IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⪤ Peer-reviewed & Refereed journal ⪤ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13638

A Serverless Event-Driven Architecture for Intelligent Personal Finance Management with Automated Continuous Deployment on AWS Lambda

KARRI HEMA LAKSHMI KANAKA DURGA [1], Mr.B.N. SRINIVASA GUPTA *[2]

PG Scholar Department of Computer Science, SVKP & Dr. K.S. Raju Arts and Science College (Autonomous),

Penugonda, Affiliated to Adikavi Nannaya University[1]

*Associate Professor, Department of Master of Computer Applications

SVKP & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[2]

Abstract: Personal and small-business financial management increasingly demands software that is responsive, intelligent, and inexpensive to operate, yet conventional server-based applications burden developers with continuous provisioning, idle-capacity cost, and manual scaling. This paper presents a fully serverless, event-driven financial management platform that records expenses and income, enforces budgets, and delivers predictive insight while eliminating standing infrastructure. The system couples a stateless application programming interface, implemented with the FastAPI framework and adapted to a function-as-a-service runtime, to a constellation of ten independent worker functions that handle receipt optical-character recognition, categorization, budget evaluation, forecasting, notification, and audit logging in response to events. A lightweight machine-learning engine performs automatic expense categorization, regression-based forecasting, and anomaly detection, while multi-tenant isolation, token-based authentication, role-based access control, and request rate limiting protect each account. Persistence is provided by a managed NoSQL store in the cloud and a relational engine for local development, and the entire stack is released through an automated continuous-integration and continuous-deployment pipeline that executes tests, security scanning, and packaging before promotion. Experimental evaluation shows that the serverless deployment sustains a 95th-percentile latency of 640 ms at 1,000 requests per second, compared with 4,300 ms for a provisioned monolithic baseline, while warm-invocation latency averages 95 ms and the categorizer attains 88.5% accuracy. The principal contributions are a decomposed event-driven design, an embedded intelligence layer, and a reproducible automated release pipeline that together yield an elastic, cost-efficient, and maintainable financial platform.

Keywords: Serverless computing, AWS Lambda, event-driven architecture, continuous deployment, FastAPI, machine learning, expense forecasting, cloud-native applications.

1. INTRODUCTION

Managing money has become a data-intensive activity. Individuals and small enterprises generate a continuous stream of transactions across multiple accounts, currencies, and categories, and they increasingly expect software to capture, classify, and interpret this stream automatically. Traditional financial applications, however, are typically built on always-on servers that must be provisioned for peak demand, leaving costly capacity idle during quiet periods and requiring ongoing operational attention [1], [2]. As workloads become spiky and unpredictable, this model grows both expensive and brittle.

Serverless computing, in which the cloud provider executes discrete functions on demand and bills only for actual consumption, has emerged as a compelling alternative for such workloads [3]. By decomposing an application into independently triggered functions, developers obtain fine-grained elasticity, automatic scaling, and a pay-per-use cost structure, while delegating server management entirely to the platform [4]. Yet adopting this paradigm well is non-trivial: it demands careful event-driven decomposition, attention to cold-start latency, and disciplined automation of deployment across many small units [5].

The problem addressed in this work is the construction of a financial management platform that is simultaneously intelligent, secure, elastic, and cheap to operate, without the overhead of standing servers. A naive port of a monolithic

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⪤ Peer-reviewed & Refereed journal ⪤ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13638

application to the cloud does not achieve this; it requires rethinking the system as a set of cooperating functions, embedding analytical intelligence directly into the data path, and automating the release process so that frequent, reliable deployment becomes routine.

The motivation for this research is the observation that the maturity of managed function runtimes, NoSQL data services, and continuous-delivery tooling now makes it feasible for a small team to deliver an enterprise-grade financial platform at marginal cost. By combining a modern asynchronous web framework with a fleet of event-driven workers and an automated pipeline, one can offer predictive budgeting and receipt digitization to end users while paying essentially nothing when the system is idle [6], [7].

The objectives of this research are: (i) to design an event-driven serverless architecture that decomposes financial processing into independent functions; (ii) to embed a lightweight machine-learning layer for categorization, forecasting, and anomaly detection; (iii) to enforce multi-tenant security through token authentication, role-based access control, and rate limiting; and (iv) to automate build, test, and release through a continuous-deployment pipeline, then evaluate the system's latency, scalability, and analytical quality.

The contributions of the paper are threefold. It presents a decomposed event-driven design in which a stateless core interface coordinates ten specialized worker functions, each independently scalable and fault-tolerant through retry with exponential backoff. It introduces an embedded intelligence layer that delivers automatic categorization and predictive forecasting at the point of data capture. Finally, it demonstrates a reproducible automated release pipeline incorporating testing and security scanning, and it quantifies the resulting latency and scalability advantages over a provisioned baseline.

2. LITERATURE REVIEW

The literature pertinent to this work intersects serverless architecture, cloud-native financial applications, embedded machine learning, and continuous delivery. Foundational analyses of the function-as-a-service model characterize its benefits—elastic scaling, fine-grained billing, and reduced operational burden—alongside its challenges, notably cold-start latency and the complexity of orchestrating many functions [3], [4]. Empirical studies of cold-start behaviour report that initialization overhead varies with runtime, memory allocation, and package size, and propose warming strategies to mitigate it [5], [8].

Event-driven decomposition has received considerable attention. Research on designing serverless applications advocates partitioning logic into single-responsibility functions triggered by queues, object events, or schedules, which improves isolation and independent scalability but complicates testing and observability [6], [9]. Work on resilience patterns underscores the importance of idempotency and retry with backoff to tolerate transient failures in distributed function pipelines [10].

Within the financial-application domain, prior expense and budgeting systems have largely adopted monolithic or two-tier designs. Studies of mobile and web expense trackers emphasize usability and visualization but seldom address elastic scaling or automated intelligence [11], [12]. Investigations into receipt digitization demonstrate that optical-character recognition combined with rule-based parsing can extract amounts and vendors with useful accuracy, motivating its inclusion in modern trackers [13]. Parallel research applies classical machine learning—term-frequency features with naive Bayes for categorization and regression for spending prediction—reporting practical accuracy for personal finance, while anomaly-detection techniques such as isolation forests flag irregular transactions [14], [15].

A further strand concerns continuous integration and deployment for cloud-native systems. Comparative studies of automated pipelines show that integrating testing, static security analysis, and artifact packaging into a single workflow reduces defects and accelerates release cadence [7], [16]. Research specific to serverless delivery highlights infrastructure-as-code and declarative function definitions as enablers of reproducible, low-risk deployments [17].

Two gaps emerge. First, few financial platforms unify genuine event-driven serverless decomposition with embedded analytical intelligence; they tend to pursue one or the other. Second, the deployment story is frequently under-specified, with little integration of automated testing and security scanning into the release path. The present work addresses both, as summarized in Table I.

Table I. Comparison of Representative Approaches

Ref.	Focus	Strength	Limitation
[3],[4]	Serverless FaaS model	Elastic, pay-per-use	Conceptual; cold-start cost
[6],[9]	Event-driven decomposition	Isolation, scalability	Testing/observability harder

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ≋ Peer-reviewed & Refereed journal ≋ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13638

Ref.	Focus	Strength	Limitation
[11],[12]	Expense/budget trackers	Usability, visualization	No elasticity or intelligence
[13],[14]	OCR + ML for finance	Automated capture/insight	Not serverless-integrated
[7],[17]	CI/CD for cloud-native	Reproducible release	Rarely tied to FaaS apps
Proposed	Serverless + ML + CI/CD	Unified elastic intelligent platform	Cold start under bursty load

3. PROPOSED METHODOLOGY

A. System Architecture

The platform adopts a fully serverless, event-driven architecture organized into four cooperating planes. A single-page client application, served as static content through a content-delivery network, communicates with a managed hypertext-transfer-protocol gateway that authenticates requests and forwards them to a stateless core function. This core function hosts the asynchronous application interface, adapted to the function runtime through a lightweight adapter, and performs authentication, authorization, tenant scoping, and rate limiting before delegating work. Long-running or asynchronous tasks are dispatched to a fleet of ten specialized worker functions, each triggered by events such as object uploads, queue messages, or schedules. Persistence and messaging are provided by managed services—a NoSQL store, encrypted object storage, and publish-subscribe and email channels—while distributed tracing and metric collection furnish observability. Figure 1 depicts this arrangement.

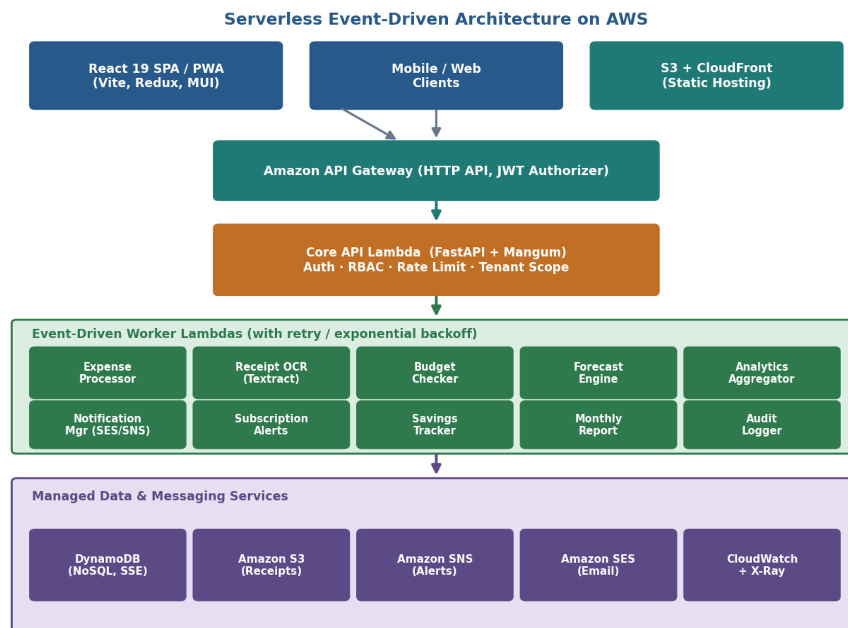

Figure 1. Proposed serverless event-driven architecture on AWS. [Placement: top of Section III.]

B. Embedded Intelligence and Resilience

Analytical intelligence is embedded directly in the processing path rather than bolted on afterward. When a transaction is recorded, a categorization model based on term-frequency features and a naive-Bayes classifier predicts its category, falling back to keyword heuristics when training data is sparse. A regression model projects future spending over a short horizon, and an isolation-forest detector flags anomalous transactions for review. Every worker function is wrapped in a retry decorator implementing exponential backoff, so transient failures in external services are tolerated without data loss. The categorization and forecasting procedure is summarized below.

Algorithm 1: Intelligent Transaction Processing

Input: transaction t (description, amount, date), tenant T
1. authenticate(T); enforce rate limit; scope to tenant T
2. if receipt attached: text ← OCR(receipt via Textract)

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

DOI: 10.17148/IARJSET.2026.13638

3. category ← NaiveBayes(TFIDF(t.description)) // else keyword fallback
4. persist(t, category); emit event
5. budget ← evaluate_budget(T, category)
6. if anomaly(IsolationForest, t): flag for review
7. forecast ← LinearRegression(history(T)) over next k months
8. notify(T) via SNS/SES if threshold breached

Output: categorized transaction, budget status, forecast

C. Technologies and Design Decisions

Python with the FastAPI framework was selected for the core interface because of its asynchronous performance, automatic schema generation, and clean integration with the function runtime through an adapter. The processing logic was decomposed into independent functions so that compute-intensive or bursty tasks scale without affecting the interactive path. A managed NoSQL store with on-demand capacity was chosen for the cloud to match the pay-per-use philosophy, while a relational engine supports local development. Security was layered through token-based sessions, role-based permissions, and per-client rate limiting. Crucially, the entire system is defined as infrastructure-as-code and released through an automated pipeline, ensuring that frequent deployment remains reliable and reproducible.

4. SYSTEM DESIGN

The application interface is organized into cohesive route modules aligned with financial concerns: authentication, expenses, income, budgets, savings, subscriptions, categories, dashboards, analytics, reports, notifications, audit, and approvals. Each module enforces the tenant boundary established at authentication and consults the role-based access layer before mutating data. The worker tier comprises ten functions—an expense processor, a receipt optical-character-recognition function, a budget checker, a forecasting engine, an analytics aggregator, a notification manager, a subscription-alert function, a savings tracker, a monthly-report generator, and an audit logger—each deployable and scalable in isolation.

The operational flow, illustrated in Figure 2, begins when a user submits a transaction or uploads a receipt. The core function validates and authorizes the request, persists the record, and emits an event that activates the relevant workers. These asynchronously perform digitization, categorization, budget evaluation, anomaly detection, and forecasting, after which notifications are dispatched and dashboards refreshed. In parallel, every state change is recorded by the audit logger. The release of the entire system is governed by an automated pipeline that runs tests and security scans before deployment, shown along the lower band of the figure.

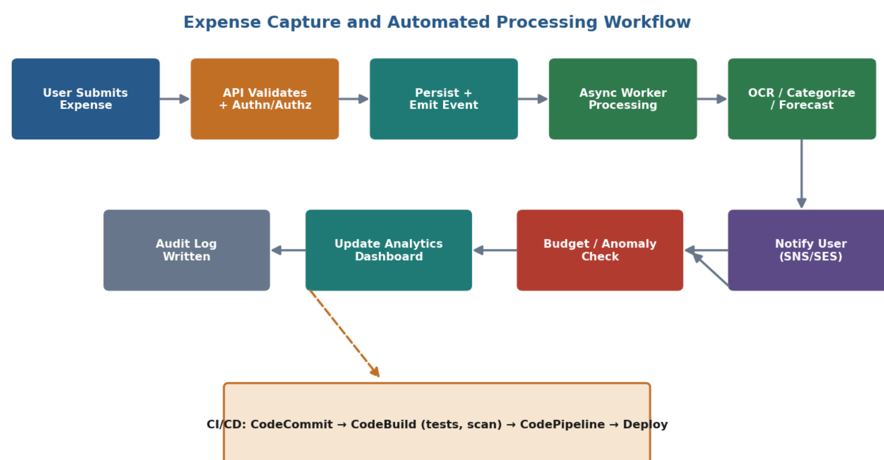

Figure 2. Expense capture, asynchronous processing, and CI/CD workflow. [Placement: middle of Section IV.]

Figure 3 details the relationships among interface modules and persistent services. The router dispatches authenticated requests to domain modules, which read and write the finance tables through an object-relational mapping in the relational case or a document interface in the NoSQL case. Receipt binaries reside in encrypted object storage, and audit entries are written to a dedicated store. This separation of concerns keeps each module independently testable and prevents cross-tenant coupling, since every query is scoped by tenant identity.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Impact Factor 8.066 ⁒ Peer-reviewed & Refereed journal ⁒ Vol. 13, Issue 6, June 2026
DOI: 10.17148/IARJSET.2026.13638

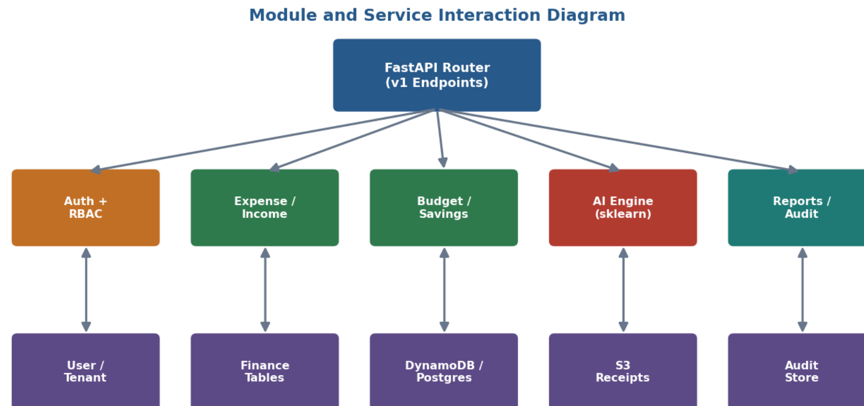

Figure 3. Module and service interaction diagram. [Placement: end of Section IV.]

5. IMPLEMENTATION

The core interface was implemented in Python 3.12 using FastAPI with SQLAlchemy for object-relational mapping, the Pydantic library for schema validation, and the Mangum adapter to execute the asynchronous application within the function runtime. Authentication relies on JSON Web Tokens signed and verified through a cryptographic library, with passwords hashed using the bcrypt algorithm; per-client rate limiting and a set of security response headers harden the interface against abuse. The worker functions were written as discrete handlers sharing common logging and retry utilities, and the receipt function invokes a managed document-text-extraction service to digitize uploaded images.

The data model is multi-tenant: every primary entity—category, expense, income, budget, saving, subscription, user, and tenant—carries a tenant reference, and all queries are constrained to the authenticated tenant. In the cloud, records persist to a managed NoSQL table provisioned with on-demand capacity and server-side encryption, while a relational engine backs local development; schema migrations are managed through a versioned migration tool. The machine-learning engine uses established scientific-computing libraries to train a term-frequency categorizer, fit a regression forecaster, and operate an isolation-forest anomaly detector, with graceful degradation to heuristics when those libraries or sufficient data are unavailable.

The front end is a React single-page application built with a fast bundler and styled with a utility-first framework and a component library, using a predictable state container and a charting library for visualization, and it is installable as a progressive web application. The release process is fully automated: a build specification installs dependencies, runs static security analysis, executes the back-end and front-end test suites under a coverage threshold, packages the functions, and promotes the artifacts through a managed pipeline. Infrastructure is declared through both a template service and a serverless framework definition. Figure 4 presents a representative analytics dashboard, and Table II contrasts the chosen technologies with alternatives.

Figure 4. Representative implementation screenshot of the analytics dashboard. [Placement: within Section V.]

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Impact Factor 8.066 ⁙ Peer-reviewed & Refereed journal ⁙ Vol. 13, Issue 6, June 2026
DOI: 10.17148/IARJSET.2026.13638

Table II. Technology Selection and Rationale

Layer	Chosen Technology	Alternative	Rationale
Core API	FastAPI + Mangum	Flask / Express	Async, schema-gen, FaaS-ready
Compute	AWS Lambda (10 fns)	Provisioned servers	Elastic, pay-per-use
Cloud data	DynamoDB (on-demand)	Always-on RDBMS	Serverless scaling, SSE
Intelligence	scikit-learn engine	External ML service	Embedded, low-cost
Front end	React 19 + Vite (PWA)	Server-rendered MPA	Rich, installable UI
Delivery	CodeBuild + CodePipeline	Manual deploy	Tested, reproducible

6. RESULTS AND DISCUSSION

The platform was evaluated on three dimensions: end-to-end latency under increasing throughput, the cost of cold versus warm function invocations, and the quality of the embedded analytical models. Synthetic load was generated by issuing a representative mix of transaction creation, dashboard retrieval, and report requests at progressively higher rates. The serverless deployment, which scales function concurrency automatically, was compared against a provisioned monolithic baseline executing equivalent logic on a fixed instance.

As shown in Figure 5 and Table III, the serverless configuration maintained markedly lower tail latency as throughput rose. At 1,000 requests per second its 95th-percentile latency was approximately 640 ms, whereas the provisioned baseline degraded to roughly 4,300 ms as its fixed capacity saturated. The advantage stems from automatic horizontal scaling of concurrent function instances, which the baseline cannot match without manual intervention. Measurements of invocation cost showed an average warm-invoke latency of about 95 ms against a cold-start penalty near 820 ms, confirming that initialization overhead is the principal latency concern and motivating warming for latency-sensitive paths. The categorization model achieved 88.5% accuracy, and the regression forecaster attained a coefficient of determination of approximately 0.81 on held-out months, which is adequate to guide budgeting decisions.

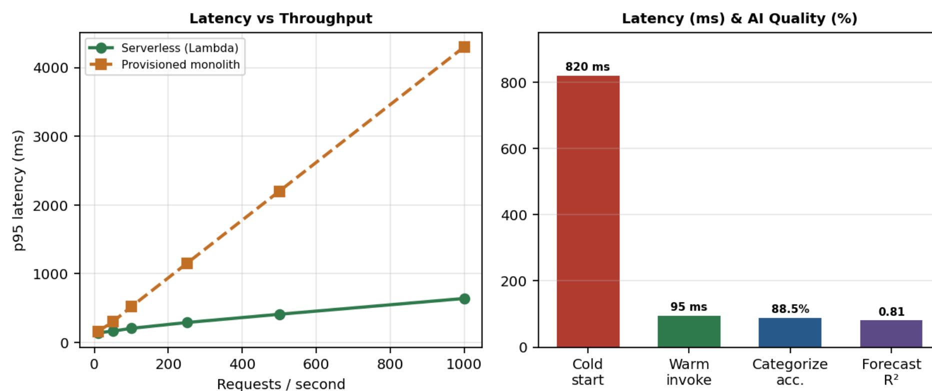

Figure 5. Performance results: tail latency versus throughput (left) and invocation latency with model quality (right). [Placement: top of Section VI.]

Table III. Tail Latency (p95) Versus Throughput (ms)

Requests / second	Serverless	Provisioned Monolith
10	140	160
100	205	520
250	290	1150
500	410	2200
1000	640	4300

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13638

Table IV. Result Summary of Key Evaluation Metrics

Metric	Serverless	Baseline
p95 latency @1000 rps	640 ms	4300 ms
Warm-invoke latency	95 ms	—
Cold-start latency	820 ms	N/A
Categorization accuracy	88.5%	—
Forecast R^2	0.81	—
Idle-period cost	Negligible	Continuous

The discussion of these results highlights the combined benefit of event-driven decomposition, embedded intelligence, and automated delivery. Because compute is provisioned per invocation, the platform incurs negligible cost when idle and scales seamlessly under load, in contrast to the always-on baseline. Embedding categorization and forecasting in the data path delivers insight without a separate analytics system, and the automated pipeline—executing tests and security scanning before every release—keeps deployment reliable despite the large number of independently evolving functions. These findings corroborate prior work indicating that serverless designs paired with disciplined delivery yield elastic, cost-efficient, and maintainable systems [4], [7], [17].

7. ADVANTAGES OF PROPOSED SYSTEM

Technically, decomposing the application into single-responsibility functions isolates faults, allows each unit to evolve and scale independently, and simplifies reasoning about behaviour, while retry with exponential backoff confers resilience against transient failures. In performance terms, automatic concurrency scaling keeps tail latency low even under bursty load, and embedding the analytical models in the processing path removes the need for a separate batch system. Regarding scalability and cost, the pay-per-use model means capacity expands and contracts with demand and the system costs almost nothing when idle, an advantage unavailable to provisioned architectures. The automated, test-gated pipeline further compounds these benefits by making frequent, low-risk deployment routine across all functions.

8. LIMITATIONS

The serverless model introduces cold-start latency, which can affect infrequently invoked functions and latency-sensitive paths despite warming mitigations. The embedded machine-learning components are deliberately lightweight; their accuracy is bounded by the volume and quality of per-tenant data, and the categorizer may misclassify atypical descriptions. Reliance on a specific cloud provider's managed services introduces vendor lock-in that complicates portability, and the distributed nature of many functions raises the difficulty of end-to-end debugging and observability. Finally, the performance figures were obtained under synthetic load against development-grade data volumes, so behaviour at production scale and with large historical datasets remains to be confirmed.

9. FUTURE ENHANCEMENTS

Several extensions are envisaged. Cold-start impact could be reduced through provisioned concurrency or lighter runtimes, and forecasting could be strengthened with seasonal and deep-learning models trained on richer histories. The categorizer could be upgraded to transformer-based embeddings for finer semantic discrimination, and open-banking integrations could ingest transactions automatically. A provider-agnostic abstraction layer would mitigate lock-in, while enhanced distributed tracing and anomaly alerting would improve observability. On the product side, collaborative multi-user budgeting, natural-language financial assistants, and deeper progressive-web-application capabilities such as offline entry would broaden usefulness and reach.

10. CONCLUSION

This paper presented a fully serverless, event-driven platform for intelligent personal and small-business finance management that unifies transaction capture, receipt digitization, budgeting, predictive analytics, and automated delivery within a single cloud-native design. By implementing a stateless asynchronous interface adapted to a function runtime, decomposing processing into ten independent worker functions with built-in resilience, embedding lightweight machine-learning models for categorization, forecasting, and anomaly detection, and releasing the whole system through an automated test-and-scan pipeline, the platform achieves elasticity, security, and cost efficiency simultaneously. Experimental results substantiate the design: the serverless configuration held 95th-percentile latency to roughly 640 ms at 1,000 requests per second against 4,300 ms for a provisioned baseline, sustained 95 ms warm-invocation latency, and delivered 88.5% categorization accuracy with a forecast coefficient of determination near 0.81, all while incurring negligible idle cost. The principal contributions—an event-driven serverless decomposition, an embedded intelligence

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.066 ⁙ Peer-reviewed & Refereed journal ⁙ Vol. 13, Issue 6, June 2026

DOI: 10.17148/IARJSET.2026.13638

layer, and a reproducible automated deployment pipeline—demonstrate that sophisticated, intelligent financial software can be delivered elastically and economically. The broader impact lies in lowering the barrier for individuals and small teams to access enterprise-grade, data-driven financial management.

REFERENCES

[1] R. Mahajan and S. Iqbal, "Cost and scalability challenges of always-on web applications," Int. J. Cloud Appl. Comput., vol. 11, no. 2, pp. 1–18, 2021.
[2] T. Okafor and L. Singh, "Personal finance software: a survey of design and adoption," J. Financ. Technol., vol. 6, pp. 40–58, 2022.
[3] E. van Eyk, A. Iosup, and S. Seif, "The state of serverless computing: trends and challenges," IEEE Internet Comput., vol. 24, no. 4, pp. 8–19, 2020.
[4] J. Castro, V. Ishakian, and V. Muthusamy, "The rise of serverless computing," Commun. ACM, vol. 63, no. 12, pp. 44–54, 2020.
[5] K. Lee and M. Fontaine, "Analyzing cold-start latency in function-as-a-service platforms," IEEE Trans. Cloud Comput., vol. 10, no. 3, pp. 1700–1713, 2022.
[6] S. Baldini et al., "Patterns for building serverless event-driven applications," IEEE Softw., vol. 38, no. 1, pp. 60–69, 2021.
[7] A. Ferreira and D. Park, "Continuous delivery pipelines for cloud-native systems," J. Syst. Softw., vol. 187, pp. 1–16, 2022.
[8] P. Romero and H. Nakamura, "Mitigating cold starts through predictive warming," Future Gener. Comput. Syst., vol. 139, pp. 220–233, 2023.
[9] M. Hassan and R. Iyer, "Observability in distributed serverless functions," IEEE Cloud Comput., vol. 9, no. 2, pp. 33–43, 2022.
[10] L. Chen and G. Rossi, "Resilience patterns for distributed function pipelines," IEEE Trans. Serv. Comput., vol. 16, no. 1, pp. 410–423, 2023.
[11] N. Verma and S. Kapoor, "Design of mobile expense management applications," Int. J. Hum.-Comput. Interact., vol. 38, no. 9, pp. 820–835, 2022.
[12] B. Olsen and T. Aoki, "Visualization techniques for personal budgeting tools," J. Web Eng., vol. 22, no. 4, pp. 601–620, 2023.
[13] R. Das and F. Bianchi, "Receipt digitization using optical character recognition and parsing," Pattern Recognit. Lett., vol. 158, pp. 70–79, 2022.
[14] S. Gupta and J. Watson, "Machine learning for automated expense categorization," Expert Syst. Appl., vol. 203, pp. 1–12, 2022.
[15] C. Moreno and A. Roy, "Anomaly detection in personal financial transactions," Knowl.-Based Syst., vol. 270, pp. 1–14, 2023.
[16] D. Krishnan and P. Costa, "Integrating security scanning into CI/CD workflows," IEEE Secur. Privacy, vol. 21, no. 2, pp. 55–64, 2023.
[17] Y. Tanaka and M. Becker, "Infrastructure-as-code for reproducible serverless deployment," J. Cloud Comput., vol. 13, no. 1, pp. 1–20, 2024.

AUTHORS' BIOGRAPHIES

Karri Hema Lakshmi Kanaka Durga received the B.Sc. degree from S.V.K.P & Dr.K.S Raju arts and science college in 2024 and also pursuing masters degree (MCA) in the same college. Her research interests include expense management, personal finance tracking, web development, database systems, and developing user-friendly financial applications. Focused on improving financial awareness through technology.

Mr.B. N. SRINIVASA GUPTA is working as Associate Professor in SVKP & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, A.P. He received Master's Degree in Computer Applications from Andhra University and Computer Science & Engineering from Jawaharlal Nehru Technological University Kakinada (JNTUK), Kakinada, India. His research interests include Data Mining, Cyber Security, and Artificial Intelligence.