IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

Vendor Management and Procurement System: A Web-Based Workflow-Automation Approach Using Flask, SQLite, and Role-Based Access Control

Nagendra Prasad S[1] and Prof. Kusuma N[1]
Student, Department of Master of Computer Applications[1]
Faculty, Department of Master of Computer Applications[2]
Maharaja Institute of Technology, Mysore, Karnataka, India

Abstract: *Procurement is a core operational function in every organisation, yet in many small and medium-sized enterprises it still runs across spreadsheets, email threads, and paper files rather than a single system of record. This fragmentation slows vendor approvals, obscures the status of purchase requests, and leaves procurement data prone to duplication and inconsistency, particularly where several people must be involved because no one source of truth exists. This paper presents the Vendor Management and Procurement System (VMPS), a web-based application that consolidates vendor registration and approval, procurement request handling, purchase order generation, inventory tracking, and reporting into a single, centrally administered platform. The system is built on the Python Flask framework using a modular Blueprint architecture, with SQLite as the backing data store and Tailwind CSS, JavaScript, and Jinja2 templating on the client side. Three role-based accounts—administrator, employee, and vendor—are enforced through password-hashed, session-managed authentication, so that each actor can reach only the functionality relevant to their responsibilities. A vendor registers and submits company information for administrative review; once approved, that vendor becomes eligible to receive purchase orders. Employees raise procurement requests, administrators convert approved requests into purchase orders, and completed purchases automatically update inventory records, with dashboard reporting generated directly from this transactional data rather than compiled by hand. The system was developed through a staged requirement analysis, design, implementation, and testing methodology, and was validated using white-box, black-box, unit, integration, output, and user-acceptance testing, with all recorded test cases passing. This paper details the system's architecture, database design, implementation, and testing outcomes, and discusses planned enhancements including cloud deployment, mobile access, barcode integration, and supplier performance analytics.*

Keywords: Vendor management, procurement automation, e-procurement, role-based access control, Flask, SQLite, purchase order managreement, inventory management, web application.

1. INTRODUCTION

Purchasing is one of the core functions of every organisation, spanning vendor registration, purchase requisitions and approvals, order management, and inventory updates. In practice, a large share of this activity still lives in spreadsheets, email inboxes, or paper records. Once procurement information is scattered across more than one source, more people must be pulled in to reconcile it, and keeping the records accurate becomes correspondingly harder. As procurement activity intensifies, manually entering vendor data, tracking purchase requests through approval, generating purchase orders, and maintaining inventory levels becomes increasingly error-prone. Most procurement software on the market compounds this problem by being function-specialised: one tool generates purchase orders, another registers vendors,

DOI: 10.48175/IJARSCT-37765

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

and critical functions such as vendor approval, inventory updates, and procurement tracking end up siloed from one another. The result is duplicated listings, slow approvals, conflicting records, and—because these tools are rarely built with differentiated access in mind—weak control over who can see or change what.

Organisations do not want to switch between several systems to manage a single procurement cycle; they want one application that manages vendor data, registration approval, purchase authorisation, and supply tracking on a unified platform. This paper describes the Vendor Management and Procurement System (VMPS), built to close that gap using freely available, well-understood web technologies: Flask and SQLite on the backend, and HTML5, Tailwind CSS, JavaScript, and Jinja2 on the frontend. The system automates the traditional purchase operations—authenticated, rolerestricted access; centralised data administration; purchaseorder and inventory tracking; and reporting—so that organisations can concentrate on vendor relationships rather than administrative record-keeping. The remainder of this paper reviews related work, states the problem and objectives, describes the system architecture and database design, details the implementation and technology stack, reports the testing methodology and results, and discusses limitations and planned future enhancements.

II. RELATED WORK

Research adjacent to this project falls into three broad strands: enterprise-scale vendor and ERP-based supplier management, domain-specific vendor management case studies, and platform proposals for mobile or web-based e-procurement.

Menon's study of Enterprise Resource Planning (ERP) systems in United States supply chains shows how real-time analytics, process automation, and uniform evaluation criteria improve vendor selection and performance management, helping organisations manage compliance, reduce procurement risk, and build long-term supplier partnerships [1]. A related discussion of ERP-based vendor management reinforces the same conclusion—that ERP adoption improves procurement efficiency, transparency, compliance, and supplier performance through workflow automation, performance monitoring, and data-driven risk management [2]. Both studies, however, focus on ERP implementation at the level of large corporate systems rather than on a self-contained, web-based platform that also handles vendor onboarding, approval, quotation comparison, and document management; VMPS is scoped precisely at that smaller, self-contained level.

Shinde and Sonavane examine vendor management practice within the pharmaceutical supply chain of a single manufacturer, using ERP tooling to support selection, approval, performance evaluation, and quality control, and find that structured vendor management is associated with greater supply-chain and operational efficiency [3]. Their work is chiefly descriptive of organisational practice rather than a reusable software platform, and does not offer a generalpurpose system for vendor registration, approvals, and communication that another organisation could deploy directly— one of the gaps VMPS is intended to address.

Sahroni and Angrian propose an Android-based eprocurement and vendor management system supporting mobile vendor registration, tender notification, vendor rating, and bid approval, allowing vendors to bid from any location and shortening procurement cycles [4]. Their system is confined to the Android platform; VMPS instead targets a browser-based, role-differentiated web application that any device with a browser can reach, while retaining the same emphasis on registration, approval, and tracking.

Adewale et al. examine advanced vendor-management models—vendor scorecards, multi-criteria decision analysis, AI, blockchain, and cloud computing—aimed at maximising economic value in global supply chains [5]. Their contribution is largely theoretical and technology-forward rather than an implemented, day-to-day operational tool, which is the space VMPS occupies.

Chaudhary and Deshmukh propose a Vendor Management System intended to improve supplier performance within Just-in-Time and supply-chain management, built around the systematic certification, monitoring, evaluation, and improvement of vendor performance [6]. Their emphasis is on the supplier-evaluation methodology itself rather than an automated, web-based registration-to-purchase-order pipeline.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

Across this body of work, ERP systems, vendor-evaluation models, and e-procurement platforms are consistently shown to raise procurement efficiency and transparency, but the existing literature is concentrated on enterprise-scale ERP theory, single-organisation case studies, or platform-specific mobile proposals. What is comparatively underexplored is a compact, role-based web application that unifies vendor registration and approval, procurement-request handling, purchase-order processing, inventory tracking, and reporting behind a single centralised database—the gap VMPS is built to fill.

III. PROBLEM STATEMENT

An organisation buying products and services from multiple vendors needs to track vendor data, purchase requests, approvals, purchase orders, and inventory consistently as procurement activity grows. Most procurement software available today is function-specialised—generating purchase orders here, registering vendors there—so that vendor approval, inventory updates, procurement tracking, and reporting remain siloed from one another. This produces duplicated listings, slow approvals, and conflicting records, and is compounded by weak role-based access control in many existing tools, even though procurement inherently involves multiple user types with different authorisations. VMPS was designed to resolve these issues through a single, centralised, web-based application that provides vendor registration and approval, procurement management, purchase-order processing, inventory management, and user administration against one comprehensive database, so that organisations can operate with a more organised and less labour-intensive procurement process.

IV. OBJECTIVES AND SCOPE

A. Objectives

Integrate vendor administration, purchase processing, in-ventory management, and reporting into a single online application, so that organisations no longer need to consult multiple systems to see the whole of their procurement activity.

Reduce manual intervention in vendor registration, vendorapproval, purchase-order processing, and inventory maintenance by automating these routine steps, improving consistency across records.

Enforce access security through role-based authentication,so that administrators, vendors, and employees can only perform the tasks relevant to their role.

Support quicker registration, approval, and purchase-orderconsolidation through a web-based interface, backed by a single consolidated database for vendor, procurement, purchase-order, inventory, and user data.

Validate data automatically to reduce manual entry errors,and provide dashboards and reports that give administrators visibility into purchasing activity and inventory levels.

Adopt a modular architecture that can absorb futurefeatures—cloud deployment, mobile apps, barcode integration, and advanced procurement analytics—without disrupting the existing system.

B. Scope

The platform is designed to be role- and industry-agnostic: administrators, employees, and vendors each operate through a secure, differentiated login, vendor registrations are reviewed and certified by administrators before a vendor may participate in procurement, and employees submit purchase requests that supervisors turn into purchase orders while inventory is kept current. Every module reads from and writes to a single SQLite database, so that procurement data can move between modules—from a request, to an order, to an inventory update, to a report—without duplication. Individual job seekers are not the intended users here; rather, the primary users are the organisation's own administrators, employees, and its registered vendors, with the same design suitable for adoption by a small or medium-sized business regardless of the specific goods or services it procures.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

V. SYSTEM ARCHITECTURE AND METHODOLOGY

VMPS is organised as a layered web application: a browserrendered frontend, a Flask-based backend organised into Blueprints, and a single SQLite database that every module reads and writes. Figure 1 shows the overall system architecture.

Frontend Layer

The client-facing pages—login, dashboards, vendor management, procurement requests, purchase orders, inventory, and reports—are built with HTML5 and styled with Tailwind CSS utility classes, which gave dashboards, forms, and tables a

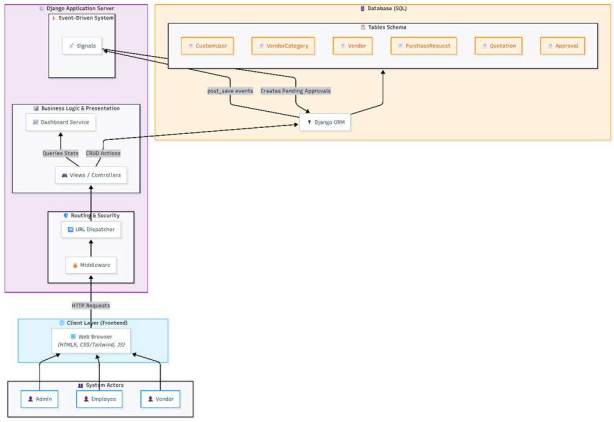

Figure 1: System architecture of the Vendor Management and Procurement System.

consistent layout without hand-writing large amounts of custom CSS. Hero Icons are used throughout the navigation and menu elements. JavaScript adds client-side validation, dynamic forms, search, and confirmation dialogs, so that common interactions do not require a full page reload.

Authentication and Role Management

A single authentication module governs sign-in for administrators, employees, and vendors alike. Passwords are hashed before being stored in SQLite, sessions are managed to guard against session hijacking, and every backend route checks the caller's role before executing, so that a user can only reach the functions assigned to their role. Input is validated before it is processed, and users can log out to end their session securely.

Backend and Business Logic Layer

The backend is implemented in Python using the Flask framework, with the codebase organised into Flask Blueprints—Authentication, Vendor Management, Procurement, Purchase Orders, Inventory, and Reporting—so that each functional area can be developed, tested, and maintained largely independently of the others while still sharing the same underlying database connection. Each user action performed through the web interface is validated by its corresponding Blueprint, the relevant business logic is applied, and the SQLite database is read or updated accordingly before a response is returned to the browser.

Database Layer

All company data—user accounts, vendor records, procurement requests, purchase orders, and inventory items—is held in one SQLite database, connected through Python database libraries. Separate tables were designed for users, vendors, procurement requests, purchase orders, and inventory records, linked by primary and foreign keys to avoid data duplication and preserve referential integrity across modules. Figure 2 shows the entity-relationship design.

The principal entities are: a *User* entity holding an encrypted password and a role that determines which modules the account can reach; a *Vendor* entity holding company details, contact information, and an approval status, which must be approved before that vendor can be issued purchase orders; a *Procurement Request* entity tied to the employee who

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

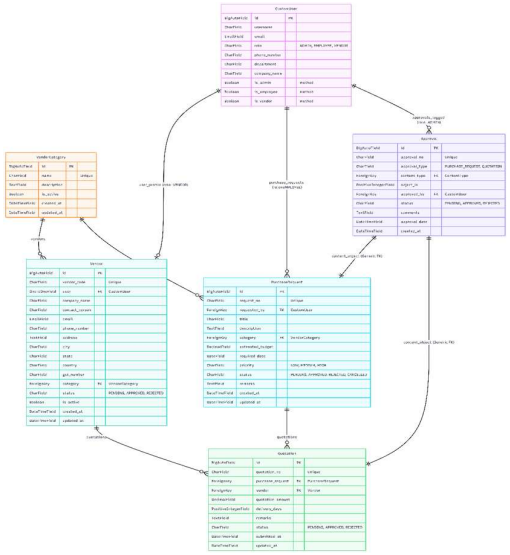

Figure 2: Entity-relationship diagram of the VMPS database.

raised it, which can be converted into a purchase order once approved; a *Purchase Order* entity linking one vendor to one approved procurement request, carrying order date and status; and an *Inventory Item* entity tracking product, quantity, and stock status, updated whenever a purchase order is completed. A user can raise many procurement requests; a vendor can receive many purchase orders; an approved procurement request generates exactly one purchase order; and a completed purchase order updates the corresponding inventory record.

E. Vendor Registration and Approval Workflow

A prospective vendor registers through the public-facing signup flow and submits company name, contact information, and address. The submission is held in a pending state until an administrator reviews it; the administrator may approve, reject, or edit the vendor's details. Only an approved vendor is eligible to be issued purchase orders, and administrators can revisit vendor status at any time to suspend or reinstate a vendor's eligibility for procurement activity.

Procurement and Purchase-Order Workflow

An employee raises a procurement request describing the goods or services required. Administrators view outstanding requests, and an approved request is converted into a purchase order addressed to a specific, approved vendor. Purchase orders can be amended before the procurement is finalised, and both procurement requests and purchase orders retain a full history for later reference and status tracking.

Inventory and Reporting

Once a purchase order is completed, the corresponding inventory record is updated automatically, so stock levels reflect the latest transaction without a separate manual step. The reporting module then draws directly on this same transactional data to produce dashboard summaries and procurement, vendor, purchase-order, and inventory reports, giving administrators a basis for purchasing decisions without compiling figures by hand.

DOI: 10.48175/IJARSCT-37765

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

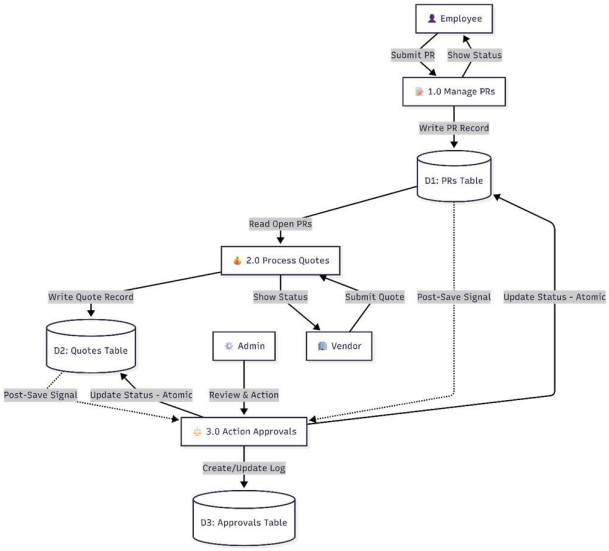

Figure 3: Data flow diagram for the core procurement cycle.

End-to-End Flow

The complete cycle proceeds as follows: (1) a user authenticates and reaches their role-specific dashboard; (2) a vendor registers, or an employee raises a procurement request; (3) an administrator reviews and approves the vendor registration or procurement request; (4) an approved request is converted into a purchase order against an approved vendor; (5) the purchase order is fulfilled and the corresponding inventory record is updated; and (6) dashboard statistics and reports reflect the updated procurement and inventory state. Figure 3 traces this flow at the data level, showing how each module reads from and writes to the shared SQLite database.

VI. IMPLEMENTATION DETAILS

A. Technology Stack

Language: Python
Backend framework: Flask, organised with Flask
Blueprints
Database: SQLite
Frontend: HTML5, Tailwind CSS, JavaScript
Templating: Jinja2, with template inheritance for shared navigation, sidebar, and dashboard layouts
IDE: Visual Studio Code, using an isolated Python virtual environment

Development Methodology

Development proceeded in five stages. *Requirement analysis* examined existing procurement processes to identify functional needs around user identification, vendor approval, procurement administration, and dashboard reporting, alongside security and usability considerations. *System and database design* produced the module breakdown—Authentication, Vendor Management, Procurement Requests, Purchase Orders, Inventory, and Dashboard/Reporting—and the accompanying SQLite schema. *Environment setup* established the Python virtual environment and Flask Blueprint scaffolding. *Core module development* built authentication and role-based

DOI: 10.48175/IJARSCT-37765

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

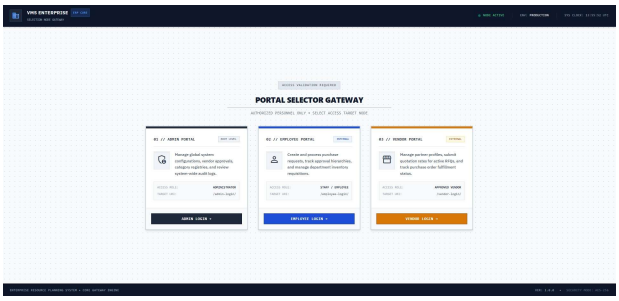

Figure 4: VMPS login page, offering differentiated access for administrators, employees, and vendors.

access first, then vendor registration and approval, then the procurement, purchase-order, and inventory modules, with input validation and password hashing applied throughout. *Testing and integration*, described in Section VII, verified each module individually before validating the system as a whole.

Hardware and Software Requirements

The application runs comfortably on consumer-grade hardware: an Intel Core i5-class processor or better, a minimum of 8 GB of RAM, and around 500 GB of available storage. The software stack—Python 3.x, Flask, SQLite, and Visual Studio Code—is open source and free to set up, and the application itself was developed and tested on Windows. No specialised hardware or paid infrastructure is required, which keeps both the development and the ongoing operating cost low.

VII. SYSTEM TESTING AND VALIDATION

Testing proceeded in two phases: each module was verified individually, and the integrated system was then tested end to end across the frontend, backend, and SQLite database. Six complementary testing methods were applied. *White-box testing* exercised the internal logic and validation rules of each module—CRUD operations, database transactions, and both well-formed and malformed input—before modules were integrated. *Black-box testing* verified functional behaviour from the user's perspective without reference to internal implementation, checking navigation, usability, and system output against expected results across a range of procurement scenarios. *Unit testing* covered login and role-based access, vendor registration and approval, purchase-order and requisition handling, and the reporting and inventory modules individually. *Integration testing* then verified that these modules communicated correctly once combined, tracing data flow across authentication, vendor, procurement, inventory, and reporting components. *Output testing* checked that procurement figures, vendor approval and registration status, purchaseorder accuracy, inventory updates, and generated reports all matched expected results. Finally, *User Acceptance Testing* exercised the vendor registration and approval process, purchase requisitions, inventory management, and reporting under realistic use, together with a usability and navigation review.

Across the fifteen recorded test cases—covering login with valid and invalid credentials, vendor registration, approval, update and deletion, procurement request creation and approval, purchase-order generation, inventory update, dashboard display, and procurement, vendor, and inventory report generation—every case produced the expected outcome. No significant functional defects were found during validation; the system connects to the database correctly, manipulates data as expected, and was judged ready to support the procurement activity it was designed for.

VIII. RESULTS AND DISCUSSION

The completed system demonstrates that a single Flask application, backed by one SQLite database and organised into Blueprints, can support the full procurement lifecycle—vendor registration through to inventory update and reporting—without requiring separate tools for each stage. Consolidating these functions removed the need to reenter or reconcile vendor and procurement data across systems, and the role-based authentication layer meant that administrators,

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, July 2026
Impact Factor: 8.2

employees, and vendors could share one application without exposing functionality outside their responsibilities. Because purchase orders and inventory records are linked directly, inventory figures update automatically as procurement completes, removing a manual reconciliation step that is common in spreadsheet-based processes. The modular Blueprint structure also proved convenient during development itself: individual modules—vendor management, procurement, purchase orders, inventory—could be built, tested, and revised largely independently, which kept the testing process in Section VII tractable and made later changes to one module unlikely to disturb the others.

Two limitations are worth noting. First, the current system is a single-organisation deployment without multi-tenant support, so extending it to multiple organisations would require additional data isolation. Second, several planned capabilities—cloud hosting, mobile access, barcode-based stock updates, and automated email notifications—remain future work rather than part of the present implementation, and their absence means some procurement steps (for example, being notified the moment a purchase order changes status) still rely on a user actively checking the dashboard.

IX. CONCLUSION AND FUTURE SCOPE

This paper presented the Vendor Management and Procurement System, a web-based application built with Flask, SQLite, HTML5, Tailwind CSS, JavaScript, and Jinja2 that consolidates vendor administration, procurement requests, purchase-order processing, inventory management, and reporting behind a single, role-restricted platform. The modular Blueprint architecture kept development and testing manageable and leaves the system well positioned for incremental extension. Validation across white-box, black-box, unit, integration, output, and user-acceptance testing found every recorded test case behaving as expected, supporting the conclusion that the system is ready to streamline day-to-day procurement work for a small or medium-sized organisation.

Planned future enhancements include: deploying the application to a cloud platform to improve accessibility and data-backup resilience; adding automated email notifications for vendor approval, procurement, and purchase-order status changes to reduce the need for users to check the dashboard manually; building companion mobile applications so employees, administrators, and vendors can carry out procurement tasks off-site; integrating barcode and QR-code scanning to reduce manual data entry during stock counts; and introducing supplier-performance analytics that track procurement patterns and purchase history over time to support better-informed procurement decisions. The modular design of the current system is intended to accommodate each of these additions with minimal disruption to the modules already in place.

REFERENCES

1. K. Menon, R. K. Katuri, and K. Desi, “Enterprise Resource Planning (ERP) Systems for Enhancing Performance Management and Vendor Selection in US Supply Chains,” *Procurement Secretariat*, 2024/2025.
2. “Vendor Management & ERP: Improving Procurement Efficiency, Transparency, and Compliance,” 2024.
3. P. C. Shinde and U. Sonavane, “Study on Vendor Management Systems in Supply Chain Management,” 2024/2026.
4. T. R. Sahroni and B. Angrian, “Development of an AndroidBased E-Procurement and Vendor Management System,” 2019.
5. T. T. Adewale, A. B. N. Abbey, C. Mokogwu, A. Q. OlufemiPhillips, and I. O. Adewole, “Advancing Vendor Management Models to Maximize Economic Value in Global Supply Chains,” 2023.
6. M. Chaudhary and S. S. Deshmukh, “Effective Vendor Management System,” 2016.