Volume XI Issue 2 Year 2026 | Page 539-549 | ISSN: 2527-9866
Received: ***15-05-2025*** | *Revised:* ***20-05-2025*** | *Accepted:* ***26-05-2025***

An Offstage WebView Architecture for API-Less Digital Public Service Integration in Local Government Super Applications

Fina Febrianti[1], Alim Hardiansyah[2], Yulian Ansori[3]
[123]Sultan Ageng Tirtayasa University, Cilegon City, Banten, Indonesia, 42434
e-mail: *3337220061@untirta.ac.id[1], alim.hardiansyah@untirta.ac.id[2], yulian.ansori@untirta.ac.id[3]*
*Correspondence: *3337220061@untirta.ac.id*

Abstract: *The fragmentation of digital public services in Indonesian local governments, compounded by the near-universal absence of publicly accessible APIs on government service websites, prevents direct integration into unified super applications through conventional mechanisms. This study proposes a Flutter-based offstage WebView scraping architecture that resolves this barrier without requiring backend modification or server-side infrastructure. The architecture introduces a hidden WebView processing layer that fully renders the target government website and executes a dual-strategy JavaScript extraction pipeline, combining DataTables API retrieval with DOM fallback scraping, before transmitting structured data to a native Flutter presentation layer. This separation of the retrieval and rendering layers eliminates the interface inconsistency of conventional WebView integration while removing the deployment overhead of server-side approaches. Evaluated through implementation in the Lebak Super Application, the technique achieved a 97.0% extraction success rate across one hundred trials, a mean end-to-end response time of 4.31s under Wi-Fi conditions, and a peak memory consumption of 6.2 MB during the extraction phase. All thirty functional black box testing scenarios passed. These results demonstrate that the proposed architecture is a viable transitional integration strategy for local governments operating under limited API infrastructure readiness.*

Keywords: *Flutter, offstage WebView, API-less integration, super application, web scraping*

1. Introduction

The development of information and communication technology has dramatically reshaped government management, especially in Indonesia[1]. The regulation on Electronic-Based Government System (Sistem Pemerintahan Berbasis Elektronik, or SPBE), issued by Presidential Regulation No. 95 of 2018, obliges the government to provide digital public services that are integrated, efficient and oriented to the needs of citizens[2]. In this context, local governments are becoming more and more accountable to transform their service ecosystems into a single digital architecture[3]. However, a fragmented digital public service architecture remains a persistent challenge. This happens when each government agency develops and manages its own service website, forcing the public to access services on different sites. This approach impairs the level of efficiency, inclusiveness, and public trust in the service[3].

An increasing number of local governments are developing super applications to unify various digital public services into a single interface[4]. This fragmentation challenge is illustrated by Lebak Regency's eighteen independently managed digital service websites across their various agencies[5]. Although the existing initiatives in Indonesian cities support the applicability of the super application model[4], they simultaneously expose a significant technical challenge that stems from local government service websites. These websites, which are managed by local government agencies, predominantly do not have machine-readable data exchange endpoints, which makes traditional API integrations impossible. This is evident from the systematic reviews

that identified the majority of Indonesian local governments persist in developing fragmented digital services with limited interoperability[3], [6] and is further substantiated by the national SPBE evaluation data, which show that interoperability is the lowest-scoring domain across assessed agencies[7]. An initial phase of this study involved a systematic observational check of all eighteen agency websites in Lebak Regency. This check confirmed that none provided a machine-readable data exchange endpoint[5]. This correlates with the works of Setijadi et al.[3] and Kencono et al.[8] in that the lack of interoperability infrastructure has been identified as a major and systemic obstruction to the realization of SPBE in various local governments in Indonesia. Although the case study shall focus on Lebak Regency, the example drawn is likely to be valid for many other local governments in Indonesia as well. Multiple systematic reviews on the SPBE implementation conducted across almost all regions of Indonesia show that the local governments/administrations are mostly affected by fragmented application processes and a scarcity of interoperability infrastructure at the local government level[3], [6]. So, it could be suggested that the sample drawn from this study is likely to be valid for other local governments in Indonesia that are affected by similar situations.

Three commonly mentioned integration methods each have significant drawbacks in the current situation. First, the conventional WebView method introduces inconsistencies in the mobile application interface, as it embeds external website interfaces directly into mobile applications[9]. Next, server-side scraping methods place a burden of dependency on an infrastructure that is unlikely to exist within resource-constrained local government teams[10]. Integration via API gateways presupposes the existence of a machine-readable endpoint, which is highly unlikely for most implementations of government services in Indonesia[11]. In the absence of APIs, constrained infrastructure, and the need for consistent interfaces, none of these methods addresses all three concerns.

The research essentially identifies the absence of a client-side integration method that does not require backend alteration, is independent of server-side processes, and preserves the client-side interface[3]. Previous studies regarding WebView integration, server-side scraping, and Flutter-based development[11] analyze the aforementioned areas as distinct issues, rather than as interrelated aspects of a single problem.

This study fills the gap by proposing a Flutter-based scraping technique using an off-stage WebView. In this solution, the WebView is placed in an offscreen rendering mode, allowing it to load the target webpage and execute all its client-side scripts without showing the UI to the end user. Next, an injecting JavaScript scraping code is executed to grab the data, and Flutter's UI is drawn using the native components. The proposed method is implemented and tested in the Lebak Super Application for the Lebak Regency Department of Industry and Trade's Commodity Price Information service using black box testing and a multidimensional performance evaluation framework.

This study presents three contributions. The first contribution is technical and consists of the development of a dual-strategy extraction pipeline that integrates DataTables API retrieval with DOM fallback scraping, offering a degree of fault tolerance that is missing from single-strategy implementations. The second contribution is architectural and is characterized by the strict separation of the hidden retrieval layer and the native presentation layer, which resolves interface inconsistency without reliance on a server-side component. The third contribution is practical, and it encompasses the idea that this method allows local government teams to join together numerous, disparate services without the need for any backend integration, and thus, facilitates the advancement of the SPBE agenda in scenarios characterized by insufficient API infrastructure readiness.

2. Literature Review

The incorporation of digital public services poses an ongoing challenge for smart governance, especially among local governments where services are created autonomously and lack interoperability[3]. Through a systematic review, Prihatmanto et al. have shown that interoperability frameworks are essential for the integration of public services[3], but their study is limited to broad and high-level architectural suggestions, and they do not put forward a practical approach to mechanisms that could be operable on the client side in API-absent environments, which are the criteria that indicate the core of the issue being dealt with in this study.

Nanavati et al. present evidence that Flutter is a solid framework for producing highly cross-platform applications featuring responsive native interfaces. Studies conducted on Flutter focus on internal application performance and not on the rendering architecture, and the external data fetching capabilities that Flutter provides are largely ignored by researchers, as is the use of offstage WebViews for the purpose of structuring data extraction[3], [11]. It is also noted that WebView integration has been researched in relation to super applications. Zhang et al. argue that within super applications, shared WebViews create security risks and isolation violations[12]. Although offstage rendering has positive effects in this area, their design mostly tackles security contrary to the problems of data extraction and the use of native UI design. Weichbroth interviewed various experts on the subject, and his findings pointed to the use of inconsistent interfaces as a major hindrance to usability in mobile applications[13]. This study will also solve the issue of interface consistency by removing the WebView layer, which is also located in plain sight. In the context of extraction, Sharma and Borkar stated that for websites that rely on JavaScript, dynamic scraping that is rendering-aware, is superior to static HTML parsing[10]. In a similar vein, Tacuri et al. stated that structured scraping improves extraction flexibility across diverse web resources, and thus, improves the flexibility of extraction across web resources[14]. Both studies hold empirical evidence for the technical aspects of this study; however, both studies solely utilize server-side run-time environments, making their frameworks infeasible for those local governments that lack the ability to allocate resources as well as maintain external web scraping resources[3].

Table 1. Comparison with Previous Research

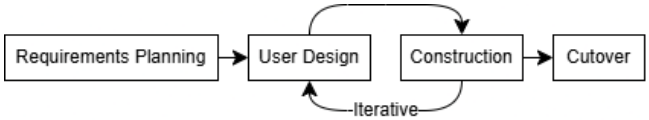
Reference	Focus	Method	Key Finding	Limitation Relative to This Study
Prihatmanto et al. (2024)	Smart government architectures	Systematic literature review	Interoperability frameworks are critical for public service integration	Provides theoretical basis only; no practical client-side integration mechanism for API-absent environments
Nanavati et al. (2024)	Flutter application performance	Performance analysis	Flutter supports responsive and consistent cross-platform UI	Addresses internal app performance only; does not explore WebView-based external data extraction
Zhang et al. (2025)	WebView-based super-app ecosystems	Security and static analysis	Shared WebView environments introduce interface isolation issues	Focused on security analysis; does not propose structured data extraction or native UI presentation
Tacuri et al. (2025)	Multi-source web scraping	Tool evaluation	Controlled scraping improves extraction flexibility	Relies entirely on server-side infrastructure; impractical for resource-constrained local government teams

Reference	Focus	Method	Key Finding	Limitation Relative to This Study
Sharma & Borkar (2024)	Dynamic web scraping strategies	Comparative experimental	Dynamic rendering outperforms static HTML parsing	All evaluated strategies require server-side runtime environments; no client-side alternative proposed
Weichbroth (2025)	Mobile application usability	Practitioner insights and literature analysis	Identifies critical usability challenges and outlines future research directions for mobile apps	Focuses on general usability challenges and practitioner perspectives; does not propose a constructive technical architecture to resolve UI inconsistencies caused by API-less integration

Among the four integration methods discussed in the context of this study, including the conventional WebView, server-side scraping, API gateway integration, and the offstage WebView approach, the method for the offstage WebView approach is the only integration method that eliminates all visible WebView rendering, server-side infrastructure dependency, and API availability, and offers a fully native user interface. This method is therefore relevant as a possible integration approach under the existing API infrastructure conditions.

3.Methods

The current research utilizes an applied research design with a developmental approach, producing a working software artifact to tackle specific technical integration challenges within local government digital public services. The chosen development process is Rapid Application Development (RAD), which is selected due to its appropriateness with rapid development cycle iterative prototyping[15]. In RAD, the development process is divided into four key stages: requirements planning, user design, construction, and cutover. The design and construction stages are undertaken repetitively with organized team reviews, to ensure that all requirements pertaining to functionality and aesthetics are met. Subsequently, the process proceeds to the cutover phase. The phases are illustrated in Figure 1.



This research focuses on the Commodity Price Information service of the Lebak Regency Department of Industry and Trade that is available on their public-facing website[16]. This service provides information on the commodity prices at ten traditional markets in Lebak Regency. Each entry lists the commodity, the unit of measure, the current price, the previous price, and the percentage of price change. The website uses a server-side rendering framework, and client-side data hydration is implemented with jQuery and the DataTables library. This service is chosen because it is a concrete example of the class of government digital services that include a publicly available web interface, no machine-readable data exchange endpoint, and use of a third-party JavaScript library for rendering content.

Two methods were used for data collection. The first method used an observational analysis to capture the frontend structure of the target website. The analysis also aimed to capture the structure of the Document Object Model (DOM) for data tables. Finally, the analysis also aimed to capture the DataTables and jQuery components and the initialization behavior. The analysis provided the foundation for the design of the JavaScript extraction script and the choice of DOM selectors. The

second method used the collection of quantitative data for performance and reliability in the execution of defined evaluation scenarios on the built system. The two-layer architecture of this system is comprised of a hidden data retrieval layer built on top of the offstage InAppWebView component and a native Flutter presentation layer. The entire system architecture is shown in Figure 2.

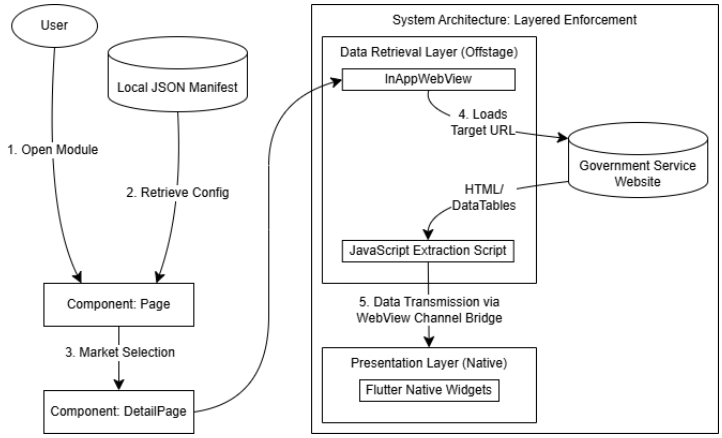


Figure 2. System architecture of the scraping implementation

The WebView component operates in offstage mode, meaning it processes web content within the application, remains in memory, and operates without showing any output to the user. Upon completion of the page load, a script that extracts information and executes JavaScript is inserted. The script first attempts to use the DataTables JavaScript API to obtain the initialized DataTables object in the browser's memory to retrieve the data. The primary modification is a fallback approach, which performs a series of CSS selector-based, direct, table row data extractions. The extracted data is then organized as a JSON object, and a text-based JavaScript communication bridge is used to send the data to the Flutter-based application. An approximate time of 2,500 milliseconds is inserted as a delay after the page is rendered. This provides additional time for the client-side rendering to run to completion, as during the initial tests of the website, typical completion of jQuery and DataTables took less than 2,000 milliseconds using standard Wi-Fi, and a 500-millisecond buffer was inserted for latency issues. A complete overview of the execution of the Data Extraction pipeline is shown in Figure 3.

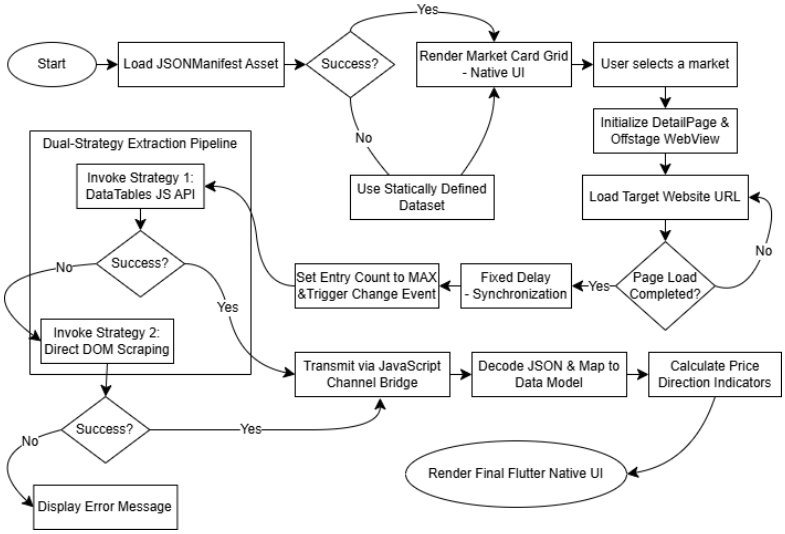


Figure 3. Execution logic of the extraction pipeline

In an effort to provide an empirically based technical contribution and move beyond an evaluation that is only concerned with functional correctness, this study implements a multifaceted evaluation framework composed of functional correctness testing, performance benchmarking, reliability assessment, and robustness testing. Functional correctness was assessed using black box testing[17]. This comprised thirty structured test scenarios, separated into four categories: page loading, data extraction, data accuracy, and user interface functionality. Each test scenario documented input conditions, expected results, actual results, and a determination of pass/fail. To ensure the results corresponded to actual runtime behavior, testing was performed on a real Android device.

The performance benchmarks indicated the method's responsiveness and the consumption of computational resources. CPU and memory use were evaluated in three states: idle state (application loaded, no extraction active), extraction phase (WebView rendering and JavaScript injection), and rendering phase (completion of native Flutter widget population). The standard diagnostic tool for empirical mobile assessments, Android Profiler, was used to document the presence of consistent behavior for five repeated trials for each state, and mean and standard deviation of each state were reported.

Response time, which measures the time taken for WebView to initialize and render the native widgets, was assessed for thirty independent runs under controlled network conditions. For comparison, the same target WebView was tested under the same thirty runs, without offstage rendering or data extraction, under the same controlled network conditions and device configurations. Additionally, the total end-to-end latency was assessed and compared under three simulated network scenarios, in accordance with mobile application testing standards[18]. These network conditions were set as high-speed Wi-Fi with a nominal bandwidth of 20 Mbps, standard 4G LTE cellular data, which offers approximately 10 Mbps, and constrained network conditions simulating ultra-slow 3G data, with a bandwidth of 1.5 Mbps. The comparison was meant to describe the proposed solution's latency across the realistic end-use environments of Lebak Regency.

Reliability assessment analyzed the success rate for extracting data using the dual strategy pipeline. One hundred extraction attempts were made, and the standard network conditions during the attempts were observed. The primary extraction success rate was defined by the number of attempts that resulted in the complete and structurally valid data. The success rate of each strategy was evaluated separately. This included attempts resolved by the primary DataTables API, as well as attempts resolved by the fallback strategy (DOOM fallback). This also provided a quantitative measure of the pipeline's tolerance to fault tolerance.

Robustness testing examined the system under two types of adversarial conditions. The first involved assessing the system’s ability to cope with document object model (DOM) changes. This was done by making front-end structural changes to the target website and adjusting some of the primary table class attributes and the nesting depth. Then, the number of successful extraction attempts was recorded to evaluate system performance without requiring modifications to the extraction script. This test was intended to determine the level of sensitivity of the proposed approach to front-end changes, which is the predominant front-end maintenance risk in client-side web scraping. The second adversarial condition involved the variability of network conditions. This involved assessing extraction success and measuring response times for the three tiers of network performance during the benchmarking phase of the project, to assess if the network performance (and the resultant network conditions) had an adverse effect on extraction success.

All quantitative measurements will be given in the form of mean values plus/minus one standard deviation. The performance data will be provided together with results of black box testing in order

to give a complete multi-dimensional evaluation of the provided technique, so that reproducibility as well as comparative evaluation in future implementations will be supported.

4. Results and Discussion

This integration technique underwent a thorough multidimensional assessment for validation of both technical and practical aspects. The results reveal extensive, detailed coverage of the system's performance in areas including functional correctness, resource usage, reliability of data extraction, and adaptability under various scenarios.

Table. 2 Functional Black Box Testing Summary

Functional Group	ID	Test Scenarios	Pass	Fail
Page Loading	TC-PL	6	6	0
Data Extraction	TC-DE	7	7	0
Data Accuracy	TC-DA	8	8	0
User Interface Functionality	TC-UI	9	9	0
Total		30	30	0

Black box testing across all 30 scenarios achieved a 100% pass rate, confirming the functional reliability of the extraction and integration mechanisms.

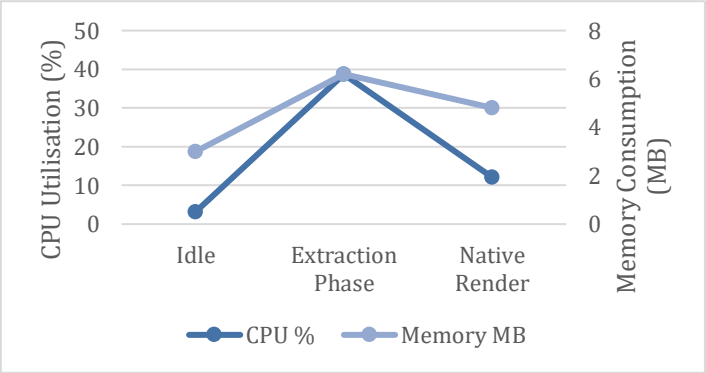


Figure 4. Application Resource Consumption Phase

For resource consumption, the peak CPU utilization recorded was 38.7% with a memory usage peak of 6.2 MB for the transient data extraction process. Both metrics stabilized quickly post-render[18]. These metrics are within normal operating bounds for mobile applications that perform data retrieval. This aligns with the resource usage profiles for Flutter applications that have been documented[19], [20].

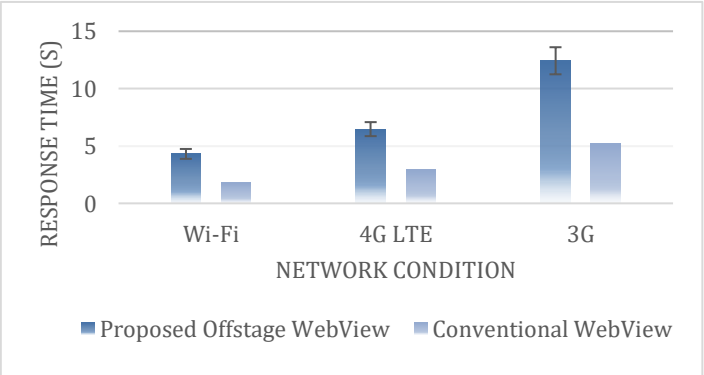


Figure 5. End-to-end response time comparison

The average end-to-end response times for Wi-Fi, 4G LTE, and 3G tethering were 4.31s, 6.74s, and 12.43s respectively. For context, a standard WebView implementation that was loading the same site in question averaged 1.82s for Wi-Fi, 2.91s for 4G LTE, and 5.20s on 3G tethering. The proposed method adds an unavoidable latency overhead of 2.4s to 7.2s for a trade-off for interface independence on client-side data and UI rendering. Response times are usually acceptable for Wi-Fi and 4G networks in urban settings, but the latency of 3G is too high and warrants improvements to local caching[18].

The dual-strategy extraction pipeline achieved a 97.0% success rate after 100 trials. The first of the two strategies, the DataTables API, resolved 89 of the trials. The second of the two strategies, the DOM fallback, resolved 8 of the trials. The remaining 3 cases of failure were attributed to target server latency that exceeded the 2,500 milli second sync window. Data demonstrates empirical resilience over a single-strategy pipeline which is expected to be mostly hypothetical.

Table. 3 System Robustness Against Simulated DOM Modifications

Modification Scenario	Simulated Front-End Change	Extraction Success Rate
Condition (i)	Alteration of primary table class attributes	91.0%
Condition (ii)	Modification of DataTables wrapper nesting depth	74.0%
Condition (iii)	Renaming of jQuery-initialized table identifier	83.0%

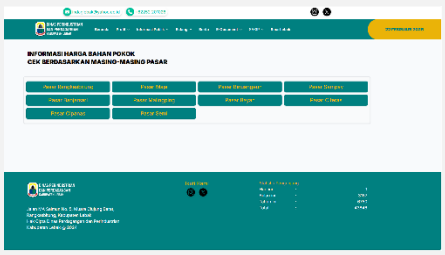
Testing for robustness by simulating changes to the DOM reveals a success rate between 74.0% and 91.0%. The results show that the fallback mechanism provides some level of resilience, but not complete, against changes to the front-end structure[14].

Table. 4 Extraction Success Rate Across Network Conditions

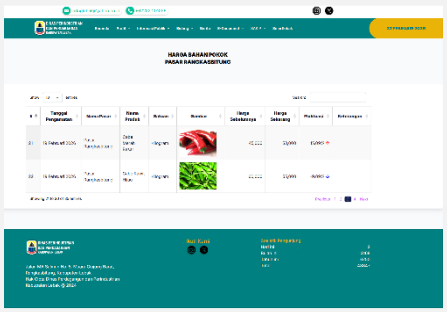
Network Condition	Extraction Success Rate	Primary Cause of Failure
Wi-Fi	97.0%	Baseline server timeout
4G LTE	95.0%	Occasional script timeout
3G	88.0%	Increased jQuery initialization timeout

Extraction success rates of 97.0% with a Wi-Fi connection dropped to 88.0% with a slow 3G connection. This is within the expected range of latency variability with slow cellular network conditions[21]. Adaptive synchronization delays that align to the state of the network will help with future implementation. Native Flutter implementations and the case-government services stand side by side in Figures 6.

(a)



(b)



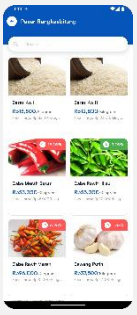


Figure 6. Visual separation between the external web rendering and the application interface: (a) market list, (b) commodity price

The empirical results of the study support the suggested technique's structural implementation. The advanced extraction success rate, combined with the dual-strategy fault tolerance, improves systems utilizing single-strategy scraping techniques. Also, the architectural separation between hidden data retrieval and the native presentation no longer creates the interface inconsistency issue that previous implementations of Flutter WebView experienced. The latency overhead compared to traditional WebView is fundamentally unavoidable since it fully client-side renders, extracts, and builds the precise UI in the same process. This cost, however, replaces the burden on the server side that would otherwise be created by alternative scraping methods[22], which in the case of resource-constrained local governments, represents a positive trade-off.

The proposed approach is constrained by four main limitations. In terms of scalability, each additional integrated service requires a separate WebView instance, which places proportional memory pressure that can be a bottleneck on lower specification devices when considering an entire comprehensive government super application. Therefore, a shared WebView pool with strict session isolation is an appropriate architectural enhancement. In the absence of session isolation, shared WebViews have been empirically proven to leak data and cause identity crisis issues in super-application ecosystems[12], [23]. In terms of maintainability, DOM changes on target websites create challenges to Web scraping of government services. Therefore, if a government website is updated, the scraping selectors need to be updated as well, and this maintenance cost increases with the number of government services integrated. Concerning security, testing in production is required in the context of strict channel and content control to mitigate the risks of expanded security surface within the WebView[12], [24]. With respect to the legal and ethical implications, automated extraction of publicly available government data is in a legal gray area in Indonesia[25]. Though the commodity price data in this research has no legal restrictions and no private data, in the absence of similar data and services, the implementation will require special legal assessments. This approach is best understood as a transitional integration mechanism. Its potential lies in allowing the development of advanced applications to outpace the development of supporting API infrastructure. In this sense, it provides a meaningful solution to the standardized government API requirements that are a prerequisite to achieving sustainable SPBE interoperability[2], [8].

5. Conclusions

This study shows that the Flutter-based offstage WebView scraping method can be implemented as a local government 'super' application building block in the context of a missing API infrastructure. Three key contributions to the field have been developed through this research. The first contribution is technical. It is related to the two-pronged extraction strategy, which relies on the use of the DataTables API with DOM fallback scraping as the extraction method. With this dual-pronged extraction strategy, an extraction success rate of 97.0% was obtained. Also, this

strategy set a benchmark of fault tolerance, which is absent from other 'single-pronged' strategies. In this elimination strategy, the fallback mechanism managed to autonomously recover 8.0% of all elimination attempts.

The second contribution is architectural. Specifically, it separates the hidden data retrieval layer from the Flutter drawing layer. This separation resolves Flutter's interface inconsistencies that arise from using WebView, without the server-side infrastructure dependencies found in other scraping approaches. The combination of offstage rendering, client-side extraction, and a native user interface has not yet been documented in the context of a government super application. The third contribution is the use of a practical example. This approach allows local government development teams to integrate fragmented digital services without backend access. This capability advances the State's Policy of Digital Government integration of services and the Electronic-Based Government System (SPBE) initiative, even when the necessary API infrastructure is still developing. The approach is validated in the Lebak Super Application, which has been tested with thirty black box testing scenarios and four quantitative evaluation dimensions, and demonstrates the approach is viable within a representative local government context in Indonesia.

The best characterization for this technique would be a transitional integration mechanism. Regarding its current constraints, sensitivity to front-end markup changes, latency overhead in case of degraded network conditions, and linear increase in memory capacity when more service instances are run concurrently will effectively shape the scope and goals of future research. This entails the implementation of possibly adaptive synchronization delays with real-time measurements of network conditions, automation of DOM changes detection with an alerting mechanism, and the implementation of local data caching to reduce latency on slow networks. Additionally, the architecture of a shared WebView pool will address the memory pressure caused by concurrent services and cover cross-framework evaluations in addition to the DataTables and jQuery scope. Longitudinal comparative studies against server-side scraping of various government service typologies will further help to more firmly establish the basis for a wider adoption.

References

[1] R. Wagola, A. Nurmandi, Misran, and D. Subekti, "Government Digital Transformation in Indonesia," in *HCI International 2023 Posters*, Copenhagen: Springer, Jul. 2023, pp. 286–296. doi: https://doi.org/10.1007/978-3-031-36001-5_37.

[2] Presiden Republik Indonesia, *Peraturan Presiden Nomor 95 Tahun 2018 tentang Sistem Pemerintahan Berbasis Elektronik*. Indonesia, 2018.

[3] A. Setijadi Prihatmanto, R. Andrian, W. Danar Sunindyo, and R. Sutriadi, "Transforming Public Services: A Systematic Review of Smart Government Frameworks, Architectures, and Implementation Challenges," *IEEE Access*, vol. 12, pp. 135799–135810, Aug. 2024, doi: 10.1109/ACCESS.2024.3450907.

[4] J. R. Sumirat and A. R. Syahputra, "'Super App Precision' Sebagai Bentuk Pelayanan Publik di Era Masyarakat 5.0," *JUSTIN: Jurnal Sistem dan Teknologi Informasi*, vol. 12, no. 1, pp. 1–7, Jan. 2024, doi: 10.26418/JUSTIN.V11I3.65162.

[5] Pemerintah Kabupaten Lebak, "Portal Resmi Kabupaten Lebak." Accessed: May 08, 2026. [Online]. Available: https://lebakkab.go.id/

[6] D. Mutiarin, H. Lawelai, A. Sadat, Nastia, A. A. M. Wijaya, and L. M. A. Sa'ban, "Interoperability Governance: An Analysis of the Impact of Digitization of Public Services on Local Government," *LOGOS: Journal of Local Government Issues*, vol. 7, no. 2, pp. 111–128, Sep. 2024, doi: 10.22219/LOGOS.V7I2.30388.

[7] Kementerian Pendayagunaan Aparatur Negara dan Reformasi Birokrasi, *Keputusan Menteri Pendayagunaan Aparatur Negara dan Reformasi Birokrasi Nomor 13 Tahun 2024 tentang Hasil*

Evaluasi Sistem Pemerintahan Berbasis Elektronik Pada Instansi Pusat dan Pemerintah Daerah Tahun 2023. Indonesia, 2024.

[8] B. D. Kencono, H. H. Putri, and T. W. Handoko, "Transformasi Pemerintahan Digital: Tantangan dalam Perkembangan Sistem Pemerintahan Berbasis Elektronik (SPBE) di Indonesia," *JIIP - Jurnal Ilmiah Ilmu Pendidikan*, vol. 7, no. 2, pp. 1498–1506, Feb. 2024, doi: 10.54371/JIIP.V7I2.3519.

[9] M. Mahendra and B. Anggorojati, "Evaluating the performance of Android based Cross-Platform App Development Frameworks," in *ICCIP 2020*, Tokyo: Association for Computing Machinery, May 2021, pp. 32–37. doi: 10.1145/3442555.3442561.

[10] K. Sharma and G. M. Borkar, "Comparative Analysis of Dynamic Web Scraping Strategies: Evaluating Techniques for Enhanced Data Acquisition," *Advancements in Communication and Systems*, pp. 241–252, Feb. 2024, doi: 10.56155/978-81-955020-7-3-22.

[11] J. Nanavati, S. Patel, U. Patel, and A. Patel, "Critical Review and Fine-Tuning Performance of Flutter Applications," in *ICMCSI 2024*, Lalitpur: IEEE, Apr. 2024, pp. 838–841. doi: 10.1109/ICMCSI61536.2024.00131.

[12] M. Zhang, S. Wang, G. Zheng, Y. Zhao, and H. Wang, "Demystifying Cookie Sharing Risks in WebView-based Mobile App-in-app Ecosystems," in *ASE 2025*, Seoul: IEEE, Jan. 2025, pp. 1693–1704. doi: 10.1109/ASE63991.2025.00142.

[13] P. Weichbroth, "Usability Issues With Mobile Applications: Insights From Practitioners and Future Research Directions," *IEEE Access*, vol. 13, pp. 91301–91311, Feb. 2025, doi: 10.1109/ACCESS.2025.3573503.

[14] A. Tacuri, S. Firmenich, A. Fernandez, F. Riva, M. Urbieta, and G. Rossi, "Web Scraping by End Users," *IEEE Access*, vol. 13, pp. 205027–205044, Nov. 2025, doi: 10.1109/ACCESS.2025.3636662.

[15] H. Suwandi, H. Harlinda, and S. H. Mansyur, "Implementation of a School Information System Using Rapid Application Development Method," *JUTIF: Jurnal Teknik Informatika*, vol. 3, no. 6, pp. 1501–1512, Dec. 2022, doi: 10.20884/1.JUTIF.2022.3.6.332.

[16] Pemerintah Kabupaten Lebak, "Situs Resmi Disperindag Kabupaten Lebak." Accessed: May 15, 2026. [Online]. Available: https://disperindag.lebakkab.go.id/

[17] P. Kusuma Ayuningtyas, D. Atmodjo WP, and P. Rachmadi, "Performance And Functional Testing With The Black Box Testing Method," *IJPSAT: International Journal of Progressive Sciences and Technologies*, vol. 39, no. 2, pp. 212–218, Jul. 2023, doi: 10.52155/ijpsat.v39.2.5471.

[18] A. Kaczmarczyk, P. Zajac, and W. Zabierowski, "Performance Comparison of Native and Hybrid Android Mobile Applications Based on Sensor Data-Driven Applications Based on Bluetooth Low Energy (BLE) and Wi-Fi Communication Architecture," *Energies (Basel).*, vol. 15, no. 13, p. 4574, Jun. 2022, doi: 10.3390/EN15134574.

[19] R. Rua and J. Saraiva, "A large-scale empirical study on mobile performance: energy, run-time and memory," *Empir. Softw. Eng.*, vol. 29, no. 1, Dec. 2023, doi: 10.1007/S10664-023-10391-Y.

[20] W. Oliveira, B. Moraes, F. Castor, and J. P. Fernandes, "Analyzing the Resource Usage Overhead of Mobile App Development Frameworks," in *EASE 2023*, Association for Computing Machinery, Jun. 2023, pp. 152–161. doi: 10.1145/3593434.3593487.

[21] A. Alhammadi *et al.*, "Revolutionizing Mobile Broadband: Assessing Multicellular Networks in Indoor and Outdoor Environments," *IEEE Access*, vol. 12, pp. 120840–120863, Aug. 2024, doi: 10.1109/ACCESS.2024.3451961.

[22] L. Hidayati, L. P. Kusuma, D. Agustini, and V. Y. P. Ardhana, "Implementasi Web Scraping untuk Pengumpulan Data Media Sosial Lingkup Pemerintah Provinsi NTB," *SIMIKA: Jurnal Sistem Informasi dan Informatika*, vol. 7, no. 1, pp. 63–72, Mar. 2024, doi: 10.47080/SIMIKA.V7I1.3200.

[23] L. Zhang *et al.*, "Identity Confusion in WebView-based Mobile App-in-app Ecosystems," in *USENIX security 2022*, Boston: USENIX, 2022, pp. 1597–1613.

[24] T. T. Nguyen and B. Stock, "Studying the Privacy Risks in Android WebView's Web Permission Enforcement," in *ACM ASIACCS 2025*, Hanoi: Association for Computing Machinery, Aug. 2025, pp. 1309–1322. doi: 10.1145/3708821.3710821.

[25] K. Logos, R. Brewer, C. Langos, and B. Westlake, "Establishing a framework for the ethical and legal use of web scrapers by cybercrime and cybersecurity researchers: learnings from a systematic review of Australian research," *International Journal of Law and Information Technology*, vol. 31, no. 3, pp. 186–212, Nov. 2023, doi: 10.1093/IJLIT/EAAD023.