API-First Big Data Pipelines: Designing Scalable REST Integration Layers for Saudi E-Government Services.

Abstract

Saudi e-government services now reach most residents through digital channels, and each transaction both depends on data held by other agencies and produces data that could inform policy. Integration is still often built case by case, with point-to-point interfaces and batch exports that are hard to secure, evolve and scale. This paper designs an API-first integration layer in which every exchange between agencies is defined first as a versioned REST contract, and in which the events produced by those exchanges feed governed big data pipelines. A design science approach is used, drawing on an integrative synthesis of 30 refereed studies (2020–2025) covering API design and specification, API evolution and governance, REST and GraphQL performance, API security, microservice scaling, big data and stream processing, and digital government in Saudi Arabia and elsewhere. Ten requirements are derived. The resulting architecture combines a contract registry and design rules, a gateway that enforces identity, authorisation and quotas, stateless microservices with dependency-aware autoscaling, an event backbone, stream and batch processing with governed storage, and an analytics and open data tier, all under a common governance plane. An analytical evaluation shows strong evidence for design rules, machine-readable contracts and the scalability of stream processing, mixed evidence for automated security testing, autoscaling and protocol choice, and limited evidence on cross-agency contract governance and on Saudi implementation. No system was built, measured or deployed for this study and no government data were used; the architecture is offered as a basis for empirical testing.

UNDER PEER REVIEW IN IJAP

1. Introduction

Saudi Arabia has moved a large share of public services online. The Absher platform alone had 23 million authenticated users when it was used to run a national health survey (Alluhidan et al., 2022), and a repeated assessment of government websites found remarkable improvement in the maturity of online services compared with surveys from 2006 to 2012 (Alzahrani, 2022). Behind

each digital service sits an integration problem. Renewing a licence or registering a business may require data from several agencies, and every completed transaction creates records that planners and analysts could use.

Two design traditions address these needs separately. Web APIs, most often built in the REST style, expose application functions in a uniform way, and their quality depends on design rules, specifications and security controls (Bogner et al., 2023; Santos Filho et al., 2025). Big data pipelines move and transform large volumes of data for analysis, and their quality depends on the choice and tuning of processing frameworks (Ahmed et al., 2020; Henning & Hasselbring, 2024). In government, the two are frequently built by different teams: interfaces for transactions on one side, exports and warehouses for reporting on the other. The result is duplicated logic, inconsistent data definitions and analytical data that lag behind operations.

An API-first approach offers a way to join them. In this approach, the contract of each interface is designed, reviewed and published before any code is written, and the same contracts define the events that flow into analytical pipelines. Research on digital government has called for stronger empirical evidence on API technologies in government settings (Quiña-Mera et al., 2023) and for design principles that build transparency into government systems (Matheus et al., 2021), but there is no integrated architecture that links contract-first REST design with scalable pipelines for Saudi e-government. This paper proposes one and evaluates it against published evidence.

2. Research Aim and Questions

The aim is to design an API-first integration layer that connects Saudi e-government services through scalable REST interfaces and feeds big data pipelines from the same contracts, and to evaluate the design analytically. The objectives are (1) to identify integration requirements arising from the Saudi e-government setting and from the literature; (2) to synthesise evidence on API design, evolution, security, scaling and data processing; (3) to design an architecture that meets the requirements; and (4) to assess the strength of evidence behind each design decision and to identify residual risks.

Three questions guide the study. RQ1: What requirements should an integration layer for Saudi e-government services satisfy? RQ2: What does peer-reviewed evidence indicate about mechanisms for designing, securing, scaling and analysing API-based exchanges? RQ3: How

UNDER PEER REVIEW IN IJAP

can these mechanisms form a coherent API-first architecture, and how well is each element supported?

3. Method: Design Science with Evidence-Based Evaluation

The study follows the design science sequence of problem identification, objective setting, design and evaluation. Because the artefact is an architecture and no implementation was available, evaluation is analytical: each design choice is assessed against the published evidence for the mechanism it uses.

The evidence base was assembled through an integrative review. In November 2026 an open index of scholarly publications was queried, and the bibliographic details of every candidate were confirmed against registration metadata. Searches combined interface terms (REST API, OpenAPI, GraphQL, API evolution, API security, OAuth), platform terms (microservices, Kubernetes, autoscaling), data terms (big data pipeline, Hadoop, Spark, stream processing) and government terms (e-government, digital government, interoperability, open government data, Saudi Arabia). Eligible items were English-language journal articles subject to peer review, published in a final issue from 2020 to 2025, with an abstract reporting an experiment, a tool, a case, a model or a systematic review relevant to API-based integration or large-scale data processing. Conference papers, theses, items without abstracts and studies limited to user interfaces were not included.

Two anchor studies framed the review: the performance comparison of GraphQL and REST services in a live information system by Lawi et al. (2021) and the controlled experiment on RESTful design rules and understandability by Bogner et al. (2023). The 30 included articles comprise four on API design quality and specification, three on evolution and governance, three on protocol and runtime performance, two on security, three on microservice scaling, five on big data and stream processing, and ten on digital government, six of them on Saudi Arabia.

Evidence for each criterion was graded as supportive where several studies demonstrate a mechanism with measured or validated effects, as mixed where mechanisms exist, but results depend strongly on context or carry significant trade-offs, and as limited where the corpus offers little direct evidence. Each attribution to a source was compared with that source's reported findings. No prototype was built, no load test was run, no government system was accessed, and

UNDER PEER REVIEW IN IJAR

no data were generated, so the evaluation reflects the literature rather than observed performance.

4. Saudi E-Government Context and Requirements

Saudi e-government has progressed from information websites to transactional and increasingly data-driven services. Alzahrani (2022) found marked improvement on a five-stage maturity model, and Zygiaris and Maamari (2023) identified technology adequacy, organisational change, open government and socially inclusive government as the factors enabling the move from e-government to digital government. Hashim (2024) linked e-government to smart city initiatives that rely on the Internet of Things, big data and cloud computing, while noting citizens' concerns about the security of such environments. Quality at the front end remains uneven: an assessment of Saudi mobile government websites found usability and accessibility problems and non-compliance with international web standards (Al-Sakran & Alsudairi, 2021).

Government data can serve purposes beyond the original transaction. A web survey administered through Absher between 2019 and 2020 reached a 24.6% response rate, and the authors concluded that e-government platforms could, with better study design, supply inexpensive data for health research (Alluhidan et al., 2022). Such reuse depends on interoperable, well-described data.

4.1 Lessons from Digital Government Elsewhere

International studies show how integration and data reuse can be designed. A science mapping of digital government research identified five main areas, including transformation, cybersecurity, participation, open data and transparency, and adoption models (Güler & Büyüközkan, 2023). Matheus et al. (2021) defined 16 design principles for digital transparency, tested them in three cases and concluded that technological and institutional foundations must precede the deployment of tools. An ontology-based architecture for open government data covered the whole value chain and was demonstrated with building energy data from Tehran (Masoumi et al., 2022). Multi-agent systems combined with deep reinforcement learning have been proposed to coordinate departments with incompatible formats and protocols, reporting a task completion rate of 95% and a communication latency of 120 milliseconds in a municipal setting (Mequanenit et al., 2025). Shared platforms between agencies can also reduce resource

UNDER PEER REVIEW IN IJAR

constraints, as a design science study of an Ethiopian government network showed (Gebeyehu & Twinomurinzi, 2022).

4.2 Requirements

From this context and the technical literature, ten requirements were derived (Table 1). They concern the quality and machine-readability of contracts, their evolution, security, commercial and usage terms, performance and scalability, resilience, data processing and governance. Together they answer RQ1.

Table 1. Requirements for an API-first integration layer for Saudi e-government services

No.	Requirement	Why it matters	Main sources
Q1	Contracts follow shared design and naming rules	Comprehension and reuse depend on rule adherence	Bogner et al. (2023); Palma et al. (2022)
Q2	Contracts are machine-readable and analysable	Dependencies and semantics enable automation	Martin-Lopez et al. (2022); Muhamad et al. (2022)
Q3	Versions evolve without breaking consumers	Change is the main source of integration failure	Lercher et al. (2024)
Q4	Identity, authorisation and object access are enforced	Public interfaces are frequent targets	Santos Filho et al. (2025); Munonye and Péter (2022)
Q5	Usage limits and service levels are explicit	Agencies share capacity and obligations	Fresno-Aranda et al. (2025)
Q6	Protocols suit each workload	Response time and resource use trade off	Lawi et al. (2021); Shyam Mohan and Goswami (2025)
Q7	Services scale elastically and cost-effectively	Demand peaks at deadlines and campaigns	Blinowski et al. (2022); Santos et al. (2025)
Q8	Failures and ageing are detected and contained	Orchestration alone misses some faults	Flora et al. (2022)
Q9	Events feed stream and batch pipelines at scale	Analytics must keep pace with operations	Henning and Hasselbring (2024); Ahmed et al. (2020)
Q10	Governance covers access, retention, transparency and open data	Public trust and reuse require it	Matheus et al. (2021); Masoumi et al. (2022)

5. Evidence on API-First Mechanisms

The literature is reviewed under six headings corresponding to the layers of the eventual design.

5.1 Design Rules and Machine-Readable Contracts

Design rules matter for comprehension. In a controlled experiment with 105 participants, Bogner et al. (2023) found that for 11 of 12 RESTful design rules, snippets violating the rule led to

UNDER PEER REVIEW IN IJAR

significantly worse comprehension, and that experience did not offset the effect. Naming is part of quality: analysing 19 REST APIs for Internet of Things applications, Palma et al. (2022) found linguistic antipatterns to be prevalent and validated a tool to detect them.

Contracts can carry more meaning than they usually do. Muhamad et al. (2022) transformed 106 OpenAPI documents into semantic web services described in RDF, enabling queries over operations and parameters, and reported that storing contract resources in a relational schema improved reuse efficiency by about 48%. Martin-Lopez et al. (2022) noted that OpenAPI cannot describe dependencies between input parameters and proposed a dedicated language with seven automated analysis operations, such as checking whether a request satisfies all dependencies.

5.2 Evolution, Governance and Usage Terms

Interfaces change, and change is where integration breaks. From 17 interviews in 11 companies, Lercher et al. (2024) identified six strategies and six challenges in the evolution of microservice APIs; strategies centred on backward compatibility, versioning and close collaboration, while challenges included the effort of impact analysis, poor communication of changes and consumers relying on outdated versions. APIs also govern behaviour. Tracing the history of one large platform's interfaces, van der Vlist et al. (2022) showed how a simple interface for data access evolved into a layered governance arrangement that shaped what developers could do. Usage terms can be formalised: Fresno-Aranda et al. (2025) proposed a model of API pricing plans and limits that extends OpenAPI, assessed its expressiveness against 268 real APIs and validated it with 54 pricing models. For government, the same structure can express quotas and service levels between agencies.

5.3 Protocol and Runtime Choices

Protocol choice involves trade-offs. Comparing REST and GraphQL in a live management information system with intensive queries, Lawi et al. (2021) found REST up to 50.50% faster in response time and 37.16% higher in throughput, while GraphQL used 37.26% less CPU and 39.74% less memory; they recommended GraphQL where data requirements change often. A mapping of 84 GraphQL studies found growing adoption but called for more rigorous empirical evidence in industry and government (Quiña-Mera et al., 2023). The implementation platform matters too. Shyam Mohan and Goswami (2025) found Node.js faster for read requests and

UNDER PEER REVIEW IN IJAP

Spring Boot better for create, update and delete operations under high load, with both failing with error rates of 10% to 75% at 64,000 virtual users.

5.4 Security

Public interfaces are targets. Broken object-level authorisation, in which a caller reaches objects belonging to others, is a common weakness; Santos Filho et al. (2025) transformed OpenAPI contracts into Petri nets and detected such attacks from server logs with accuracy above 95% in their scenarios. Delegated authorisation is another weak point. Modelling the OAuth flow as a supervised learning problem, Munonye and Péter (2022) built seven classifiers that detected vulnerabilities with accuracy above 90%, although only 54% of results matched known vulnerabilities.

5.5 Scaling Services

Microservices promise horizontal scalability, but evidence is nuanced. Blinowski et al. (2022) implemented one application as a monolith and as microservices on two platforms and found that a monolith performed better on a single machine, that, in the cloud environment tested, adding resources to one instance was cheaper than adding instances, and that performance fell once the instance count passed a threshold. Orchestration does not remove all risk: fault injection experiments showed that Kubernetes probes detected various faults but neither detected nor acted on software ageing (Flora et al., 2022). Dependencies between services complicate autoscaling, and Santos et al. (2025) showed that reinforcement learning agents aware of such dependencies could achieve up to 50% lower latency than default scaling mechanisms.

5.6 Big Data and Stream Processing

Pipelines need disciplined design. Tardio et al. (2020) proposed an iterative methodology that derives pipeline components from big data requirements and tested it in two use cases. Khan et al. (2023) integrated storage services into pipelines and ranked storage options using five parameters, among them cost, network performance and server-side encryption. For batch processing, Ahmed et al. (2020) showed that Hadoop and Spark each have more than 150 parameters, that performance depends heavily on data size and configuration, and that Spark achieved up to 14 times speed-up on one workload after tuning. For streams, Henning and Hasselbring (2024) ran over 740 hours of experiments with up to 110 microservice instances

UNDER PEER REVIEW IN IJAR

handling as many as a million messages each second; all five frameworks scaled roughly linearly given enough resources, but none was superior in every case, and the choice strongly affected operating cost. Fragkoulis et al. (2024) traced two generations of stream processing systems and their approaches to state, fault tolerance, elasticity and out-of-order data. To answer RQ2, the evidence supports rule-based contract design, machine-readable and analysable contracts, explicit versioning, contract-derived security testing, measured scaling decisions and pipeline frameworks selected and tuned for the workload.

6. Proposed API-First Architecture

The design rests on one principle: every exchange between agencies, and every data flow derived from it, is defined first as a reviewed, versioned contract. Contracts are the single source from which interface code, tests, security checks, quotas, events and analytical schemas are generated or validated.

6.1 Design-Time Layer

A contract registry holds OpenAPI documents for all agency interfaces, extended with inter-parameter dependencies (Martin-Lopez et al., 2022), usage plans and service levels (Fresno-Aranda et al., 2025) and semantic annotations that link parameters to shared government vocabularies (Muhamad et al., 2022). Automated linting checks each contract against design and naming rules before review, reflecting the comprehension effects reported by Bogner et al. (2023) and the antipatterns found by Palma et al. (2022). Each contract declares its version, its compatibility with earlier versions and a deprecation date, and the registry notifies every consuming agency of planned changes, addressing the evolution challenges reported by Lercher et al. (2024).

6.2 Run-Time Layers

Figure 1 presents the run-time architecture. Channels such as portals, mobile applications and agency systems call a national API gateway, which authenticates callers through a shared identity service, enforces OAuth scopes and object-level authorisation, applies quotas from the contract and records every call. Security tests are generated from contracts and applied before and after release, drawing on the detection approaches of Santos Filho et al. (2025) and Munonye and Péter (2022).

UNDER PEER REVIEW IN IJAR

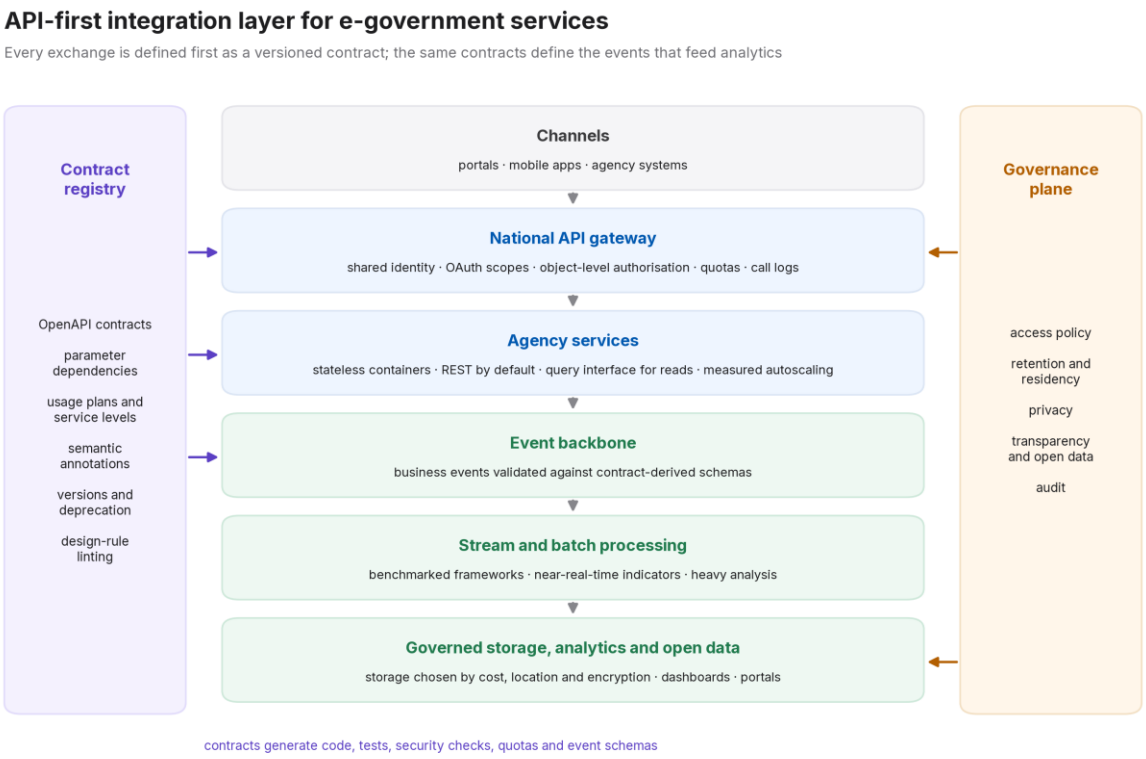


Figure 1. Run-time architecture of the API-first integration layer for Saudi e-government services.

Behind the gateway, agency services are stateless and packaged as containers. REST is the default for transactions because of its response time and throughput (Lawi et al., 2021), while a query interface may be offered for read-heavy consolidated views where data needs vary. Scaling policy is set from measurements: vertical scaling first where it is cheaper, horizontal scaling within tested limits (Blinowski et al., 2022), dependency-aware autoscaling for chains of services (Santos et al., 2025) and health checks supplemented by ageing detection (Flora et al., 2022).

Each completed transaction publishes a business event whose schema is derived from the contract. An event backbone carries these events to stream processing for near-real-time indicators and to batch processing for heavy analysis, using frameworks chosen through benchmarks against the expected load (Ahmed et al., 2020; Henning & Hasselbring, 2024). Processed data land in governed storage selected by cost, location and encryption criteria (Khan et al., 2023) and are published to analytics, dashboards and open data portals, following the ontology-based approach of Masoumi et al. (2022). A governance plane spans all layers, holding

UNDER PEER RE

policies on access, retention, residency, privacy and transparency in line with the principles of Matheus et al. (2021).

6.3 From Contract to Insight

Figure 2 follows one service from design to analysis. A contract is drafted and checked against rules, reviewed by producing and consuming agencies, and published with its usage plan. Code stubs, tests and security checks are generated from it. At run time, calls pass the gateway and update the service's records, and each call emits an event that is validated against the contract's event schema before entering the pipelines. When a contract changes, the new version is published alongside the old one, consumers are notified and pipeline schemas are migrated before the old version is retired. To answer RQ3, the architecture unites design-time contract governance, a secured and measured run time, and contract-derived pipelines under one governance plane.

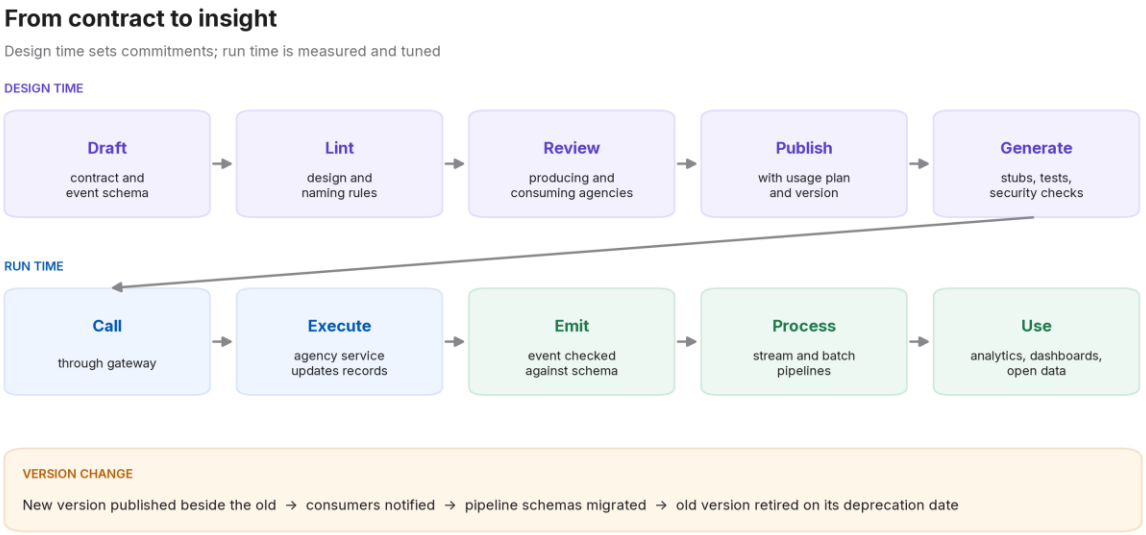


Figure 2. From contract to insight: the life of one API-first service, including version change.

7. Analytical Evaluation

Table 2 rates the evidence for the design response to each of the ten requirements. The ratings describe the literature, not a tested implementation.

Table 2. Analytical evaluation of the API-first architecture against the ten requirements

Criterion	Design response	Evidence rating	Residual risk
Q1 Design rules	Automated linting before review	Supportive	Rules must be agreed across agencies
Q2 Machine-readable contracts	OpenAPI with dependencies and semantics	Supportive	Tooling maturity varies
Q9 Pipelines at scale	Benchmarked stream and batch frameworks	Supportive	Cost depends on framework choice
Q4 Security	Gateway enforcement; contract-derived tests	Mixed	Field accuracy unproven
Q6 Protocols	REST default; query interface for reads	Mixed	Results depend on workload
Q7, Q8 Scaling and resilience	Measured scaling; dependency-aware autoscaling; ageing checks	Mixed	Instance limits and ageing
Q5 Usage terms	Usage plans and service levels in contracts	Mixed	No government use reported
Q3 Evolution	Versioning, deprecation dates, notifications	Mixed	Consumers lag behind versions
Q3, Q10 Cross-agency governance	Shared registry and governance plane	Limited	Not studied across agencies
Saudi implementation	Context-derived requirements	Limited	No Saudi integration studies

Three criteria are rated supportive. Design rules and naming quality have experimental and tool-based evidence (Bogner et al., 2023; Palma et al., 2022). Machine-readable, analysable contracts are supported by working transformations and analysis tools (Martin-Lopez et al., 2022; Muhamad et al., 2022). Stream and batch processing at scale are supported by extensive benchmarks and surveys (Ahmed et al., 2020; Fragkoulis et al., 2024; Henning & Hasselbring, 2024), although the choice of framework affects cost.

Five criteria are rated mixed. Contract-driven security testing achieves high detection accuracy in controlled scenarios but has limited field validation (Munonye & Péter, 2022; Santos Filho et al., 2025). Protocol choice depends on workload, with REST faster and GraphQL more frugal (Lawi et al., 2021). Elastic scaling is constrained by instance limits, costs and dependencies (Blinowski et al., 2022; Santos et al., 2025), and resilience is incomplete because orchestration does not detect ageing (Flora et al., 2022). Usage plans and quotas have a validated model but no

UNDER PEER REVIEW IN IJAR

government application (Fresno-Aranda et al., 2025). Evolution strategies are documented, yet the challenges of communication and outdated consumers persist (Lercher et al., 2024).

Two criteria are rated limited. No study in the corpus evaluates contract governance across many government agencies, the setting in which versioning and change communication are hardest. Nor does any study examine API-first integration or pipeline performance in Saudi e-government; the Saudi studies concern maturity, transition factors, usability and data reuse rather than integration technology.

8. Discussion, Implications and Limitations

The evidence suggests that the strongest gains from an API-first approach come at design time. Rules, contracts and automated checks have strong support and are relatively cheap, whereas the run-time elements, such as protocol, scaling and autoscaling, involve trade-offs that can be settled only by measurement. The design therefore treats contracts as fixed commitments and run-time choices as tunable.

Linking interfaces to pipelines through contracts addresses a common weakness of government analytics: data definitions that drift between operational and reporting systems. If every event is validated against a contract, analytical data inherit the same meaning and versioning as the transaction that produced them. The history of platform APIs also shows that interfaces are instruments of governance (van der Vlist et al., 2022). For government, contracts can encode not only data structures but also who may use which data, under which limits and with what transparency.

For Saudi agencies, the analysis points to three priorities: a shared contract registry with common design rules and vocabularies; a gateway with contract-derived security testing and quotas; and benchmarking of processing frameworks and scaling policies against real service loads before commitment. Researchers could evaluate contract governance across agencies, measure REST and alternative protocols on government workloads and study how event-driven analytics affects policy decisions.

The study has limitations. A single researcher selected the 30 journal articles, and conference papers, which carry a large part of API and systems research, were not considered. The corpus also contains no study of integration technology in Saudi government. Public information on

UNDER PEER REVIEW IN IJAP

national integration platforms was not analysed. The architecture has not been implemented, load-tested or reviewed by government architects, and its evaluation shows only how strongly the literature supports each choice.

9. Conclusion

Saudi e-government services need integration that is secure, evolvable and scalable, and analytics that keep pace with operations. This paper proposed an API-first architecture in which reviewed, versioned REST contracts govern every exchange between agencies and generate the tests, security checks, quotas and event schemas that feed big data pipelines. Published evidence strongly supports its design rules, machine-readable contracts and scalable processing, supports its security, protocol, scaling, quota and evolution choices with qualifications, and is thin on cross-agency governance and Saudi implementation. Prototyping the architecture with real services and loads, together with the agencies that would use it, is the necessary next step.

UNDER PEER REVIEW IN IJAR

References

1. Ahmed, N., Barczak, A. L. C., Susnjak, T., & Rashid, M. A. (2020). A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench. *Journal of Big Data*, *7*(1), Article 110. https://doi.org/10.1186/s40537-020-00388-5

2. Alluhidan, M., Alsukait, R. F., Alghaith, T., Saber, R., Alamri, A., Al-Muhsen, S., Alhowaitan, F., Alqarni, A., Herbst, C. H., Alazemi, N., & Hersi, A. S. (2022). Effectiveness of using e-government platform "absher" as a tool for noncommunicable diseases survey in Saudi Arabia 2019–2020: A cross-sectional study. *Frontiers in Public Health*, *10*, Article 875941. https://doi.org/10.3389/fpubh.2022.875941

3. Al-Sakran, H. O., & Alsudairi, M. A. (2021). Usability and accessibility assessment of Saudi Arabia mobile e-government websites. *IEEE Access*, *9*, 48254–48275. https://doi.org/10.1109/access.2021.3068917

4. Alzahrani, A. I. (2022). A periodical analysis of e-government maturity in Saudi Arabia. *Transforming Government: People, Process and Policy*, *16*(1), 18–31. https://doi.org/10.1108/tg-05-2021-0083

5. Blinowski, G., Ojdowska, A., & Przybylek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, *10*, 20357–20374. https://doi.org/10.1109/access.2022.3152803

6. Bogner, J., Kotstein, S., & Pfaff, T. (2023). Do RESTful API design rules have an impact on the understandability of web APIs? *Empirical Software Engineering*, *28*(6), Article 132. https://doi.org/10.1007/s10664-023-10367-y

7. Flora, J., Goncalves, P., Teixeira, M., & Antunes, N. (2022). A study on the aging and fault tolerance of microservices in Kubernetes. *IEEE Access*, *10*, 132786–132799. https://doi.org/10.1109/access.2022.3231191

8. Fragkoulis, M., Carbone, P., Kalavri, V., & Katsifodimos, A. (2024). A survey on the evolution of stream processing systems. *The VLDB Journal*, *33*(2), 507–541. https://doi.org/10.1007/s00778-023-00819-8

UNDER PEER REVIEW IN IJAR

9. Fresno-Aranda, R., Fernandez, P., Gamez-Diaz, A., Duran, A., & Ruiz-Cortes, A. (2025). Pricing4APIs: A rigorous model for RESTful API pricings. *Computer Standards & Interfaces, 91,* Article 103878. https://doi.org/10.1016/j.csi.2024.103878

10. Gebeyehu, S., & Twinomurinzi, H. (2022). A collaborative consumption digital platform for government organizations using design science. *Digital Government: Research and Practice*, *3*(1), 1–14. https://doi.org/10.1145/3510615

11. Güler, M., & Büyüközkan, G. (2023). A survey of digital government: Science mapping approach, application areas, and future directions. *Systems*, *11*(12), Article 563. https://doi.org/10.3390/systems11120563

12. Hashim, H. (2024). E-government impact on developing smart cities initiative in Saudi Arabia: Opportunities & challenges. *Alexandria Engineering Journal*, *96,* 124–131. https://doi.org/10.1016/j.aej.2024.04.008

13. Henning, S., & Hasselbring, W. (2024). Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud. *Journal of Systems and Software*, *208,* Article 111879. https://doi.org/10.1016/j.jss.2023.111879

14. Khan, A. Q., Nikolov, N., Matskin, M., Prodan, R., Roman, D., Sahin, B., Bussler, C., & Soylu, A. (2023). Smart data placement using storage-as-a-service model for big data pipelines. *Sensors*, *23*(2), Article 564. https://doi.org/10.3390/s23020564

15. Lawi, A., Panggabean, B. L. E., & Yoshida, T. (2021). Evaluating GraphQL and REST API services performance in a massive and intensive accessible information system. *Computers*, *10*(11), Article 138. https://doi.org/10.3390/computers10110138

16. Lercher, A., Glock, J., Macho, C., & Pinzger, M. (2024). Microservice API evolution in practice: A study on strategies and challenges. *Journal of Systems and Software*, *215*, Article 112110. https://doi.org/10.1016/j.jss.2024.112110

17. Martin-Lopez, A., Segura, S., Muller, C., & Ruiz-Cortes, A. (2022). Specification and automated analysis of inter-parameter dependencies in web APIs. *IEEE Transactions on Services Computing*, *15*(4), 2342–2355. https://doi.org/10.1109/tsc.2021.3050610

18. Masoumi, H., Farahani, B., & Shams Aliee, F. (2022). Systematic and ontology-based approach to interoperable cross-domain open government data services. *Transforming*

UNDER PEER REVIEW IN IJAR

Government: People, Process and Policy, *16*(1), 110–127. https://doi.org/10.1108/tg-08-2021-0132

19. Matheus, R., Janssen, M., & Janowski, T. (2021). Design principles for creating digital transparency in government. *Government Information Quarterly*, *38*(1), Article 101550. https://doi.org/10.1016/j.giq.2020.101550

20. Mequanenit, A. M., Nibret, E. A., Herrero-Martín, P., García-González, M. S., & Martínez-Béjar, R. (2025). A multi-agent deep reinforcement learning system for governmental interoperability. *Applied Sciences*, *15*(6), Article 3146. https://doi.org/10.3390/app15063146

21. Muhamad, W., Suhardi, & Bandung, Y. (2022). Transforming OpenAPI Specification 3.0 documents into RDF-based semantic web services. *Journal of Big Data*, *9*(1), Article 55. https://doi.org/10.1186/s40537-022-00600-8

22. Munonye, K., & Péter, M. (2022). Machine learning approach to vulnerability detection in OAuth 2.0 authentication and authorization flow. *International Journal of Information Security*, *21*(2), 223–237. https://doi.org/10.1007/s10207-021-00551-w

23. Palma, F., Olsson, T., Wingkvist, A., & Gonzalez-Huerta, J. (2022). Assessing the linguistic quality of REST APIs for IoT applications. *Journal of Systems and Software*, *191*, Article 111369. https://doi.org/10.1016/j.jss.2022.111369

24. Quiña-Mera, A., Fernandez, P., García, J. M., & Ruiz-Cortés, A. (2023). GraphQL: A systematic mapping study. *ACM Computing Surveys*, *55*(10), 1–35. https://doi.org/10.1145/3561818

25. Santos Filho, A., Rodríguez, R. J., & Feitosa, E. L. (2025). Automated broken object-level authorization attack detection in REST APIs through OpenAPI to colored Petri nets transformation. *International Journal of Information Security*, *24*(2), Article 83. https://doi.org/10.1007/s10207-024-00970-5

26. Santos, J., Reppas, E., Wauters, T., Volckaert, B., & De Turck, F. (2025). Gwydion: Efficient auto-scaling for complex containerized applications in Kubernetes through reinforcement learning. *Journal of Network and Computer Applications*, *234*, Article 104067. https://doi.org/10.1016/j.jnca.2024.104067

UNDER PEER REVIEW IN IJAR

27. Shyam Mohan, J. S., & Goswami, K. (2025). Performance analysis and comparison of Node.js and Java Spring Boot in implementation of RESTful applications. *Software: Practice and Experience*, *55*(7), 1209–1233. https://doi.org/10.1002/spe.3418

28. Tardio, R., Mate, A., & Trujillo, J. (2020). An iterative methodology for defining big data analytics architectures. *IEEE Access*, *8*, 210597–210616. https://doi.org/10.1109/access.2020.3039455

29. van der Vlist, F. N., Helmond, A., Burkhardt, M., & Seitz, T. (2022). API governance: The case of Facebook's evolution. *Social Media + Society*, *8*(2), Article 20563051221086228. https://doi.org/10.1177/20563051221086228

30. Zygiaris, S., & Maamari, B. E. (2023). The journey from e-government to digital transformation: The case of Saudi Arabia. *Electronic Government, an International Journal*, *19*(1), 95. https://doi.org/10.1504/eg.2023.127578

UNDER PEER REVIEW IN IJAR