

Access Automation & Security

Event-driven application engineering for a 20,000+ user community

LEGIONER SOLUTIONS

Built structured access and operations workflows that connect user identity, role assignment, persistent requests, and staff actions in one application.

20,000+ user community

Community scale; membership-based count.

~70% less manual administration

Project-reported estimate across automated activities.

01 / THE OPERATIONAL CHALLENGE

Membership approvals, access requests, and resource fulfillment relied on recurring staff coordination. The application replaced free-form intake with guided requests, explicit status, and completion notifications, making outstanding work easier to track.

02 / MY ENGINEERING SCOPE

Application and integration development spanning event routing, command and form handlers, role-based checks, MongoDB persistence, spreadsheet synchronization, operational displays, and automated notifications. Staff retained responsibility for approval decisions and in-game fulfillment.

03 / CYBERSECURITY RELEVANCE

The work connects identity context with application authorization, maintains records of approvals and request state, and protects intentional data changes during selected concurrent operations. The security focus is practical: who can act, what state can change, and how staff see the result.

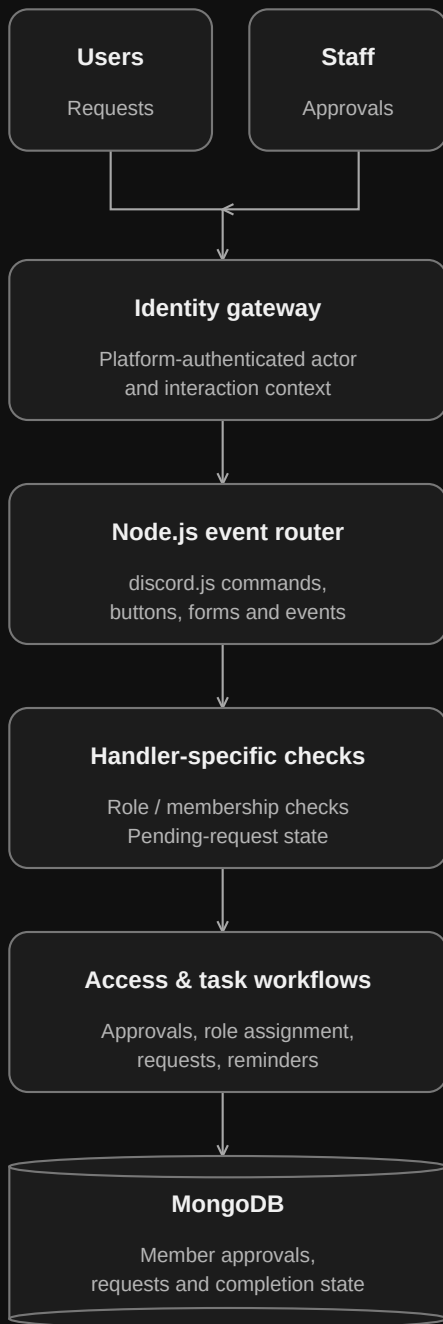
TECHNOLOGY

Node.js / JavaScript / discord.js v14 / MongoDB
Google Sheets API / JSON persistence / async-mutex

Architecture & Data Flow

Identity context, workflow processing, persistence, and operational integrations

ACCESS & WORKFLOWS



TELEMETRY & INTEGRATIONS



DESIGN NOTES

Simplified logical architecture. Authorization checks are distributed across selected handlers. Local JSON, locks, and temporary state are process-scoped. External services supply identity and telemetry.

Security Controls in Practice

How access requests move through the system and preserve operational context

01 / IDENTITY & AUTHORIZATION

The platform supplies authenticated user identity with each interaction. Selected commands check configured role groups before modifying membership state; cross-community onboarding checks primary-community membership and the required role before assigning secondary roles.

02 / REQUEST LIFECYCLE

Guided inputs collect the target server and character details. Applicable workflows check for an existing pending request, then create a MongoDB record marked incomplete and publish a staff work card. After staff fulfillment, the application records completion, refreshes the queue, disables completed controls, and notifies the requester.

03 / DATA INTEGRITY & CONCURRENCY

Character-record removals use a process-local mutex to serialize the helper's read/modify/write path. Updates include a lastModified marker; spreadsheet imports skip retained records already marked as manually corrected. These controls reduce specific overwrite paths while preserving intentional staff edits.

04 / OPERATIONAL SAFEGUARDS

Additional controls include typed command options, required form inputs, duplicate-pending-request checks, private responses in selected workflows, configured telemetry-source matching, and RAW spreadsheet writes. Persisted reminders restore scheduled work after application restarts.

05 / ENGINEERING JUDGMENT & NEXT STEPS

The architecture separates workflow automation from human fulfillment. The next hardening priorities are consistent authorization on every mutation, atomic request deduplication, revision-checked database writes, and retry/reconciliation tracking when database updates succeed but notifications fail.

RECRUITER TAKEAWAY

Hands-on experience connecting identity, access workflows, backend logic, data persistence, and operational troubleshooting in a community application.