WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Automated Web Service Renewal Manager: Automating Domain and Hosting Renewal Tracking for Web Service Providers

Preshita Tadas[1],Swayam Gagapalliwar[2],Yash Pidurkar[3],Omkar Gumdelwar[4],Harshil Sheladia[5]

Department of Robotics & Artificial Intelligence, Priyadarshini College of Engineering, Nagpur, India

ABSTRACT—

Every website depends on two things renewing on time — its domain registration and its hosting plan — and most organizations still track both by hand, in a spreadsheet or a calendar reminder. That works fine for one or two sites. It stops working once an agency is responsible for dozens or hundreds of client sites, because a missed renewal date doesn't just cause an inconvenience, it takes a client's website and email offline with no warning. This paper describes the Automated Web Service Renewal Manager, a system we designed to move renewal tracking out of spreadsheets and into a monitored, notification-driven pipeline. It centralizes client and service data in a relational database, checks expiry dates daily through a scheduler, and sends a staged sequence of reminders (90, 30, 7, and 1 day before expiry) instead of one last-minute warning. We back this design with real numbers on why it matters: industry downtime surveys put the cost of an outage at $100 to over $300,000 an hour depending on business size, and registry-level data shows that staged reminders are directly responsible for the majority of on-time renewals rather than customers renewing unprompted. We describe the architecture, data model, development plan, and a simple worked estimate of the staff time this kind of automation is expected to save at moderate scale.

Keywords—*web service management, domain renewal, hosting renewal, automation, notification systems, IT service management (ITSM)*

INTRODUCTION

Every site on the internet runs on two things that need renewing: a domain name and a hosting plan. For someone managing one or two personal sites, remembering these dates is easy. The problem shows up once an organization is responsible for dozens or hundreds of client sites at once — a web agency, a freelancer with a growing client base, or a managed-service provider. At that point, tracking renewal dates by memory, spreadsheet, or calendar reminder stops being a minor inconvenience and starts being a real operational risk.

The cost of getting this wrong is immediate and visible, and it isn't small. Industry downtime surveys consistently put the cost of an outage well above what most teams assume: ITIC's 2024 survey of over 1,000 firms found that a single hour of downtime exceeds $300,000 for more than 90% of mid-size and large enterprises [11], and while a small client site obviously isn't losing six figures an hour, small-business estimates still typically land between $100 and $5,000 per hour once lost sales, support load, and reputation cost are counted [11]. A domain that lapses stops resolving, email tied to that domain stops working, and the client notices before the agency does — and none of it needed to happen, since the date was known in advance.

What is more frustrating is that the fix for this is not complicated. Registry-level data backs this up directly: Nominet's Domain Name Industry Report found that 60% of renewing customers acted because of a reminder from their registrar, against only about 7% who renewed after a reminder from the registry itself, and a full 98% of surveyed customers recalled receiving a renewal reminder at all [12]. In other words, reminders are already doing most of the work of keeping domains alive — the actual gap is that most small organizations aren't running a reminder system reliable enough to make use of that.

This paper describes a system we built to close that gap: the Automated Web Service Renewal Manager. It centralizes renewal data, checks it automatically every day, and sends a staged sequence of reminders well before anything actually expires. We settled on four goals early on and kept coming back to them whenever a design decision got complicated:

(1) put all renewal information in one place instead of scattered spreadsheets; (2) monitor expiry dates without needing a person to check them; (3) notify at intervals that actually give someone time to act, not just a last-minute warning; and (4) keep the system usable at a scale of hundreds of tracked services, not just a handful. The fourth point mattered more than it might sound like it should — a design that only works for ten services isn't really solving the problem this paper is about.

RELATED WORK

Web Engineering as a discipline was framed early on by Deshpande et al., who argued that building maintainable web systems requires the same planning discipline as any other software lifecycle, not an ad-hoc approach [1]. Taylor et al. looked specifically at maintenance issues — documentation, coding standards, structured data handling — and found these to be strong predictors of a system's long-term reliability [2],

which is directly relevant here since a renewal tracker is exactly the kind of long-lived system that has to keep working correctly for years without much attention.

The W3C's own working note on web service lifecycles frames a service as moving through distinct phases from creation to eventual retirement [3], and renewal is really just the recurring transition point in that lifecycle — the moment a service either continues or lapses. Di Lucca et al.'s work on reverse engineering web applications is a useful reminder that most real systems accumulate structure informally over time [4], which is part of why we wanted a system with an explicit, well-defined data model from the start rather than one that grows organically. Jiang et al.'s pattern-based approach to subscription management in SaaS systems maps closely onto our own design problem, since a domain or hosting renewal is functionally a subscription with a hard expiry date [5], and Walther et al.'s study of subscription renewal intention in cloud enterprise systems gives a behavioral angle on why timely, well-placed reminders actually change renewal outcomes rather than just documenting them [6].

On the IT service management side, Iden and Eikebrokk's systematic review of ITSM implementation lays out the general case for treating IT processes as manageable, measurable workflows [7], and MacLean and Titah's more recent review of 126 ITSM studies found that the degree of automation in a service process correlates directly with improvements in service quality and reduced errors [8] — which is essentially the hypothesis this project tests at a small scale. Keller and Ludwig's WSLA framework for specifying and monitoring service-level agreements [9] and Hollenbeck's RFC on the Extensible Provisioning Protocol for domain name mapping [10] round out the more technical side of this picture, since domain lifecycle events are, at the protocol level, already fairly well standardized — the gap is not in the protocols but in how consistently organizations actually watch them.

PROBLEM ANALYSIS AND MOTIVATION

Looking at how renewal tracking is actually done in practice, a few problems keep showing up regardless of organization size:

- Manual tracking: most teams rely on spreadsheets, calendar entries, or email folders, all of which require someone to remember to check them.
- High error rate: manual systems are vulnerable to missed updates and simple forgetfulness, and the error rate gets worse, not better, as the number of tracked services grows.
- Poor visibility: when renewal data is spread across multiple spreadsheets or inboxes, there is no single place to see what is coming up.
- Scalability limits: manual tracking does not scale linearly — doubling the number of client sites more than doubles the effort needed to track them reliably.
- Reactive discovery: problems are usually found after the fact, when a site is already offline, instead of being caught with enough lead time to act.

The downstream impact of a missed renewal goes beyond the website itself: a lapsed domain typically also breaks any email hosted on it, opens a window for a competitor or squatter to register the expired name, damages the client relationship, and — as the downtime figures in Section I show — carries a real financial cost even for a small client site. None of these outcomes are surprising once a renewal is missed; they are consequences that were entirely avoidable if the date had been tracked properly.

PROPOSED SYSTEM ARCHITECTURE

We went back and forth on whether a three-tier architecture was overkill for a project this size, since a simpler script-plus-database setup would have gotten a basic version running faster. We settled on three tiers anyway, mainly because it keeps the parts that change independently the interface, the business rules, and the stored data — from being tangled together, which matters once notification logic or the data model needs to change later without touching everything else. For a system meant to run unattended for months at a time, that separation felt worth the extra setup.

A. Components

- Presentation layer: a web dashboard where a user can see active services, upcoming renewals, renewal history, and notification logs.
- Business logic layer: backend services handling authentication, renewal-date calculations, notification scheduling, and (eventually) payment or provider-API integrations.
- Data layer: a relational database storing client records, service details, renewal dates, and notification history.
- Email service: an SMTP-based notification engine responsible for generating and sending reminder emails.
- Scheduler: a background process that checks renewal dates daily and triggers notifications when a date matches the notification schedule.

SYSTEM DESIGN AND COMPONENTS

A. Data Model

The core entities are relatively simple: Clients (contact details, billing information, communication preferences), Services (domain names, hosting provider, registration and expiry dates, renewal cost, current status), Notifications (a full audit trail of what was sent, to

whom, and whether it was delivered), and a Renewal Schedule table tracking which notification milestones — 90, 30, 7, and 1 day before expiry
— have already fired for a given service.

B. Notification Strategy

Rather than a single last-minute email, the system sends a staged sequence of reminders, which lines up with the Nominet finding cited earlier — reminders are what actually drive timely renewals, so it is worth sending more than one [12]. The default schedule we used is:

- 90 days before expiry — an initial heads-up notification.
- 30 days before expiry — a primary reminder that treats the renewal as something to act on soon.
- 7 days before expiry — an urgent reminder.
- 1 day before expiry — a final, critical alert.

METHODOLOGY AND APPROACH

We followed an iterative, modular build process so each layer could be tested as it was built rather than all at once at the end. With four of us on the project, splitting work by layer also made it easier to divide tasks without three people editing the same file — one pair naturally gravitated toward the backend and scheduler, the other toward the frontend and email templates.

- Phase 1 (Weeks 1–2) — Requirements and design: analyzing user needs and writing system design documentation.
- Phase 2 (Weeks 3–4) — Database implementation: building and securing the data persistence layer.
- Phase 3 (Weeks 5–7) — Frontend development: building the dashboard for service management and monitoring.
- Phase 4 (Weeks 8–10) — Backend services: implementing scheduling, notification logic, and integrations.
- Phase 5 (Weeks 11–12) — Email integration: setting up the SMTP service, templates, and delivery tracking.
- Phase 6 (Weeks 13–15) — Testing: unit tests, integration tests, and user acceptance testing.
- Phase 7 (Weeks 16–17) — Deployment: production rollout, user training, and documentation.

For the technology stack, we leaned toward a React or Vue.js frontend, a Node.js/Express or Django backend exposing a RESTful API, PostgreSQL or MySQL for storage, a cron-based or dedicated task scheduler (e.g., APScheduler or Bull) for the daily renewal check, SMTP with retry logic for email delivery, and a standard cloud platform (AWS, Azure, or GCP) for hosting so the system can scale without a re-architecture. Node.js and Express were the more likely pick within the team simply because more of us had already used them in earlier coursework, and picking a stack nobody has to learn from scratch matters when the whole build is compressed into a 17-week window.

IMPLEMENTATION STRATEGY

A. System Workflow

End to end, the system works as: a user registers and sets up an organizational profile; they enter service details including domain/hosting expiry dates; that data is stored centrally; a background scheduler checks all tracked expiry dates daily; when a date matches a notification milestone, the system generates and sends the corresponding email; every notification is logged for audit purposes; and the user can mark a service as renewed, which resets its tracking cycle.

B. Security Considerations

Because the system holds client data, domain credentials, and potentially payment information, we treated security as a first-class requirement rather than an afterthought:

- All data in transit encrypted with TLS/HTTPS.
- Database encryption at rest, with secure backup procedures.
- Role-based access control to limit what each user type can see or change.
- Periodic security audits and vulnerability checks.
- Full audit logging for compliance and dispute resolution.

EXPECTED BENEFITS AND A WORKED ESTIMATE

The main operational change this system is meant to produce is straightforward: renewal tracking stops depending on someone remembering to check a spreadsheet. To put a rough number on that instead of just asserting it, consider a small agency managing 200 client services manually. If checking, updating, and following up on a single service's renewal status takes even 10 minutes a month per service — a conservative estimate once you count opening the spreadsheet, cross-checking dates, and sending a manual reminder — that is roughly 33 hours of staff time every month spent on a task that contributes nothing beyond keeping dates from slipping through the cracks. Automating the checking and the first-line notification does not eliminate every minute of that, since renewals still need to be actioned, but it removes the checking itself, which

is most of it. This is our own illustrative estimate, not a measured result — the real number would depend on the agency's actual client count and current process, but it gives a sense of scale.

Combined with the downtime figures from Section I, the business case follows fairly directly: even at the low end of the small-business downtime range (~$100/hour) [11], a handful of missed renewals a year costs more in lost client trust and emergency fixes than the system takes to build and run. A system that reliably notifies ahead of a deadline also comes across as more professional than one that reacts to a problem after it has already happened.

CONCLUSION AND FUTURE WORK

The Automated Web Service Renewal Manager targets a problem that is easy to underestimate precisely because it looks simple: tracking a handful of dates. In practice, that simplicity breaks down fast once the number of tracked services grows, and the consequences of getting it wrong are immediate, visible to the client, and — as the downtime and reminder-effectiveness data in this paper show — measurable in real financial and behavioral terms, not just theoretical ones. Nothing about the fix is clever. It just has to actually run every day without anyone remembering to check it, which is precisely what a spreadsheet can't promise. By centralizing renewal data, monitoring it automatically, and notifying at well-timed intervals, the system removes a preventable failure mode rather than just making it easier to manage manually.

The same pattern — centralized data, automated monitoring, staged notification — is not really specific to domain renewals, and could reasonably be applied to software license renewals, contract reviews, or general subscription management. A few directions we think are worth pursuing next: integrating payment gateways so renewals can be completed automatically rather than just flagged; connecting directly to registrar and hosting-provider APIs for real-time status verification instead of relying on manually entered dates; adding a simple predictive layer that flags services at higher risk of being missed based on past renewal behavior; and building a mobile app so notifications and quick actions are not limited to email.

ACKNOWLEDGMENT

We thank Prof. Preshita Tadas, Assistant Professor, Department of Robotics & Artificial Intelligence, Priyadarshini College of Engineering, Nagpur, for her guidance and feedback throughout this project during the 2026–27 academic session.

REFERENCES

- Y. Deshpande, S. Murugesan, and A. Ginige, "Web Engineering," IEEE Software, vol. 20, no. 5, pp. 40–45, 2003.
- M. J. Taylor, J. McWilliam, J. Sheehan, and A. Mulhaney, "Maintenance issues in the web site development process," in Proc. 34th Hawaii Int. Conf. System Sciences, 2002.
- H. He, M. Potts, and I. Sedukhin, "Web Service Management: Service Life Cycle," W3C Working Group Note, 2004.
- G. A. Di Lucca, A. R. Fasolino, and P. Tramontana, "Reverse Engineering Web Applications," IEEE Trans. Software Eng., vol. 30, no. 8, pp. 540–554, 2004.
- Z. Jiang, W. Sun, K. Tang, J. L. Snowdon, and X. Zhang, "A Pattern-Based Design Approach for Subscription Management," in Service-Oriented Computing, Springer, 2009, pp. 500–515.
- S. Walther, S. Sarker, D. Sedera, and T. Eymann, "Exploring Subscription Renewal Intention of Operational Cloud Enterprise Systems," J. Strategic Inf. Syst., vol. 22, no. 4, pp. 273–290, 2013.
- J. Iden and T. R. Eikebrokk, "Implementing IT Service Management: A Systematic Literature Review," J. Inf. Syst. Manage., vol. 3, no. 1, pp. 1–13, 2013.
- MacLean and R. Titah, "Implementation and Impacts of IT Service Management in the IT Function: A Systematic Review," Inf. Syst. J., vol. 33, no. 1, pp. 123–156, 2023.
- Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," IBM Research, 2003.
- S. Hollenbeck, "Extensible Provisioning Protocol (EPP) Domain Name Mapping," RFC 5731, 2009.
- L. DiDio, "ITIC 2024 Hourly Cost of Downtime Survey," Information Technology Intelligence Consulting, 2024. [Online]. Available: https://calyptix.com/wp-content/uploads/ITIC-2024-Hourly-Cost-of-Downtime- Survey-Results-Part-2.pdf
- Nominet, "Domain Name Industry Report," 2009. [Online]. Available: https://news.namebay.com/?p=2296