

Design and Implementation of a Centralized Webhook Management System for Monitoring, Logging, and Event Control

Dr. Sarika S. Jadhav¹, Dr. Navnath S Bagal², Sarvesh Pandit³, Hrishikesh Garje⁴, Shruti Ghate⁵ and Aditya Ganesh⁶, 1-6Department of Computer Engineering, Padmabhooshan Vasantdada Patil Institute of Technology, Pune, India. Email: sarikasjadhav@gmail.com, navnathsbagal@gmail.com, sarveshpandit29@outlook.com, hrishikeshgarje554@gmail.com, shrutighate7@gmail.com, adityapganesh@gmail.com.

Abstract— In today's software world many apps rely on real-time data sharing between platforms like payment gateways, code repositories and communication tools. This sharing happens through event-driven systems like webhooks which let systems automatically send data when certain events occur. Webhooks are efficient and widely used. Managing multiple webhook sources across different services can be tough. It is hard to keep track of everything because there is no control. It is difficult to see what is happening with event flows. Logging is often inconsistent. It takes a while to detect failures. These challenges can make systems less reliable. Make debugging harder. To solve these problems, we propose a centralized webhook management system. This system brings together webhook processing, monitoring, logging and alerting into one platform. A backend service, built with server-side technologies acts as the main controller. It receives webhook events from services like payment systems and repository platforms. Each event is checked, categorized and processed efficiently to ensure execution. By having control, efficient event processing, detailed logging and proactive monitoring the proposed system improves overall reliability and maintainability. It also supports scalability allowing new services and webhook sources to be added with changes. This approach provides a solution for managing complex event-driven architectures in modern applications. It ensures performance and easier troubleshooting. The system helps to handle event-driven communications in a way. It makes it easier to add services and webhook sources. The system provides a platform, for webhook management. It improves the reliability of the system. The system makes it easier to debug and troubleshoot issues.

Index Terms— Web-Hook Orchestration, Stripe Service, Slack Service, Monitoring & Analysis in SaaS, Event-driven Architecture, API integrations.

I. Introduction

In today's software world applications usually work with services. They often need to talk to services like payment systems, version control systems and notification tools. One good way to make this communication happen is through webhooks. Webhooks let systems send data to applications when something specific happens. This makes them great for real-time connections. GitHub and Stripe use webhooks a lot. They use them to tell applications about events, like code changes or payment confirmations. This event-driven approach is good because it makes systems respond faster and work automatically. However, when many webhook sources are involved it can get tricky. It becomes hard to keep track of events find failures and make sure every event is processed correctly. Missed events can happen. Events can be processed late. There may not be logs. These issues can make it take a time to fix problems and reduce how reliable the system is another big concern is that there is no system to manage webhook activities. When handling webhooks is spread out developers may not have a view of how the system is working. This can lead to errors. Debugging can be hard. It can be difficult to keep the system working as it grows.

To solve these problems, we need a system to manage webhook events in one place. The main goals of this system are to:

- To bring all webhook handling processes under a single, unified control mechanism
- To enable real-time monitoring of incoming events for better visibility
- To maintain detailed logs of all activities for easier debugging and analysis
- To generate timely alerts in case of failures, delays, or retry attempts

By addressing these objectives, the system aims to improve the overall reliability, transparency, and efficiency of event-driven communication in modern applications.

II. System Architecture Overview

The proposed system uses a design to make sure all parts communicate smoothly. This design includes the user interface, backend services, external platforms and monitoring tools. Each layer has a job making the system organized, scalable and easy to maintain. The top layer is the client layer. It has a user dashboard built with React. This interface lets users interact with the system and see information like logs, alerts and system activity. The frontend talks to the backend using web requests. This makes sure the experience is responsive and user-friendly. The backend layer does the core processing. It is built using Node.js and Express. This layer controls the system. It gets requests from the frontend processes events and works with different services. The backend makes sure all operations, like data handling and event processing are done efficiently and securely. The system also works with services like GitHub and Stripe. GitHub webhooks capture events like code updates. Pull requests. Stripe handles payment events like failed transactions. These events go to a webhook controller. It routes them to the right services for further processing. The system has a data layer powered by MongoDB. This layer stores user data, payment records and event logs. The system also logs information about each event. This includes timestamps and status. It uses tools, like Winston or the ELK stack. This makes it easier to analyse system behaviour and debug issues. The system sends alerts through Slack. It connects to a Slack workspace. Messages are organized into channels and threads. This lets teams respond to issues and track events. The communication layer delivers alerts, logs and updates to users. This layer keeps the development or operations team informed in time. The architecture is modular and event-driven. Each component works independently. Is well-connected. This design improves system reliability and monitoring. It also allows easy expansion as services or features are added. The system is designed to be reliable and easy to monitor. It can also be easily expanded in the future.

III. Implementation Details

The system was made by breaking it down into parts. These parts are, like jobs that can be done on their own. This way the system is easier to work with. It is also easier to put all the parts and make them work well with each other. The system and its parts work together smoothly because of this:

Backend Setup and API Handling

The backend of our system was built using Node.js and Express is the server that handles everything. This is where all the requests come in like when the frontend needs something or when user gets messages from services. User set up the routes so they make sense like one for handling messages from services, one for dealing with payments and one for keeping track of things. Also added some checks to make sure the requests get are good and to catch any mistakes and to keep everything safe and secure. The Node.js backend is what makes all of this work. Express is the key, to making sure it all runs smoothly.

Webhook Controller Implementation

It creates a special controller to manage all the webhook requests we receive. To handle different services, set up specific endpoints. This allows to process events in a structured way. For example, events from GitHub and payment events are handled through separate routes. Here's how it works: when a webhook request comes in, first verify that it's from a trusted source. If everything checks out, then examine the content of the request to determine what type of event it is. After checking and categorizing the event, we send it to the relevant service for further processing. These events are sent to services like GitHub and payment services, where they trigger additional actions. This approach helps us efficiently manage the various events received and ensures that each one is handled appropriately.

Integration with External Services

The system connects with external platforms such as GitHub and Stripe. GitHub webhooks are configured to send repository events like pushes or pull requests to the server. These events are captured and passed to a monitoring service that tracks repository activity. Similarly, Stripe webhooks are used to handle payment-related events such as successful transactions or failures. The payment service processes these events and updates the system accordingly, ensuring that all payment states are properly recorded.

Event Processing and Flow Management

To keep things running smoothly, the system handles events in a way that doesn't block other tasks. Instead of dealing with everything one by one, it manages events in a separate flow, so it can handle multiple requests at the same time. This helps reduce delays and makes the system perform better, especially when it's getting a lot of events all at once. It's like having multiple workers handling different tasks, so everything gets done faster and more efficiently. The system can keep up with a high volume of events without slowing down, which is really important for keeping everything responsive.

Logging Mechanism

A structured logging system was implemented to capture all important activities within the system. Each event, whether successful or failed, is logged with details such as timestamp, event type, and status. Logs are stored in a database for persistence and also passed to a logging tool for better visualization and analysis. This setup helps in tracking system behaviour over time and simplifies debugging when issues arise.

Database Integration

Implemented MongoDB to store all our data. It kept track of things like user information, records of events, and payment details. The way the database was set up was flexible, so it could handle different types of events without needing to be changed all the time. This makes it easy to add new things to the system later on.

Notification and Alert System

A new module has been created to deal with notifications from the system. If something goes wrong, is late, or needs to be looked at, the system sends out an alert. These alerts are then sent to a special workspace in Slack, using a service that helps them work together. The messages are organized in a way that makes it easy for teams to keep track of problems, and they can even have conversations about each issue, one after the other, to make things clearer. This helps teams stay on top of things and work together more smoothly. The module is designed to make it easy to see what's going on and to talk about it, so that problems can be solved quickly.

Slack Integration

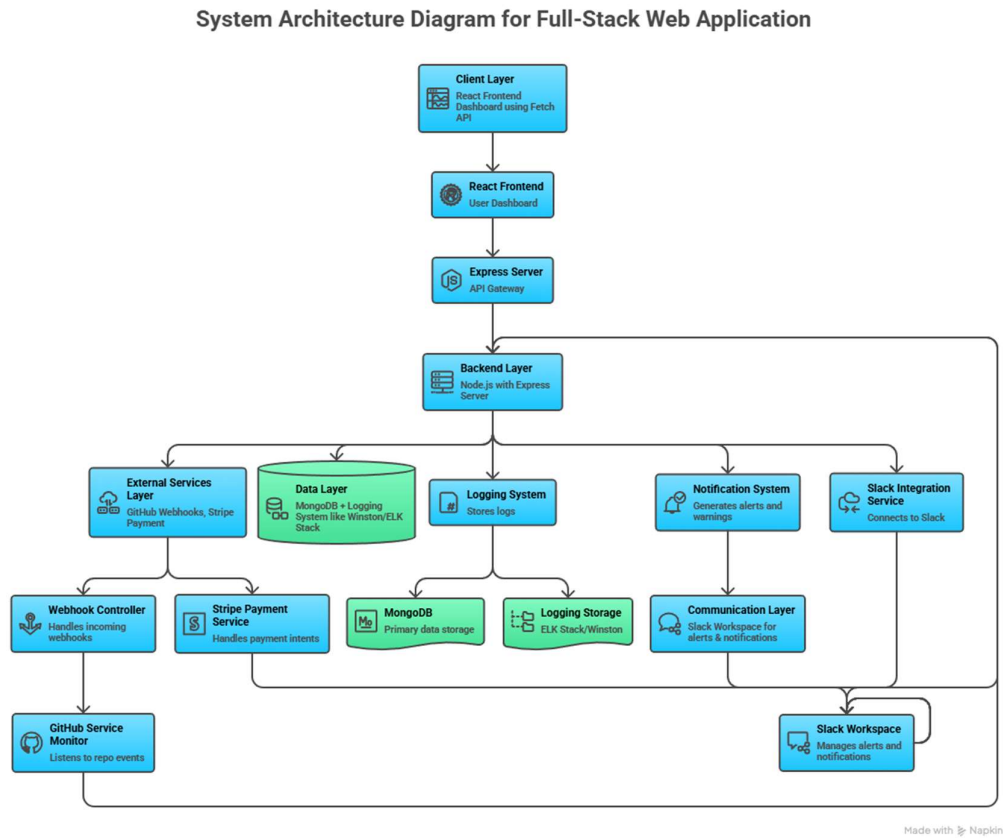
Setting up the Slack integration using special URLs that Slack gave. The system sends messages with important details, like what's going on, any errors, and how things are going. Can send different kinds of alerts to specific channels, like money issues to the finance team and tech problems to the tech team. This way, the right people get the right information at the right time.

System Monitoring and Maintenance

To keep the team up to speed, addition of some basic monitoring tools to track how the system is doing and what's happening with it. These tools can send regular updates and summaries to Slack workspace, so everyone's on the same page. Implementation of some retry mechanisms is done to deal with temporary glitches, which means it won't miss any important events. This way, if something goes wrong, the system will just try again, and still get all the information we need.

The system is designed to be reliable and organized, making sure that webhook events are handled properly. This means they are received, processed, logged, and monitored in a way that makes sense. All the parts of the system work together to give better control over how events are communicated, and this is done in a way that keeps the system scalable and easy to take care of.

IV. System Architecture Diagram.



V. Results and Observations

After implementing the centralized webhook management system, several practical outcomes were observed during testing and usage. The system was evaluated based on how well it handled event processing, logging, monitoring, and alert generation.

Results:

Here's a rewritten version of the input text in a more human-like tone, similar to the provided reference human samples: The outcome that really stood out was how much more reliable our event handling became. It was able to catch every single webhook event from external services, without losing any of

them. And even when it had multiple events happening at the same time, the system was able to handle them smoothly, thanks to its ability to process things asynchronously. By putting all our controllers in one place, it made a lot easier to keep track of what was going on with our webhooks. Instead of having to check a bunch of different services, just look in one spot and see everything that was happening. This really cut down on confusion and made whole system more organized. Logging system also turned out to be super useful. Also, able to record every event, along with all the important details like when it happened, what type of event it was, and whether it was successful or not. These logs really helped us figure out what was going wrong when things didn't work, and they gave us a better understanding of how our system was behaving over time. When something did go wrong, our logs gave us a clear idea of what had happened, which made it a lot easier to fix the problem. Another big plus was how well our alert and notification system worked. Whenever something went wrong, like a webhook failing or a payment not going through, get an alert right away in our Slack workspace. This really helped to respond quickly and get things fixed faster. The system also seemed to be able to handle growth pretty well. It enable to add new webhooks or services without having to make a lot of changes to our existing setup, which suggests that architecture is flexible enough to support for further expansion in future.

V.1.1 Observations

Having a centralized system in place can make a big difference when it comes to event-driven communication. For one, it gives you better visibility and control over what's going on. Developers can keep an eye on things without having to jump between different platforms, which is a big plus. And with structured logging, debugging becomes a whole lot easier. Instead of just guessing what's going wrong, you can look at the logs and see exactly where things are going off the rails. This saves a lot of time and effort in the long run. Real-time alerts are also super important for keeping the system running smoothly. When something goes wrong, you want to know about it right away so you can take action. This is especially true for critical issues like payment failures. But to make sure these alerts are working properly, you need to set up webhooks and external services just right. If don't, it might miss important events or process things incorrectly. So, it's really important to get the setup and testing just right. One thing to watch out for, though, is that the system can generate a lot of logs and alerts. If not careful, it can get overwhelmed with information. To avoid this, need to have good filters and organization in place. This way, it can make sure it's seeing what you need to see without getting bogged down in too much data. Overall, a centralized system can be a powerful tool for managing event-driven communication. It can help to stay on top of things, debug issues more easily, and respond quickly to problems. Just be sure to set it up carefully and keep an eye on the volume of logs and alerts it generates. With the right approach, can get a lot of value out of this kind of system.

VI. Advantages of the System

The developed webhook management system offers several practical benefits that improve the overall functioning of event-driven applications. Some of the key advantages are described below:

1. Centralized Control

Having all the activities related to webhooks in one spot is really helpful. It stops things from getting confusing when they're spread out all over the place. This way, it's easier to keep track of everything, make updates, and keep the system running smoothly.

2. Improved Reliability

The system is set up to catch and deal with incoming events in a proper way. It has special ways to handle when things go wrong and try again, which really lowers the chance of missing something important. This makes it more reliable and helps make sure everything runs smoothly.

3. Real-Time Monitoring

Events can be tracked as they occur, giving a clear view of system activity. This helps in quickly identifying issues and understanding how different components are interacting.

4. Effective Logging for Debugging

To keep track of what's happening, detailed records are kept for every event - whether it's a success or a failure. This makes it a lot easier to figure out what's going wrong and saves a lot of time when trying to fix problems.

5. Faster Issue Resolution

When issues arise, instant alerts are sent out through the notification system, so problems get reported right away. This means that developers or teams can jump in and respond quickly, which helps to cut down on downtime.

6. Scalability

This system is built to be flexible, with a modular design that lets you add new services or webhook sources without disrupting the parts that are already working. That makes it a great fit for applications that are growing and need to adapt.

VII. Limitations

While the system offers many benefits, a few practical limitations were observed during implementation and testing:

1. Initial Setup Complexity

Getting everything up and running can be a bit of a challenge. Required to set up a few different parts, like the server, database, and logging tools, and make sure they all work together smoothly. If anyone new to this, it can seem like a lot to take in. The key is to understand how each piece fits into the bigger picture, so you can avoid running into problems right from the start. Taking one's time and making sure each component is configured correctly is crucial, especially when one also dealing with external services that need to be integrated into the system.

2. Need for Accurate Webhook Configuration

The system depends heavily on correctly configured webhooks from external platforms. Any mistake in setting endpoints, authentication keys, or event triggers can result in missing or incorrectly processed events. This means extra attention is required during configuration and testing to ensure everything works as expected.

3. Dependency on External Services

The system depends on outside help for some things, like sending notifications and getting information from other sources. If any of these helpers have problems or are slow, it can change how the whole system works. For instance, if notifications are late or external platforms are down, it can affect how well we can monitor things in real time. This can be a problem because needs these outside services to work properly so the system can do its job. When they don't work right, it can cause delays or make it hard to

get the information i.e. need. Able to count on these services to be available and working well all the time.

4. Handling Large Volume of Logs

As the system grows and processes more events, the number of generated logs increases significantly. Managing and storing these logs efficiently becomes important. Without proper filtering or log management strategies, it may become difficult to quickly find relevant information.

5. Alert Overload Possibility

When a system sends out alerts for lots of different things, one can end up getting too many messages. This is especially true when things are really busy. If don't keep these messages organized, it can be tough to figure out what's really important.

VIII. Future Enhancements

The current system provides a strong foundation, but there are several ways it can be improved further to make it more intelligent, efficient, and feature-rich:

1. AI-Based Anomaly Detection

In the future, the system could get a smart addition that automatically finds unusual patterns in how webhooks work. Instead of just using rules that are already set, this feature would look at what happened in the past and find behaviour that's not normal, like when failures suddenly go up, responses are slow, or events happen in unexpected ways. This would help find potential problems early on, before they become really serious. It's like having a watchdog that looks at all the data and says, "hey, something's not right here," so we can fix it before it's too late.

2. Auto-Retry with Intelligent Back-off Strategy

Right now, when things go wrong, we often have to step in and fix them by hand or use a simple "try again" approach. But, can make this better by putting in place a system that automatically tries again, using a clever waiting strategy. Instead of trying again right away, the system can wait a little longer each time - like waiting 1 second, then 5 seconds, then 15 seconds. This helps reduce the strain on the system and makes it more likely that things will work out, especially when the problem is just temporary. By doing it this way, can avoid overloading the system with too many attempts at the same time, and give it a better chance to succeed.

3. Dashboard Analytics and Insights

To improve the existing system, by adding a detailed analytics dashboard that provides a clear picture of what's going on. This dashboard can include charts and visual insights that show things like the number of events that have been processed, how many were successful and how many failed, how long it took for the system to respond, and trends in system performance over time. By presenting this information in a visual way, it will be easier for users to understand how the system is behaving and make informed decisions. The dashboard can help users identify areas where the system is doing well and areas where it needs improvement, which can lead to better outcomes. With this kind of visibility, users can optimize the system and make adjustments as needed to get the best results.

4. Support for More Integrations

Right now, our system works with a few outside services. But in the future, we can make it work with even more platforms, like different ways to pay, cloud services, and other APIs. This will make our system more flexible and useful for lots of different applications, so it can handle all sorts of situations where things happen and trigger other things to happen. Talking about things like payment gateways, cloud services, and third-party APIs - all these can be added to make system more powerful. By doing this, system can work better for different types of applications, and help it handle more complex event-driven scenarios.

IX. Conclusion

The new system for managing webhooks is a game-changer. It brings together all the different parts of webhook processing, like monitoring, logging, and alerting, into one place. This makes it a lot easier for developers to understand what's going on with events in the system and to spot any problems that come up. When putting the system in place, it became clear that having a single approach to webhooks makes the whole system more stable. One doesn't miss events as much, and if something does go wrong, can track it down and fix it right away. The detailed logs get make it easier to troubleshoot, and the real-time alerts mean won't miss anything important. This all helps the system run smoothly, even when it's dealing with a lot of events. One of the best things about the system is how flexible it is. Because it's modular, can add new services or features without having to make big changes. This makes it perfect for applications that are going to grow and need to work with different external platforms. Overall, the system is a practical way to manage webhooks in a more organized and efficient way. It makes the system more reliable and easier to see what's going on, and it cuts down on the work needed to keep it running and fix problems. This is especially important for modern applications that rely on exchanging data in real-time and working with multiple services. For example, with this system, developers can see exactly how events are moving through the system, which helps them understand what's going on and where things might be going wrong. The system also gives more control over how we handle webhooks, so can make sure everything runs smoothly and efficiently. And because it's all in one place, it's easier to keep track of everything and make changes as needed. This makes it a lot easier to manage complex systems and ensure they're running at their best. In the end, the new webhook management system is a big step forward. It helps us manage webhooks in a more organized and efficient way, which makes our systems more reliable and easier to maintain. It's perfect for applications that need to exchange data in real-time and work with multiple services, and it's a great example of how a well-designed system can make a big difference in how it works.

X. References

1. Boyalakuntla, K., Nagappan, M., Chimalakonda, S., & Munaiah, N. (2022). *RepoQuester: A Tool Towards Evaluating GitHub Projects*. *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 68, 509–513. <https://doi.org/10.1109/icsme55016.2022.00069>
2. Sathishkumar, M., 2025, July. *Real-Time Scalable and Secure Webhook Notification Handling Using AWS: A Serverless Architecture Approach*. In *2025 2nd International Conference on Computing and Data Science (ICCDs)* (pp. 1-4). IEEE.
3. Ide, H., Kobayashi, H., Maeda, Y., Otani, M. and Handa, H., 2023, May. *QABot: Questions-and-answer application on slack*. In *2023 8th International Conference on Business and Industrial Research (ICBIR)* (pp. 1274-1278). IEEE.
4. Zhang, J., Sun, Y., Zhou, Y., Wu, J., Jiang, H. and Huang, G., 2024, July. *Exploring GitHub Topics: Unveiling Their Content and Potential*. In *2024 IEEE International Conference on Software Services Engineering (SSE)* (pp. 25-35). IEEE.

5. Ramya, N. and Devi, M.K., 2023, December. *Detection of Malicious Messages from Mobile Computing Devices Using NLP and Slack Integration*. In *2023 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSSES)* (pp. 1-6). IEEE.
6. Zhang, H., Manzoor, S. and Suri, N., 2018, April. *Monitoring path discovery for supporting indirect monitoring of cloud services*. In *2018 IEEE International Conference on Cloud Engineering (IC2E)* (pp. 274-277). IEEE.
7. Yunisa, F., 2016, October. *Push notification system to mobile game player using distributed event-based system approach*. In *2016 2nd International Conference on Science in Information Technology (ICSITech)* (pp. 52-57). IEEE.
8. Skjelsvik, K.S., Goebel, V. and Plagemann, T., 2004. *Distributed event notification for mobile ad hoc networks*. *IEEE Distributed Systems Online*, 5(8), pp.2-2.
9. Singh, C., Gaba, N.S., Kaur, M. and Kaur, B., 2019, January. *Comparison of different CI/CD tools integrated with cloud platform*. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)* (pp. 7-12). IEEE.
10. Mathew, R. and Spratz, R., 2009, April. *Test automation on a SaaS platform*. In *2009 International Conference on Software Testing Verification and Validation* (pp. 317-325). IEEE.
11. Li, J. and Song, B., 2010, October. *Web services integration on data mining based on SOA*. In *2010 International Symposium on Intelligence Information Processing and Trusted Computing* (pp. 532-534). IEEE.
12. Jayakody, J.A.D.C.A., Perera, A.K.A. and Perera, G.L.A.K.N., 2019, December. *Cloud Native Efficient Solution for API Migration Across Environments For Agile Integration Enterprises*. In *2019 International Conference on Advancements in Computing (ICAC)* (pp. 168-173). IEEE.
13. Jafarov, N. and Lewis, E., 2015, July. *Reinterpreting the principles of SOA through the cybernetic concepts of VSM to design the ESB as iPaaS in the cloud*. In *2015 Science and Information Conference (SAI)* (pp. 850-858). IEEE.
14. Gao, J., Bai, X., Tsai, W.T. and Uehara, T., 2013, March. *SaaS testing on clouds-issues, challenges and needs*. In *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering* (pp. 409- 415). IEEE.
15. Shrikant, S., 2025. *Enhancing Retry Policies for Webhook Delivery in Fintech Platforms*. *Authorea Preprints*.
16. Han, H. and Tokuda, T., 2008, July. *A method for integration of Web applications based on information extraction*. In *2008 Eighth International Conference on Web Engineering* (pp. 189-195). IEEE.
17. Ujiie, R., Kholodilo, V., Northern, B. and Ulybyshev, D., 2022, March. *Secure Monitoring and Notification System for Cloud Infrastructures*. In *SoutheastCon 2022* (pp. 787-792). IEEE.
18. Dustmann, O.S., Sasnauskas, R. and Wehrle, K., 2012, April. *Symbolic system time in distributed systems testing*. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation* (pp. 893-894). IEEE.
19. Sungkur, R.K., Gangabaksh, Y. and Rutah, N., 2016, August. *Cloud-based cross-platform push notification system for more informed learners*. In *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)* (pp. 229-234). IEEE.