTP, може бути у формі JSON, HTML, XML або простого тексту. JSON — найпоширеніший формат файлів, який використовується для REST API. Як приклад можна взяти соціальну мережу, яка застосовує REST API для доступу до профілів користувачів, публікацій та взаємодії із застосунками. Наприклад, Twitter, котрий надає можливість взаємодії зі своїм API через REST для отримання та публікації твітів.

3. GraphQL — мова запитів для API. Це середовище виконання на боці сервера, яке здійснює запити на основі визначеного набору даних. GraphQL має конкретні випадки використання. Його архітектура здатна декларувати конкретну інформацію, яка вам потрібна. На відміну від архітектури REST, де HTTP обробляє клієнтські запити та відповіді, GraphQL запитує дані за запитом. Сервіс GraphQL визначає типи та поля цих типів, а потім надає функції для кожного поля та типу. Ця служба отримує запити GraphQL для перевірки та виконання. Спочатку перевіряється запит, щоб переконатися, що він стосується визначених типів і полів. Потім здійснюються відповідні функції для отримання бажаного результату. Прикладом використання є інтернет-магазин, де GraphQL використовують для запитів різноманітних даних, зокрема інформації про товари, описи категорій та замовлення.

4. gRPC — платформа з відкритим кодом, розроблена на основі Remote Procedure Call (RPC) і може бути реалізована в усіх середовищах розробки. Ця технологія забезпечує можливість чіткого, зручного та двонапрямленого зв'язку та координації між клієнтом і сервером, а також спрощує побудову підімкнених систем. Цю технологію розроблено на основі HTTP/2. Однією з цікавих особливостей цієї технології є зв'язок між службами та всіма центрами оброблення даних із можливістю відстеження, балансування навантаження перевірки рівня доступу та працездатності згаданої служби. Прикладом використання є мікросервісна архітектура, де gRPC використовується для забезпечення ефективного міжсервісного зв'язку.

5. WebSocket — протокол на основі TCP, який використовують для зв'язку між браузером користувача та сервером. Цей протокол дає змогу програмістам передавати дані між браузером і сервером двостороннім способом без повторних запитів. WebSocket підтримується в усіх сучасних браузерах і є однією з найпростіших комунікаційних технологій у сфері розроблення вебзастосунків. Використовуючи WebSocket, можна обмінюватися даними між браузером і сервером для стабільного передавання даних і інформації. Інакше кажучи, протокол WebSocket дає можливість програмістам постійно передавати дані між браузером і сервером без потреби у повторному надсиланні запитів. Використовуючи цю вебфункцію, ви можете здобути більше переваг від зв'язку між програмами. Цей протокол використовується для багатьох вебзастосунків, таких як програми чату, онлайн-ігри, регулярні оновлення тощо. Прикладом застосування є жива чат-система, яка використовує WebSocket для негайного передавання повідомлень між користувачами.

6. WebHook — спосіб серверів надсилати повідомлення клієнтам, коли щось відбувається. Тобто повідомлення надсилаються автоматично, коли їх подія запускається у вихідній системі. Вони використовують зворотні виклики HTTP або запити POST для доставляння корисних даних, які містять інформацію про події. Клієнти реєструють свої вебхуки на серверах, надаючи URL-адреси, які можуть отримувати корисні дані. WebHook — це керовані подіями зворотні виклики HTTP та асинхронні операції. Прикладом використан-

ня є система керування контентом, де WebHooks використовуються для надсилання сповіщень про оновлення або події, зокрема створення нового контенту або оновлення даних. Наприклад, GitHub використовує WebHook для надсилання сповіщень про коміти та зміни в репозиторіях.

7. MQTT — це простий протокол обміну повідомленнями, розроблений для обмежених пристроїв, мереж із високою затримкою та пристроїв із низькою пропускною здатністю. Це відповідне рішення для обміну даними для зв'язку між машинами (M2M), яке відіграє важливу роль в Інтернеті речей (IoT). Комунікація MQTT працює як система публікації та підписки. Пристрої транслюють повідомлення на певну тему. Усі пристрої, які підписалися на цю тему, отримають це повідомлення. Одним із основних застосувань є надсилання повідомлень для керування виходами, читання, а також публікація даних із давачів, підімкнених до інтернету. Прикладом використання є Інтернет речей (IoT), де MQTT використовується для передавання даних між пристроями IoT та серверами. Наприклад, системи контролю за «розумними будинками», де пристрої застосовують MQTT для взаємодії з центральним сервером.

Вимогам для інформаційної системи, де присутні IoT-системи, більш відповідає WebSocket, який забезпечує миттєве відправлення даних у реальному часі. Arduino має можливості для підімкнення до WebSocket, прийняття та відправлення даних.

Компоненти архітектури API

Розуміючи ці компоненти та впроваджуючи їх в архітектуру API, організації можуть забезпечити успішне доставляння API, яке є надійним, безпечним та простим у використанні. Це, зі свого боку, допоможе їм максимізувати цінність їхніх API і створити позитивний досвід для користувачів у відкритих специфікаціях API. Тому для розроблення правильного та безпечного API потрібно сформувати та інкапсулювати компоненти. Розглянемо далі ці компоненти.

1. *Модель даних* — інкапсулює структуру даних, якими обмінюватимуться між клієнтами та серверами через API. Це охоплює такі елементи, як об'єкти, атрибути та зв'язки між об'єктами.

2. *Кінцеві точки* — це конкретна URL-адреса, яка дає змогу клієнтам робити запити, а сторона сервера отримує відповіді на http-запити. Хороша архітектура веб-API повинна мати чітко визначені кінцеві точки, щоб забезпечити повернення правильних даних заголовка типу вмісту для успішної відповіді на тіло запиту http-заголовка даного запиту, що є простим процесом.

3. *Автентифікація та авторизація* — забезпечує автентифікацію (підтвердження того, що користувачі є тими, за кого вони себе видають) і авторизацію (визначаючи, до яких даних або дій вони можуть отримати доступ). Це можна зробити за допомогою токенів, OAuth 2.0 та інших методів із кодами стану.

4. *SDK* — пакети розроблення програмного забезпечення (SDK) надають розробникам прості у використанні бібліотеки програмування, інструменти керування API та документацію для швидкого створення програм, які отримують доступ до служб API.

5. *Версії* — надають змогу паралельно існувати різним версіям API, гарантуючи, що наявні програми не зламаються, коли вносяться зміни до основного або менш основного номера версії базової кодової бази.

6. *Аналітика та моніторинг* — дають можливість контролювати та аналізувати API для швидкого виявлення будь-яких проблем із продуктивністю або ризиків безпеки, а також відстежувати шаблони використання та поведінку користувачів. Це допомагає організаціям ухвалювати кращі рішення щодо своїх програм API, надавати кращі послуги своїм клієнтам і гарантувати відсутність передчасної оптимізації.

Вибір відповідної API-архітектури

Вибір оптимальної API-архітектури залежить від потреб проєкту та специфіки вимог до системи. Для визначення правильного напрямку варто брати до уваги наведені далі фактори.

1. Потреби проєкту — може бути REST, оскільки має просте, стандартизоване та масштабоване API, також це може бути SOAP для більш структурованих систем із високими вимогами до безпеки та надійності.

2. Гнучкість керуванням даними — GraphQL, забезпечує потрібну більшу гнучкість та точність у виборі даних, що повертаються із сервера.

3. Ефективність міжсервісного зв'язку — gRPC, являє собою мікросервісну архітектуру, де важлива ефективність та низький час відповіді між сервісами.

4. Реальний час та стримінг даних — WebSocket, відповідає вимогам для негайної взаємодії в реальному часі, наприклад для чатів або онлайн-ігор.

5. Підписка на події та оновлення — може бути WebHook або MQTT для сценаріїв, де важливі сповіщення про події або оновлення, зокрема реакція на зміни в системі або IoT-сценарії.

6. Стандартизація та інтеграція — SOAP або REST, якщо важлива стандартизована інтеграція з великою кількістю партнерів чи систем.

ВИСНОВКИ

Останніми роками використання API набуває успішного розвитку, а отже, складність екосистеми API дедалі стрімко зростає. Сучасна архітекту-

ра API зазвичай становить кілька рівнів, які працюють разом для створення комплексної системи API, і охоплює різні компоненти та служби, які підтримують розгортання та використання API. Систему API слід розробляти з огляду на безпеку та заходи для запобігання неавторизованому доступу, захисту конфіденційних даних і захисту від потенційних загроз безпеці.

За результатами дослідження для комунікації системи IoT з інформаційною системою було вибрано WebSocket, оскільки має можливість отримувати та відправляти дані в реальному часі. Також для оптимізації API-сервера потрібно на боці Arduino вдосконалити код та відправляти дані, коли вони відрізняються від попередніх. Також для інших було встановлено, що найкраща архітектура API для конкретного типу системи залежить від її вимог.

Список використаної літератури

1. ***Мартін Р. С.*** *Clean Architecture. Фабула, 2019. 368 с.*

2. ***Introduction*** *to GraphQL [Електронний ресурс]. URL: https://graphql.org/learn/.*

3. ***Болл К. Дж.*** *Hacking Apis. Breaking Web Application Programming Interfaces. No Starch Press, 2022. 368 с.*

4. ***Lombardi A.*** *WebSocket Lightweight Client-Server Communications. O'Reilly Media, 2015. 144 с.*

5. ***Richardson L.*** *RESTful Web APIs: Services for a Changing World 1st Edition. O'Reilly, 2013. 406 с.*

6. ***Learn REST:*** *A RESTful Tutorial [Електронний ресурс]. URL: https://www.restapitutorial.com/.*

D. Mironov, A. Tushych, D. Kozlov

RESEARCH OF API ARCHITECTURE FOR CLIENT-SERVER COMMUNICATION

API architectures are an important component of modern systems. They provide interaction between various programs and services, and also provide access to data and functionality to other systems. Equally important is the impact of APIs on business. There are several API trends that affect business: API usage expansion, API usage simplification, API security. These trends have a significant impact on business, they allow companies to: increase interaction with customers, collaborate with partners, optimize internal processes. However, choosing the right API architecture is a difficult task. There are many different API architectures, each with its own advantages and disadvantages. Also, different types of systems have different API architecture requirements. Possible problems can be a lack of sufficient documentation, which makes it difficult for developers to choose the right solution, also incompatibilities between different protocols, data formats, request and response methods, which can make it difficult to interact between different systems, and the cost of implementing an API architecture can be significant, which can be important factor for developers. Future prospects may include: creation of new architectures, expansion of API usage, API automation. This study is devoted to the study of API architectures for communication between the client and the server. The goal of the study is to determine which API architecture is best for a specific type of system with specific requirements. To achieve this research goal, an analysis of existing API architectures will be conducted. The analysis will include a study of the advantages and disadvantages of different architectures, as well as their compatibility with the requirements of different types of systems. The results of the study aim to consider the following API architectures: SOAP, REST, GraphQL, gRPC, WebSocket, WebHook, MQTT, and also help developers in choosing the optimal API architecture to meet the specific needs of their projects.

Keywords: API architecture; SOAP; REST; GraphQL; gRPC; WebSocket; WebHook; MQTT; client server.