УДК 004.455.2:004.738.1

Т. І. Коробейнікова[1], Л. А. Савицька[2]

УДОСКОНАЛЕНИЙ МЕТОД РОЗРОБКИ API ПІДВИЩЕНОЇ ШВИДКОДІЇ

1 − Національний університет «Львівська Політехніка», Львів

2 − Вінницький національний технічний університет, Вінниця

Анотація. Дана робота присвячена дослідженню методів та засобів розробки API з метою розробки удосконалених підходів для підвищення швидкодії API, його захищеності, орієнтованості в першу чергу на мобільний трафік, на кросплатформенність, можливість користувачам самостійно популяризувати його, знаходити нові сфери застосування та простоти розробки.

Ключові слова: методи та засоби розробки API, Web-API, підвищення швидкодії API, проблеми розробки API, інтеграція додатків, REST-API, middleware, вимоги до Web-API.

Аннотация. Данная работа посвящена исследованию методов и средств разработки APE с целью разработки усовершенствованных подходов для повышения быстродействия APE, его защищенности, ориентированности в первую очередь на мобильный трафик, на кросплатформенность, возможность пользователям самостоятельно популяризировать его, находить новые сферы применения и простоты разработки.

Ключевые слова: методы и средства разработки APE, Web-APE, повышение быстродействия APE, проблемы разработки APE, интеграция приложений, REST-API, middleware, требования к Web-API.

Abstract. This work is devoted to the study of methods and tools for developing ARI in order to develop improved approaches to improve the performance of ARI, its security, focus primarily on mobile traffic, cross-platform, the ability to promote it, find new applications and ease of development.

Keywords: methods and means of API development, Web-API, increase of ARI performance, problems of ARI development, application integration, REST-API, middleware, Web-API requirements.
DOI: https://doi.org/10.31649/1999-9941-2021-50-1-31-35.

Вступ

Із розвитком сучасних інтернет-технологій перед програмістами часто постає задача переносу продукту на інші платформи, розширення функціоналу та створення додаткових клієнтів веб-сервісів. Нині за оцінками експертів близько 75% інтернет трафіку припадає на мобільні пристрої. Відповідно, набагато збільшилась потреба у мобільних клієнтах інтернет-додатків та веб-сервісів [1, 2]. Все більшого поширення набуває стратегія «mobile first». Відповідно, підвищуються вимоги до швидкості та продуктивності API (Application Programming Interface), який забезпечує зручну інтеграцію веб-сервісів, конфіденційність та захищеність.

Актуальність

Нині популярність веб-сервісів та платформ багато в чому залежить від внутрішньої структури ПЗ та його ієрархії. Розробкаефективного та швидкодіючого API дозволяє легко інтегрувати продукт в інші сервіси та додатки. Добре організована серверна частина програми є запорукою безпроблемного та швидкого розширення функціоналу та легко масштабування. Продукт, який пропонує API розробникам, дає можливість користувачам самим популяризувати його, знайти нові сфери застосування програми. API має бути орієнтований в першу чергу на мобільний трафік, на кросплатформенність продукту та його захищеність від DDOS-, csrf-атак та інших кіберзагроз. Тому виникає необхідність у методі вдосконалення розробки API для підвищення його швидкодії та захищеності.

Мета

Метою статті є розробка удосконаленого методу розробки API підвищеної швидкодії та захищеності за рахунок використання додаткового шару функцій проміжної обробки та принципу незбереження стану клієнта.

Для досягнення поставленої у статті мети необхідно вирішити задачі, що наведені нижче:

1. Проаналізувати сучасні методи та засоби розробки API, виконати їх порівняльну характеристику;
2. Визначити основні вимоги до створення ефективного та швидкісного API;
3. Розробити метод розробки API для підвищення його швидкодії та захищеності.

Аналіз сучасних методів та засобів розробки API та їх порівняльна характеристика

До основних технологій розробки API відносять SOAP. SOAP (англ. Simple Object Access Protocol) − протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML. Спочатку SOAP призначався, в основному, для реалізації віддаленого виклику процедур (RPC). Зараз протокол використовується для обміну повідомленнями в форматі XML, а не тільки для виклику процедур. SOAP може використовуватися з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTP та інші. Проте, його взаємодія з кожним із цих протоколів має свої особливості, які

потрібно відзначити окремо. Найчастіше SOAP використовується разом з HTTP. SOAP є одним із стандартів, на яких ґрунтується технологія веб-сервісів.

Основним конкурентом SOAP є REST архітектура. REST (англ. Representational State Transfer, «передача репрезентативного стану») — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів [3]. Системи, що підтримують REST, називаються RESTful-системами. У загальному випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL. Кожна URL в свою чергу має строго заданий формат.

Відсутність додаткових внутрішніх прошарків означає передачу даних в тому ж вигляді, що і самі дані. Тобто ми не загортаємо дані в XML, як це робить SOAP і XML-RPC, не використовуємо AMF, як це робить Flash і так далі. Просто віддаємо самі дані. Але це може спричинити за собою проблеми пов'язані з безпекою передачі даних.

Основні принципи REST:

- кожна одиниця інформації повинна мати ID;
- зв’язок між ресурсами (гіперпосилання);
- використання стандартних методів (CRUD);
- ресурси можуть мати багато представлень;
- незбереження стану клієнта.

Порівнюючи SOAP з REST, слід сказати, що SOAP підтримує WS-Security (Web Services Security), яка надає додаткові функції безпеки. Крім того, SOAP підтримує набір властивостей ACID атомарність, узгодженість, ізольованість та довговічність. Хоча, зазвичай, інтернет-додатки не вимагають транзакційної надійності такого рівня. Серед недоліків SOAP слід відзначити невиправдано великий об’єм повідомлень. SOAP використовує XML, формат якого надлишковий і містить багато некорисної інформації. Відповідно, це суттєво впливає на швидкодію системи. Крім цього, це накладає певні обмеження на формат відповідей та формат представлення даних. Одним із суттєвих недоліків є те, що SOAP не підтримує кешування запитів.

Серед переваг REST API слід зауважити, що REST дозволяє багато різних форматів даних, а не тільки XML, а також забезпечує кращу підтримку браузерних клієнтів, так як підтримує JSONP. REST має кращу продуктивність та масштабованість, так як не зберігає стан клієнта. На відміну від SOAP, запити REST можуть бути закешовані. Розробка API та створення клієнтів є також набагато легшою. У швидкості REST набагато випереджає SOAP за рахунок незбереження стану клієнта, підтримки різноманітних форматів даних та кешування.

Розглянемо основні інструменти для розробки API:

- API на Sinatra мовою Ruby;
- API мовою Python на основі Flask;
- API на основі Slim мовою PHP;
- API на NodeJS.

Ruby – інтерпретована, повністю об’єктно-орієнтована мова з чіткою динамічною типізацією. Серед її особливостей слід відзначити лаконічний і простий синтаксис, наявність автоматичного прибиральника сміття, реалізацію багатьох шаблонів програмування, незалежну від операційної системи підтримку невитискальної багатопоточності, кросплатформенність та ін. Sinatra – безкоштовний і відкритий програмний каркас, призначений для розробки веб-додатків. Серед його переваг слід виділити простоту та гнучкість, а також підтримку «прошарків» (middleware) – компонентів, які знаходяться між сервером і вашим додатком які відстежують або/і маніпулюють HTTP-запитами та відповідями для надання різної функціональності. Одним з головних недоліків є те, що Sinatra не підтримує архітектурний шаблон MVC [4].

Python – високорівнева мова програмування, орієнтована на підвищення ефективності та читабельності коду. Він відрізняється чітким послідовним синтаксисом, продуманою модульністю та масштабованістю. Flask – мікрофреймворк для створення веб-додатків. Це мінімалістичний програмований каркас, який надає лише базові можливості. За замовчуванням він не включає в себе рівень абстракції бази даних, валідацію форм та інші додаткові можливості. Але Flask підтримує розширення, які можуть задовольнити фактично будь-які потреби.

PHP – скриптова мова програмування, яка є однією з найпоширеніших мов, що використовуються у сфері веб-розробок. Серед її переваг можна виділити ефективність, наявність інтерфейсів до багатьох баз даних, універсальність. До її недоліків можна віднести відсутність строгої типізації, відсутність підтримки Unicode у версіях до 6.0. Крім того, PHP дозволяє вставляти html-код безпосередньо в код скрипта. Це порушує принцип розподілення програмної логіки від її представлення. Slim – це PHP мікрофреймворк, який призначений для розробки API та невеликих веб-додатків. Він містить дуже зручну структуруURL з шаблонами сторінок, флеш-повідомленнями, зашифрованими cookies та іншими особливостями. Основ-

ним призначенням цього фреймворку є саме розробка API. Він містить додаткові компоненти, які дозволяють захистити систему від підробки крос-доменних запитів та кешування запитів.

NodeJS – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Використання JavaScript у якості серверної мови надає такі переваги як асинхронна однопотокова модель модель виконання запитів, неблокуючий ввід/вивід. Однією з головних переваг платформи NodeJS є пакетний менеджер модулей npm (node package manager) [5]. Його ідея – створення маленького програмного блоку, який вирішує одну проблему, але робить це добре і якісно. Ці маленькі побудовані блоки дозволяють компонувати величезні проекти з цих невеликих частин. Це відповідає принципу відкритості та багаторазового використання коду. NodeJS надзвичайно зручний для створення швидких масштабованих мережевих додатків, оскільки дозволяє одночасно обробляти величезну кількість з'єднань з високою пропускною здатністю, що є рівноцінним високій масштабованості. У порівнянні із традиційними веб-сервісами, де кожне з'єднання (запит) породжує новий потік, навантажуючи оперативну пам'ять системи і, врешті-решт, розбираючи цю пам'ять без залишку, NodeJS працює набагато економніше [6]. Він працює в єдиному потоці, при викликах використовує неблокуюче введення/виведення, яке дозволяє підтримувати десятки тисяч конкурентних з'єднань. Крім того, ця платформа чудово поєднується з NoSQL базами даних, зокрема MongoDB, які є дуже гнучкими при зміні чи масштабуванні та здатні до розподілених обчислень.

Основні вимоги до розробки ефективного та швидкісного API

Удосконалений ефективний та швидкісний API повинен мати такі характеристики:

1. Масштабованість. API має бути структурований і спроектований так, щоб була можливість легко масштабувати систему або розширити її функціонал без погіршення швидкості та ефективності роботи.
2. API повинен мати стандартизований та зручний формат запитів. Крім того, для підвищення швидкодії системи слід використовувати засоби кешування запитів. В свою чергу, відповіді сервера повинні мати явне чи неявне позначення як кешовані чи некешовані з метою попередження отримання клієнтами застарілих або невірних даних у відповідь на подальші запити.
3. API повинен мати стандартизований та зручний формат відповідей та підтримувати різні формати представлення даних. Це дозволить забезпечити кросплатформенність системи, розширити сфери використання продукту, полегшить його популяризацію та інтеграцію в інші веб-сервіси.
4. Виконання принципу «незбереження стану клієнта» (stateless). Серверна сторона не повинна якимось чином залежати від того, з мобільного чи настільного пристрою підключився клієнт. Всі запити від клієнта мають бути складені так, щоб сервер отримав всю необхідну інформацію для виконання запиту без необхідності зберігання стану клієнта в період між запитами.
5. API має реалізовувати модель клієнт-сервер. Відділення потреб клієнта від потреб сервера, який зберігає дані, полегшує перенесення коду клієнтського інтерфейсу на інші платформи, а спрощення серверної частини покращує масштабованість. Такий підхід дозволяє клієнту та серверу розвиватися незалежно один від одного.
6. Має бути передбачена можливість повернення помилок виконання запиту. Причому помилки мають бути чітко описані, щоб не тільки користувач знав, що йому необхідно зробити, але й ви легко орієнтувалися, коли користувач надсилає вам запит для вирішення проблеми. Але необхідно також уникати зайвої надлишковості, щоб не заплутувати користувачів.
7. Забезпечення захищеності та конфіденційності системи. API має бути захищений від таких загроз як DDOS-атаки, csrf-атаки.
8. Застосування проміжних серверів може суттєво підвищити масштабованість системи за рахунок балансування навантаження і розподіленого кешування. Причому завдяки ієрархічності структури клієнт зазвичай не може визначити, взаємодіє він напряму з сервером чи з проміжним вузлом системи.

Дотримання вищенаведених вимог дозволить створити якісний та ефективний API, який буде забезпечувати надійність, масштабованість, портативність компонентів, прозорість системи взаємодії, здатність еволюціонувати та легко змінюватися по мірі необхідності [7].

Розробка удосконаленого методу розробки API для підвищення його швидкодії та захищеності

Так як API повинен працювати швидко, а також легко масштабуватися та бути здатним до безпроблемної зміни чи розширення функціоналу, запропонуємо використати нереляційну базу даних. Так як вона не потребує опису схеми таблиць та використовує JSON-подібні документи, то нереляційна база даних є дуже гнучкою та зручною при зміні та масштабуванні. Крім того, вона здатна до більш швидкого вилучення простих структур даних, а також має розподілений доступ до даних, розміщених на різних серверах. До того ж, є можливість використовувати MapReduce для задач обробки даних. Це потужний інструмент для розподілених обчислень.

Для покращення швидкодії та захищеності API пропонується створити проміжний шар за допомогою функцій проміжної обробки. Це допоможе збалансувати навантаження, і за рахунок цього підвищити швидкодію. Крім цього, в цьому додатковому шарі буде відбуватися перевірка механізму CORS, захист від крос-доменних підробок запитів та автентифікація. Це допоможе суттєво підвищити конфіденційність та захищеність системи.

Функції проміжної обробки можуть виконувати такі завдання:

- виконання будь-якого коду.
- внесення змін до об'єктів запитів і відповідей.
- завершення циклу «запит-відповідь».

виклик наступного проміжного обробника зі стеку.

Приклад виклику функції проміжної обробки зображено на рисунку 1.

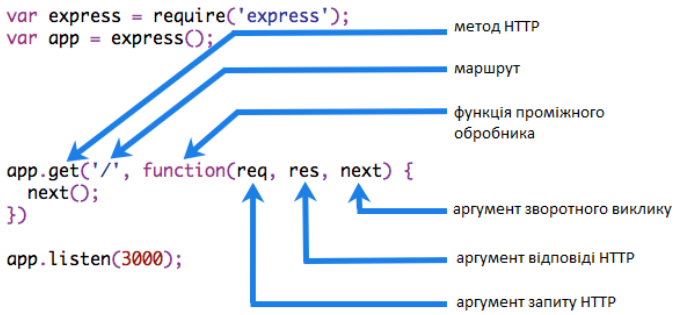

Рисунок 1 – Елементи виклику функції проміжної обробки

Запити будуть проходити через функції проміжного шару перед тим, як виконати своє основне завдання. Ці функції мають доступ до об’єкту запиту, об’єкту відповіді та до наступної функції проміжної обробки. Кожна з цих функцій зможе певним чином змінювати запит чи відповідь, переправити запит на іншу адресу, передавати керування іншій функції. Крім цього, проміжна функція може генерувати кінцеву відповідь та віддати її користувачеві, зупиняючи тим самим рух у проміжному стеку. Схема проміжного шару зображена на рисунку 2.

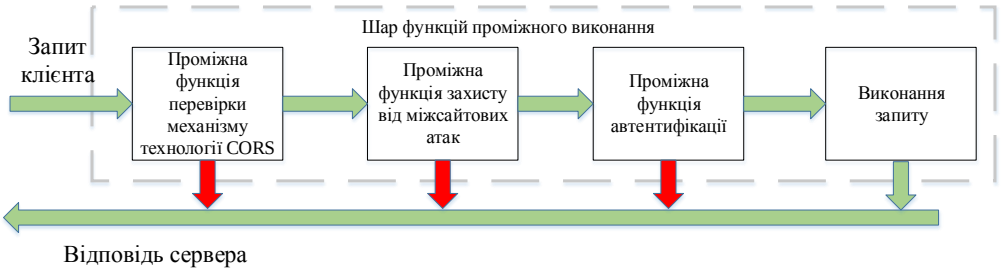

Рисунок 2 – Шар функцій проміжної обробки

Висновки

У даній статті був проведений аналіз методів та засобів розробки API, виконано їх порівняльну характеристику та аналіз їх відповідності визначеним вимогам, було визначено основні вимоги до створення удосконаленого ефективного та захищеного API. Також було проаналізовано критичні задачі, які виникають у процесі розробки API, що дало можливість запропонувати шляхи їх вирішення.

Запропонований удосконалений метод розробки API дозволяє забезпечити підвищення швидкодії та захищеності системи за рахунок створення додаткового шару функцій проміжної обробки, який також допоможе збалансувати навантаження. Використання NoSQL баз даних зробить систему більш гнучкою та масштабованою і дозволить орієнтувати її на виконання конкретних задач.

Список літератури

[1] О. І. Гороховський, Т. І. Трояновська, О. Д. Азаров, *Інформаційна технологія доставки контенту у системах комп’ютеризованої підготовки спеціалістів: монографія*. Вінниця, Україна: ВНТУ, 2016, 160 с.

[2] Oleg I. Pursky, *Big Data Processing: methods, models and information technologies: monograph*. Steyr, Austria: Shoida GmBH, 2019, 234 p.

[3] С. Ньюмен, *Создание микросервисов*. Питер, Россия: ООО Издательство «Питер», 2016, 300 с.

[4] М. Таллоч, *Знакомство с Windows Azure. Для ИТ-специалистов*. Москва, Россия: ЭКОМ Паблишерз, 2014, 154 с.
[5] Подробное описание возможностей разработки с Microsoft Azure Cloud Services. [Електронний ресурс]. Режим доступу: http://habrahabr.ru/company/microsoft/blog/242543. Дата звернення: Лют., 20, 2016.
[6] О. Г. Тімінський, «Виникнення, розвиток і проблеми інформаційних технологій управління», *Управління розвитком складних систем*, № 25, с. 86−90, 2016.
[7] М. В. Князєва та ін., *Рівень розвитку техніки і технологій в XXI столітті. Частина 1: Серія монографій*. Одеса, Україна: КУПРІЄНКО СВ, 2019, 227 с.

Стаття надійшла: 11.12.2020.

References

[1] O. I. Horokhovskyi, T. I. Troianovska, O. D. Azarov, *Informatsiina tekhnolohiia dostavky kontentu u systemakh kompiuteryzovanoi pidhotovky spetsialistiv: monohrafiia.* Vinnytsia, Ukraina: VNTU, 2016, 160 s.
[2] Oleg I. Pursky, *Big Data Processing: methods, models and information technologies: monograph*. Steyr, Austria: Shoida GmBH, 2019, 234 p.
[3] S. N'jumen, *Sozdanie mikroservisov*. Piter, Rossija: OOO Izdatel'stvo «Piter», 2016, 300 s.
[4] M. Talloch, Znakomstvo s Windows Azure. Dlja IT-specialistov. Moskva, Rossija: JeKOM Pablisherz, 2014, 154 s.
[5] Podrobnoe opisanie vozmozhnostej razrabotki s Microsoft Azure Cloud Services. [Elektronnyi resurs]. Rezhym dostupu: http://habrahabr.ru/company/microsoft/blog/242543. Data zvernennia: Liut. 20, 2016.
[6] O. H. Timinskyў, «Vynyknennia, rozvytok i problemy informatsiўnykh tekhnolohiў upravlinnia», *Upravlinnia rozvytkom skladnykh system*, № 25, s. 86−90, 2016.
[7] M. V. Kniazieva ta in., *Riven rozvytku tekhniky i tekhnolohii v XXI stolitti. Chastyna 1: Seriia monohrafii*. Odesa, Ukraina: KUPRIIeNKO SV, 2019, 227 s.

Відомості про авторів

Коробейнікова Тетяна Іванівна – кандидат технічних наук, доцент кафедри безпеки інформаційних технологій.

Савицька Людмила Анатоліївна – кандидат технічних наук, доцент кафедри обчислювальної техніки.

Т. И. Коробейникова[1], Л. А. Савицкая[2]

УСОВЕРШЕНСТВОВАННЫЙ МЕТОД РАЗРАБОТКИ API ПОВЫШЕННОГО БЫСТРОДЕЙСТВИЯ

1 − Национальный университет «Львовская Политехника», Львов

2 − Винницкий национальный технический университет, Винница

T. I. Korobeinikova[1], L. A. Savytska[2]

IMPROVED METHOD OF INCREASED SPEED API DEVELOPMENT

1 − National university «Lvivska Politechnika», Lviv

2 − Vinnytsia National Technical University, Vinnytsia