Trail AI: A Delivery Assurance and Evidence-Based Audit System for Software Agencies

Capstone Project Thesis Document

Student Name: Jhonn Vincent Arcipe

Student ID: BDSE-0922-114

Institution: Lithan EduClaas

Program: Lithan Software Engineering Program

Date: January 12, 2026

Executive Summary

Trail AI is an event-driven middleware system designed to bridge the "Execution Gap" in software development agencies—the persistent disconnect between communication (Slack), project tracking (Jira), and code execution (GitHub). Unlike passive integration tools that merely link data, Trail AI generates **"Proof Packets"**: tamper-evident, hash-chained narratives that verify the complete lifecycle of a task from initial commitment through execution to final authorization.

The system implements three core innovations:

1. **Passive Handshake Protocol:** Automatic detection of task acceptance via Jira/GitHub signals with "Explicit Rejection" capability, creating an authoritative record of intent without manual developer overhead.
2. **Optimistic Closure Protocol:** Evidence-based closure proposals (PR merged + approvals + CI passing) that auto-finalize after a set verification window (e.g., 24 hours) unless a human manager intervenes.
3. **AI-Enhanced Proof Packets:** Exportable audit artifacts containing Gemini-generated,

client-friendly summaries of technical work, designed for agency client reporting, billing verification, and dispute resolution.

Built using a modern technology stack (Bun, Hono, Drizzle ORM, Supabase, and Gemini AI), Trail AI is architected for distribution via the GitHub Marketplace and Atlassian Marketplace. The frontend is deployed on Vercel for global CDN performance, while the backend runs on Fly.io or Railway for cost-effective serverless execution. This thesis documents the system design, implementation plan, evaluation methodology, and market validation strategy.

Table of Contents

1. Introduction

1.1 Context

Software development agencies operate in a distributed, tool-fragmented environment. Teams use Slack for synchronous communication, Jira for project tracking, and GitHub for version control and continuous integration. Each tool maintains partial state, and no single system captures the complete narrative of task execution from responsibility acceptance to final delivery authorization.

This fragmentation creates what we term the "Execution Gap": a systematic inability to provide stakeholders (clients, project managers, auditors) with a verifiable, unified narrative of delivery. The consequences are measurable:

- **Billing disputes:** Clients dispute completion because they lack empirical evidence beyond status updates.
- **Administrative overhead:** Managers spend ~20% of their time context-switching between tools to verify status.
- **Audit liability:** When disputes arise, there is no single source of truth that captures both the commitment handshake and the final authorization decision.
- **Implicit accountability:** Tasks assigned in Slack often lack explicit acceptance, resulting

in misaligned expectations.

1.2 Proposed Solution

Trail AI transforms project management from an administrative burden into a high-value delivery assurance service by:

1. **Capturing Intent (Passive Handshake):** Automatically detecting when work begins (e.g., Jira moves to "In Progress") to log a formal start time, while offering developers an easy "Reject" option if the assignment is incorrect.
2. **Automating Evidence:** Collecting GitHub signals (PR merge, approvals, CI status).
3. **Optimistic Finalization:** Proposing closure and auto-approving if policy is met after a set window (24h), removing manager bottlenecks while preserving veto power.
4. **Generating Proof Packets:** Creating shareable, tamper-evident delivery receipts enriched with AI summaries.

1.3 Key Innovation Claims

- **Deterministic Evidence:** Closure eligibility is anchored to objective GitHub signals (PR merged, approvals, CI pass), not probabilistic AI judgments.
- **Tamper-Evident Audit Trail:** Hash-chained event logs ensure integrity of historical records.
- **Client-Ready Narratives:** Gemini AI transforms technical commit logs into business-readable release notes.
- **Optimistic Human Gate:** Final closure decisions assume success to reduce friction but remain under human control via a veto mechanism, preserving organizational trust.

1.4 Project Scope

In Scope (MVP):

- Slack Bolt App for task notifications and intervention commands.
- GitHub App for event capture (PR merge, approvals, CI status).
- Jira REST API integration for bi-directional status sync.
- Hash-chained event logging for data integrity.
- 3-tier policy evaluation (Agile, Standard, Hardened).
- **Optimistic Closure Engine** (Auto-close timer via pg-boss).
- Proof Packet generation (Web, PDF, JSON export) with **Gemini AI Summaries**.

Out of Scope (Future Releases):

- Manual time tracking/stopwatch.
- Individual performance scoring.
- Code quality assessment or grading.
- Vision-based screenshot verification (moved to v1.1).

2. Problem Statement

2.1 The Client Trust Gap

Software agencies deliver custom work to non-technical clients (marketing departments, product companies, startups). These clients need evidence that work was completed, tested, and authorized before payment. Currently, agencies provide:

- Informal email updates ("The login feature is done").
- Screenshots or demo videos (difficult to share securely; lack authoritative metadata).
- Jira board access (overwhelming and non-technical).

Result: Billing disputes, client requests for "proof," and loss of trust.

2.2 Context-Switching & Administrative Debt

Project managers and team leads verify task status by manually checking Slack, Jira, and GitHub. This context-switching adds cognitive load and increases the risk of stale data. Trail AI solves this by automating the state transitions based on actual work activity.

2.3 The Audit Black Hole

Standard integrations show *what* changed, but not *who* authorized it or *when* the responsibility was accepted. Trail AI fills this gap with the "Handshake" and "Closure" events.

- **Commitment Handshake:** When did the developer formally accept responsibility?
- **Formal Handover:** Who authorized the final delivery? Was it auto-closed due to policy compliance?
- **Override History:** If closure was vetoed, why?

2.4 Implicit Accountability

Tasks assigned in Slack often lack formal acceptance. Trail AI formalizes this by listening for work-start triggers (like moving a Jira card) and logging it as a formal commitment, giving the developer a chance to "Reject" if the assignment was a mistake.

2.5 Problem Summary Table

Problem	Impact	Current Tools	Trail AI Solution
Client Trust Gap	Billing disputes, payment delays	Email + Screenshots	**Proof Packets** (shareable, tamper-evident, AI-summarized)
Context-Switching	Manager time waste, stale Jira boards	Manual checking + updating	**Auto-sync via webhooks** + Optimistic Closure

Audit Black Hole	Dispute liability, missing records	Jira activity + PR logs (disconnected)	**Hash-chained event log** + Override taxonomy
Implicit Accountability	Misaligned expectations, missed deadlines	Slack conversation (informal)	**Passive Handshake** (Formal intent logging)

3. Related Work & Competitive Analysis

3.1 Existing Integration Approaches

3.1.1 Atlassian Jira GitHub Integration

- **What it does:** Surfaces GitHub development info in Jira.
- **Limitations:** Shows what was pushed, but not who accepted responsibility. No formal approval or override taxonomy. Does not produce a shareable, client-ready narrative.

3.1.2 LinearB/gitStream

- **What it does:** PR workflow automation and engineering metrics.
- **Limitations:** Focused on PR throughput and internal efficiency, not delivery narratives or agency-client trust. No client-shareable receipts.

3.1.3 Workato/Workbot for Slack

- **What it does:** Low-code integration platform.
- **Limitations:** Requires non-technical users to build complex workflows. No opinionated "delivery assurance" flow. No tamper-evident guarantees.

3.2 Comparative Analysis (Refined)

Dimension	**Jira/GitHub**	**gitStream**	**Trail AI (Revised)**
Workflow Friction	High (Manual Updates)	Low (CI Automation)	**Zero (Passive Detection)**
Closure Logic	Manual	Rule-based	**Optimistic + Human Veto**
Client Value	None (Internal Only)	None (Internal Only)	**AI Summaries + Proof Packets**
Audit Integrity	Mutable Logs	Log Files	**Hash-Chained**

			Ledger
Primary Use Case	Engineering Ops	Code Velocity	**Agency/Client Trust**

3.3 Market Gap

Existing tools optimize for internal engineering efficiency (reducing PR review time). Trail AI targets a different job-to-be-done: **external trust and billing verification** for agencies. This gap is not addressed by LinearB or Jira Automation because their target buyers are internal engineering VPs, not Agency Owners dealing with billing disputes.

4. Research Methodology

4.1 Philosophy & Approach

Following Saunders' Research Onion framework, Trail AI employs a **Design Science** research strategy with mixed methods validation. The philosophy is **Pragmatism**: focusing on "what works" to solve the practical problem of delivery transparency.

4.2 Validation Plan

The project will be validated through a structured pilot with 2-3 software studios.

Success Criteria:

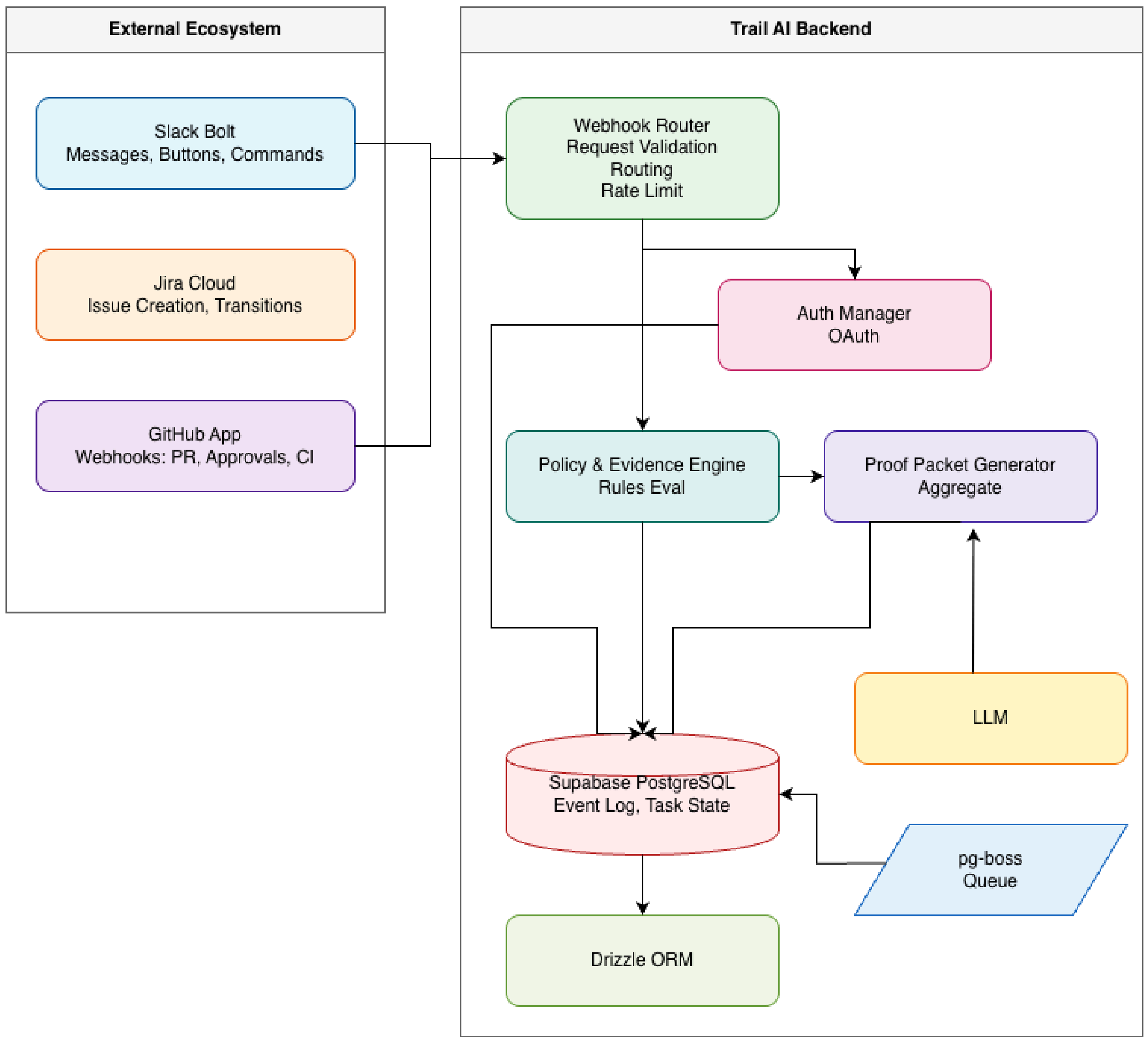
- **Evidence Density:** $\geq$ of Jira tickets closed with an associated Proof Packet.
- **Intervention Rate:** < 10% of auto-closures require manager override (proving the optimistic model works).
- **Adoption Friction:** Zero manual "Start Task" clicks required (measured via system logs).
- **Business Impact:** Reduction in client "status clarification" requests.

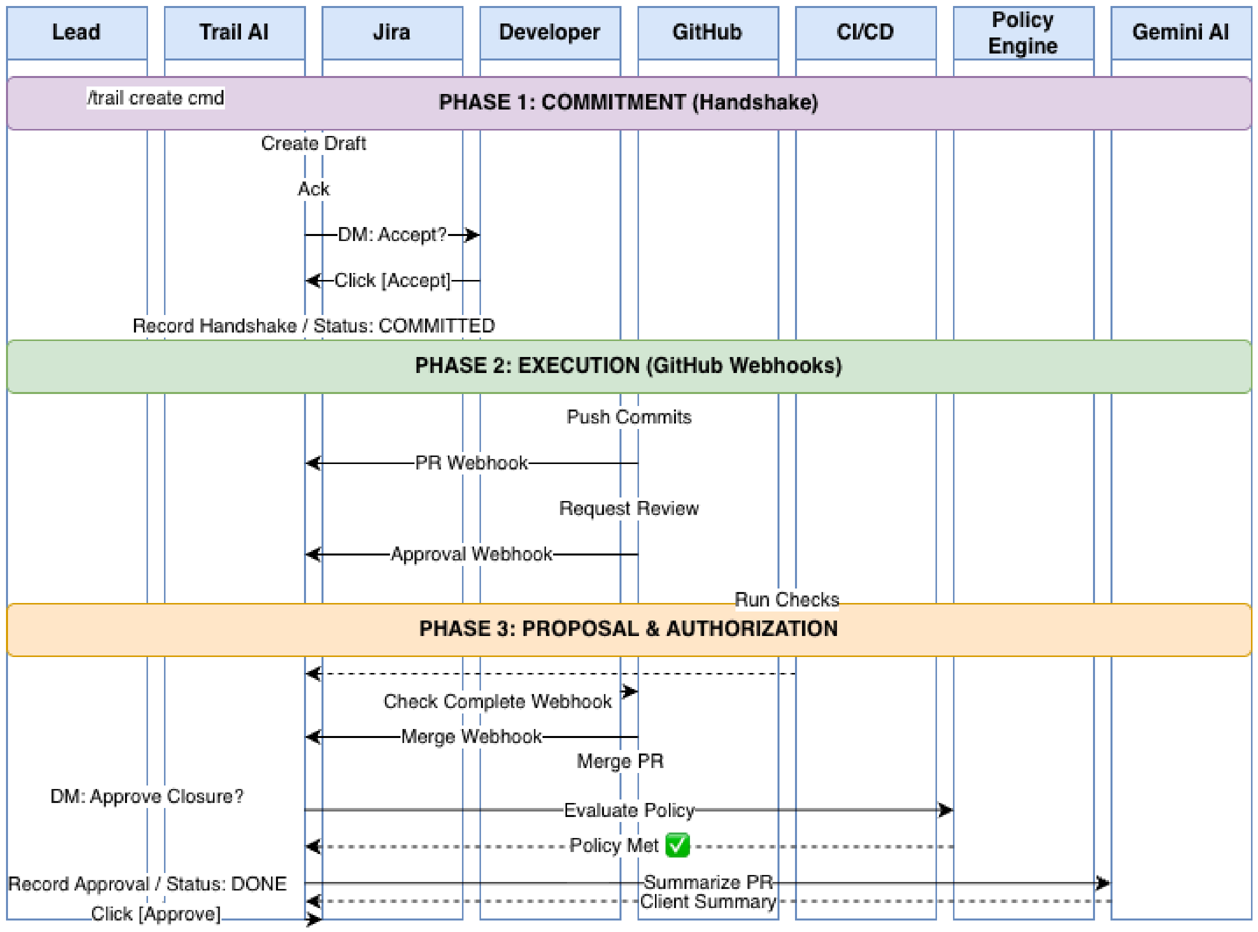
5. System Design & Architecture

5.1 Architectural Principles

- **P1. Passive Ingestion:** The system listens to existing tools (Jira/GitHub) rather than demanding new user inputs.
- **P2. Optimistic Trust:** Assume the developer is correct if all CI/Policy checks pass; allow managers to veto, rather than blocking for approval.
- **P3. Tamper-Evident Audit Trail:** The event log is append-only with hash-chaining. Any attempt to delete or modify a historical entry breaks the cryptographic chain.
- **P4. Privacy by Design:** The system processes metadata only (URLs, IDs, timestamps, status strings). No proprietary source code is ingested.

5.2 System Architecture Diagram





5.3 End-to-End Event Flow

1. **PHASE 1: PASSIVE COMMITMENT (Handshake)**
   - **Trigger:** Developer moves Jira Ticket to "In Progress".
   - **Trail AI Action:** Logs "Handshake" event using trigger_source: jira_webhook.
   - **Notification:** Slack DM to Dev: "Tracking started for **[Task Name]**. Click **[Reject]** if this is an error."
2. **PHASE 2: EXECUTION**
   - **Trigger:** GitHub PR Merged + CI Passed.
   - **Trail AI Action:** Verifies Policy (Tier 1/2/3). Aggregates Evidence.
3. **PHASE 3: OPTIMISTIC CLOSURE**
   - **Action:** System drafts Proof Packet and schedules auto_close job (24h delay).
   - **Notification:** Slack message to Lead: "Task Complete. Auto-closing in 24h. Click **[Veto]** to stop."
   - **Result:** If timer expires, Status updates to **CLOSED**.

5.4 Core Components

5.4.1 Webhook Router

Ingests and routes incoming events. Validates request signatures using **HMAC-SHA256**.

- **Slack Verification:** Verifies HMAC-SHA256(signing_secret, request_body) matches the signature header.
- **GitHub Verification:** Verifies HMAC-SHA256(secret, payload) matches the X-Hub-Signature-256 header.

5.4.2 Auth Manager

Manages OAuth tokens for Slack, GitHub, and Jira. Uses jose library to encrypt tokens at rest in Postgres (AES-256).

5.4.3 Policy & Optimistic Engine

Evaluates closure criteria. If Policy Met (PR Merged + CI Pass), it schedules a auto_close job via pg-boss for 24 hours in the future. If a Manager clicks "Veto" in Slack, the job is cancelled.

5.4.4 Proof Packet Generator

Aggregates metadata into a structured JSON artifact. Includes the **Hash Chain Visualization** logic to allow frontend verification of data integrity.

5.4.5 Gemini AI Integration

- **Function:** Summarizes PR description and commit messages into a 1-2 sentence client-facing summary.
- **Example:** "The login timeout issue was resolved by updating the session expiration logic. Tested with concurrent users."
- **Guardrails:** Gemini output is generated with temperature=0 for determinism.

5.5 Data Integrity: Hash-Chained Event Log

Every event is stored with a prev_hash pointing to the event_hash of the preceding record. This creates a blockchain-like structure where modifying a past event invalidates all subsequent hashes, making tampering immediately detectable on the Proof Packet viewer.

5.6 User Stories

The system design is driven by three primary user personas:

1. The Developer (Alex)

- *Goal:* Wants to focus on code, not Jira updates.
- *User Story:* "As a Developer, I want my task status to update automatically when I move a card in Jira or merge a PR, so that I don't have to manually update three different tools."
- *Acceptance Criteria:* Moving Jira card to 'In Progress' triggers Slack notification; Merging PR moves Jira card to 'Done' (pending closure).

2. The Team Lead (Sarah)

- *Goal:* Wants to ensure quality without becoming a bottleneck.
- *User Story:* "As a Team Lead, I want completed tasks to close automatically if they pass all tests, but I want the ability to veto them if I see a non-technical issue."
- *Acceptance Criteria:* System auto-closes tasks after 24h; 'Veto' button in Slack cancels auto-close.

3. The Agency Client (Mark)

- *Goal:* Wants to know what he is paying for without reading code.
- *User Story:* "As a Client, I want a weekly summary of delivered features written in plain English, verified by a secure link, so I can approve invoices with confidence."
- *Acceptance Criteria:* Proof Packet contains Gemini-generated summary; Hash chain verifies data has not been edited by the agency.

5.7 Storyboard Walkthrough

1. **Scene 1: The Invisible Handshake.** Alex starts work on Monday. He drags a card "Fix Login" to "In Progress" in Jira. Instantly, Trail AI logs the timestamp. Alex gets a quiet Slack DM: "Tracking started." He keeps coding, uninterrupted.
2. **Scene 2: The Evidence.** Alex pushes code. GitHub runs tests. Trail AI watches silently. When Alex merges the PR, Trail AI sees: *Tests Passed + Code Reviewed*.
3. **Scene 3: The Optimistic Proposal.** Sarah (Lead) is in a meeting. She gets a Slack notification: *"Fix Login is complete. Auto-closing tomorrow at 9 AM."* She checks the summary. It looks good. She does nothing. The system works for her, not against her.
4. **Scene 4: The Proof.** Mark (Client) clicks a link in his invoice. He sees a "Proof Packet." It doesn't show confusing git hashes. It says: *"Login bug fixed; session reliability increased to 99%."* Mark sees the "Verified" green checkmark. He pays the invoice.

6. Implementation Plan

6.1 Technology Stack & Rationale

Component	**Technology**	**Rationale**
Runtime	**Bun**	Native TypeScript support, 3x faster start-up than Node.js, and built-in test runner.
API Framework	**HonoJS**	Ultra-lightweight, runs natively on Bun, optimized for high-throughput webhook processing.
Database	**Supabase (PostgreSQL)**	Open-source, built-in Auth,

		and RLS (Row Level Security) for secure multi-tenancy.
ORM	**Drizzle**	Type-safe migrations, zero-runtime overhead, and best-in-class performance with Bun.
Task Queue	**pg-boss**	Critical for the **Optimistic Closure** timer. Uses Postgres to manage jobs, removing the need for Redis.
AI Model	**Gemini 1.5 Flash**	Chosen for its large context window (for reading long git logs) and cost-effectiveness compared to GPT-4.
Integrations	**@slack/bolt, jira.js**	Official SDKs ensure reliability and type safety when handling external APIs.
Frontend	**React + TypeScript**	Standard for interactive dashboards; needed for the "Hash Chain Visualizer" component.
Hosting	**Fly.io / Railway**	Serverless-like container orchestration that supports the persistent connections needed for background jobs.

6.2 Development Timeline (Sprint Strategy)

Given the immediate deadline, the plan prioritizes the "No-Friction" features:

- **Phase 1: Foundation (Jan 12-13):** Initialize Bun/Hono, Drizzle schema, and Supabase Auth.

- **Phase 2: Passive Handshake (Jan 14-16):** Build Jira/GitHub webhook listeners to auto-detect work start.
- **Phase 3: Optimistic Engine (Jan 17-19):** Implement pg-boss queues for the 24h auto-close timer and Veto logic.
- **Phase 4: AI & Proof Packets (Jan 20-22):** Integrate Gemini for summaries and build the React viewer.
- **Phase 5: Polish & Visuals (Jan 23-24):** Final testing and building the Hash Chain visualizer.

6.3 Deployment Architecture

- **Local:** Bun server on localhost:3000, Supabase local emulation.
- **Production:** Docker container on Fly.io (Backend), Vercel (Frontend). Logs streamed to standard output for observability.

7. Evaluation Framework

7.1 Success Criteria (Pilot Metrics)

Metric	Definition	Target	Why It Matters
Evidence Density	% of Jira tickets closed with associated Proof Packet	$\geq$	Measures adoption of core value.
Intervention Rate	% of auto-closures vetoed by managers	< 10%	Low intervention means the Optimistic model is accurate.
Adoption Friction	Count of manual "Start" clicks	**Zero**	Proves the passive detection works.
Value Perception	Qualitative Survey Score	Positive	Do clients prefer the AI summaries over raw git logs?

7.2 Evaluation Method

- **Quantitative:** Automated logging of every event timestamp and user interaction.
- **Qualitative:** Semi-structured interviews with Leads and Developers to assess if the "Passive" flow felt invisible or intrusive.

8. Risk Analysis & Mitigation

8.1 Technical Risks

- **Risk 1: API Rate Limiting**
  - *Description:* High-volume webhooks could trigger platform limits.
  - *Mitigation:* Implement pg-boss job queue for asynchronous processing with exponential backoff retries.
- **Risk 2: OAuth Token Expiration**
  - *Description:* Tokens expire, causing sync failures.
  - *Mitigation:* Proactive token refresh (refresh 24h before expiration) using encrypted tokens in Postgres.
- **Risk 3: Webhook Spoofing**
  - *Description:* Malicious actors faking PR merges.
  - *Mitigation:* Strict HMAC-SHA256 signature verification on all incoming routes.
- **Risk 4: Database Tampering**
  - *Description:* Direct DB access to alter logs.
  - *Mitigation:* Append-only table structure and Hash-Chaining (visualized on frontend) to make tampering immediately obvious.

8.2 Product Risks

- **Risk 5: Notification Fatigue**
  - *Mitigation:* Batch notifications. Only notify managers on "Exceptions" (Veto needed) rather than every success.
- **Risk 6: Surveillance Perception**
  - *Mitigation:* Explicitly market as "Delivery Assurance" (protecting the dev's work) rather than "Employee Tracking."

9. Commercialization Strategy

9.1 Business Model

- **Freemium:** Free for 1 Project (Proof of Concept).
- **Starter ($9/mo):** Affordable for small studios; basic exports.
- **Agency Tier ($49/mo):** Includes **AI Summaries**, Custom Branding, and unlimited retention.

9.2 Go-to-Market

- **Phase 1:** Direct Outreach to Agencies.
- **Phase 2:** GitHub Marketplace Launch.
- **Differentiation:** "We are the only tool that uses AI to write your client reports and Blockchain-style hashing to prove they are real."

10. Conclusion & Future Work

Trail AI addresses the "Execution Gap" not by adding more tools, but by intelligently listening to the tools developers already use. By shifting from **Manual inputs** to **Passive Detection**, and from **Manager Approval** to **Optimistic Verification**, the system removes the friction that plagues traditional agency software.

The addition of Gemini AI transforms the "Proof Packet" from a technical log into a valuable business asset, automating client communication. This thesis presents a robust, secure, and user-centric solution for the modern software agency.

Future Work:

- **Vision Verification:** Using Gemini Pro Vision to validate screenshot artifacts.
- **Billing Integration:** Automatically generating Stripe invoices based on Proof Packets.

11. References

1. Atlassian Support. (2025, September 24). "Integrate Jira with GitHub." Retrieved from https://support.atlassian.com/jira-cloud-administration/docs/integrate-jira-software-with-github/
2. Atlassian Automation Tutorials. (2025). "Jira Automation Rule on Pull Request Approval." Retrieved from https://www.atlassian.com/devops/automation-tutorials/jira-automation-rule-pullrequest-approval
3. gitStream Documentation. (2025). "Execution Model-Trigger Control." Retrieved from https://docs.gitstream.cm/execution-model/
4. Google AI for Developers. (2026, January 7). "Structured Outputs | Gemini API." Retrieved from https://ai.google.dev/gemini-api/docs/structured-output
5. Google AI for Developers. (2024, July 28). "Function Calling with the Gemini API." Retrieved from https://ai.google.dev/gemini-api/docs/function-calling
6. Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). "Design science in information systems research." MIS Quarterly, 28(1), 75-105.
7. Saunders, M. N. K., Lewis, P., & Thornhill, A. (2019). Research Methods for Business Students (8th ed.). Pearson.
8. Slack Developer Docs. (2025). "Verifying Requests from Slack." Retrieved from https://docs.slack.dev/authentication/verifying-requests-from-slack/
9. GitHub Docs. (2024, December 31). "Pricing Plans for GitHub Marketplace Apps." Retrieved from https://docs.github.com/en/apps/github-marketplace/selling-your-app-on-github-marketplace/pricing-plans-for-github-marketplace-apps
10. Workato Documentation. (2025). "Post GitHub Milestones with Workbot for Slack Use Case." Retrieved from https://docs.workato.com/getting-started/use-cases/github-workbot-for-slack.html

11. LinearB Blog. (2024, March 18). "Pull Request Best Practices: Our Tips." Retrieved from https://linearb.io/blog/pull-request-best-practices-our-tips
12. Graphite. (2025, November 19). "Understanding Lead Time vs Cycle Time." Retrieved from https://graphite.com/guides/lead-time-vs-cycle-time
13. Designli. (2025, December 18). "What is Feature Creep and How to Avoid It?" Retrieved from https://designli.co/blog/what-is-feature-creep-and-how-to-avoid-it
14. Forbes Technology Council. (2025, August 15). "How to Design a Focused MVP That Proves Value Fast." Retrieved from https://www.forbes.com/councils/forbestechcouncil/2025/08/15/how-to-design-a-focused-mvp-that-proves-value-fast/
15. Fly.io Documentation. (2025). "Deploying Bun Applications." Retrieved from https://fly.io/docs/languages-and-frameworks/bun/
16. Vercel Documentation. (2025). "Deploying React Applications." Retrieved from https://vercel.com/docs/frameworks/react

12. Appendices

Appendix A: Event Log Schema (Drizzle)

```
// events table
export const events = pgTable('events', {
  id: uuid('id').primaryKey().defaultRandom(),
  prev_hash: text('prev_hash'), // NULL for first event
  event_hash: text('event_hash').notNull(),
  event_type: text('event_type').notNull(), // 'handshake', 'pr_merged', 'optimistic_close', etc.

  // NEW: Distinguish between auto-detected events and manual overrides
  trigger_source: text('trigger_source').default('automatic'), // 'jira_webhook', 'github_webhook',
'manual_veto'

  payload: jsonb('payload').notNull(),
  created_at: timestamp('created_at').defaultNow().notNull(),
  workspace_id: text('workspace_id').notNull(),
});

// Hash chain verification trigger
// Ensures no UPDATE/DELETE on events table
```

Appendix B: Proof Packet JSON Schema

```
{
  "$schema":
"[http://json-schema.org/draft-07/schema#](http://json-schema.org/draft-07/schema#)",
  "type": "object",
  "properties": {
    "packet_id": { "type": "string" },
    "task": {
      "type": "object",
      "properties": {
        "jira_id": { "type": "string" },
        "title": { "type": "string" },
        "assigned_to": { "type": "string" }
      }
    },
    "ai_summary": {
      "type": "string",
      "description": "Gemini-generated client-friendly summary of the work delivered."
    },
    "handshake": {
      "type": "object",
      "properties": {
        "accepted_by": { "type": "string" },
        "method": { "type": "string", "enum": ["manual_click", "passive_jira_move"] },
        "accepted_at": { "type": "string", "format": "date-time" }
      }
    },
    "execution": { "type": "object" },
    "authorization": { "type": "object" },
    "audit": { "type": "object" }
  },
  "required": ["packet_id", "task", "ai_summary", "handshake", "execution"]
}
```

Appendix C: Pilot Recruitment Criteria

- **Team size:** 5-20 developers.
- **Tech stack:** GitHub + Jira + Slack (primary tools).
- **Agency type:** Web/mobile dev studios, SaaS consultancies, digital agencies.
- **Geographic:** Preferably English-speaking (for onboarding and feedback).
- **Commitment:** Willingness to use Trail AI for 4-6 weeks of actual work (not a demo trial).

Appendix D: Interview Questions (Qualitative Feedback)

For Leads/PMs:

1. Did Trail AI reduce the number of times you had to manually check Jira/GitHub to verify status?
2. Did the **Optimistic Closure** flow feel safe, or did you feel the need to verify manually anyway?
3. Did the **AI-generated Proof Packet** reduce the time you spent writing client updates?
4. Would you pay for this product? At what price?
5. What feature would you add next?

For Developers:

1. Did you notice the **Passive Handshake** (auto-tracking), or did it happen invisibly?
2. Did auto-synced Jira status reduce your manual Jira updates?
3. Any concerns about privacy or surveillance?
4. Did the system feel helpful or annoying?

For Clients (if accessible):

1. Did the Proof Packet help you understand what was delivered without needing a meeting?
2. Did it reduce questions about billing or "is it really done?"
3. Would you like to receive these automatically with each sprint?