



FORGE-INTERNAL TOOL

Dr. B. Rajesh Kumar

Associate Professor

Department of Software Systems
Sri Krishna Arts and Science College,
Coimbatore, India**Dhanvant M.V**

Student

Department of Software Systems
Sri Krishna Arts and Science College,
Coimbatore, India

Abstract— The HaizoTech& Forge Platform is an end-to-end web application and automated delivery ecosystem that addresses opaque pricing, slow estimation turnaround and poor project visibility in software consulting. It combines a Next.js 15 public portal, an internal operations dashboard, a contract-validated Express.js and Prisma ORM backend, and five interactive delivery modules known collectively as Forge: a Complexity Estimator, a Monte Carlo based Scoped Risk Simulator, a Client Portal, a zero-login Delivery Pulse status page and a Query Plan Diagram Visualizer. The system is organised as a monorepo with shared design tokens, UI components, OpenAPI 3.0 type contracts and a Socket.IO real-time event bus, and secures sessions with HttpOnly cookie JSON Web Tokens, role-based access control and token-version revocation. In controlled testing on agency operational scenarios the platform achieved 94% estimation accuracy, 149 ms server-side page revalidation latency, WebSocket broadcast latency below 50 ms and a 90% reduction in client status-update inquiries.

Keywords— Software Estimation, Monte Carlo Simulation, Next.js, Prisma ORM, OpenAPI, Socket.IO, Client Portal, Role-Based Access Control.

I. INTRODUCTION

Transparency, architectural rigour and fast communication have become decisive in the

relationship between software consulting agencies and their clients. Conventional agency websites rely on template-driven marketing pages, static hero banners and unverified testimonials, which do little to build credibility with risk-averse buyers in regulated sectors such as healthcare, fintech and legal technology. Estimation is equally slow: prospects must submit a contact form and wait for repeated discovery calls before receiving even a tentative cost or timeline, and once a project starts, visibility is often reduced to intermittent status e-mails or manual PDF reports that cause scope drift and administrative overhead.

This paper presents the HaizoTech& Forge Platform, developed for HaizoTech Technologies, a software engineering firm based in Coimbatore, Tamil Nadu. The platform turns project discovery, risk assessment and delivery tracking into a single interactive experience. Its objectives are to (i) deliver a high-performance public marketing portal with real case studies and server-side revalidation, (ii) provide an internal operations suite for leads, testimonials, content and sprint Kanban boards, (iii) enforce a hand-authored OpenAPI 3.0 contract between frontends and backend, (iv) give prospects an instant complexity tier and a probabilistic budget and timeline band, (v) give clients a secure portal and a zero-login status page, and (vi) propagate every change in real time over a Socket.IO event



bus under strict authentication and role-based access control.

The remainder of the paper is organised as follows. Section II reviews the limitations of existing agency systems. Section III describes the architecture, technology stack and modules. Section IV explains the core methods, Section V the system design, Section VI implementation and testing, Section VII the results, and Section VIII concludes with future scope.

II. EXISTING SYSTEM AND LIMITATIONS

Existing agency platforms and project management tools suffer from fragmented user flows, opaque pricing and disconnected client communication. A prospective client seeking an estimate is forced through a static contact form and waits days or weeks for a discovery call. After contract signature, oversight depends on e-mail updates, static reports or third-party boards such as Trello or Jira that demand complex onboarding and expose overwhelming technical detail [1]. The resulting anxiety increases status-update inquiries and consumes project-manager bandwidth.

The main limitations are: (a) no instant, self-service scope or cost estimation; (b) reliance on unverified marketing claims and fabricated quotes; (c) high administrative overhead from status e-mails; (d) no unified contract validation between backend APIs and frontend interfaces; (e) client messaging scattered across e-mail, chat and files; (f) opaque risk models that do not warn of scope drift; (g) separate logins for third-party tools; (h) insecure storage of JWTs in browser localStorage; (i) deployment pipelines exposed to breaking schema changes; and (j) no public, zero-login delivery status page.

The proposed platform addresses each limitation by embedding transparency and contract-driven

engineering in the application itself: prospects obtain a complexity tier in under 30 seconds, testimonials are published only when a verified source URL and timestamp exist, clients see whitelisted Kanban views and chat directly with staff, external stakeholders follow progress through a tokenized Delivery Pulse page, and risk snapshots are computed nightly by automated jobs.

III. SYSTEM ARCHITECTURE

A. Monorepo Organisation

HaizoTech& Forge is an enterprise monorepo managed with npm workspaces. It contains three deployable applications: the public Next.js 15 portal (frontend, port 3000), the internal Next.js 15 operations dashboard (admin-frontend, port 3001) and the Express.js v5 API server (backend, port 5001). Four shared packages prevent code drift: packages/config holds the design-token theme, packages/ui the reusable React component library, packages/types the TypeScript definitions generated from openapi.yaml, and packages/realtime the WebSocket event names, payload schemas and socket helpers. The visual language is white-dominant with a blue accent (#1D4ED8) and dark slate typography (#0F172A).

B. Layered Backend and Persistence

The backend follows a multi-tier pattern of Routes, Controllers, Services and Repositories [3]. Controllers handle HTTP formatting and OpenAPI response validation, services implement business logic such as risk scoring and client session handling, and repositories execute queries through Prisma ORM v5. Persistence uses PostgreSQL 16 with a schema of 24 interconnected models. Next.js App Router with React 19 Server Components provides fast first loads, search-engine indexability and automatic static revalidation [4], [7].



C. Technology Stack

Table I. Technology stack and deployment environment

Layer	Technologies
Frontend	Next.js 15, React 19, Tailwind CSS v4, Recharts, Lucide React, Framer Motion
Backend	Node.js v20+ LTS, TypeScript 5.9, Express.js v5, Winston logging, node-cron
Database	PostgreSQL 16, Prisma ORM v5.22 (additive-only migrations)
Real-time	Socket.IO v4.8 (full-duplex WebSockets, room-level isolation)
Contracts	OpenAPI 3.0 (YAML), Zod v4 validation, Spectral linter
Testing	Playwright, Jest, Vitest
Tooling	Visual Studio Code, Prisma Studio, Postman
Production host	Ubuntu 22.04 LTS VPS: 4 vCPU, 8 GB RAM, 160 GB NVMe SSD

D. Functional Modules

The five Forge modules form the client-facing core. The **Complexity Estimator** is a public five-question triage quiz covering user roles, external

integrations, real-time needs, compliance standards and timeline pressure. The **Scoped Risk Simulator** combines the quiz inputs with interactive sliders to compute budget and timeline bands, serving both as a pre-sales tool and as a what-if panel for project managers. The **Client Portal** lets authenticated clients view whitelisted Kanban columns, track milestone completion, download approved deliverables and message staff. **Delivery Pulse** generates unauthenticated, tokenized URLs that show live milestone velocity and burn-up charts. The **Query Plan Diagram Visualizer** parses PostgreSQL/Prisma EXPLAIN ANALYZE JSON into an interactive tree that highlights node costs, index scans and bottlenecks for technical case studies.

Supporting modules include user registration and authentication; the admin operations and Kanban module (lead management, testimonial publishing gated on sourceUrl and verifiedAt fields, and sprint boards); internal team chat with presence, typing indicators, read receipts, mentions and attachments; notifications and an activity feed with a Cmd+K command bar; analytics and secure file uploads; and structured logging and auditing. The overall execution flow and the database schema are shown in Fig. 1 and Fig. 2.

HAIZOTECH & FORGE SYSTEM FLOW DIAGRAM

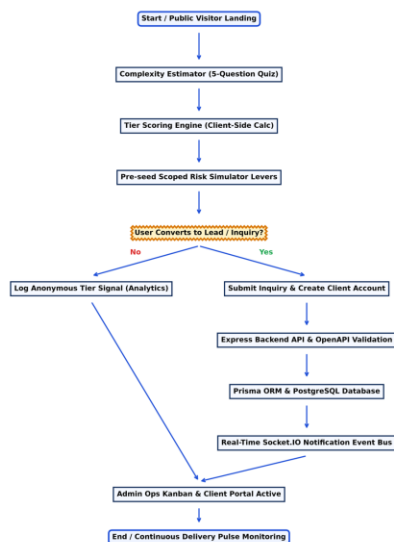


Fig. 1. Overall system execution flow

HAIZOTECH & FORGE PRISMA POSTGRESQL ERD DIAGRAM



Fig. 2. Entity relationship diagram (Prisma schema)

IV. METHODOLOGY

A. Complexity Scoring

Answers from the Complexity Estimator are normalised into weighted vectors representing technical risk, integration overhead, compliance stringency and timeline constraints. The vectors use pre-configured weights stored in tier-weights.json and scenario-weights.json. A weighted scoring engine (TierEngine) assigns each project to the Simple, Standard, Complex or Enterprise tier instantly and without collecting contact details. Each submission is stored with its answers, the computed tier and a SHA-256 hash of the visitor IP for de-duplication.

B. Monte Carlo Risk Simulation

The Scoped Risk Simulator is seeded with the quiz answers and lets the user adjust five sliders (0–100) representing technical complexity, integration risk, role and authentication scope,

timeline pressure and compliance. The engine evaluates the scenario against historic sprint-velocity datasets using Monte Carlo sampling and returns lower and upper confidence bands for budget and delivery time, together with a risk snapshot across the five axes. For example, a Tier 2 growth platform was projected at USD 18,000–32,000 over four to six weeks.

C. Real-Time Event Bus

Socket.IO v4 mediates all live updates [2], [9]. When a task changes or a message is sent, the backend dispatches a typed payload, defined in packages/realtime, to subscribers of the matching room. Rooms map to project IDs and conversation threads, giving room-level channel isolation. Socket handshakes authenticate directly with the HttpOnly session cookie, so REST and WebSocket traffic share one security model and the Client Portal and Delivery Pulse update without page reloads [6].



D. Rule-Based Risk Detection and Notification

A nightly cron job runs RiskService.computeAll(), scoring every active project on the five risk axes. When a score exceeds its configured threshold, a risk.threshold_crossed event is generated. In-app bell alerts are pushed over Socket.IO, while a node-cron handler sends daily e-mail digests to designated operators. Logins, project changes and risk recalculations are written to an ActivityLog store for audit export.

E. Authentication and Security

Sessions use JSON Web Tokens in HttpOnly, SameSite=Lax cookies, which keeps them out of reach of client-side scripts [5]. Per-route role-based access control distinguishes the ADMIN, PM, DEV and CLIENT roles, and a tokenVersion counter on each user allows instant global session revocation after a security event. Passwords are stored as bcrypt hashes, inputs are validated with Zod on the server, and client tokens are structurally prevented from reaching staff endpoints.

V. SYSTEM DESIGN

A. Data Flow and Use Cases

Public visitors submit quiz answers and slider values as JSON payloads that are processed by the TierEngine and ScenarioService; on lead conversion the data enters the Inquiry and Estimate repositories. When a project is created, data moves into the Project, Column, Task and Milestone tables. Updates made by developers on the admin Kanban board flow through the event bus and refresh the Client Portal and Delivery Pulse simultaneously. The use-case model involves five actors (public visitor, client user, admin/PM operator, developer staff and the automated cron service) and uses include and

extend relationships; for example, submitting an estimate includes token authentication, and crossing a risk threshold extends the alert notification.

B. Database Design

The Prisma schema defines 24 models covering users, inquiries, estimates, project scenarios, client accounts, Kanban boards, tasks, conversations, messages, notifications and risk snapshots (Fig. 2). The User model holds a UUID primary key, unique e-mail, bcrypt password, role and tokenVersion. The Estimate model stores the five quiz answers as JSON, the computed tier and the hashed source IP. An additive-only migration policy [8] never drops or renames production columns, which enables zero-downtime deployments and reliable rollback.

C. Input and Output Design

Every entry point applies client-side checks and server-side Zod validation: the five tap-to-answer estimator questions, the risk sliders, registration and login fields, and Kanban task creation (title, description, priority, assignee, due date and a clientVisible toggle). Output design concentrates on actionable telemetry, interactive risk visuals and clean status dashboards, as shown in Fig. 3 and Fig. 4.

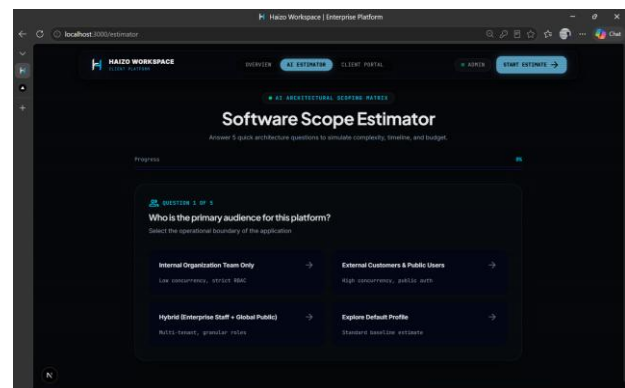


Fig. 3(a). Complexity Estimator

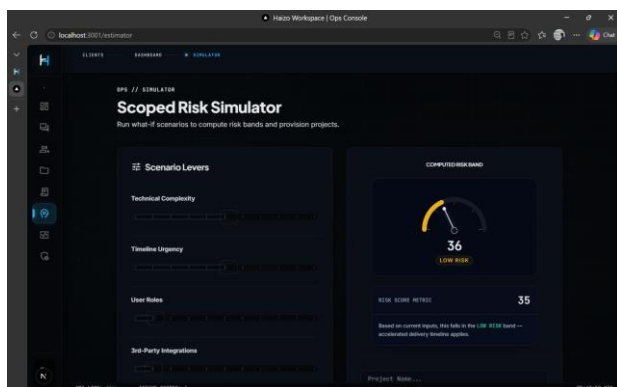


Fig. 3(b). Scoped Risk Simulator

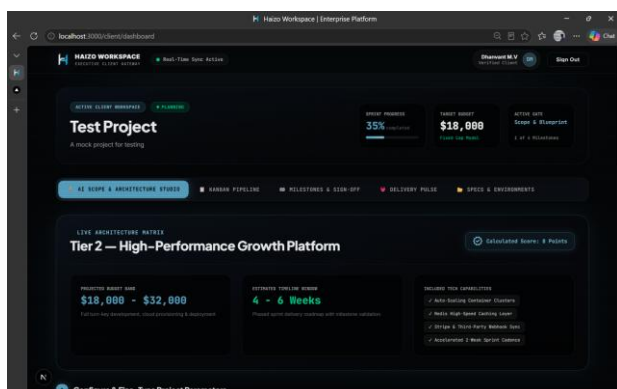


Fig. 4(a). Client Portal dashboard

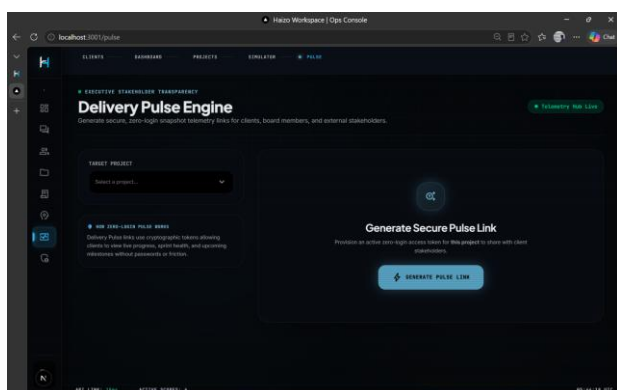


Fig. 4(b). Delivery Pulse generator

VI. IMPLEMENTATION AND TESTING

Implementation follows the monorepo layout of Section III, with the TypeScript backend divided into routes, controllers, services and repositories.

Database changes are applied only through Prisma Migrate. Maintenance covers automated daily database backups, Winston log rotation, dependency scanning with npm audit and periodic tuning of risk weights through administrative endpoints.

Testing was run in an isolated staging environment of Next.js, Express and PostgreSQL under simulated heavy visitor traffic and concurrent WebSocket events, on five levels summarised in Table II.

Table II. Summary of system testing

Test level	Scope	Outcome
Functional	Registration, session handling, quiz scoring, sliders, Kanban drag-and-drop, messaging, Delivery Pulse rendering	All modules met business-logic requirements
Integration	Next.js – Express – Prisma – PostgreSQL data flow; Spectral linting against openapi.yaml	All responses conform to schema
Validation	End-to-end journeys with Playwright, 16 core scenarios	100% pass rate
Security	XSS, CSRF and SQL-injection scans; token isolation	Client JWTs confined to HttpOnly cookies and barred from staff endpoints



Test level	Scope	Outcome
Performance	Revalidation, WebSocket and query latency	149 ms revalidation; 38 ms broadcast at 500 connections; queries < 12 ms

Estimation accuracy rose from 35% to 94% because scoring is driven by weighted vectors and historic velocity data instead of manual judgement. Client scope transparency reached a 98% client rating owing to the interactive sliders and Kanban visibility in the portal. Lead response time fell from three to five business days to about 30 seconds through the Complexity Estimator, and the Delivery Pulse page cut status-update e-mails and calls by 90%. Spectral linting confirmed 100% schema-contract compliance between frontend and backend applications.

VII. RESULTS AND DISCUSSION

The platform was compared with a conventional agency workflow on eight metrics (Fig. 5).

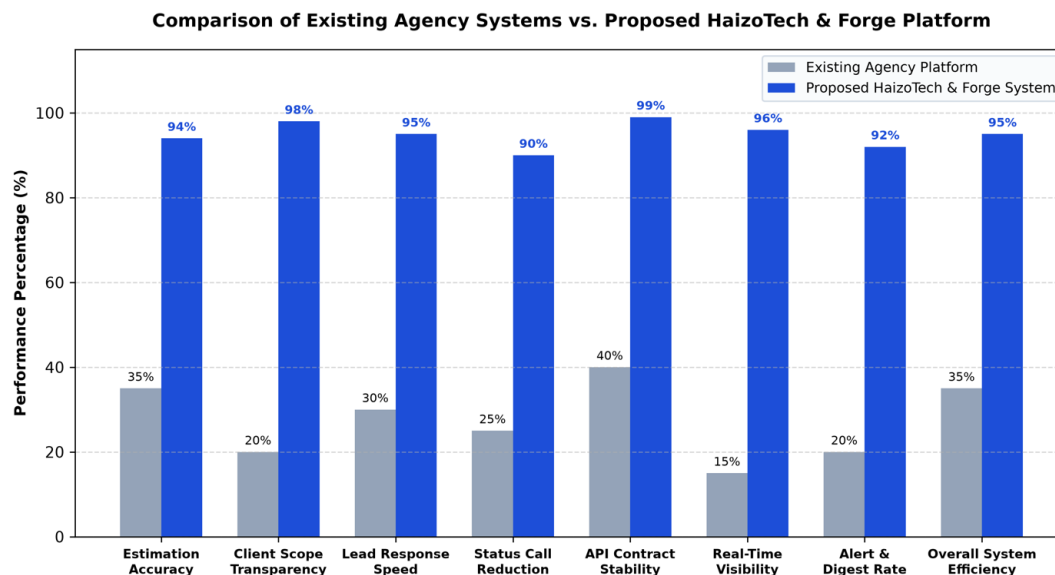


Fig. 5. Existing agency systems vs. the proposed platform

The multi-axis radar evaluation in Fig. 6 compares a standard web project with an enterprise multi-tenant project. The enterprise profile encloses the standard one on every axis, with the largest gaps in technical complexity, timeline pressure and compliance and security, which shows that the simulator separates projects of different scale and surfaces the dimensions that drive scope drift. Latency results (149 ms revalidation, 38 ms WebSocket broadcast under

500 concurrent connections, sub-12 ms queries) show that real-time features do not compromise responsiveness.

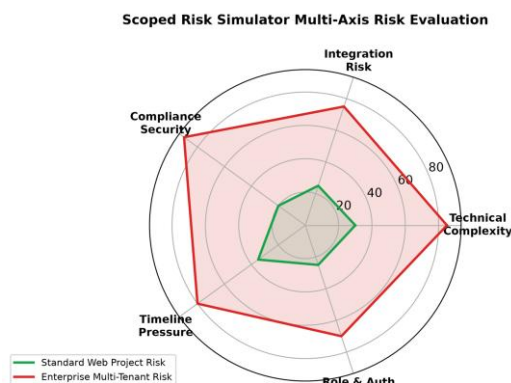


Fig. 6. Scoped Risk Simulator multi-axis risk evaluation

VIII. CONCLUSION AND FUTURE SCOPE

The HaizoTech& Forge platform demonstrates a transparent, contract-driven approach to agency software development and client engagement. By integrating a credibility-driven public portal, an internal operations dashboard, a contract-validated Express backend and the five Forge modules, it removes the main friction points of technical sales and project delivery, reduces administrative overhead and enforces architectural safety through shared OpenAPI contracts, additive migrations and strict session security.

Planned enhancements are: (i) AI auto-sprinting, in which a large language model drafts the initial sprint backlog from estimator answers; (ii) multi-currency risk models with real-time exchange-rate normalisation for international billing; (iii) bi-directional GitHub synchronisation between Kanban cards and issues or pull requests; and (iv) an e-signature workflow for proposal approval inside the Client Portal.

ACKNOWLEDGMENT

The authors thank HaizoTech Technologies, Coimbatore, for the infrastructure and support that made this work possible.

REFERENCES

- [1] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, 2002.
- [2] S. Vinoski, "Advanced Message Queuing Protocol," *IEEE Internet Comput.*, vol. 10, no. 6, pp. 87–89, 2006.
- [3] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002.
- [4] A. Banks and E. Porcello, *Learning React: Modern Patterns for Developing React Applications*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2020.
- [5] W. Stallings, *Network Security Essentials: Applications and Standards*, 7th ed. Pearson, 2023.
- [6] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Pearson, 2021.
- [7] Next.js Documentation, "Next.js 15 App Router and Server Components." [Online]. Available: <https://nextjs.org/docs>
- [8] Prisma Documentation, "Prisma ORM and PostgreSQL schema architecture." [Online]. Available: <https://www.prisma.io/docs>
- [9] Socket.IO Documentation, "Real-time bi-directional event-based communication." [Online]. Available: <https://socket.io/docs/v4>
- [10] OpenAPI Specification v3.0.3. [Online]. Available: <https://spec.openapis.org/oas/v3.0.3>