World Journal of Advanced Engineering Technology and Sciences

eISSN: 2582-8266

Cross Ref DOI: 10.30574/wjaets

Journal homepage: https://wjaets.com/

(Review Article)

Modernizing legacy financial systems with Api-Driven Architectures

Surendra N Koritala *

Azure Cloud Architect, Sr. IEEE Member, USA.

World Journal of Advanced Engineering Technology and Sciences, 2025, 14(03), 450-461

Publication history: Received on 01 February 2025; revised on 25 March 2025; accepted on 27 March 2025

Article DOI: https://doi.org/10.30574/wjaets.2025.14.3.0159

Abstract

Today's financial industry is progressing to adopt novel solutions that would help address the problem of outdated IT infrastructures together with the demands of a complex digital environment. This research paper provides a review of how such modernization can be attained, with particular emphasis in the value of architectures that employ APIs in harmonizing the link between traditional tense financial systems and contemporary applications. From secondary research, the context examines the main API structures: RESTful, GraphQL, and SOAP to emphasise mainly about these architectures: Scalability, Interoperability, and Performance. Through a detailed examination of architectural workflows as well as code samples, this paper shows that APIs enable basic capabilities such as routing, data sharing, and authentication while grappling with the challenges posed by legacy systems. RESTful APIs are introduced here as the simplest and quickest approach; GraphQL for its fine-grain data transfer; and, SOAP as providing the highest level of security for data transmission while dealing with sensitive financial transactions. Taken together, these architectures collectively help the financial institutions to gain in operational flexibility, advance user experiences and undergird innovation. These results point out that the API-driven modernization is not simply the technological transformation but the evolutionary necessity for maintaining the competitiveness of the financial sector at the international level. APIs enable organizations to support compliance needs, embrace new technologies, improve the client experience, and maintain core systems of record's integrity. Through the analysis of technical concepts and comparative effects on financial organizations, this paper establishes their API's importance in building the future. When applying the developed API-First approach, the research finds that modernisation is crucial for closing the remaining gap between core systems and digitalisation to support the sustainable and adaptable business growth of financial institutions.

Keywords: Finance; API; Architecture; Cash

1. Introduction

Today's financial industry is progressing to adopt novel solutions that would help address the problem of outdated IT infrastructures together with the demands of a complex digital environment. This research paper provides a review of how such modernization can be attained, with particular emphasis in the value of architectures that employ APIs in harmonizing the link between traditional tense financial systems and contemporary applications.

From secondary research, the context examines the main API structures: RESTful, GraphQL, and SOAP to emphasise mainly about these architectures: Scalability, Interoperability, and Performance [1]. Through a detailed examination of architectural workflows as well as code samples, this paper shows that APIs enable basic capabilities such as routing, data sharing, and authentication while grappling with the challenges posed by legacy systems.

RESTful APIs are introduced here as the simplest and quickest approach; GraphQL for its fine-grain data transfer; and, SOAP as providing the highest level of security for data transmission while dealing with sensitive financial transactions

* Corresponding author: Surendra N Koritala

Copyright © 2025 Author(s) retain the copyright of this article. This article is published under the terms of the Creative Commons Attribution Liscense 4.0.

[2]. Taken together, these architectures collectively help the financial institutions to gain in operational flexibility, advance user experiences and undergird innovation [3].

These results point out that the API-driven modernization is not simply the technological transformation but the evolutionary necessity for maintaining the competitiveness of the financial sector at the international level [4]. APIs enable organizations to support compliance needs, embrace new technologies, improve the client experience, and maintain core systems of record's integrity.

Through the analysis of technical concepts and comparative effects on financial organizations, this paper establishes their API's importance in building the future. When applying the developed API-First approach, the research finds that modernisation is crucial for closing the remaining gap between core systems and digitalisation to support the sustainable and adaptable business growth of financial institutions.

One of the driving reasons for integrating APIs into financial legacy systems is to increase their efficiency [5]. Through APIs, an organization can eliminate manual workflows, decline the overall system intricacy of managing different platforms, and support real-time decision making.

APIs reduce response time or time required to query data, automate processes and enhance the clients' experience. Secondly, they enhance integration of the system and application which makes it easier for the banking industry to adopt new business models like the open banking and digital wallets [6]. The connectivity with external services could also be an advantage because it enables the financial institutions to develop new products and services as well as enhance customer satisfaction, competitive advantage.

The last of the (privately-generated) driving factors is the need to address regulatory compliance as another force that is driving organizations toward API-driven architectures [7]. The financial institutions are in press to conform to the laid down strict regulatory guidelines that covers the fields such as data privacy, security and reporting. These regulations have evolved over time, and the legacy systems that exist today cannot integrate with new systems, and are often not transparent enough for compliance with all these new conditions [8].

With API integration for their systems, modern organizations can apply more adaptable and flexible compliance mechanisms, accounting for real time reporting of regulatory obligations, as well as contingency for future changes [9]. On the financial level APIs also make data and transactions more transparent for financial institutions which will help them to upgrade their physical security, audit, and other procedures to protect customer's personal information.

Also, the rising need for tailor-made and customer-centric approaches to handling financial services is contributing to the effort of upgrading legacy systems [10]. Customers expect personalized integrated experiences when engaging firms through web and application, mobile applications, banking, and chatbots.

As with other traditional financial systems which are rigid and tend to function in a closed end, they are unable to offer such experiences [11]. APIs help financial institutions to connect to a whole host of third parties including: analytics providers; artificial intelligence providers; and customer relationship management (CRM) systems to name but a few so that financial institutions can offer products and services that are unique to particular consumers [12].

This integration also helps to shift more power directly into the hands of customers over the value of their financial information and to build more trust and customer loyalty [13]. Nonetheless, integrating new technologies into legacy stock exchanges with API-based designs raises several questions that must be answered to achieve efficient results.

A primary issue is the difficulty of incorporating new API-based solutions with existing legacy systems at the middle of this integration setting [14]. They have found that often the existing system infrastructure is not optimized to support modern day APIs and there might therefore be compatibility problems. In addition, there is no API standardization among different APIs or more so the platforms, which poses a problem with interoperability [15].

Business and commercial banks should employ appropriate human capital, software, and other facilities that can correct the transition from outdated complex structures to the API-centered ones. A further issue is that in the course of the integration the security and privacy of data must be protected [16].

Banks and other financial organizations manage various customer's data, such as name, address, transactions, and other financials. Since API acts as a bridge between two systems in the context of data transfer, its use has put emphasis on making data more secure before they are transferred through AES encryption, authentication, and access controls.

2. Methodology

The methodology of this research paper involves conducting a thorough secondary analysis to examine how building with APIs forward architecture can affect positive changes on the then outmoded financial structures. This approach undertakes literature review, case studies, technical documentation and reports as the components of the knowledge base in constructing API driven modernization options.

In this regard, based on the analysis of materials scattered across various sources, this work sets the following objectives: To define the key components of API architectures, to outline the promising practices and possible difficulties, and to indicate the directions for the transition from legacy architectures to the contemporary efficient ones.

First of all, the study entailed thorough identification from scholarly articles indexing other referenced publications as well as whitepapers and industry examples. The prime area of concentration was on gathering information and intelligence about API and architecture like RESTful APIs, GraphQL, SOAP and other models like API gateways and microservices architecture.

Hence, the specifications of world-class BaaS providers and other organizations, actively using APIs, were analysed to determine their actual application. This review also embraced reports and analyses that consulting firms produced as part of their work on API and modernization of legacy systems, illustrating financial and operational gains of API incorporation.

He divided the lessons learnt from these sources according to how applicable they are to the problem posed by legacy systems, the technical viability of the solutions and the resultant strategic benefits. Thus, when examining these architectures a comparative approach was used.

Various types of API were compared according to their appropriateness when integrated into financial systems in terms of scalability, security, and performance. RESTful APIs which are stateless, lightweight communication paradigms were compared based on how they can support high transaction rate of financial operations.

GraphQL which is used for querying data was the solution in the contexts where different and diverse parameters from several traditional databases should be retrieved in specific ways. Although SOAP is a relatively older standard than REST it was contemplated for its reliability and compliance characteristics when dealing with financial data.

API gateways and microservices as well as other hybrid architectures were also examined to determine their application in facilitating integration between legacy systems and modern applications.

To observe the real use of these architectures, code snippets from public repositories and case studies about technical solutions were examined. For example, a sample code showing how a RESTful API can be developed in order to enable incorporation of a centralized core banking system with a new generation of mobile application was discussed.

The kind of analysis that was undertaken was to deconstruct these snippets in order to determine how data is retrieved, analysed, and delivered effectively and securely. Likewise, examples of GraphQL queries were analysed regarding their applications to reduce the complexity in information exchange in handling financial systems. These examples were accompanied by lessons drawn from API lifecycle management tools and frameworks that are instrumental in assisting with the deployment and support for APIs in the financial context.

The final element of the research methodology was the analysis of case studies of modernisation of financial systems that have used APIs. These examples collected from financial institutions, fintech startups, and technology providers proved how API-driven architectures mirror issues that arise with traditional architectures.

For example, a case of a renowned bank discussed how it employed microservices and API gateways to break a monolithic old structure into the microservices structure. This transition did not only enhance the performance of the system, but also allowed it to launch new customer facing services faster.

Another real example focused on how a fintech firm has utilized GraphQL APIs for assembling real-time financial data from several outdated sources to share with its clients. Such examples were examined in order to make a review of common patterns, difficulties and results with the aim at developing recommendations for future applications.

Security and compliance were, therefore, particularly focus points in the methodology since the financial industry has a high level of regulatory compliance standards. Specifically, the research targeted the way API architectures provide protection of data and compliance with specific standards including PCI DSS, GDPR, and ISO 27001.

Exploratory content analysis was conducted on materials provided by the Open Web Application Security Project (OWASP) and the Financial Services Information Sharing and Analysis Center (FS-ISAC) to inform security features like authentication, encryption, and API tested for vulnerability.

Some of the more targeted API security practices that were studied include OAuth 2.0 which is used for Authentication, JWT (Json Web Tokens) for Token Authentication and API Throttling or rate limiting. Similarly, the methodology covered a brief evaluation of the API management tools and platforms used in most industries.

Application programming interfaces or APIs management tools such as Apigee, Postman, AWS API Gateway were reviewed to discover their correlation for end to end API management. These tools offer features like API specification, test, deployment, and monitoring as well as serving analytics that are central to the success of APIs for an organization.

Rate limiting, caching and monitoring were compared regarding the operational efficiency of the evaluated financial systems. Similar information was gathered from developer communities and forums where successes and failures as well as practical issues and possible ways to use API in legacy systems are being considered.

For the purpose of offering both the advantages as well as the flip side of API-driven architectures, the research approach also looked into the problems of implementing the architecture. Some of the key questions include the high first-time cost of modernization, API obsolescence problems in integration with existing, heavily-burdened legacy ones, and staff re-education.

Recommendations on the mitigation of these challenges were made by synthesising literature from industry gurus and professionals. For example, phased implementation approaches in which APIs are introduced gradually along with traditional systems were examined as a viable approach to avoid disruption.

Finally, to ensure that the diverse findings of the research were also forward-looking in terms of identifying the lessons, trends, or technologies that have not yet been fully understood in connection with API modernization, the research employed forward-looking research questions. Some concepts like event-driven architectures, serverless approach, and AI-enhanced API management were discussed to analyse their potential influence on the further modernization of the financial systems.

These trends were also looked at with the current level of API utilization in order to give a perspective of how legacy system modernization has evolved. (Note that by using the method of secondary research only in this work, one can create a diverse and informed approach to API-driven architectures for the modernization of legacy financial systems.)

The examination of literature review as well as of technical as well as practical case studies and best practices sources offers a significant background for conclusion of effective recommendations for the financial institutions while starting their modernization processes. In this methodology it is demonstrated how far APIs can go and also the practical questions and factors that need to know so the success intended is achieved.

3. Results and discussion

3.1. API Architectures

An evaluation of the architecture flowcharts used in APIs helps in understanding the operation of various types of APIs and their support structures in financial modernization. These flowcharts act as the representation of the working or real time architectural representation of the APIs or systems that are driven by APIs.

For these diagrams, we can distill knowledge as to how these API architectures facilitate the interactions between old and new systems, being concretely scalable, secure, and performant. The emphasis is made on the definitions of API architectures, the work-flows they imply, and the concrete advantages of their application for the financial systems update.

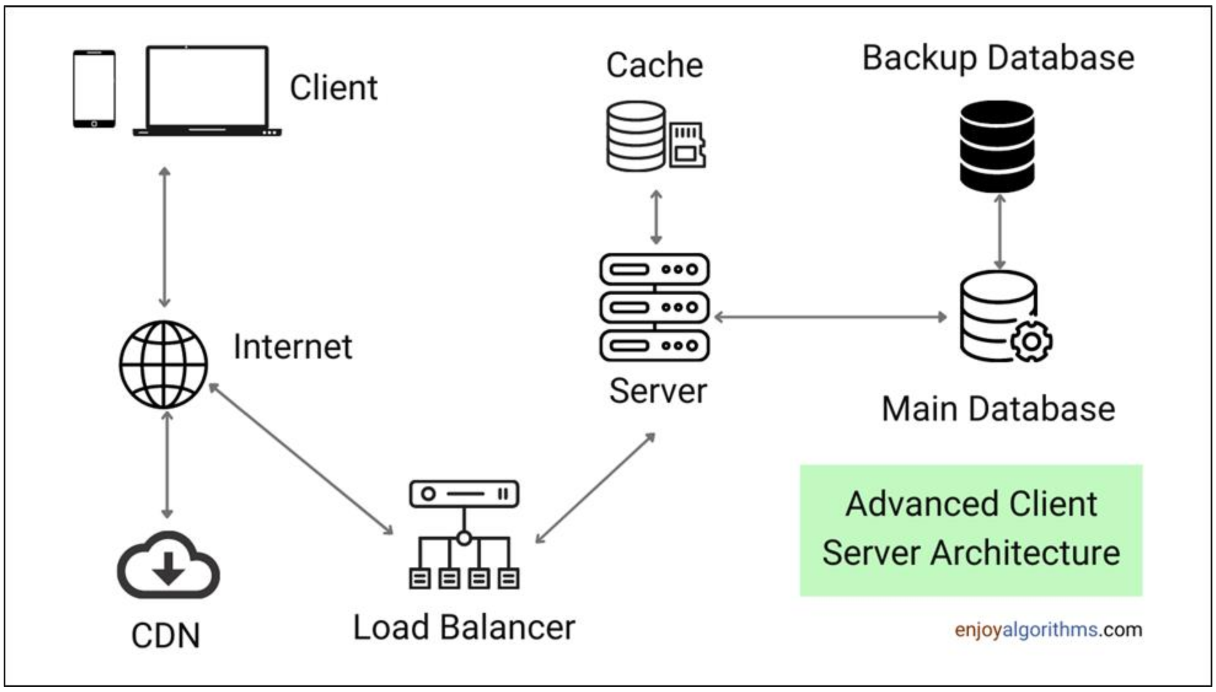


Figure 1 RESTful API Architecture

The most extensively implemented architecture in financial system modernization is the RESTful APIs because of its stateless nature, simplicity and ease of implementation. The flowchart on RESTful API architecture will usually start with client requests coming from the WEB or mobile application.

These requests are funnelled through an API gateway, this is a single-entry point into the architecture, which handles traffic management, access authentication, and the routing of requests. Finally, the gateway relays the approved requests to a certain part in the system, and microservice applications deal with the actual business processing.

These microservices also communicate with the application's legacy core systems through the adaptors or middleware that allow for easy data exchange while hiding the complexity of interactions. When breaking down this architecture the API gateway feature emerges as being important. This has the function of controlling authentication such as OAuth 2.0 and also regulating traffic to avoid congestion.

Also, the application of microservices improves the modularity; it means that financial institutions can always change or update various functionalities without affecting other sections. For instance, a microservice engaged in handling transactions could be changed or expanded independently from the customer authentication module. This flexibility makes RESTful APIs especially suitable for complex large-scale financial systems where new innovations and the efficient implementation of new features are continuously being developed.

GraphQL APIs offer a better solution in case of specific data fetch scenarios where such a pattern is required. In contrast to REST API that uses multiple URLs for various disparate data structures, GraphQL works with a single URL but enables clients to be particular about the data they want.

Starting point of flowchart of GraphQL API architecture is a query interface through which clients make their request. These queries are compared to a dictionary with a priori defined structure that explains the kinds of data that are contained therein and the connections between them. Resolver functions work as an interface between the application's interface or controller and data sources such as inner database, external database, other applications, or third-party applications.

In analysing this architecture, it becomes clear that the ability to maximize the flow of data is one of its key advantages. The capability of clients to request only the necessary data they need makes GraphQL effective in decreasing overall bandwidth consumption as well as avoiding instances of both over and under ingestion of data. This is especially helpful to financial systems that handle vast amounts of data information like customers' profile, transactions' history, and accounts details.

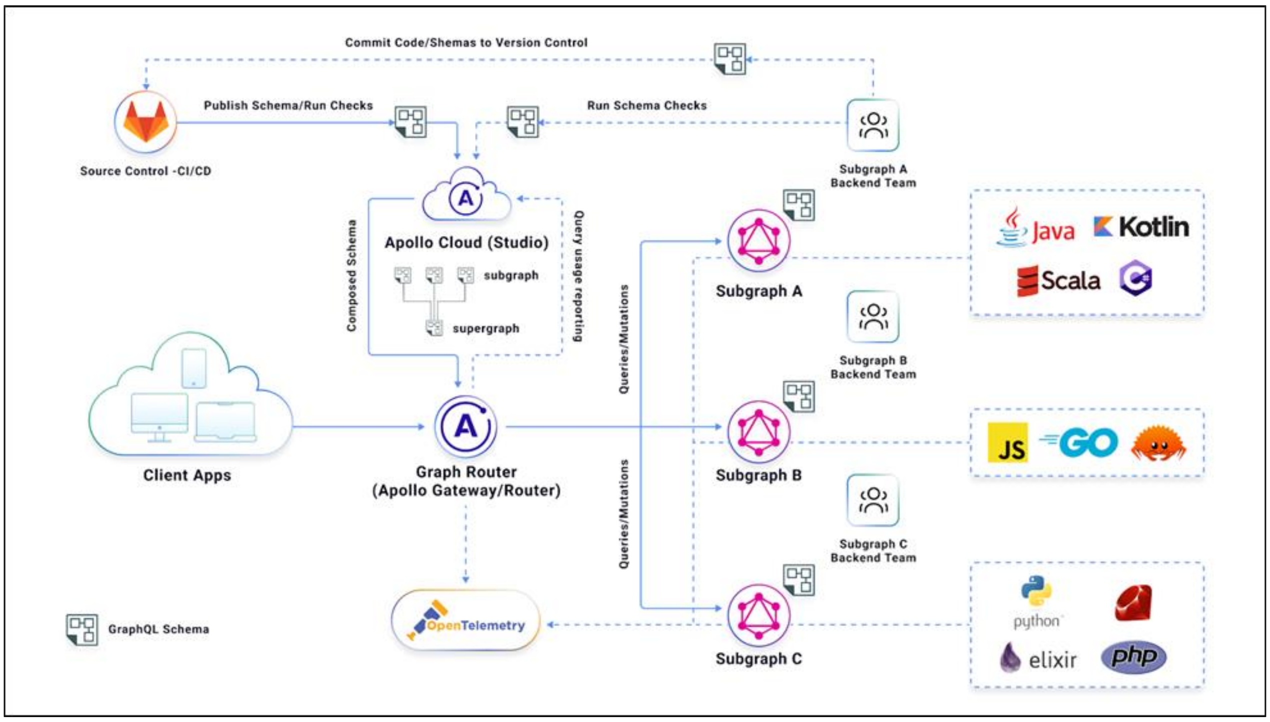


Figure 2 GraphQL API Architecture

For example, in implementing GraphQL, a financial application can pull data from multiple legacy databases and present a consolidated response to the client, making efficiency and usage more effective. The schema-first approach also helps to add clarity into the overall API design which in turn will help the developers to implement.

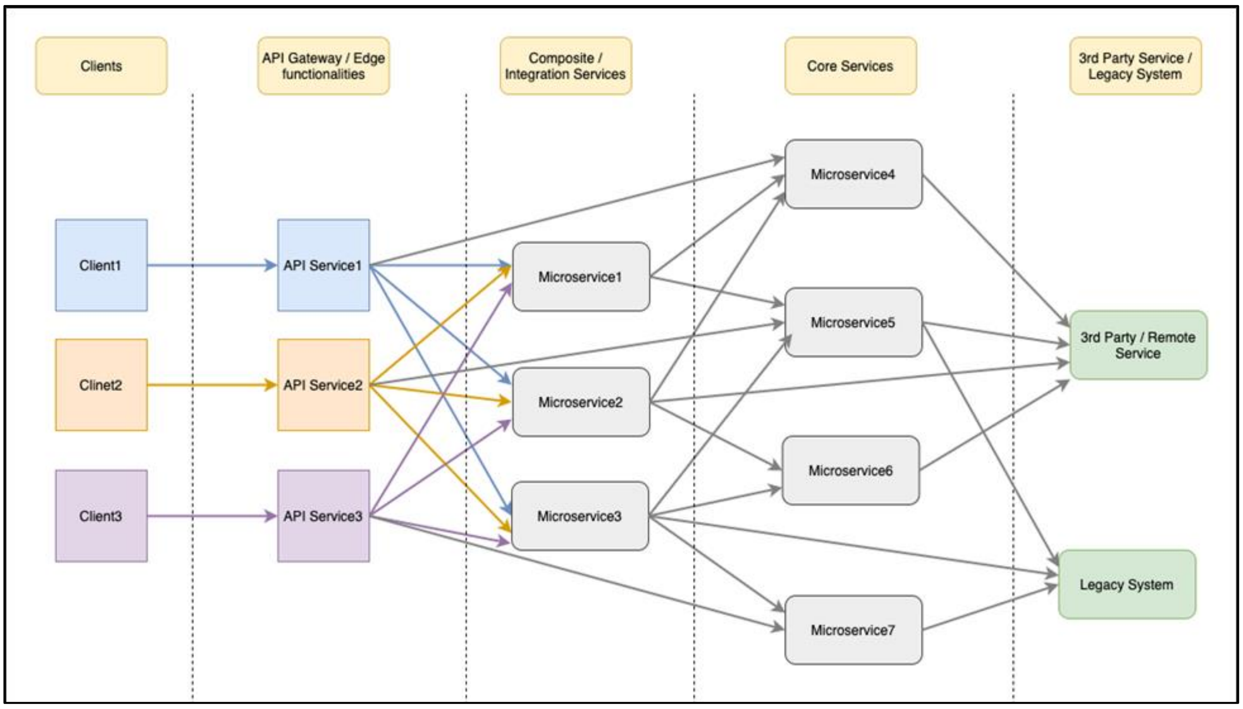


Figure 3 API Gateway with Microservices

The use of an API in conjunction with microservices represents a combination of the approach - where, the various app modules are designed according to their individual needs, while their control is centralized. In this architecture, the flowchart often shows that the API gateway is the initial point of client's requests.

This means that the gateway forwards these requests to the correct microservices depending on encoded rules. Many microservices are tightly focused on some functionality, like authentication, processing of transactions or generation of reports. These microservices query the underlying databases and interact with modern APIs along with the heritage systems to fulfil the received requests.

The following architecture analysis explains why it is scalable and robust. When the load is divided into multiple microservices, it becomes possible for financial systems to process large transactions per second without huge system lags. Further, the separation of concerns facilitates the development and deployment of microservices ultimately allowing financial institutions to adapt to agile processes.

For instance, the new regulation expected to change the reporting service means that modifications can be done and deployed without compromising other services such as transactions. The API gateway goes a step further in enhancing this structure by concentrating the functions of security, caching, and monitoring, and generating high uniformity across the system.

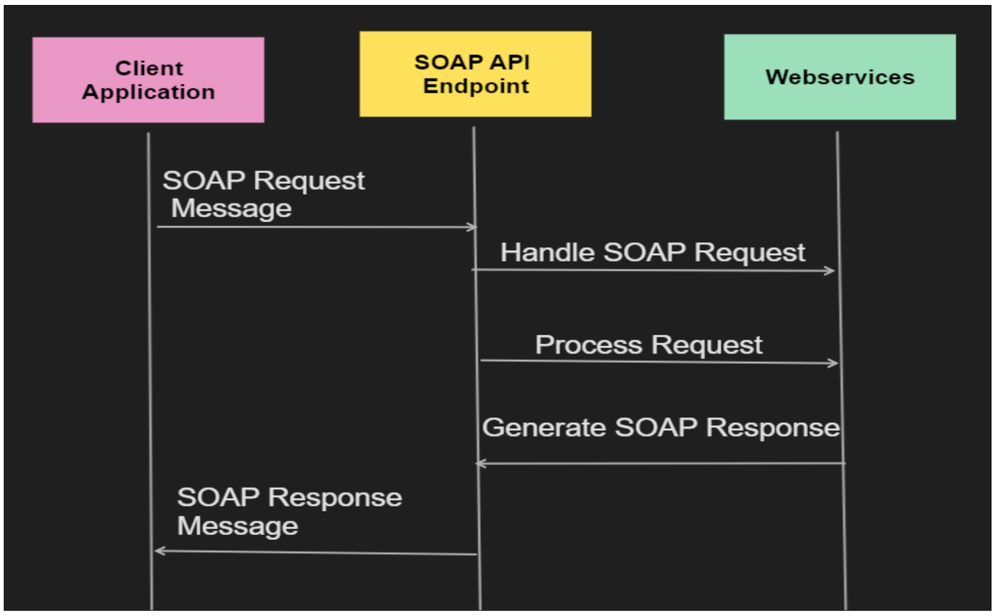


Figure 4 SOAP API Architecture

While the use of SOAP APIs has become somewhat dated in the modern world, they are still valuable for applications that demand highly secure financial systems. In most cases of a flowchart for a SOAP API architecture, client applications are assumed to develop requests in the SOAP message format.

These messages are checked against a Web Services Description Language (WSDL), which is a file containing descriptions of the procedures and data types. The messages are often accompanied with security headers to guarantee their content's credibility as well as its confidentiality. It is then processed to the server side where an application interacts with the old systems to either pull or push data.

SOAP API design carries the next potential benefits when utilized in an environment where compliance with the standards may be completely crucial. For instance, the financial systems that require implementing the PCI DSS requirements for the payment data security will appreciate the integration of SOAP, with the XML encryption and signature.

While SOAP is more resource-consuming than REST, or GraphQL APIs, the reliability and robustness of the SOAP protocol make the technology appropriate for important and secure operations such as interbank deals and messages.

Based on the comparison of these API architectures, the features of their applicability and advantages in various modernisation processes can be identified. RESTful APIs do very well in terms of simplicity and extensibility, which is also very good if we are going to be dealing directly with the end-user customer-facing app that has to be high-performance.

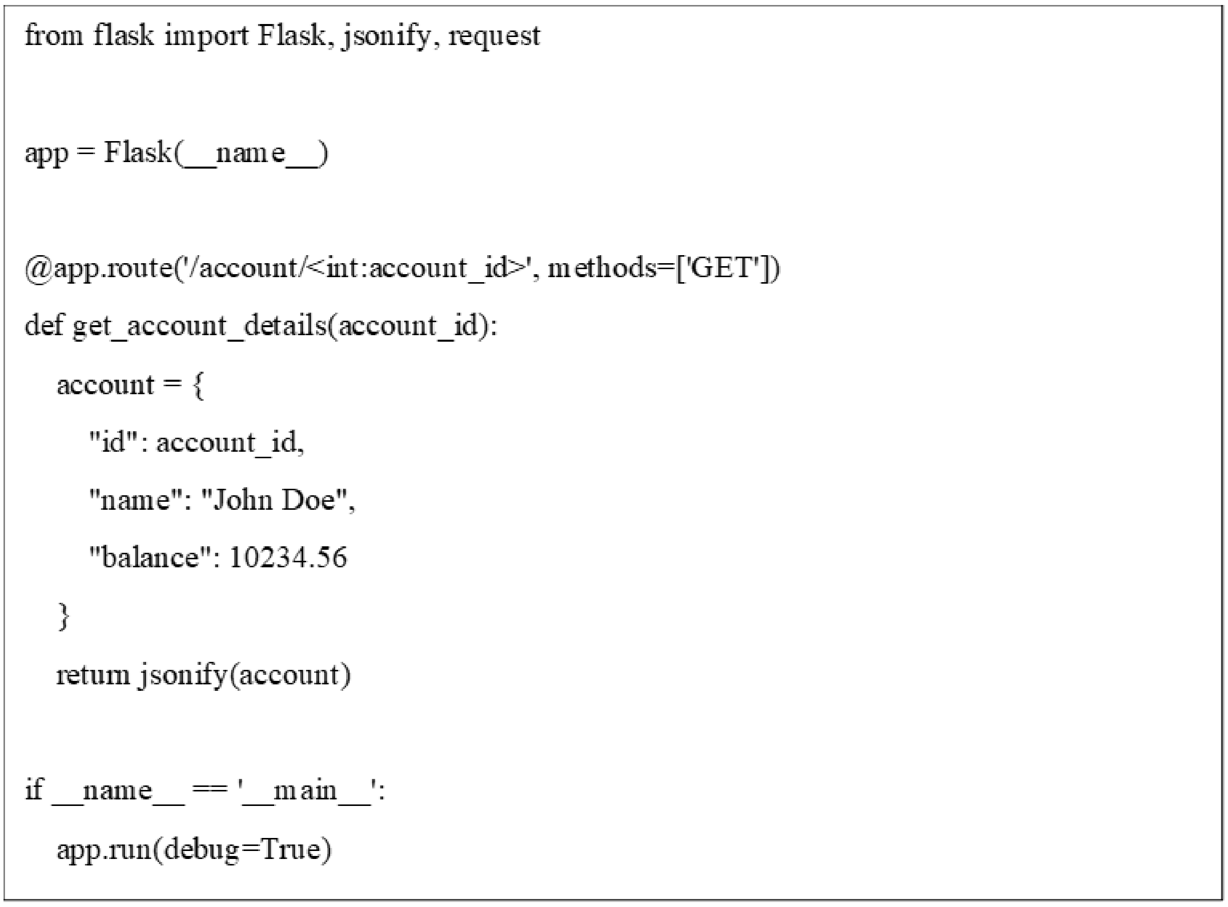
GraphQL APIs solve the problems related to data aggregation and make a conception to vary the way it can reuse the data itself, which is suitable for handling a question with various potential and routes. The integration of API gateways and microservices also offer the architectural style of modularity and resilience that can meet the changing demands of financial systems.

SOAP APIs on the other hand remain relevant especially in organizations where security and compliance matters are of major concern in specific areas where data is being used. I believe that these flowcharts and their analyses offer a map for use by the financial institutions that are considering modernization efforts.

Exploring the logical, physical, and technological aspects of each architecture provides transparent insights on the collections, compounding structures, and interrelations that are vital in decision making when choosing the right approach to fit a certain need. Incorporation of such diagrams and technical findings makes this research not only discuss theoretical frameworks but also offer specific recommendations for application.

3.2. Code implementation

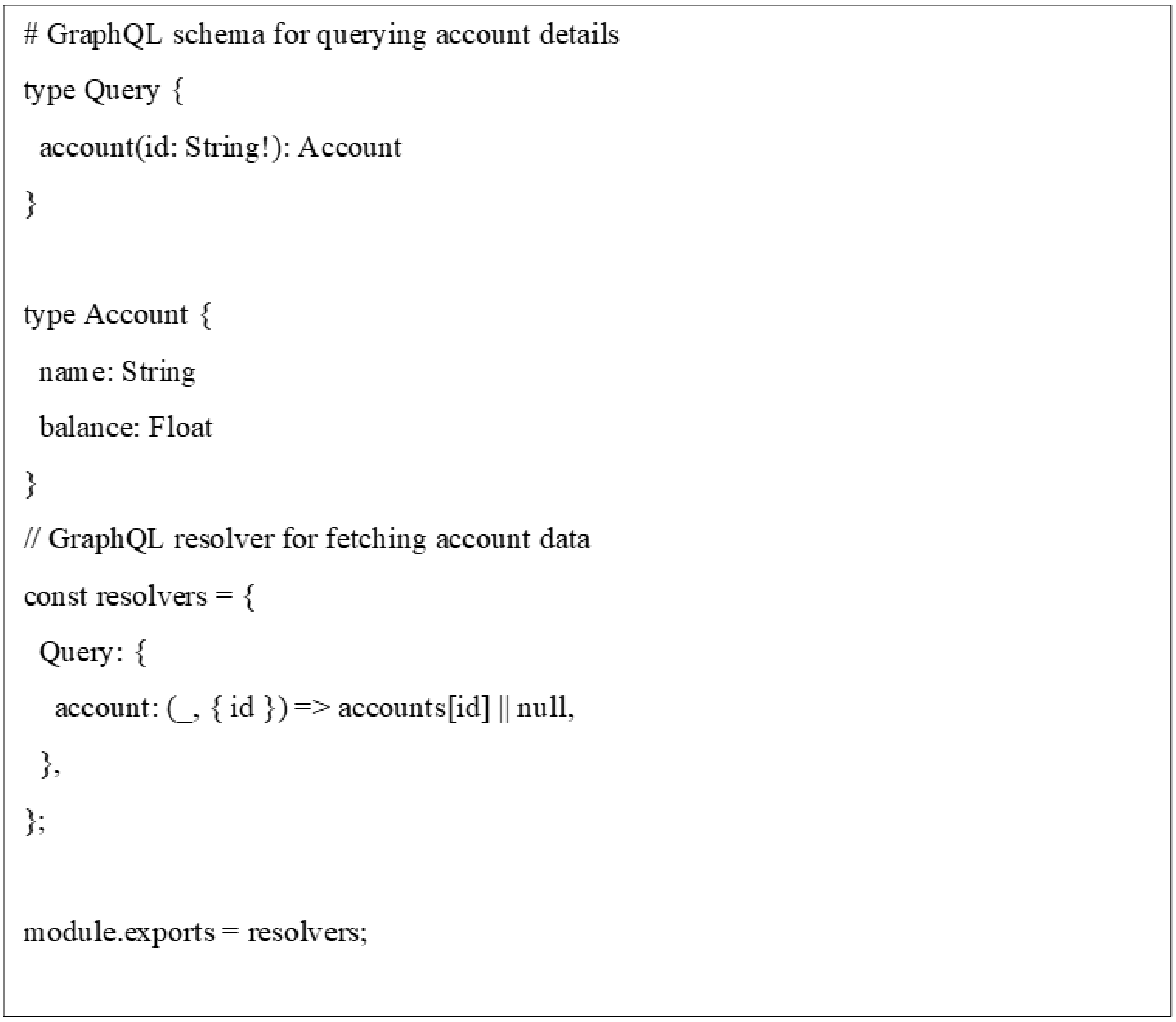
An example in a RESTful API setup is where a user requests to retrieve information regarding their account on a financial application. One example of a basic Python Flask implemented



```
from flask import Flask, jsonify, request


app = Flask(__name__)


@app.route('/account/<int:account_id>', methods=['GET'])
def get_account_details(account_id):
    account = {
        "id": account_id,
        "name": "John Doe",
        "balance": 10234.56
    }
    return jsonify(account)


if __name__ == '__main__':
    app.run(debug=True)
```

As the following code shows, RESTful APIs are about using clear and understandable endpoints. The client makes a GET request to the server with an account ID as a parameter, the server returns the relevant account info in JSON format. This simplicity and clarity make RESTful APIs a preferred choice of tool when it comes to simple data pulling.

On the other hand we have GraphQL APIs by which the data can be queried in a fine-grained manner decreasing the problem of over-querying and under-querying of the data. This is a simple GraphQL schema, consisting of two Query types for getting user and account data.



```
# GraphQL schema for querying account details
type Query {
  account(id: String!): Account
}


type Account {
  name: String
  balance: Float
}
// GraphQL resolver for fetching account data
const resolvers = {
  Query: {
    account: (_, { id }) => accounts[id] || null,
  },
};


module.exports = resolvers;
```

Accounts is a Query type and an Account type and it defines the structure of the data that we want to return to the clients. The resolver uses the query to get the account data of interest based on the ID of the account, which makes the data fetching very dynamic.

For systems using an API gateway and microservices, further example on Node.js depicts how the API gateway forwards customer requests to respective microservices.

```
const express = require('express');
const app = express();

app.get('/transaction', (req, res) => {
  res.redirect('http://localhost:5001/transaction');
});

app.get('/auth', (req, res) => {
  res.redirect('http://localhost:5002/auth');
});

app.listen(5000, () => {
  console.log('API Gateway running on port 5000');
});
```

```
from zeep import Client

wsdl = 'http://example.com/service?wsdl'
client = Client(wsdl=wsdl)

response = client.service.GetAccountDetails(accountID=123)
print(response)
```

This API gateway directs /transaction requests to one microservice while directing /auth requests to other microservice. It demonstrates API gateways as a central point that helps to address the problem of client communication with numerous services.

Last but not the least, SOAP APIs that are not often employed today are very secure. A Python sample of how the SOAP request and response look using the zeep library is as follows:

In this snippet, a client initiates a SOAP message to get account details and communicates with the web services given by WSDL. SOAP because of its rigid structure provides safe operations and dependable communication and it is perfect for regulated networks.

4. Conclusion

API based modernization of traditional financial systems is one of the revolutionary steps in the industry. RESTful API, GraphQL, and SOAP API integration help financial institutions work closely together with one or multiple APIs while increasing scalability and improving users' experiences while addressing security and compliance issues.

A look at the architectural workflows and code snippets to understand the situation makes it easy to appreciate the fact that APIs are very flexible and effective means by which one can integrate traditional systems into modern applications. From this research one learns the importance of APIs in creating and sustaining innovations that would keep the financial systems relevant in a fast-growing digital environment. Hence the need for API driven modernization as a core approach to the future of financial technology.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Zachariadis, M., & Ozcan, P. (2017). The API economy and digital transformation in financial services: The case of open banking. https://dx.doi.org/10.2139/ssrn.2975199

[2] Ünsal, E., Öztekin, B., Çavuş, M., & Özdemir, S. (2020, October). Building a fintech ecosystem: Design and development of a fintech API gateway. In 2020 international symposium on networks, computers and communications (ISNCC) (pp. 1-5). IEEE. https://doi.org/10.1109/ISNCC49221.2020.9297273

[3] Fikri, N., Rida, M., Abghour, N., Moussaid, K., & El Omri, A. (2019). An adaptive and real-time based architecture for financial data integration. Journal of Big Data, 6, 1-25. https://doi.org/10.1186/s40537-019-0260-x

[4] Premchand, A., & Choudhry, A. (2018, February). Open banking & APIs for transformation in banking. In 2018 international conference on communication, computing and internet of things (IC3IoT) (pp. 25-29). IEEE. https://doi.org/10.1109/IC3IoT.2018.8668107

[5] Kellezi, D., Boegelund, C., & Meng, W. (2019). Towards secure open banking architecture: an evaluation with OWASP. In Network and System Security: 13th International Conference, NSS 2019, Sapporo, Japan, December 15–18, 2019, Proceedings 13 (pp. 185-198). Springer International Publishing. https://doi.org/10.1007/978-3-030-36938-5_11

[6] Weir, L. (2019). Enterprise API Management: Design and deliver valuable business APIs. Packt Publishing Ltd. https://books.google.co.in/books?hl=en&lr=&id=0OikDwAAQBAJ&oi=fnd&pg=PP1&dq=financial+system+with+API+architectures&ots=XYVrJzb7ad&sig=lRYlbM9zfy85cQQDmhuKs2aFEcU&redir_esc=y#v=onepage&q=financial%20system%20with%20API%20architectures&f=false

[7] Luo, G., Li, W., & Peng, Y. (2020). Overview of intelligent online banking system based on HERCULES architecture. IEEE Access, 8, 107685-107699. https://doi.org/10.1109/ACCESS.2020.2997079

[8] Remolina, N. (2019). Open banking: Regulatory challenges for a new form of financial intermediation in a data-driven world. https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?article=1006&context=caidg

[9] Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A., Larsen, S. T., & Dustdar, S. (2018). Microservices: Migration of a mission critical system. IEEE Transactions on Services Computing, 14(5), 1464-1477. https://doi.org/10.1109/TSC.2018.2889087

[10] Elghaish, F., Abrishami, S., & Hosseini, M. R. (2020). Integrated project delivery with blockchain: An automated financial system. Automation in construction, 114, 103182. https://doi.org/10.1016/j.autcon.2020.103182

[11] Jin, H., Dai, X., & Xiao, J. (2018, July). Towards a novel architecture for enabling interoperability amongst multiple blockchains. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS) (pp. 1203-1211). IEEE. https://doi.org/10.1109/ICDCS.2018.00120

[12] Lytvyn, V., Kuchkovskiy, V., Vysotska, V., Markiv, O., & Pabyrivskyy, V. (2018, September). Architecture of system for content integration and formation based on cryptographic consumer needs. In 2018 IEEE 13th International

Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 391-395). IEEE. https://doi.org/10.1109/STC-CSIT.2018.8526669

[13] Abrishami, S., & Elghaish, F. (2019). Revolutionising AEC financial system within project delivery stages: A permissioned blockchain digitalised framework. 36th CIB W, 78, 2019. https://www.researchgate.net/profile/Faris-Elghaish/publication/335892493_Revolutionising_AEC_financial_system_within_project_delivery_stages_A_permissioned_blockchain_digitalised_framework/links/5d8765a2299bf1996f926493/Revolutionising-AEC-financial-system-within-project-delivery-stages-A-permissioned-blockchain-digitalised-framework.pdf

[14] Knewtson, H. S., & Rosenbaum, Z. A. (2020). Toward understanding FinTech and its industry. Managerial Finance, 46(8), 1043-1060. https://doi.org/10.1108/MF-01-2020-0024

[15] Perera, K. J. P. G., & Perera, I. (2018, October). A rule-based system for automated generation of serverless-microservices architecture. In 2018 IEEE International Systems Engineering Symposium (ISSE) (pp. 1-8). IEEE. https://doi.org/10.1109/SysEng.2018.8544423

[16] D'Silva, D., Filková, Z., Packer, F., & Tiwari, S. (2019). The design of digital financial infrastructure: lessons from India. BIS Paper, (106). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3505373