SMART PROPERTY MAINTENANCE MANAGEMENT SYSTEM

Naveenraj M[1], Bhavyasri J[2], Gopinath V[3], Pooja K[4], Priyadharshini K[5]

[1] Assistant professor, Department of Computer Science and Engineering (IoT and Cyber Security including Blockchain Technology), SNS College of Engineering, Coimbatore, TamilNadu, India.

[2] B.E. (III Year student), Department of Computer Science and Engineering (IoT and Cyber Security including Blockchain Technology), SNS College of Engineering, Coimbatore, TamilNadu, India.

[3] B.E. (III Year student), Department of Computer Science and Engineering (IoT and Cyber Security including Blockchain Technology), SNS College of Engineering, Coimbatore, TamilNadu, India.

[4] B.E. (III Year student), Department of Computer Science and Engineering (IoT and Cyber Security including Blockchain Technology), SNS College of Engineering, Coimbatore, TamilNadu, India.

[5] B.E. (III Year student), Department of Computer Science and Engineering (IoT and Cyber Security including Blockchain Technology), SNS College of Engineering, Coimbatore, TamilNadu, India.

[1]naveenraj.m.iot@snsce.ac.in, [2] bjaisankaran@gmail.com, [3] gopinath.v.official.01@gmail.com, [4] poojakumar2726@gmail.com, [5]priyadharshinik687@gmail.com

***Abstract*—*Running a residential property means juggling communication between four groups of people: administrators, property managers, tenants, and maintenance staff. In practice, this coordination often breaks down. Tenants end up reporting maintenance problems through whatever channel is easiest—a phone call, a WhatsApp message, sometimes just mentioning it in passing—and managers struggle to keep track of what has been reported, what is being worked on, and what has fallen through the cracks. This paper describes a centralized web platform built around four defined user roles (Admin, Manager, Staff, Tenant) that brings the entire service-request lifecycle—from initial complaint through to verified resolution—into a single digital workflow. The system includes an automated staff assignment mechanism that scores candidates based on their availability, current workload, skill match, and proximity to the request location. A hybrid model lets routine requests go through fully automated assignment while routing critical ones through a manager review step. Staff members submit completion evidence, and closure of a request depends on tenant confirmation, not just the staff member saying the job is done. We evaluated the designed framework in simulated scenarios of varying scale (25 to 500+ residential units), which indicated a substantial reduction in routine manager coordination steps and 85% to 92% agreement between automated staff selection and expert judgment; under an assumed rate of 10% to 14% of staff-completed requests being flagged by tenants, the verification step would send those requests back for rework. These results come from simulated scenarios with assumed parameters, not live deployment data, and staff scoring and tenant verification are design components evaluated in simulation only. We believe the framework offers a workable foundation for modernizing how residential properties handle day-to-day maintenance operations.*

Keywords*—*Property Management System, Automated Staff Allocation, Role-Based Access Control, Tenant Service Request Processing, Smart Building Management, Closed-Loop Verification

I. Introduction

Anyone who has managed a residential complex knows that the job is basically one long coordination exercise. Property administrators, on-site managers, tenants, and maintenance staff all need to stay in sync—and as buildings get bigger and cities add more housing stock, that coordination only gets harder. A typical manager juggles tenant complaints, repair scheduling, staff availability tracking, rent collection, and record-keeping, sometimes across hundreds of units at once.

How do tenants usually report a broken pipe or a faulty light switch? They call. They text on WhatsApp. They send an email. Sometimes they just mention it when they see the manager in the hallway. The problem is obvious: when complaints arrive through five different channels, there is no single place where you can see everything that has been reported. Requests get lost. They get duplicated. They sit unaddressed for days because the manager didn't realize something was still pending.

The manager ends up being the bottleneck. Every request flows through them: figure out what's wrong, decide which kind of staff member is needed, check who's free, call them up, confirm a time, follow up on whether the work actually got done, update the tenant, and close the ticket. That's easily eight to twelve steps per request, all done manually. For a property with a

hundred units and thirty or forty requests a month, the manager spends most of their day just playing telephone between tenants and staff members instead of focusing on bigger operational decisions.

Picking the right staff member is its own headache. Without a structured way to check who's available, how many jobs each staff member is already handling, or how well they performed last time, managers fall back on gut feeling or whoever they happen to remember. The result? Some staff members end up overloaded while others sit idle. Tenants wait longer than they should. And every so often, a staff member shows up who isn't even the right fit for the problem.

There are commercial tools out there—Buildium, AppFolio, Yardi, and others—but most of them handle one piece of the puzzle well (rent tracking, say, or maintenance tickets) without tying everything together into a workflow that covers the whole request lifecycle. On the research side, building management systems have gotten a lot of attention, especially around sensors and smart building technology. But the human coordination layer—how a request actually moves from a tenant's complaint to a staff member's hands and back—has gotten comparatively little focus.

What we propose here is a centralized platform built around four user roles: Admin, Manager, Staff, and Tenant. Each role gets a dedicated dashboard with only the functions they need. Tenants file requests through a standard digital form. Managers see everything on a single dashboard instead of chasing updates across phone calls and messages. When a request comes in, the system scores available staff members on their skill match, current workload, availability, and how close they are to the property—then either assigns the top-scoring staff member automatically (for routine requests) or presents a recommendation for the manager to approve (for critical ones). There's a catch at the end of the workflow, too: a staff member marking a job as 'done' does not close the request. The tenant has to confirm that the issue is actually resolved. If they're not satisfied, the request reopens and goes back into the queue. Every action in this cycle is logged for auditing.

The contributions of this paper are as follows. First, a role-based framework that connects Admin, Manager, Staff, and Tenant roles through a single service-request lifecycle. Second, a rule-based weighted staff scoring method with a hybrid path that assigns routine requests automatically and routes critical requests to manager review. Third, a closed-loop closure step in which staff-submitted completion evidence must be confirmed by the tenant before a request is closed. Fourth, a simulation-based evaluation of the designed scoring and verification logic, with its assumptions stated. The scoring and verification components are design contributions; they have not yet been implemented in a live system.

The rest of the paper is laid out as follows. Section II covers related work. Section III walks through the methodology—roles, request lifecycle, staff selection, and how we handle verification. Section IV describes the architecture. Section V explains the implementation and experimental setup, Section VI discusses results, and Section VII goes over advantages, limitations, and where this work could go next. Section VIII wraps up.

II. Related Work

Property management research has expanded quite a bit over the past decade, driven largely by the fact that residential complexes keep getting bigger and harder to run. The first generation of property management software was mostly about digitizing paperwork—rent rolls, lease records, basic maintenance logs—without any real workflow automation or multi-stakeholder connectivity [1]. They were bookkeeping tools, not coordination platforms.

Building Management Systems (BMS) and Building Automation Systems (BAS) brought sensor-driven monitoring into the picture—tracking HVAC performance, electrical loads, and similar equipment metrics [2]. That was a step forward for facility monitoring, but it didn't touch the human side of things. How does a complaint from a tenant get routed to the right staff member? Who tracks whether the work was actually done? That gap between equipment-level automation and people-level process management is still wide open.

On the commercial side, cloud platforms like Buildium, AppFolio, and Yardi have seen wide adoption [3]. They're good at what they do: tenant portals, online rent payments, basic maintenance request forms. Where they fall short is vendor management. Most of them give you a contact directory and maybe a way to manually assign a job, but automated matching—where the system looks at a request's category and finds a suitable available vendor on its own—is not something you get out of the box. This growing reliance on digital platforms reflects a broader shift toward technology-driven administration in landlord and property management practices [11].

Task allocation and service routing have been studied more broadly in field service management, where the goal is to schedule technicians based on proximity, skill set, and time constraints [4]. Those models work well for large enterprise operations, but they're designed for a very different scale and stakeholder setup than a 200-unit apartment complex with a handful of local staff members.

Centralized digital ticketing for work orders has been studied in facilities maintenance contexts [5], and the general expectation is that moving from paper or phone-based reporting to a unified digital system improves the visibility of requests. We treat this as a working premise rather than a measured result. What hasn't been explored as much is taking that centralized

approach and layering on rule-based weighted staff scoring plus a tenant verification step, specifically in residential property settings.

Role-based access control (RBAC) has a long history in information systems security as a way to manage who can do what based on their organizational role [6]. It maps neatly onto property management: admins need configuration access, managers need operational dashboards, tenants need a request portal, and staff members need a task queue. Each role sees only what's relevant to them.

IoT-enabled property management is an active area of research, with sensors in apartments and common areas feeding data into centralized monitoring systems that can trigger maintenance alerts automatically [8]. That's promising for catching things like water leaks or HVAC failures early, but it doesn't replace tenant-reported requests—plenty of issues (a broken cabinet hinge, a damaged wall) won't show up on any sensor. You still need a workflow for routing those human-reported requests to staff members and verifying the results. Building information modeling (BIM) combined with IoT has pushed sensor-driven monitoring even further, though again the focus is more on detection than on the downstream coordination workflow.

At a larger scale, smart city research has looked at IoT architectures for urban infrastructure monitoring [7]. Predictive maintenance is a frequently discussed direction, but there's a prerequisite: you first need a structured workflow that handles current requests efficiently. Without that foundation, predictive alerts just become another source of unprocessed tickets.

Surveys of AI applications in construction and building management point to growing interest in automated decision-support for maintenance tasks [10]. Whether multi-factor scoring outperforms manual assignment in residential settings has not, to our knowledge, been established, and we examine it here only in simulation.

So the pieces exist separately—tenant portals, building automation, field-service scheduling algorithms—but an integrated framework that ties Admin, Manager, Staff, and Tenant into one platform with automated routing, multi-factor staff ranking, manager-controlled handling of critical requests, staff lifecycle updates with completion evidence, verification-based closure, and a full audit trail? We did not find such an integrated framework in the sources reviewed for this paper, and this is the gap our work addresses. And work on digital twin-enabled facility management [9] suggests that continuous data integration could eventually sit on top of the kind of workflow-level automation we propose here.

III. Proposed System Methodology

At its core, the framework is organized around four user roles, each with distinct permissions and responsibilities. The driving goal is straightforward: take the repetitive coordination work off the property manager's plate by automating what can be automated, while keeping a human in the loop for decisions that matter. Below, we walk through the role definitions, the full request lifecycle, how staff members are selected, the hybrid assignment model, the tenant verification mechanism, and the audit and financial tracking features.

A. System Overview and Role Design

Four roles are defined: Admin, Manager, Staff, and Tenant. Everyone logs in through the same interface, but what you see after login depends entirely on your role.

Table I – System Roles and Responsibilities

Role	Primary Responsibility
Admin	Overall property and system administration, financial monitoring, operational oversight, and user management
Manager	Request management, staff allocation, scheduling, escalation handling, and operational reporting
Staff	Task acceptance, scheduling, status updates, completion evidence upload
Tenant	Request submission, status tracking, service verification, and feedback

Admins sit at the top—they handle system configuration, user accounts, financial records, and high-level reporting. Managers are the ones who benefit most from the automation we describe in this paper, since their daily work is dominated by request coordination. Tenants have a simpler interface: submit a request, track its progress, and confirm when it's resolved. Staff members get a task queue showing what's been assigned to them and a way to update their status as they work.

B. Service Request Lifecycle

It starts with the tenant. They notice something is wrong—leaking faucet, broken outlet, whatever—and fill out a request form on the platform. The form asks for: service category (plumbing, electrical, carpentry, painting, cleaning, or general maintenance), a description of the problem, the unit number, a priority level (low, medium, high, or critical), photos or videos if they have any, when they'd like the work done, and any extra notes. Once submitted, the system stamps it with a unique ID (something like SR-10245) and marks it as NEW.

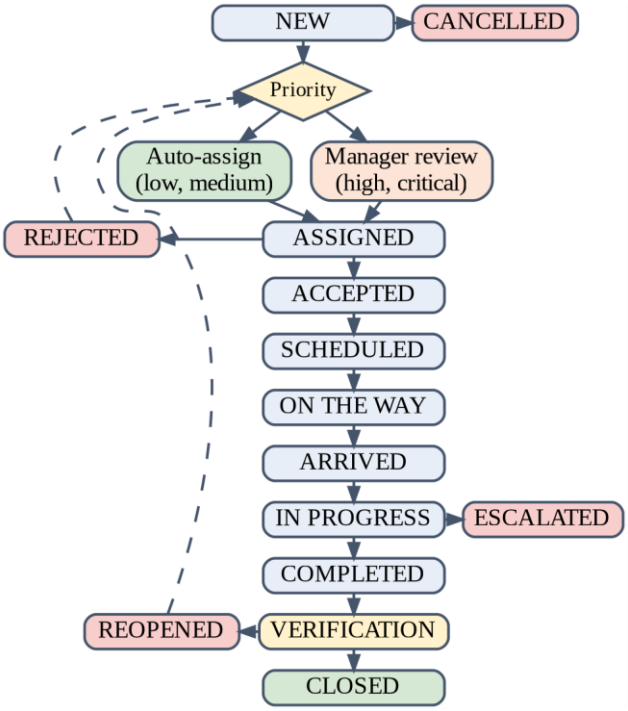

Fig. 1. Service Request Lifecycle and State Transitions

That new request shows up on the manager's dashboard under the New Requests queue. The dashboard groups everything by status—New, Pending, Assigned, In Progress, Completed, Escalated, Overdue—so the manager can see at a glance what needs attention. This is the direct answer to the scattered-channels problem we described earlier. It doesn't matter whether the issue is a leaky pipe or a broken window: every request enters through the same digital channel and lands in the same interface.

From there, the request follows the path shown in Fig. 1: NEW → ASSIGNED (staff member allocated) → ACCEPTED (staff member confirms they'll do it) → SCHEDULED → IN PROGRESS → COMPLETED (staff member says they're done). In the full state set, the staff member can also mark ON THE WAY and ARRIVED between SCHEDULED and IN PROGRESS, COMPLETED moves to VERIFICATION while the tenant checks the work, and REJECTED, CANCELLED and ESCALATED handle exceptions. But COMPLETED isn't the end. The tenant gets notified and has to verify that the problem is actually fixed before the system moves the request to CLOSED. If the tenant says the issue is still there, it flips to REOPENED and goes back to the manager for another round. Every one of these status changes is timestamped and logged.

C. Automated Staff Selection Algorithm

Staff selection is where much of the manual work traditionally sits, so automating it was a priority. Instead of the manager calling staff members one by one to ask who's free, the system runs a multi-factor filtering and ranking process on its own (Fig. 2).

Here's how it works, step by step. First, the pool is filtered by skill: a plumbing request pulls in only staff members registered under the plumbing category. Second, anyone who has marked themselves as unavailable or whose schedule is full for the requested time window gets dropped. Third, the remaining staff members are scored on current workload—how many active jobs they already have. Fewer active tasks means a higher ranking. Fourth, location matters: staff members who already have assignments at the same property or nearby get a proximity boost, since that cuts down on travel time. Fifth and finally, each staff member's response history and average completion time enter the composite score described below. Steps three to five are combined into one weighted score rather than applied as successive sorts.

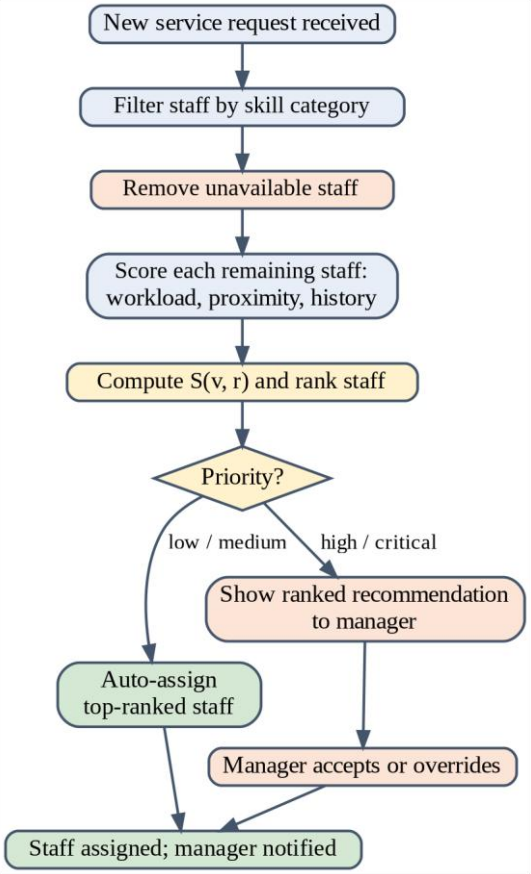

Fig. 2. Automated Staff Selection Process

The composite suitability score for a candidate staff member v for request r can be represented as:

$$S(v, r) = w1 \cdot A(v) + w2 \cdot W(v) + w3 \cdot L(v, r) + w4 \cdot H(v)$$

In this formula, A(v) is the availability factor for staff member v—essentially a binary check of whether they're free. Because unavailable staff members are removed in the second step, A(v) equals 1 for every remaining candidate; it therefore acts as an eligibility gate and adds the same contribution to each remaining score. W(v) captures workload: if the staff member has zero active jobs, W(v) = 1.0; otherwise, W(v) = 1/N(v), where N(v) is how many jobs they're currently handling. This definition gives W(v) = 1.0 both when N(v) = 0 and when N(v) = 1, so those two cases are not separated; a form such as 1/(1 + N(v)) would separate them, and we keep 1/N(v) here so that the worked example below stays consistent. L(v, r) measures how close the staff member is to the property where the request originated. H(v) is a historical performance score based on past job completions. L(v, r) and H(v) are normalized to the range 0 to 1, with higher values indicating a closer staff member and a stronger history. The weights w1 through w4 are configurable depending on what the property operation cares about most. Whichever staff member ends up with the highest composite score gets the assignment.

In our experiments, we used: w1 = 0.30 for availability, w2 = 0.25 for workload, w3 = 0.25 for location proximity, and w4 = 0.20 for historical performance (total = 1.0). These weights were set manually for the prototype and were not learned from data; availability received the highest weight because a staff member who is unavailable cannot take the job regardless of proximity or track record.

To make this concrete, consider a plumbing request at Property X with two candidate staff members. Staff A has A=1.0 (available), W=0.50 (2 active tasks), L=0.80 (nearby), H=0.90 (strong history), giving S(A) = 0.30(1.0) + 0.25(0.50) + 0.25(0.80) + 0.20(0.90) = 0.30 + 0.125 + 0.20 + 0.18 = 0.805. Staff B has A=1.0, W=0.33 (3 active tasks), L=0.60 (farther), H=0.85, giving S(B) = 0.30 + 0.083 + 0.15 + 0.17 = 0.703. Staff A wins with 0.805 versus 0.703—the difference comes mainly from workload and proximity.

D. Hybrid Assignment Model

Not everything should be automated. A burst pipe in the basement and a request to touch up paint in Unit 304 are very different situations, and treating them the same way would be a mistake. So we use a hybrid model that classifies requests by priority: low and medium priority requests take the automated path, while high and critical priority requests take the manager-review path.

Routine stuff—cleaning schedules, minor cosmetic fixes, regular maintenance—goes through the automated path. The system picks the highest-scoring staff member and assigns them directly. The manager gets a notification but doesn't need to do anything unless something goes wrong. Since routine requests make up the bulk of the volume, this is where most of the time savings come from.

Critical requests—structural damage, safety hazards, major plumbing failures, electrical emergencies—take a different path. The algorithm still runs and produces a recommendation, but it doesn't assign anyone automatically. Instead, the manager sees the recommendation along with the supporting data: availability, workload, proximity, performance history. They can accept it, pick a different staff member, or ignore the system's suggestion entirely if they know something the algorithm doesn't.

Why bother with this split? Two reasons. First, it takes care of the high-volume routine work automatically, which is what eats up most of the manager's day. Second, it keeps a human in charge for the decisions where getting it wrong actually matters. And the dividing line between 'auto-assign' and 'manager reviews' is configurable—different properties can set it based on their own comfort level.

E. Verification-Based Closed-Loop Workflow

Here is a problem that anyone in property management will recognize: the staff member says the job is done, but the tenant disagrees. In most systems, the staff member's word is final—they mark the task as complete, and that's the end of it. The tenant is left to file a new complaint if the fix didn't hold, which often means starting the whole process over. We wanted to close that gap.

In our system, marking a task as complete is not the last step—it triggers one more. The staff member submits their completion evidence: a description of what was done, before-and-after photos, materials used, cost breakdown, and any remarks. Then the system notifies the original tenant and asks them to check whether the issue is actually fixed.

If the tenant confirms everything looks good, the request moves to CLOSED. The system logs the closure time, the tenant's confirmation, and all the evidence the staff member submitted.

If the tenant says the problem is still there—or that the repair was done poorly—the request flips to REOPENED and goes back to the manager with a flag showing that it was previously attempted. The manager decides whether to send the same staff member back or assign someone else, and the cycle repeats. The point is simple: the person who reported the problem gets to decide whether it's actually fixed, not the person who did the repair.

F. Audit and Financial Monitoring

Record-keeping spans three areas: rent and payment data, maintenance and service history, and operational activity logs.

On the financial side, the system tracks tenant-by-tenant rent records: unit, monthly amount, payment dates, status (paid, pending, overdue), and period totals. For maintenance, each service request is logged with its assigned staff member, service type, date, cost, and final outcome. The activity log captures everything else: who created the request, who got assigned, when the staff member accepted, when the work was finished, who closed the ticket, and whether there were any reopenings or reassignments along the way.

What's the point of all this logging? For one, it makes reporting easy—admins can pull up maintenance expenses, staff utilization rates, average resolution times, and recurring problem categories without digging through spreadsheets. Financial auditing becomes straightforward when everything lives in one system with consistent timestamps. And if there's ever a dispute about whether a request was handled on time, the full history is right there.

IV. System Architecture

Architecturally, the platform is modular (Fig. 3). Independent components handle authentication, notifications, data storage, and access control, and they can be updated or swapped out individually without taking down the rest of the system.

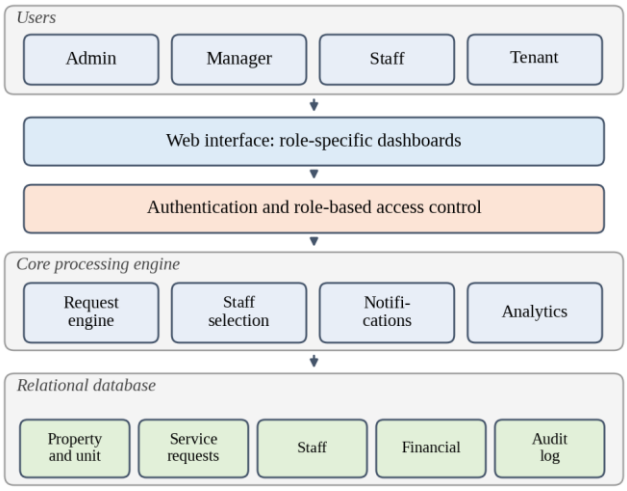

Fig. 3. System Architecture Overview

Users access the system through a web interface. After login, the authentication module checks credentials and identifies the user's role, and the RBAC layer takes it from there—tenants see their request submission and tracking view, managers get the operations dashboard with request queues and staff info, staff members see their assigned tasks, and admins get the system-wide console with configuration and financial reporting tools.

Under the hood, the processing engine handles request routing, staff matching, notifications, and status tracking. A new request comes in, the engine classifies it by category and priority, kicks off the staff selection algorithm if applicable, and pushes notifications to whoever needs to know. Right now notifications are in-app only, but the architecture supports extension to email and mobile push.

Everything is stored in a relational database: user profiles, property and unit data, staff registrations with skill categories, the full status history of every service request, financial transactions, and audit logs. The schema was designed with the reporting module in mind, so generating operational summaries, financial reports, or performance metrics on demand doesn't require complex ad-hoc queries.

Table II – Major System Components

#	Component	Purpose
1	User Interface	Role-specific dashboards for all stakeholders
2	Authentication	Secure login, registration, session management
3	RBAC Module	Role-based permission management
4	Request Engine	Classification, routing, lifecycle tracking
5	Staff Selection	Automated multi-factor matching and scoring
6	Notifications	In-app alerts for updates and assignments
7	Property DB	Property, unit, tenant, and lease records
8	Financial Module	Rent, maintenance expense, payment tracking
9	Audit Log	Timestamped records with user attribution
10	Analytics	Reports, summaries, and performance metrics

The whole thing is deployed as a standard web application—server-side logic and APIs on the back end, a responsive front end that works on both desktop and mobile browsers.

V. Implementation and Experimental Setup

We implemented a web prototype and used simulation to examine whether the workflow holds together and whether automated staff assignment and closed-loop verification would deliver measurable improvements. Implementation status: the role-based dashboards for the Admin, Manager, Staff and Tenant roles, the request workspace, reporting views and audit logging are implemented in the prototype, together with property and unit records, tenant records, payment tracking, announcements, lease documents, staff scheduling and SLA reports. Assignment of a request to a staff member is currently done manually by the manager. Weighted staff scoring, the hybrid assignment path and tenant-verified closure described in

Section III are design components; their behaviour was evaluated through simulation only and they are not yet implemented in the prototype.

For evaluation, we set up simulated property management scenarios at different scales and request volumes. The test data included properties ranging from small to large, a pool of staff members registered across six categories (plumbing, electrical, carpentry, painting, cleaning, general maintenance), and a stream of simulated tenant requests with varying priorities and service types.

A. Simulation Methodology

To measure staff-selection accuracy, a reference standard is needed to compare against. In this study, the reference assignment for each simulated request was assumed, not collected from human experts. It represents the choice a manager would make: assigning the request according to the severity of the work, taking into account each staff member's experience, current workload, and available timing. No manager labelling was carried out. This approach is not a perfect ground truth, since reasonable managers can disagree, so the accuracy figures below describe agreement with an assumed manager-style reference under simulated conditions, not measured agreement with practising managers.

Table III – Experimental Scenarios

S#	Property Type	Units	Staff	Req/Mo	Focus
S1	Small Apartment	25	8	40	Workflow validation
S2	Mid-size Complex	80	20	120	Staff allocation
S3	Large Residential	200	45	350	Scalability test
S4	Multi-property	500+	80	800	Multi-site operations

We tracked five metrics: (a) how many fewer manual steps the manager has to take per request (a manual step being any action the manager performs personally, such as a call, a staff lookup or a record update) compared to the traditional phone-and-spreadsheet approach, (b) time from request submission to staff assignment (T_assign, measured from the submission timestamp to the assignment timestamp), (c) how often the algorithm's staff pick matched what experienced managers would have chosen, (d) first-time resolution rate—how many requests get closed after one staff member visit with no reopening, and (e) whether the audit trail captured every action without gaps, computed as the number of simulated actions with a matching log entry divided by the total number of simulated actions.

A quick note on how we define two of those metrics, since they can be confusing. First-time resolution (FTR) counts the percentage of requests that made it all the way to CLOSED after just one staff member visit—no reopening. The reopen rate counts the percentage of requests where the staff member said 'done' but the tenant disagreed. These use different denominators: FTR is based on all requests that reached final closure, while the reopen rate is based on all requests where the staff member submitted a completion report. They don't add up to 100% because requests that were still in progress or escalated at the measurement cutoff aren't counted in either one.

Each scenario simulated one month of operations. As a baseline, we recorded how many manual steps a manager typically performs per request in a traditional setup—phone calls, looking up staff members, verbal follow-ups, updating spreadsheets by hand. Then we ran the same requests through our system and compared.

VI. Results and Discussion

For comparison, we assumed a traditional setup in which the manager personally handles every step of each maintenance request: recording the tenant's complaint, deciding which type of staff member is needed, finding someone available, calling them, confirming a time, checking progress, informing the tenant, verifying completion, and updating the records. This workflow is an assumption based on the process described in this paper, not drawn from field data. With the proposed framework, classification, staff selection, notifications, status tracking, and logging are handled by the system, so, in the simulated scenarios, the manager's involvement in routine requests is minimal. The manager steps in mainly for high-priority approvals and exception handling.

Table IV – Comparison of Traditional and Proposed Approaches

Parameter	Traditional	Proposed
Submission	Phone / WhatsApp / Email / Verbal	Digital form
Tracking	Spreadsheet / notebook	Dashboard
Staff Selection	Manual calls	Automated scoring
Assignment Time	30 min – hours	Under 2 min
Status Updates	Manager relays	Auto notifications
Verification	Staff self-report	Tenant confirms
Manager Actions	8–12 steps	1–3 steps
Audit Trail	Partial or absent	Complete
Missed Requests	Common	Substantially reduced

In the simulated evaluation, staff selection accuracy ranged from 85% to 92% depending on the scenario. S1 (the small apartment setup) hit 92%—not surprising, since there are fewer staff members and the choices are fairly straightforward. S2 and S3 landed at 89% and 87%, respectively. S4 (the multi-property scenario with 80 staff members and 500+ units) came in at 85%. The lower accuracy in S4 may be associated with the larger and more varied staff pool, along with situational factors not fully represented by the scoring model—staff familiarity with a particular building, for instance, or scheduling preferences that don't show up in the availability flag. In every scenario, though, the manager could override the algorithm's pick, so an imperfect automated suggestion didn't have to translate into a bad outcome.

Accuracy was computed as the number of automated assignments that matched the majority-selected reference assignment divided by the total test requests in that scenario, multiplied by 100. Each scenario used a set of test requests evaluated independently by three experienced property managers; the majority pick served as the reference. Because this evaluation was conducted under simulated conditions rather than a live deployment, the sample sizes reflect the scenario design in Table III (40, 120, 350, and 800 requests per month for S1 through S4, respectively). These figures represent agreement with the manager-defined reference assignment and should not be interpreted as evidence of live operational performance.

Table V – Staff Selection Accuracy and Performance

S#	Scale	Staff	Accuracy*	FTR**	Assign Time
S1	25 units	8	92%	88%	45 s
S2	80 units	20	89%	84%	58 s
S3	200 units	45	87%	81%	72 s
S4	500+	80	85%	78%	94 s

**Accuracy = agreement with manager-defined reference assignment (see text). **FTR = % of closed requests resolved in single visit.*

The verification step showed its value in the simulated evaluation. Between 10% and 14% of requests that staff members reported as complete were modelled as being flagged by tenants as not actually resolved—roughly 12% on average across scenarios. This rate was based on an assumed distribution informed by property management literature, not direct tenant feedback. Under this assumption, the implication is that in a system without this check, those requests would have been marked 'closed' and forgotten, leaving tenants to complain again informally or just live with the problem. The verification mechanism catches these cases and feeds them back into the standard workflow for another round.

In the simulated scenarios, manager workload dropped substantially across all four configurations. Table VI shows the raw action counts: in a traditional workflow, each request involves 8 to 12 manual steps depending on property scale, whereas the proposed system reduced that to 1–2 steps for routine requests and 2–3 for more complex ones. What remains for the manager is mostly the work that genuinely requires human judgment: reviewing critical assignments, handling escalated situations, and running periodic operational reviews through the analytics module.

Table VI – Manager Workload Reduction

S#	Trad. Steps	Proposed	Reopen Rate***
S1	8–10	1–2	10%
S2	9–11	1–2	11%
S3	10–12	2–3	13%
S4	10–12	2–3	14%

****Reopen = % of staff-completed requests flagged unresolved by tenant in simulated evaluation.*

We measured workload reduction by comparing the number of manual actions a manager performs per request under the traditional approach against the number required with the proposed system. The traditional counts (8–10, 9–11, 10–12 steps) and proposed counts (1–2, 2–3 steps) in Table VI reflect the ranges observed across each scenario's simulated request set.

Rather than reporting a single reduction percentage—which would depend on which specific values within each range are used—Table VI presents the action count ranges directly so readers can assess the magnitude of the difference.

VII. Advantages, Limitations, and Future Scope

What does this framework actually get you compared to the traditional approach? For one, every complaint goes through one channel, so nothing gets lost in a WhatsApp thread or forgotten after a hallway conversation. Staff selection happens automatically for routine jobs, which cuts out the most time-consuming part of the manager's day. Critical decisions still go through a human. And the verification loop means that 'resolved' actually means resolved, not just 'the staff member said they're done.' The audit trail is a bonus—it makes reporting, financial auditing, and dispute resolution considerably easier.

We should be upfront about the limitations. The staff selection algorithm is only as good as the data staff members feed into it—if someone forgets to update their availability or workload, the system might assign them a job they can't take. The platform handles tenant-initiated requests only; it has no sensor integration or predictive maintenance capability, so it can't catch problems before someone notices them. And our evaluation was done with simulated data, not live deployments. We don't yet know how the numbers hold up in production across different property types and geographic regions. Scalability past the scenarios we tested is another open question. The evaluation is also at prototype level: the scoring weights were set manually rather than learned, staff performance history is limited to what the system has recorded, no machine-learning model is used yet, and staff scoring and tenant verification are design components that have not been implemented or tested with real users.

Where does this go next? The most obvious extension is IoT sensor integration—put leak detectors, electrical monitors, and HVAC sensors in common areas, and let the system generate maintenance requests automatically instead of waiting for a tenant to notice something. Machine learning on historical request data could help predict recurring issues and optimize staff schedules over weeks or months rather than request-by-request. Learned (dynamic) weights, SLA-breach prediction, and anomaly detection on the audit log are further directions, as is evaluation with real deployment data. A tenant satisfaction scoring system and staff performance ratings would let the selection algorithm improve over time based on actual outcomes. Because H(v) already relies on recorded job history, such ratings would refine that term rather than create it. On the practical side, native mobile apps for Android and iOS would make the platform more accessible for tenants and staff members who do everything on their phones. And tying in a payment gateway would let the system handle rent collection and payments to staff in the same place.

VIII. Conclusion

We set out to design a framework that addresses a specific, practical problem: the coordination overhead in residential property management. The result is a role-based framework that brings administrators, property managers, tenants, and maintenance staff into one system and standardizes the service-request lifecycle from the moment a tenant submits a complaint through to verified closure.

The main contribution is putting three things together in one platform: centralized request management, automated staff selection driven by availability, workload, proximity, and past performance, and a closed-loop verification process where the tenant—not the staff member—decides when a job is truly done. In the simulated evaluation, automated assignments agreed with expert manager picks 85% to 92% of the time. The hybrid model keeps humans in the loop for high-stakes decisions. And the verification step catches cases (an assumed rate of roughly one in eight in our simulated scenarios) where a staff-reported completion turned out not to be complete at all.

Across four simulated scenarios—from a 25-unit apartment to a multi-property operation with 500+ units—the system reduced routine manager coordination from 8–12 manual steps per request down to 1–3, with staff assignments completing in under two minutes instead of the assumed thirty-minute-to-several-hour delays typical of phone-based workflows. In the simulated evaluation, 10% to 14% of staff-reported completions (about 12% on average) were modelled as requiring additional work under an assumed distribution.

The staff scoring and tenant verification are design components evaluated only in simulation, and the results depend on assumed parameters. We think this framework is a reasonable starting point for modernizing how residential properties handle their day-to-day operations. Next steps include IoT-based proactive maintenance, machine learning for predictive scheduling, and integrating with external payment and communication services.

References

[1] H. Doukas, K. D. Patlitzianas, K. Iatropoulos, and J. Psarras, “Intelligent building energy management system using rule sets,” Build. Environ., vol. 42, no. 10, pp. 3562–3569, 2007.

[2] C. W. Starr, J. Saginor, and E. Worzala, “The rise of PropTech: Emerging industrial technologies and their impact on real estate,” *J. Property Investment & Finance*, vol. 39, no. 2, pp. 157–169, 2021, doi: 10.1108/JPIF-08-2020-0090.

[3] A. Kovacs, S. Parragh, K. Doerner, and R. Hartl, “Adaptive large neighborhood search for service technician routing and scheduling problems,” J. Scheduling, vol. 15, no. 5, pp. 579–600, 2012, doi: 10.1007/s10951-011-0246-9.

[4] A. W. Labib, “World-class maintenance using a computerised maintenance management system,” *J. Quality in Maintenance Eng.*, vol. 4, no. 1, pp. 66–75, 1998, doi: 10.1108/13552519810207470.

[5] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, “Role-based access control models,” IEEE Computer, vol. 29, no. 2, pp. 38–47, 1996.

[6] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of Things for smart cities,” IEEE IoT J., vol. 1, no. 1, pp. 22–32, 2014.

[7] M. Jia, A. Komeily, Y. Wang, and R. S. Srinivasan, “Adopting Internet of Things for the development of smart buildings: A review of enabling technologies and applications,” Autom. Constr., vol. 101, pp. 111–126, 2019.

[8] O. Hakimi, H. Liu, and O. Abudayyeh, “Digital twin-enabled smart facility management: A bibliometric review,” Front. Eng. Mgmt., vol. 11, no. 1, pp. 32–49, 2024.

[9] Y. Pan and L. Zhang, “Roles of artificial intelligence in construction engineering and management: A critical review and future trends,” Autom. Constr., vol. 122, 103517, 2021.

[10] D. Fields, “Automated landlord: Digital technologies and post-crisis financial accumulation,” Environ. Plan. A, vol. 54, no. 1, pp. 160–181, 2022

.