Configuration Management for Bonzah Insurance Web Platform

Team: Anthony Campana, Eddie Gong, Emanuel Moreno-Gomez, Kenneth Gisler, Mohammed Rahman

Client: Bonzah Insurance

Course: COM S 402 – Senior Design Project

Instructor: Simanta Mitra

TA: Scott Song

Term: Fall 2025

Chapter 1 – Requirements

1.1 Purpose

This project delivers a production-ready marketing and servicing website for Bonzah Insurance. The platform provides an end-to-end quote intake flow, live chat support, blog and review management, and partner-focused content. It replaces manual lead capture with structured data stored in MySQL, enables two-way real-time chat for faster responses, and exposes content management features so marketing teams can iterate without code changes.

1.2 Scope

Included:

- Public marketing site (Next.js) with homepage sections, FAQ, testimonials, resources, and partner highlights.
- Quote flows: an embedded third-party quoter and a custom step-by-step quote wizard with validation and backend submission.

- Content features: customer reviews feed and full CRUD for blog posts (title, body, images, tags, state filters).
- Support experience: floating live-chat helper with persistent conversations, rooms, and open/closed status via WebSockets. Netlify had serverless constraints so we used HTTP Polling.
- Backend APIs: REST endpoints for posts, reviews, chats, and quote submissions; lazy table creation; serverless entry for Netlify.
- Data storage: MySQL (AWS RDS) tables for posts, reviews, chats, and quote submissions.

Out of Scope:

- Claims processing, billing, policy binding, or carrier-side integrations beyond the existing Canopy/UseCanopy link.
- Authentication and role-based access control. All admin-style pages are currently unsecured and should be protected before production.

1.3 Definitions and Acronyms

- **Next.js:** React framework used for the frontend (App Router).
- **RDS:** AWS Relational Database Service hosting the MySQL instance.
- **WS:** WebSocket, used for bidirectional chat communication.
- **Canopy:** Third-party service used for policy import and quote acceleration.
- **Quote Wizard:** Custom multi-step intake form defined in frontend/src/config/carQuoteForm.ts.

1.4 Users

- Prospective customers seeking non-owner or auto insurance quotes.
- Marketing and content editors managing blog posts and testimonials.
- Support agents responding to live chat inquiries.
- Partner managers presenting insurance partner information and embedded quoter links.

1.5 Constraints

- Database connectivity requires valid RDS credentials in backend/.env; no automatic migrations beyond runtime table checks.

- Frontend assumes API base at http://localhost:3001 unless overridden by NEXT_PUBLIC_API_BASE and NEXT_PUBLIC_WS_URL.
- Serverless deployment via Netlify reuses the Express app; cold starts re-run table ensure logic.
- Network access to RDS must allow the running host IP; local development may require VPN or IP allowlisting.

1.6 Assumptions

- MySQL schema creation and updates are handled by application-level table ensure helpers at startup.
- Public pages can call backend APIs directly; production should use HTTPS and proper CORS configuration.
- Users access the quote wizard on modern browsers with JavaScript enabled.
- Images and media are hosted externally; the backend stores metadata and URLs only.

Chapter 2 – Design

2.1 System Overview

The system consists of a Next.js frontend that renders the marketing site, quote wizard, and admin-style content pages, paired with an Express + TypeScript backend exposing REST APIs and a WebSocket server. Data is persisted in MySQL (AWS RDS). For deployment, a Netlify serverless function wraps the same Express application. Real-time chat uses WebSockets while persisting all messages via HTTP endpoints.

2.2 Architecture Breakdown

2.2.1 Frontend (Next.js)

- **Pages:** Homepage (src/app/page.tsx), privacy, partner pages, blog/review admin, assistant-client dashboard.
- **Components:** Hero sections, partner highlights, FAQ, testimonials, quote wizard (components/quote/QuoteWizard.tsx), floating chat helper (components/globals/helper.tsx).

- **Configuration:** Quote flow is config-driven via src/config/carQuoteForm.ts, defining steps, fields, and Canopy links.
- **Environment:** Uses NEXT_PUBLIC_API_BASE and NEXT_PUBLIC_WS_URL for REST and WebSocket endpoints, defaulting to localhost during development.

2.2.2 Backend (Express + MySQL + WebSockets)

- **Server Setup:** backend/src/server.ts initializes Express APIs, CORS, MySQL connection pooling, and a WebSocket server on the same HTTP server.
- **Auto-Provisioning:** Ensures required tables (posts, reviews, chats, quote_submissions) exist at startup.
- **Chat:** WebSocket rooms keyed by room_id; HTTP fallback persists messages if WebSocket is unavailable.
- **Serverless:** The Express app is exported through netlify/functions/api.ts for Netlify deployment.

2.2.3 Database Schema (Core Tables)

- **posts:** id, title, author, images (JSON), body, publish_date, last_edit_date, tag, state.
- **reviews:** id, rating, text, author, date, company.
- **chats:** id, room_id, email, message, time, status (open | closed).
- **quote_submissions:** id, payload (JSON), created_at.

2.2.4 Deployment and DevOps

- **Backend:** npm install && npm run build && npm run start (production); npm run dev for local development.
- **Netlify:** Uses netlify/functions/api.ts; environment variables must be configured in the Netlify dashboard.
- **Frontend:** npm install && npm run dev for development or npm run build && npm start for production builds.

2.3 API Surface

- **Reviews:** GET /reviews, GET /reviews/:id, POST /reviews, PUT /reviews/:id, DELETE /reviews/:id (filterable by rating, author, text, date, company).

- **Posts:** GET /posts, GET /posts/:id, POST /posts, PUT /posts/:id, DELETE /posts/:id (filterable by tag and state).
- **Chats:** GET /chats, GET /chats/open, GET /chats/:room_id, POST /chats, PUT /chats/:room_id/close, PUT /chats/:room_id/open.
- **Quotes:** POST /quote-submissions, GET /quote-submissions.
- **WebSockets:** JSON frames { type: "join" | "register" | "message", room, email, text }, broadcast per room.

2.4 UI Design

- Static section-based layout styled with Tailwind CSS utilities.
- Homepage includes announcement bar, hero CTA, embedded quoter, advantages, partners, testimonials, FAQ, contact form, and footer.
- Admin-style pages for blog creation/editing, reviews management, and chat console.
- Floating chat helper provides quick links, email capture, room joining, and real-time message streaming.

Chapter 3 – Work Done (Per Team Member)

3.1 Emanuel Moreno

- Developed major frontend components using React and Next.js
- Implemented navigation helpers and reusable UI components
- Assisted with backend API integration
- Learned and applied Docker, CI/CD workflows, and Netlify deployment

3.2 Eddie Gong

- Designed backend API routes and database schema
- Implemented authentication and data validation
- Assisted with deployment configuration

3.3 Anthony Campana

- Designed UI layouts and styling for marketing and admin interfaces.

- Implemented real-time polling chat architecture for serverless compatibility.
- Fixed critical SQL syntax and API integration issues for backend stability.
- Resolved Netlify build pipeline failures and module configuration errors.
- Conducted usability testing and created technical documentation.

3.4 Kenneth Gisler

- Managed project coordination and version control
- Integrated frontend and backend components
- Assisted with testing and bug fixes

Chapter 4 – Results Achieved

4.1 Major Deliverables

- Responsive marketing site with embedded quoter, partner highlights, FAQ, testimonials, and contact form.
- Custom quote wizard producing structured JSON submissions.
- Full CRUD for blog posts and reviews with filtering and admin-facing pages.
- Real-time chat widget with WebSocket rooms and persisted transcripts.
- MySQL-backed API with auto-provisioned tables and Netlify serverless deployment.

4.2 Client Value Delivered

- Structured capture of high-intent leads instead of ad-hoc email intake.
- Live support capabilities to reduce abandonment and improve response times.
- Marketing-controlled content publishing without developer intervention.
- Centralized quote and chat data in AWS RDS for analytics and follow-up.

4.3 Limitations

- No authentication or authorization; admin pages and APIs are currently unsecured.
- Chat lacks agent assignment and typing indicators; relies on manual monitoring.
- Some frontend components still reference http://localhost:3001; environment overrides are required before production.
- No built-in file upload pipeline; images are referenced by external URLs.

Appendix A – Setup Instructions

- **Backend:** Copy backend/.env and populate DB_HOST, DB_PORT, DB_USER, DB_PASSWORD, DB_NAME, PORT, DEV_MODE. Run cd backend && npm install && npm run dev.
- **Frontend:** Set NEXT_PUBLIC_API_BASE and NEXT_PUBLIC_WS_URL, then run cd frontend && npm install && npm run dev.
- **Database:** Optional seed file at backend/bonzah_insurance.sql. Tables are auto-created at runtime if missing.
- **Deployment:** Netlify function at netlify/functions/api.ts wraps the Express app. Configure environment variables in Netlify and point the frontend to deployed API and WebSocket URLs.

Appendix B – Repository Link

- https://git.las.iastate.edu/SeniorDesignComS/2025fall/402c/sd03_bonzah_insurance_site