AI-Augmented Real-Time Multi-Tenant Customer Support Ecosystem

Mrs. Snehal Bagal
Dept. of AI and Data Science
AISSMS IOIT, Pune, India
snehal.bagal@aissmsioit.org

Pritesh Kamdi
Dept. of AI and Data Science
AISSMS IOIT, Pune, India
pritesh.kamdi@aissmsioit.org

Shreyash Sutar
Dept. of AI and Data Science
AISSMS IOIT, Pune, India
shreyash.sutar@aissmsioit.org

Sanket Tambekar
Dept. of AI and Data Science
AISSMS IOIT, Pune, India
tambekarsanket1@gmail.com

Abstract—Messaging-based customer service requires coordinated communication, ticket ownership, tenant isolation, and timely resolution. This paper presents Resolv, a WhatsApp-centred, multi-tenant customer-support platform integrating the Meta WhatsApp Cloud API and Baileys multi-device sessions within a unified ticket workflow. The architecture combines a Next.js frontend, Express API, PostgreSQL/Prisma persistence, Redis/BullMQ background processing, Socket.IO events, and dedicated Baileys workers coordinated by a Session Manager. Workspace-scoped authorization, online-agent round-robin assignment, queued backlog dispatch, SLA escalation, and on-demand Gemini 2.5 Flash summarization with up to two historical tickets are described. The contribution is the integration and explicit modelling of these mechanisms rather than a new messaging or AI algorithm. The project team reports successful functional checks; independent latency, scalability, recovery, ROUGE, and sentiment-accuracy measurements remain unavailable. The paper therefore separates reported functional outcomes from quantitative claims and specifies a reproducible evaluation plan.

Index Terms—multi-tenancy, customer support, WhatsApp integration, distributed workers, ticket management, real-time communication, intelligence, SaaS architecture.

I. INTRODUCTION

A. Motivation

Digital messaging has become a channel for customer interaction. Businesses use messaging applications to answer questions give help coordinate service requests and keep customer relationships. When communication volume grows support teams need more than a shared inbox; they need ticket records clear assignment rules, ways for agents to work together tools to monitor service level and reports.

Traditional support workflows often depend on messaging apps, spreadsheets, manual routing and disconnected admin systems. This fragmentation makes it hard to keep a customer history to find who is responsible, for open requests and to coordinate agents. A further difficulty comes when a software-as-a-service platform serves organizations: tenant data must stay separate even though the app services and database infrastructure are shared.

Resolv solves these problems with a customer support environment that focuses on WhatsApp communication. Resolv supports two integration methods – the Meta WhatsApp Cloud API and Baileys-based multi-device sessions – and blends them with workspace-oriented tenancy, automated ticket handling, distributed session workers, instant notifications and AI-assisted conversation understanding.

B. Research Objectives

- Build a workspace where customer communication and ticket management happen in one place.
- Organize data logically to support tenants with role-based access control to ensure the right people see and do the right things.
- Connect WhatsApp Cloud API for reliable messaging and use Baileys-based WhatsApp Web sessions to manage multiple devices.
- Automate the creation of tickets their sorting round-robin assignment to agents and track service level agreements. Include escalation when deadlines are at risk.
- Enable distributed WhatsApp session handling by allowing worker nodes to register stay monitored for health get messages routed efficiently and recover from failures.
- Offer real-time messaging, instant notifications, team collaboration tools, marketing campaigns, performance analytics and subscription management.
- Include on-demand AI features like message summarization and sentiment classification—describe what they do without making claims, about how accurate or reliable the modelsre.

C. Key Contributions

1. A unified ticket and customer workflow for Meta WhatsApp Cloud API and Baileys Web Client traffic, while retaining source-specific authentication and delivery handling (Sections VI and VIII).

2. Workspace-scoped tenant data and role-based access rules applied to customer, ticket, message and notification operations (Section V).

3. A dedicated Session Manager with heartbeat-based worker health monitoring, capacity-aware placement and persistent workspace-to-worker assignments (Section VI).

4. Automated triage, eligible-online-agent round-robin routing, backlog redispatch, delayed SLA checks and manager escalation (Section VIII).

5. On-demand Gemini 2.5 Flash summaries and three-class sentiment labels based on the active ticket and up to two preceding customer tickets (Section VII).

6. Explicit mathematical abstractions and an evaluation protocol distinguishing reported functional verification from quantitative measurements not yet collected (Sections IV, XIII and XIV).

1.

D. Scope and Organization

The paper describes the implementation documented for Resolv and distinguishes implemented capabilities from interpretation and proposed future work. The system scope is WhatsApp-centred; other channels are not treated as implemented. Section II reviews related work Section III states the problem and requirements Sections IV–XII present the architecture and modules Section XIII defines the evaluation methodology and Sections XIV–XVII discuss results, limitations, future work and conclusions.

II. RELATED WORK AND RESEARCH GAP

A. Existing WhatsApp-Based Customer Support Systems

WhatsApp support can be implemented through Meta's webhook-driven Cloud API or linked-device WhatsApp Web clients. The official route supports template messaging and delivery receipts, while a Baileys-based route requires persistent authentication state, socket lifecycle management and message synchronization. A support platform must also associate messages with customers, agents and ticket states. Resolv normalizes both ingress paths into a shared ticket workflow, while preserving source-specific operational handling. This is an integration design, not evidence that competing products lack comparable features; a product-by-product benchmark would be needed for that claim.

B. Multi-Tenant SaaS Architecture and Data Isolation

Multi-tenancy allows organizations to share infrastructure while maintaining logical boundaries. Kabbedijk et al. [2] provide a systematic perspective on multi-tenancy, and Ochei et al. [3] discuss varying isolation requirements. Resolv associates tenant-owned entities with workspaceId and combines scoped data access with role-based authorization. Isolation must also be maintained in background workers, Socket.IO rooms, invitation flows and administrative operations. Application-layer scoping is the documented mechanism; database row-level security, adversarial isolation tests and formal noninterference guarantees are not established by the available evidence.

C. Distributed Worker and Session Management

Persistent WhatsApp sessions require coordination beyond conventional stateless HTTP request handling. Resolv uses a Session Manager to register workers, track heartbeats and capacities, maintain workspace-to-worker assignments, and proxy session operations. Baileys workers retain the live connection state, while Redis Pub/Sub forwards events toward the API and Socket.IO clients. A worker is considered eligible only when it is healthy, has free session capacity and remains below the configured CPU threshold. Heartbeat detection bounds and reconnection back-off are design models; observed recovery time and improvements relative to a single-worker baseline remain unmeasured.

D. Automated Ticket Assignment and SLA Management

Support-ticket scheduling must balance availability, ownership and response deadlines. Resolv routes tickets among currently online eligible agents using a persisted RoundRobinState, retains requests in QUEUED status when no agent is available, and attempts backlog dispatch when an agent becomes online or resolves a ticket. Optional phone-number mapping can override general allocation for designated agents. Redis-backed BullMQ delayed jobs evaluate SLA deadlines and trigger escalation to eligible managers. Evaluation should report assignment distribution, queue waiting time, missed deadlines and fairness under identical agent availability schedules; the round-robin equation is a model rather than a literal description of cursor persistence.

E. AI-Based Customer-Support Summarization

Customer-support summaries should capture the current problem, prior actions, unresolved constraints and relevant history without introducing unsupported facts. Dialogue-summarization research [4] motivates structured evaluation using human reference summaries and overlap-based metrics, supplemented by factuality and completeness review. Resolv retrieves the current ticket and up to two previous tickets belonging to the same customer, formats messages chronologically and invokes Gemini 2.5 Flash on demand. The returned summary is stored on the ticket to avoid repeated inference when reopened. The documented workflow establishes functionality, but not ROUGE or factual-correctness performance.

F. Sentiment Analysis in Customer Conversations

Sentiment in support conversations may reflect frustration, satisfaction or neutral information-seeking, and can shift across successive turns. Resolv requests one of three labels—Positive, Neutral or Negative—from Gemini 2.5 Flash alongside the summary. Evaluation requires a defined unit of analysis (ticket or message), independently labelled examples, class frequencies, and an explicit annotation rubric. Accuracy alone may obscure minority-class errors; macro precision, recall and F1, a confusion matrix, and inter-annotator agreement where applicable are more informative. No measured classification score is available in the supplied project materials.

G. Research Gap and Comparative Positioning

The identified research gap is architectural integration rather than the absence of any single capability in prior work: multi-tenant isolation, official and session-based WhatsApp access, persistent worker coordination, automated ticket routing, SLA escalation and historically contextualized AI assistance must operate together under a consistent customer and workspace model. Resolv presents one documented integration of these concerns. Its differentiating design choices are a shared ticket lifecycle across two WhatsApp interfaces, dedicated session workers and tenant-scoped historical AI context. These claims describe the proposed system architecture; comparative superiority in cost, accuracy, throughput or reliability requires matched experiments and cannot be inferred from feature presence alone.

.

III. PROBLEM STATEMENT AND REQUIREMENTS

A. Problem Statement

Organizations that handle customer communication using messaging and support tools can run into problems like scattered conversation records, manual ticket creation unclear assignment of tasks, limited visibility for teams and weak reporting on operations. SaaS companies also need to make sure that one customer group cannot access the data of another customer group, including messages, tickets or settings. Keeping WhatsApp Web connections running all the time adds work because the system needs to manage long-lasting socket processes, which must be set up watched and fixed when needed. Also agents can waste a lot of time reading through long message histories before they can reply or take over an escalated ticket.

The research problem is therefore to create and build a platform that brings together WhatsApp communication, support records that know about different customers, automatic ticket processes, managing sessions across different areas real-time teamwork and AI that helps understand conversations while making sure there is a clear line, between what the system does and what has not been proven to work.

B. Functional Requirements

TABLE I
FUNCTIONAL REQUIREMENTS

ID	Requirement
FR-01	Authenticate users and enforce role- and workspace-scoped access.
FR-02	Provision workspaces and initialize default configuration, subscription, and assignment state.
FR-03	Receive and send WhatsApp Cloud API messages and process webhooks.
FR-04	Manage Baileys sessions using QR or phone-pairing authentication.
FR-05	Coordinate worker registration, heartbeat, session assignment, and request proxying.
FR-06	Create, categorize, assign, reassign, and resolve support tickets.
FR-07	Monitor SLA deadlines and escalate breached tickets.
FR-08	Deliver real-time message, ticket, and notification updates.
FR-09	Support team invitations, roles, account status, and agent availability.
FR-10	Generate AI summaries and sentiment labels on demand.
FR-11	Create WhatsApp campaigns and track recipient delivery states.
FR-12	Provide analytics, subscriptions, seat limits, and Super Admin operations.

C. Non-Functional Requirements

This design keeps tenants separate makes maintenance easy handles tasks asynchronously stays fast in time and ensures recovery. These are goals, not service-level guarantees. Security relies on authorization and tenant scoping, in all API routes, workers and socket events; ; availability and scalability require testing under representative load and failure conditions.

IV. SYSTEM ARCHITECTURE AND METHODOLOGY

A. Development Approach

The development approach is modular and integration-oriented. Next.js presents role-specific dashboards and ticket interactions; Express.js implements authorization and workflow rules; PostgreSQL with Prisma persists tenant and operational records; Redis and BullMQ coordinate events and delayed jobs; Socket.IO distributes live updates; and a dedicated Session Manager coordinates Baileys worker services. This separation allows persistent WhatsApp connections to be managed independently from request-serving API processes, although scalability benefits require controlled load testing.

B. Major Modules

TABLE II
PRINCIPAL MODULES AND RESPONSIBILITIES

Module	Principal responsibility
Workspace & identity	Tenant provisioning, authentication, authorization, and configuration.
WhatsApp Cloud API	Webhook ingestion, outbound messaging, templates, and delivery receipts.
Baileys session services	Persistent WhatsApp Web sessions and message synchronization.
Ticket management	Lifecycle, triage, assignment, collaboration, and SLA handling.
AI assistance	Contextual summarization and sentiment classification.
Notifications	Persistent user notifications and real-time delivery.
Team management	Invitations, roles, active status, and online availability.
Campaigns & analytics	Broadcast processing, delivery reports, ticket analytics, and uptime.

Module	Principal responsibility
Billing & Super Admin	GoCardless subscriptions, seats, feature entitlements, and global provisioning.

C. Technology Stack

The frontend uses Next.js 15 to deliver an agent dashboard. The frontend shows a ticket table where you can filter, search, toggle column visibility and drag-and-drop columns with @dnd-kit. The frontend also offers a ticket details drawer, WhatsApp. Group views, team administration, analytics, campaigns, billing and the Super Admin portal. The Baileys chat client caches message history in IndexedDB so the frontend can render quickly with an approach.

The backend uses Node.js and Express.js to create REST endpoints and handle business logic. The backend talks to PostgreSQL through Prisma ORM. Redis in the backend powers BullMQ. Pub/Sub. The backend stores media objects in storage such, as AWS S3 or MinIO.

TABLE III
TECHNOLOGY STACK

Layer	Technology	Role
Frontend	Next.js 15	Dashboards, ticketing, chat, administration
Backend	Node.js, Express.js	REST APIs and application logic
Persistence	PostgreSQL, Prisma	Relational storage and data access
Queue / cache / events	Redis, BullMQ	Background jobs and Pub/Sub
Real-time	Socket.IO	Live messages and notifications
Authentication	AWS Cognito, Better Auth	Identity and application sessions
AI	Gemini 2.5 Flash, @google/genai	Summary and sentiment generation
WhatsApp	Meta Cloud API, Baileys	Official API and multi-device messaging
Object storage	AWS S3 / MinIO	Media storage
Billing	GoCardless	BACS Direct Debit subscriptions

D. Layered Architecture

Fig. 1 Shows the documented services grouped into five parts: a presentation tier, an application tier, a session tier that keeps running WhatsApp sockets separate a data and messaging tier and external services. By keeping the Session Manager and worker nodes, from the API persistent socket state does not get in the way of handling requests.

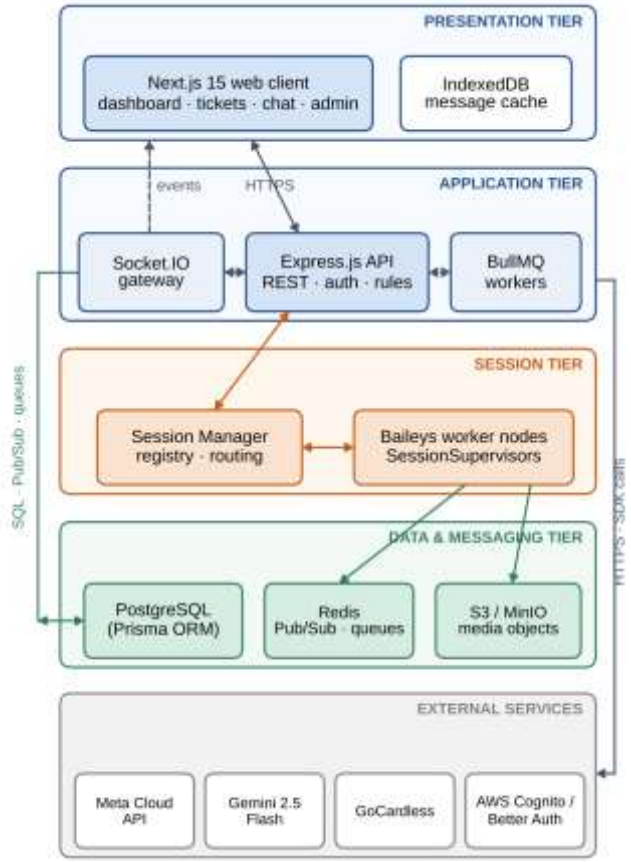


Fig. 1. Layered architecture of Resolv. Orange links denote session control, green links denote data and messaging, and the right-hand bus denotes calls to external services.

E. End-to-End Data Flow

A customer message can come in either through a Meta webhook or a Baileys worker (see Fig. 2). The message is linked to a workspace and to a customer stored and then given to the support workflow. The rules for tickets can pick a category and a description create a queued ticket and hand that ticket over to an agent who is able to help. Socket.IO sends updates to clients that are connected. Background queues run delayed checks, for service level agreements. Send out campaign messages. When the system needs it the AI service pulls the past ticket messages calls Gemini and stores the result in a structured form.

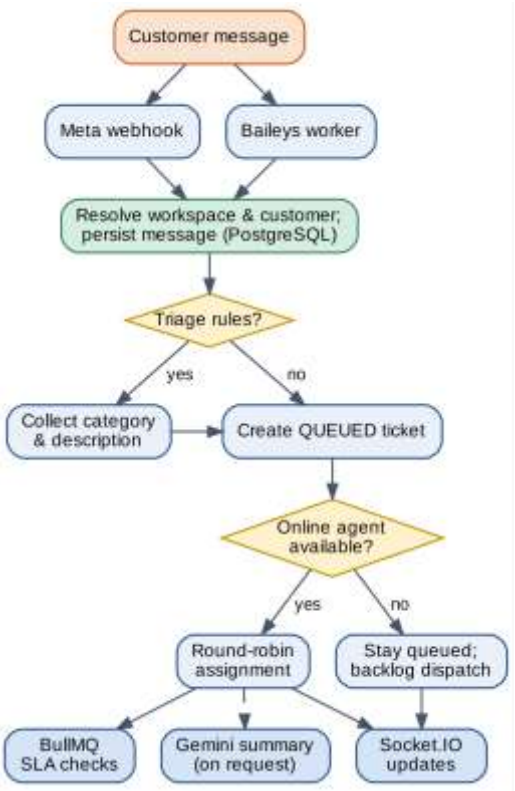


Fig. 2. End-to-end message and ticket data flow. Yellow diamonds are decision points; blue boxes are asynchronous or real-time services.

F. Notation

TABLE IV
NOTATION USED IN THE MATHEMATICAL MODELS

Symbol	Meaning
ω; ws(·)	Workspace (tenant) identifier; workspace of a record or user
ρ(u); P_a	Role of user *u*; set of roles permitted to perform action *a*
W; *w*	Set of worker nodes; a single worker
s_w; C_w	Active sessions on worker *w*; its maximum capacity
u_w; θ	CPU utilization of *w*; CPU threshold (75 %)
τ_w; T_{hb}	Last heartbeat time of *w*; heartbeat timeout (15 s)
Δ_{sweep}	Liveness-sweep interval (10 s)
d_0; β; d_{max}	Back-off base delay (3 s), growth factor, and cap (60 s)
τ_{SLA}	Workspace SLA threshold (default 3,600 s)
q_ω; U_ω	Purchased seats and current users of workspace ω

V. MULTI-TENANT WORKSPACE ARCHITECTURE

A. Logical Isolation

Resolv builds multi-tenancy on a shared PostgreSQL database using a workspace identifier called workspaceId. Records that belong to a tenant—such as users, customers, tickets, campaigns, configuration and subscription data—carry that identifier. When a user logs in the application limits queries to the records that match the user's workspace. For any record set R the tenant-scoped view for workspace ω is

$$Q(R) = r \in R : ws(r) = \omega$$

(1)An action a performed by user u on record r is allowed when the record belongs to the same workspace as the user and the user's role can carry out that action. The rule is written as

$$allow(u, a\ r) \Leftrightarrow ws(r) = ws(u) \wedge \rho(u) \in Pa$$

(2)Four roles exist: SUPER_ADMIN ADMIN, MANAGER and AGENT. Because tenant filtering happens inside the application every API route, worker job and socket event must keep the workspace scope. The documentation does not prove that database row-level security is, in place or that the system has passed penetration tests. You need to run tests to confirm that isolation works even when an attacker is involved.

B. Data Model

Fig. 3 Shows the tenant and operational entities. One workspace can own one subscription, one configuration and one round-robin state and it can also contain many users, customers, tickets and campaigns. A subscription holds its status, the seats that were bought and optional feature entitlements; Analytics and Campaigns are add-ons that are controlled through entitlements.

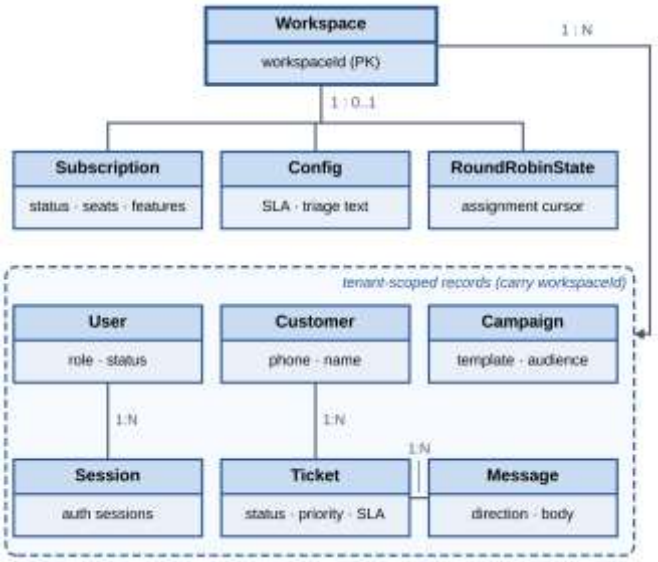


Fig. 3. Simplified tenant and operational entity relationships. The dashed region contains records scoped by the workspace identifier.

C. Workspace Provisioning

Workspaces are provisioned through two documented paths. When an administrator logs in for the time a Better Auth callback checks whether the administrator has a workspace and starts provisioning if needed. The provisioning transaction builds a Workspace links the administrator creates a default Config and a RoundRobinState record and creates an INCOMPLETE Subscription. The default SLA configuration for the Workspace is 3,600 seconds. The Super Admin portal allows enterprise provisioning (see Fig. 4). First the portal creates an AWS Cognito identity. Next the portal creates the workspace, the administrator, the subscription, the configuration and the round-robin state, in one database transaction. If that transaction fails the portal deletes the AWS

Cognito identity as a compensating rollback so that no orphaned identity remains.

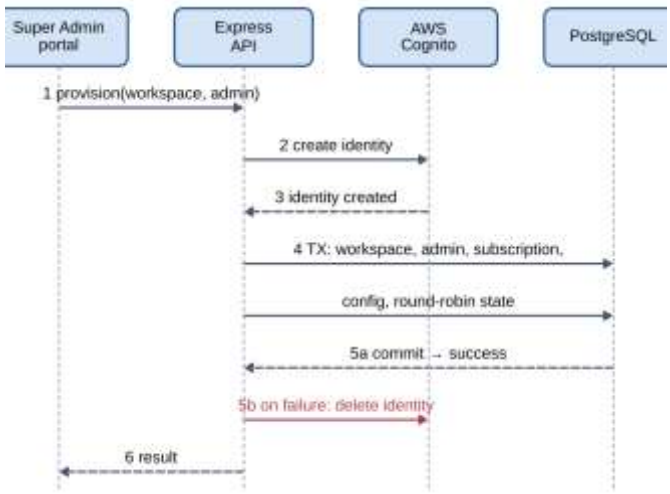


Fig. 4. Super Admin provisioning with compensating rollback of the Cognito identity when the database transaction fails.

VI. WHATSAPP INTEGRATION AND DISTRIBUTED WORKER ARCHITECTURE

A. Official WhatsApp Cloud API

The official integration gets events through a route for each workspace (/api/webhook/wa/:workspaceId) and sends messages using Meta's Graph API [8]. It can handle text, approved message templates, interactive buttons and lists and updates about message delivery. The application follows the rules for customer service messaging times. Uses approved templates when needed outside of those times.

B. Baileys Integration

The Baileys integration keeps WhatsApp Web sessions working across devices. To log in you can scan a QR code. Use an eight-digit code from your phone. The app keeps chats, contacts, groups, messages and media in sync; media files are stored in S3- storage. The chat screen can turn a message into a support ticket. Uses IndexedDB to save messages locally.

C. Session Manager and Worker Registration

A special Session Manager handles worker nodes. Each worker tells the system its name, where it can be reached and how sessions it can handle by sending a POST request to /workers/register. The worker also sends a signal every five seconds to show it is still working. The Session Manager checks every ten seconds to see if a worker is still active. If a worker does not send a signal for than fifteen seconds it is marked as not working. The health of a worker w at time t is

$$hw(t) = I(t - w \le Thb)\ Thb = 15\ s$$

where I(·) is the indicator function.

D. Placement Policy

When choosing a worker the system avoids workers that are not working or are full. It also follows the rule that workers should not use than 75% of their CPU. The group of workers that can be used at time t is

$$E(t) = w \in W : hw(t) = 1 \wedge sw < Cw \wedge uw \le \theta$$

and from this group the worker with the active sessions is picked:

$$w* = arg\ min_{w \in E(t)}\ sw$$

This is a way to assign work; the documentation does not compare this method with ways of balancing work. A record called WorkerAssignment connects each workspace to a worker and the Session Manager sends requests for a workspace, to that worker (Fig. 5)

E. Failure Detection and Recovery

Because heartbeats arrive every 5 seconds sweeps occur every Δsweep = 10 seconds. The unhealthy threshold is Thb = 15 seconds a worker that stops responding is detected at the first sweep after the timeout elapses.

The delay before failure detection called Tdet is limited by the configured intervals:

$$Thb < Tdet \le .\ Sweep \Rightarrow 15\ s < Tdet \le 25\ s$$

This bound comes from the configuration shown in Fig. 6A. It is a design property, not a recovery time. When a worker restarts, the assignments linked to the worker identifier help rebuild session supervisors. Session reconnection applies a back-off that begins at three seconds and is capped at sixty seconds. Let d0 be the delay β greater, than one be the growth factor and dmax be the cap

$dk = \min(d0 \cdot k, dmax)$ $d0 = 3$ s, $dmax = 60$ s

The documentation does not give the growth factor. Fig. 6B therefore shows a schedule using $\beta = 2$.

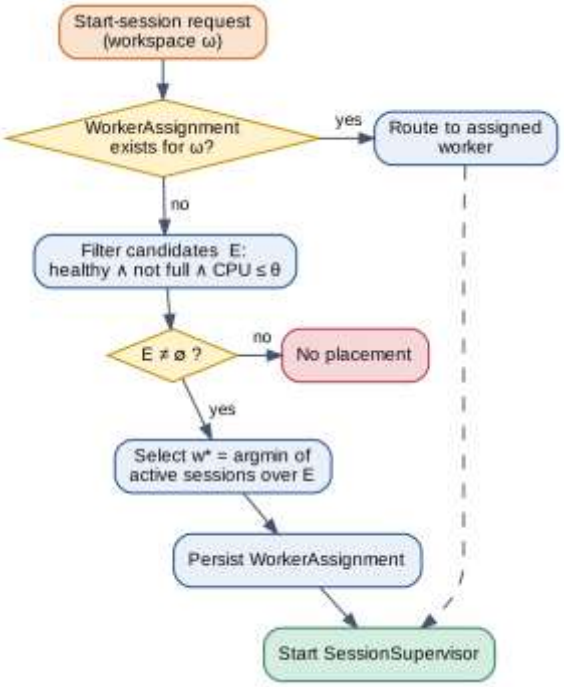


Fig. 5. Worker placement flow. An existing WorkerAssignment short-circuits selection; otherwise the least-loaded eligible worker is chosen and the assignment is persisted.

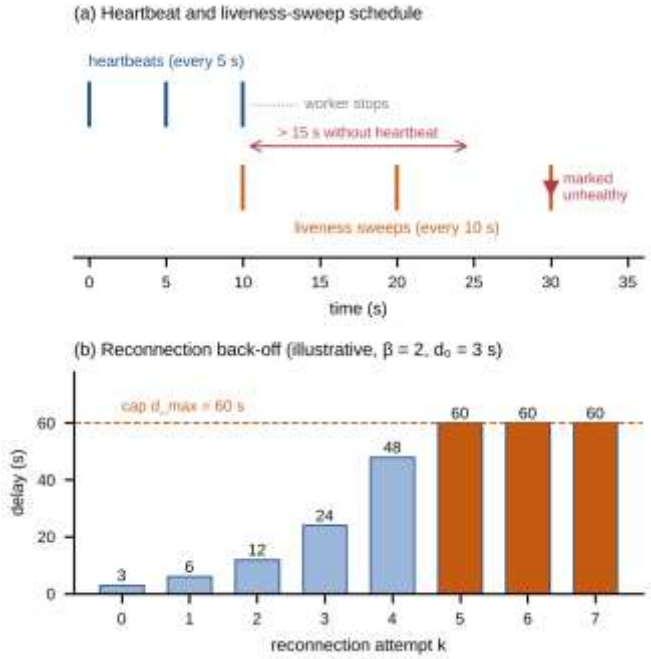


Fig. 6. (a) Heartbeat and sweep schedule implied by the configured intervals. (b) Illustrative reconnection back-off for β = 2 (the factor is assumed, not documented).

F. Distributed Event Flow

Redis Pub/Sub sends worker events to the API, which then passes them to the Socket.IO room (Fig. 7). The Session Manager keeps session coordination from the persistent Baileys socket execution. This means API instances do not store or manage WhatsApp socket state.

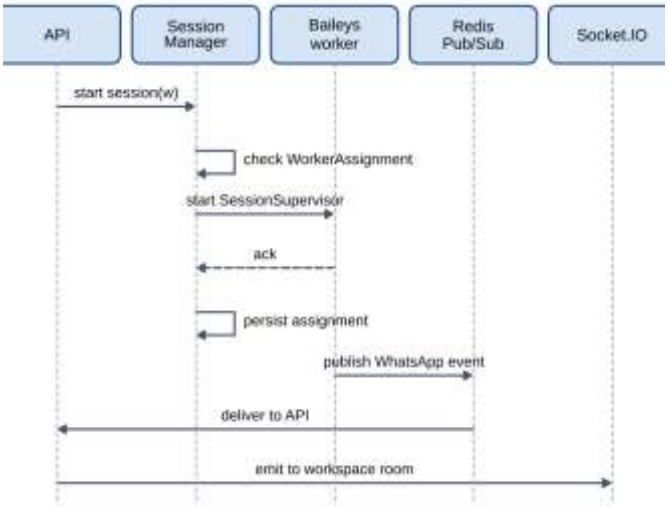


Fig. 7. Distributed session start and event flow between the API, Session Manager, worker, Redis Pub/Sub, and Socket.IO.

VII. AI-POWERED CUSTOMER SUPPORT FEATURES

Resolv integrates Google Gemini 2.5 Flash through the @google/genai SDK [9].AI processing starts when the user requests AI processing from the ticket interface not automatically for every messageThe documented capabilities are summarization and sentiment classification into Positive, Neutral or Negative.

A. Context Construction

For ticket j the pipeline retrieves all messages of ticket j and those of up to two tickets of the same customer excluding ticket j from the historical set. Writing Mj for the messages of ticket j and p1(j) p2(j) for the two recent prior tickets of that customer the context is

$$C_j = M_j \cup M_{p1(j)} \cup M_{p2(j)} \quad (8)$$

with messages formatted chronologically by direction, timestamp and body.The prompt requests a JSON response; the backend parses it and persists both fields on the ticket:

$$(s_j, \sigma_j) = G(C_j) \quad (9)$$

where G denotes the Google Gemini 2.5 Flash call sj is a summary targeted at fewer, than 100 words and σj ∈ {Positive, Neutral, Negative}. The expected output is:

```
{
  "summary": "Concise summary of the customer's
              issue and interaction.",
  "sentiment": "Negative"
}
```

B. Workflow and Safeguards

Persisted output can be retrieved later without invoking the model Fig. 8). If the API key is unavailable AI client initialization is handled so that the main API can still start

without activating AI functionality. No accuracy factuality or quality score is claimed, because no evaluation results were supplied.

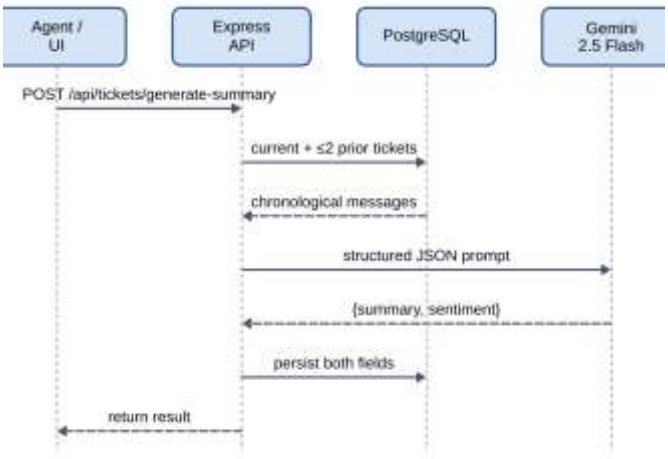


Fig. 8. On-demand AI summarization and sentiment workflow.

VIII. AUTOMATED TICKET MANAGEMENT AND WORKFLOW

A. Ticket Creation and Lifecycle

Tickets are created when people send WhatsApp messages or when a Baileys chat message is added to the support workflow. The official-API triage process works through the states AWAITING_TITLE AWAITING_DESCRIPTION QUEUED and ASSIGNED; the greeting, category choices and description prompt are all set by the workspace configuration. When a Baileys message arrives the system first finds the customer then checks the message identifier then creates the ticket stores the source message as inbound content and finally assigns the ticket either manually or, by a round-robin method; the identifier check stops duplicate tickets from being made. Tickets move through the states QUEUED, ASSIGNED, IN_PROGRESS, ON_HOLD and RESOLVED and each ticket can have a priority of LOW, MEDIUM, HIGH or URGENT.

B. Collaboration

Ticket collaboration allows public comments, private internal notes, voice notes, webcam video notes, attachments, teammate mentions references to related tickets and assignment audit information. Private notes stay separate from messages that the customer sees.

IX. REAL-TIME COMMUNICATION AND NOTIFICATIONS

Socket.IO gives two-way updates for messages, ticket changes, assignments, comments, campaign progress and WhatsApp connection status. Baileys worker events are sent via Redis Pub/Sub. Then forwarded by the main API. Workspace rooms share information at the tenant level while private rooms use the `user:${userId}` format for user-specific alerts. Notifications are stored in PostgreSQL with fields for recipient, type, title, message and metadata. The documented types are TICKET_ASSIGNED_ROUND_ROBIN, TICKET_ASSIGNED_MANAGER TICKET_TRANSFERRED TICKET_MENTIONED, TICKET_ESCALATED and SYSTEM. Users can see how many notifications are unread filter them mark one or all as read delete them and jump to the related ticket

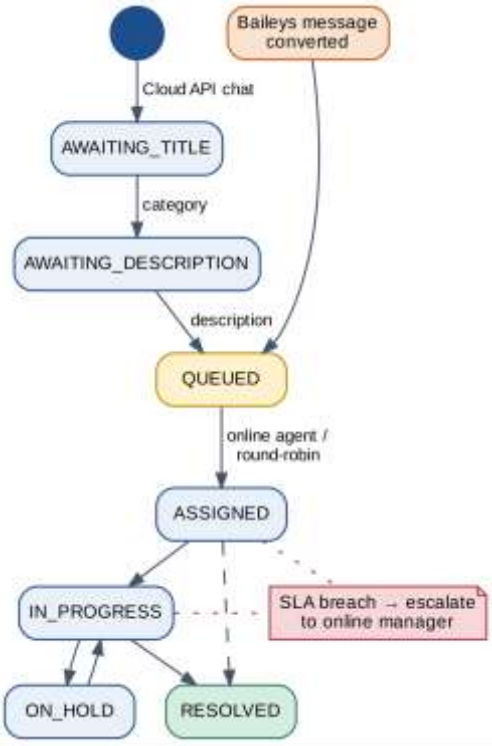


Fig. 9. Ticket lifecycle, including the interactive triage states of the Cloud API path and the SLA-escalation side path.

.

X. TEAM MANAGEMENT AND ACCESS CONTROL

Team onboarding happens by invitation only; there is no public sign-up endpoint that the documentation shows. An Admin or Manager sends an invitation that ties the invitee's email and role into a signed JWT, which can be shared by email, WhatsApp, Telegram or, by copying. When the invitee registers the email address is locked to the address that was invited. The backend checks the invitation checks if there are seats left and then creates the user and links them to the workspace. If U_ω is the set of users and q_ω is the number of seats purchased in workspace ω an invitation is accepted only if $|U| < q$. Managers cannot see or change Admin accounts; accounts can be activated, suspended or deleted. Agent /offline status is kept in UserSession with start and end timestamps, which also helps with uptime reporting.

TABLE V
ROLES AND DOCUMENTED RESPONSIBILITIES

Role	Documented responsibility
SUPER_ADMIN	Global tenant provisioning, workspace overview, seat allocation, and feature entitlement management.
ADMIN	Workspace administration, billing, settings, and team management.
MANAGER	Ticket supervision, reassignment, SLA oversight, and team coordination.

Role	Documented responsibility
AGENT	Assigned customer conversations, ticket handling, and collaboration.

XI. CAMPAIGNS, ANALYTICS, AND BILLING

A. Campaign Management

Campaigns use the official WhatsApp Cloud API and approved templates. Administrators and Managers select a template and language, upload a CSV audience or enter phone numbers, validate international numbers, remove duplicates, and map template variables (Fig. 10). Before launch the backend checks that the recipient set R_c fits the available messaging quota Q_ω:

$$|R_c| \leq Q_\omega \tag{14}$$

Recipient messages are placed in BullMQ for background dispatch; the worker sends template payloads through the Meta Graph API and stores the returned message identifiers. Webhook events update each recipient to SENT, DELIVERED, READ, or FAILED, and Socket.IO broadcasts progress and aggregate counters.

B. Analytics

The Analytics add-on reports ticket category distribution, SLA breaches attributed to agents, and agent uptime derived from UserSession intervals $[s_k, e_k]$. Over a reporting window $[T_0, T_1]$, the uptime of agent a is

$$Up_a = \frac{\sum_k (e_k - s_k)}{T_1 - T_0} \tag{15}$$

with intervals clipped to the window. Date-range and agent filters support the reporting views. Analytics and Campaigns are feature-gated by workspace entitlements.

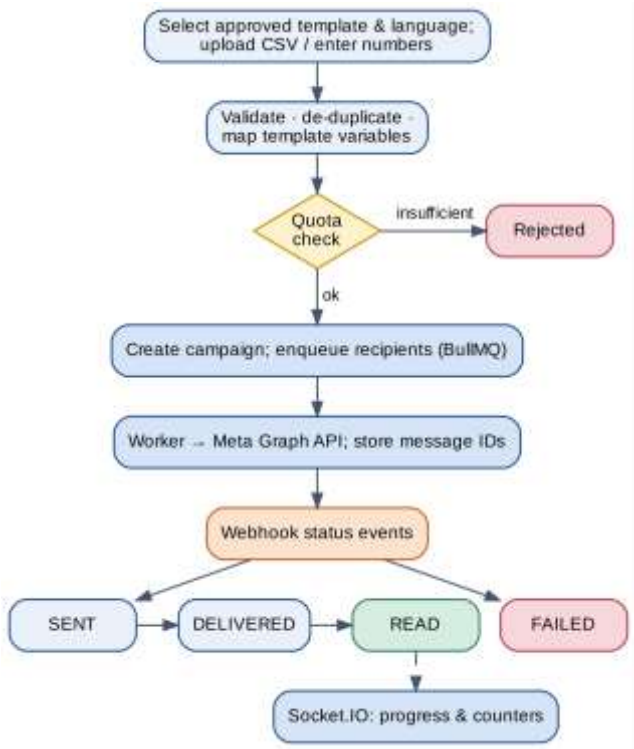


Fig. 10. Campaign pipeline from audience preparation to webhook-driven delivery states.

XII. IMPLEMENTATION DETAILS

The implementation is organized into a Next.js web application, an Express API, a Session Manager, and a WhatsApp worker service. Prisma provides typed access to PostgreSQL. The principal entities are Workspace, Subscription, User/Session/Account, Customer/Ticket/Message, Comment/Notification, Config/RoundRobinState, WhatsAppSession/WhatsAppSignalKey, WorkerAssignment, ContactTag, UserSession, and Campaign/CampaignMessage. Table VI lists representative API operations; it is not a complete specification.

TABLE VI
REPRESENTATIVE API OPERATIONS

Endpoint	Method	Purpose
/api/webhook/wa/:workspaceId	POST	Receive WhatsApp Cloud API events.
/api/tickets/generate-summary	POST	Generate ticket summary and sentiment.
/api/assign-from-chat	POST	Convert a Baileys message into a ticket.
/api/notifications	GET	Retrieve user notifications.
/api/team/invite	POST	Create a team invitation.
/api/analytics	GET	Retrieve operational analytics.
/api/campaigns/launch	POST	Launch a WhatsApp campaign.
/api/billing/checkout	POST	Initiate GoCardless authorization.
/api/billing/verify	POST	Verify billing authorization.
/api/admin/workspaces	GET/POST	List or provision workspaces.

Error handling includes authentication and authorization rejection, transaction rollback, duplicate-ticket protection, webhook signature verification, worker health checks, exponential reconnection, and graceful session cleanup.

XIII. TESTING AND EVALUATION METHODOLOGY

The project team reports successful functional operation of the principal modules. These reported passes are distinguished from independently reproducible test logs and quantitative performance measurements, which were not supplied. This section specifies the conditions required to reproduce the checks and measure performance and AI quality.

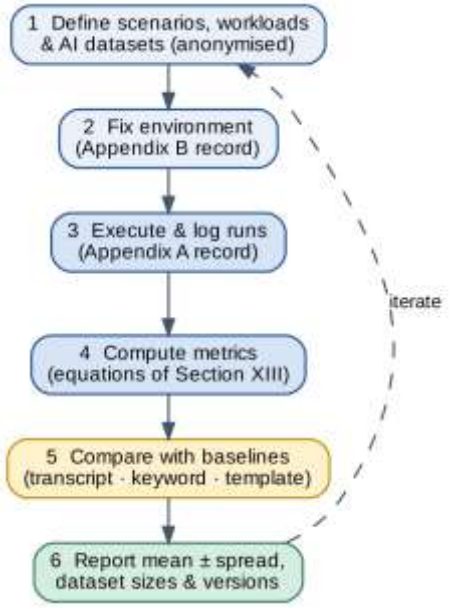


Fig. 12. Evaluation workflow: from scenario definition to baseline comparison and reporting.

A. Functional Test Cases

Functional verification should be performed in a controlled test workspace with test accounts and WhatsApp test connections, recording the date, build identifier, environment, input, observed output, pass/fail status, and evidence reference. Destructive logout and credential-cleanup tests should use non-production sessions.

TABLE VII
FUNCTIONAL VERIFICATION SCENARIOS

Area	Scenario	Expected behaviour	Status
RBAC	Manager accesses Admin account	Admin account cannot be viewed, edited, or deleted by a Manager.	Passed
Invitation	Reuse invitation token	Previously used token is rejected.	Passed
Auth	QR login	Connection established; status changes to `connected`.	Passed
Auth	Phone-pairing login	Connection established using phone pairing.	Passed
Messaging	Duplicate incoming message ID	Duplicate detected; processed only once.	Passed
Messaging	Outgoing send fails	Pending message removed; error displayed.	Passed
Ticketing	Convert same message twice	Second conversion is rejected.	Passed
Allocation	Eligible agent exists	Ticket assigned through the allocation mechanism.	Passed
Allocation	No eligible agent	Ticket remains queued until an agent is eligible.	Passed
Logout	Disconnect WhatsApp	Redis, authentication records, and IndexedDB data are cleared.	Passed (reported)
Real-time	New ticket created	Dashboard updates without a page refresh.	Passed (reported)

The expected-behaviour column describes the acceptance criterion. Passed entries reflect project-team-reported functional checks; no execution logs or environment identifiers were provided for independent verification.

B. Performance Measurements

Evaluation protocol: record the hardware, runtime, test date, number of tenants and online agents, message rate, payload sizes, run duration and at least repeated trials. Report mean and 95th-percentile delivery latency, delivery success, throughput, worker detection/recovery time and allocation fairness. A performance claim requires observed measurements and a comparable baseline.

Performance testing should use a fixed offered message rate and a documented workload. Reports must include hardware, operating system, runtime versions, database and Redis configuration, network conditions, tenant and agent counts, payload size, concurrency, and duration, and each experiment should be repeated enough to report central tendency and variability. For N successfully observed messages with send time $t_{s,i}$ and delivery-observation time $t_{r,i}$, the per-message latency is $L_i = t_{r,i} - t_{s,i}$, with mean, sample standard deviation, and 95th percentile

$$L_{avg} = \frac{1}{N}\sum_{i=1}^{N} (t_{r,i} - t_{s,i}) \tag{17}$$

$$s_L = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N} (L_i - L_{avg})^2} \tag{18}$$

$$L_{95} = L_{(\lceil 0.95\,N \rceil)},\ L_{(1)} \le \ldots \le L_{(N)} \tag{19}$$

Delivery success and cache effectiveness are ratios over fixed batches or workloads:

$$SR = \frac{n_{ok}}{n_{att}} \tag{20}$$

$$H = \frac{n_{hit}}{n_{hit} + n_{miss}} \tag{21}$$

For allocation fairness, let x_a be the tickets assigned to eligible agent a among n eligible agents over N assignments. Jain's index is

$$J = \frac{(\sum_{a=1}^{n} x_a)^2}{n \sum_{a=1}^{n} x_a^2} \tag{22}$$

where $1/n \le J \le 1$ and $J = 1$ indicates perfectly even allocation. The report must state which agents were eligible,

their online intervals, and whether special routing was enabled, and must distinguish browser (IndexedDB) cache hits from Redis cache hits.

C. AI Evaluation Plan

AI evaluation should use anonymized customer-support data and a documented annotation protocol. Summaries are assessed against reference summaries written by trained agents using ROUGE [7],

$$ROUGE - N = \frac{\sum_{g \in ref} min(c_{cand}(g), c_{ref}(g))}{\sum_{g \in ref} c_{ref}(g)} \quad (23)$$

complemented by human ratings, since lexical overlap alone does not establish factual correctness. Sentiment classification over $K = 3$ classes is assessed by per-class precision P_c, recall R_c, and macro-averaged F1,

$$F1_c = \frac{2 P_c R_c}{P_c + R_c} \quad (24)$$

$$F1_{macro} = \frac{1}{K} \sum_{c=1}^{K} F1_c \quad (25)$$

which avoids hiding poor performance on under-represented classes. With multiple annotators, inter-annotator agreement is reported via Cohen's kappa,

$$\kappa = \frac{p_o - p_e}{1 - p_e} \quad (26)$$

where p_o is observed and p_e chance agreement. If a retrieval component is part of the tested build, precision and recall at K are

$$P@K = \frac{|Rel \cap Top - K|}{K} \quad (27)$$

$$R@K = \frac{|Rel \cap Top - K|}{|Rel|} \quad (28)$$

Retrieval and draft-quality rows are *proposed procedures* applicable only if those components exist in the tested build; they are not evidence that retrieval or draft generation is implemented. Each component should be compared with a simple baseline—an unsummarized transcript, a keyword-based sentiment rule, keyword search, and a template reply—and results should include dataset size, sampling method, train/test separation, annotation instructions, class distribution, and model/prompt version. Human evaluation should use a consistent rubric and, where practical, blinded review.

TABLE IX
AI EVALUATION (PROPOSED)

Component	Dataset	Metrics	Result
Summarization	Anonymized ticket transcripts with agent-written reference summaries.	ROUGE-1/2/L [7]; human ratings of correctness, completeness, readability.	Functional check: Passed; quantitative scores not measured
Sentiment	Labelled ticket messages (positive, neutral, negative).	Accuracy, macro P/R/F1, confusion matrix, κ.	Functional check: Passed; quantitative scores not measured

XIV. RESULTS AND DISCUSSION

A. Verified and Measured Outcomes

The project team reports that the functional verification scenarios operate as expected, including authentication, duplicate-message handling, ticket creation, agent allocation and AI response generation. The supplied records do not include independently inspectable test logs or a full reproducible test environment; these outcomes are therefore described as team-reported verification. The presence of a valid Gemini summary or sentiment label is not a measurement of model quality. No benchmark dataset, human reference summaries, labelled sentiment set, measured latency, message throughput, concurrent-session capacity or recovery-time distribution was supplied. Quantitative performance and AI-quality results remain pending and must not be reported as numerical findings.

B. Hypotheses to Be Tested

- The design points to three effects that need to be tested through experiments; these are ideas, not actual results.
- H1. Using a cache first when showing things makes it faster to open conversations that were already synced: the time it takes to show something when using a cache is less than when not using a cache.
- H2. Checking message identifiers stops the message from creating more than one ticket when it is turned into a ticket more, than once: each message creates exactly one ticket.
- H3. Using AI summaries that fit the situation helps agents understand tickets faster as long as the summaries are accurate and cover everything needed.

C. Qualitative Architectural Discussion

The architecture brings tenant-scoped records, dual WhatsApp integration, distributed session coordination, ticket automation, real-time collaboration and administrative workflows into one application. Workspace identifiers provide the logical isolation mechanism; the Session Manager separates session coordination from persistent socket execution; BullMQ

decouples delayed and campaign work from synchronous requests; and Socket.IO with Redis Pub/Sub supports event propagation. These are documented design choices; their scalability, reliability and security properties remain subjects, for further testing.

XV. ADVANTAGES AND LIMITATIONS

A. Advantages

- Centralizes WhatsApp communication and support ticket workflows in a workspace-oriented application.
- Supports both the Cloud API and Baileys-based sessions.
- Separates session coordination from worker execution through a Session Manager.
- Automates triage, agent round-robin assignment, backlog dispatch and SLA escalation.
- Provides AI summaries and sentiment labels on demand.
- Supports real-time updates, private notifications, internal notes, mentions and audit information.
- Integrates team administration, campaigns, analytics, billing and global workspace provisioning.

B. Limitations and Evaluation Needs

Logical tenant isolation depends on workspace scoping across API controllers, background jobs and socket events. The Session Manager's in-memory worker registry requires evaluation for service restart and multi-instance deployment. Worker recovery and reconnection are documented,. Recovery times and behavior under simultaneous failures have not been measured. AI summaries and sentiment labels may contain errors. No empirical quality results are available. The platform depends on services—Meta, Google, AWS Cognito and GoCardless—and its performance, security, concurrency and usability have not been quantitatively established.

XVI. FUTURE WORK

- Evaluate AI ticket classification and priority recommendations using labelled support data.
- Investigate analytics for ticket volume SLA risk and recurring issues.
- Extend triage with governed chatbot automation and human handoff.
- Add channels such as email, SMS and website live chat if required by product scope.
- Evaluate replicated worker registries, automated worker replacement, multi-region deployment and distributed tracing.
- Benchmark API latency, message throughput, concurrent sessions, worker resource use and recovery time.
- Develop an AI evaluation dataset and human-review protocol.

XVII. CONCLUSION

Resolv demonstrates an integrated design for WhatsApp-centred multi-tenant customer support. Its architecture unifies Meta Cloud API and Baileys session traffic within a common ticket workflow, uses workspace-scoped access rules, delegates persistent sessions to a Session Manager and workers, and combines online-agent allocation, SLA escalation, real-time notifications and Gemini-assisted ticket context. The contribution lies in the coordinated implementation and documented models of these components, not in inventing their underlying algorithms. Functional scenarios are reported as passing by the project team; objective claims about security assurance, scalability, response latency, recovery performance and AI accuracy require additional controlled experiments. Future evaluation should publish reproducible workloads, labelled AI test sets, baseline comparisons and raw measurements.

REFERENCES

[1]W. J. Reinartz, M. Krafft, and W. D. Hoyer, "The customer relationship management process: Its measurement and impact on performance," *J. Marketing Res.*, vol. 41, no. 3, pp. 293–305, 2004, doi: 10.1509/jmkr.41.3.293.35991.

[2]J. Kabbedijk, C. P. Bezemer, S. Jansen, and A. Zaidman, "Defining multi-tenancy: A systematic mapping study on the academic and the industrial perspective," *J. Syst. Softw.*, vol. 100, pp. 139–148, 2015, doi: 10.1016/j.jss.2014.10.034.

[3]L. C. Ochei, J. M. Bass, and A. Petrovski, "Degrees of tenant isolation for cloud-hosted software services: A cross-case analysis," *J. Cloud Comput.*, vol. 7, art. 22, 2018, doi: 10.1186/s13677-018-0121-8.

[4]G. Feigenblat *et al.*, "TWEETSUMM—A dialog summarization dataset for customer service," in *Findings Assoc. Comput. Linguistics: EMNLP 2021*, 2021, pp. 245–260, doi: 10.18653/v1/2021.findings-emnlp.24.

[5]Socket.IO, "Introduction," Socket.IO Documentation. [Online]. Available: https://socket.io/docs/v4/. Accessed: Nov. 19, 2026.

[6]Taskforce.sh, "BullMQ Documentation." [Online]. Available: https://docs.bullmq.io/. Accessed: Nov. 19, 2026.

[7]C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out: Proc. ACL-04 Workshop*, 2004, pp. 74–81.

[8]Meta Platforms, "WhatsApp Cloud API Documentation." [Online]. Available: https://developers.facebook.com/docs/whatsapp/cloud-api/. Accessed: Nov. 19, 2026.

[9]Google, "Gemini API Documentation." [Online]. Available: https://ai.google.dev/gemini-api/docs. Accessed: Nov. 19, 2026.

[10]Prisma, "Prisma ORM Documentation." [Online]. Available: https://www.prisma.io/docs. Accessed: Nov. 19, 2026.

[11]Redis, "Redis Documentation." [Online]. Available: https://redis.io/docs/latest/. Accessed: Nov. 19, 2026.

[12]GoCardless, "GoCardless API Documentation." [Online]. Available: https://developer.gocardless.com/. Accessed: Nov. 19, 2026.