Open Codelabs: An AI-Native Platform for Workshops

Jai-Chang Park
Google Developer Expert (GDE), Dart-Flutter

February 19, 2026

Abstract

Open Codelabs is an open-source system for running end-to-end programming workshops with two first-class roles: facilitator and attendee. Unlike lightweight static codelab viewers, it integrates content authoring, real-time classroom operation, progress and help orchestration, submissions and assessment, certificate gating, optional code-server workspaces, and multi-surface AI assistance. This manuscript is positioned as a systems and reproducibility artifact, not as a new machine-learning algorithm: the contribution is operational integration, transparent implementation disclosure, and measured behavior under workshop-like loads. We place the system in context with research literature and production platforms, then provide implementation-grounded evidence from code paths, tests, and benchmark protocols with explicit validity limits. To improve reproducibility and transparency, we include a detailed appendix that discloses the concrete prompt templates used by the AI subsystems (generation, guide authoring, consultant, quiz assist, and text-improvement workflows) and an API-route catalog. The report is written as a submission-ready technical report with explicit references to implementation artifacts, benchmark commands, and source anchors.

1 Introduction

Workshop-scale programming education has three persistent engineering constraints. First, operational drift occurs because participant environments vary by operating system, toolchain version, and local configuration. Second, instructors need high-frequency visibility into attendee state (progress, blockers, unresolved requests), yet many codelab systems are optimized for content delivery rather than live facilitation. Third, maintaining up-to-date learning materials has become more expensive as framework and runtime ecosystems change quickly.

Open Codelabs addresses these constraints by combining authoring, execution, communication, and governance into one platform [49, 32, 15]. The design target is not just feature count, but operational coherence: a facilitator should be able to prepare materials, run a live session, intervene in real time, validate outcomes, and close the loop with feedback and certificates without switching systems.

1.1 Why We Built It: Operational Motivation

The project was initiated from a recurring gap in real workshop operations: most available tools solve one slice of the workflow well, but facilitators run sessions across multiple disconnected surfaces. In practical terms, instructors often combine a codelab renderer, a separate communication channel, manual progress tracking, external quiz/feedback forms, and ad-hoc certificate handling. This fragmentation increases operational latency exactly when intervention speed matters (for example, when many attendees block on the same setup failure).

The second motivation was quality drift in workshop content maintenance. Framework versions, CLI commands, and setup conventions change rapidly, while workshop documents are frequently copied from prior sessions and patched manually. We needed AI assistance, but not as a one-shot "generate and trust" mechanism. The design requirement became a staged pipeline (plan → draft → review → revise) with explicit structure contracts and optional grounding tools, so facilitators can inspect intermediate artifacts before publication [12, 43, 41].

The third motivation was governance and deployment realism. Many education and enterprise environments require self-hosting, auditable logs, and predictable failure modes. Therefore, the system was designed to run as a reproducible full stack (Rust/Axum backend + SvelteKit frontend + SQLite default) with Docker-first deployment, explicit API contracts, and role-scoped authorization boundaries [49, 54, 19, 51].

From this motivation, five non-negotiable product requirements were derived:

- **R1 Unified facilitator loop:** authoring, live operation, assessment, and closure in one system.
- **R2 Intervention-time observability:** real-time progress/help/chat signals for rapid instructor response.
- **R3 Verifiable completion semantics:** explicit requirement gates for quiz/feedback/submission and certificate issuance.
- **R4 Controllable AI pipeline:** operator-visible prompts, staged outputs, and tool-scoped grounding rather than opaque one-pass generation.
- **R5 Practical self-hosting:** low-friction deployment path with minimal baseline resources and transparent operational datasheets.

Conceptually, Open Codelabs was built to close the system-integration gap between static codelab tooling and classroom operations platforms. Instead of optimizing only content rendering (as in pure codelab tools) or only assignment workflows (as in classroom/assessment systems), it treats workshop execution itself as the primary systems problem [5, 4, 67].

This report makes five systems contributions:

- A complete implementation-level specification of facilitator and attendee capabilities, including role-scoped control paths and invariants.
- A transparent AI subsystem datasheet covering staged generation, prompt governance, tool exposure boundaries, telemetry, and persistence contracts.
- A workload-grounded stack rationale for Rust/Axum and SvelteKit/Bun, tied to concrete workshop operational requirements.
- Empirical operations evidence across REST, WebSocket, deployment, and quality-signal paths, with reproducible command-level artifacts.
- Open reproducibility assets: route catalog, database datasheet, benchmark harness/results, and verbatim prompt appendix.

The manuscript therefore targets systems, software-engineering, and AI-in-education artifact evaluation criteria. It does not claim a new learning algorithm, a new foundation model, or statistically generalizable educational-outcome gains from the current internal pilot.

2 Related Work

2.1 Collection Protocol

We collected related references from two sources: (i) research papers on programming education and LLM-assisted instruction, and (ii) established public systems that represent adjacent design spaces (codelab tooling, classroom assignment systems, browser IDEs, and assessment infrastructure). The research-paper subset used the query family around `programming education` with relevance filtering for classroom operations, guidance/feedback automation, and AI tutoring. The resulting list used in this report contains 15 references (10 research papers and 5 web systems).

2.2 Curated Reference Set

Table 1: Related work and systems relevant to Open Codelabs.

#	Work	Source	Year	Relevance to Open Codelabs
1	ClassCode [63]	Paper	2020	Classroom-scale programming tutorials with instructor progress visibility; directly relevant to real-time facilitator dashboards.
2	Automated grading/feedback review [10]	Paper	2023	Summarizes design choices for autograding and feedback systems, informing quiz/submission architecture.
3	Generative AI benchmark vs. tutors [60]	Paper	2023	Provides empirical tutor baselines for LLM-assisted introductory programming support.
4	ChatGPT in Python course [9]	Paper	2024	Reports student interaction patterns and trust issues relevant to Ask-Gemini style usage.
5	GitSEED [59]	Paper	2024	Git-backed educational assessment design, related to workspace branch/folder snapshot strategies.
6	Question-aware knowledge tracing [6]	Paper	2025	Suggests learner-state modeling ideas for adaptive support queues and intervention timing.
7	Pedagogical-agent fine-tuning [62]	Paper	2025	Shows SFT strategies to align LLM behavior with pedagogical constraints.
8	CodeRunner Agent [8]	Paper	2025	LLM support in educational coding workflows with context-aware advising.
9	LLM in programming education review [68]	Paper	2025	Consolidates risks/opportunities, supporting guardrail and oversight requirements.
10	LLM applications survey [61]	Paper	2025	Broad taxonomy for balancing automation with human facilitator control.
11	Google Codelabs tools [5]	Web	2026	Baseline for step-based codelab authoring and content rendering model.
12	GitHub Classroom [4]	Web	2026	Assignment and cohort management reference for classroom logistics.
13	code-server documentation [3]	Web	2026	Browser IDE and workspace access patterns relevant to optional workspace mode.
14	CloudCoder [2]	Web	2026	Browser exercise platform with prior patterns for coding task delivery and tracking.
15	Marmoset [67]	Web	2026	High-scale submission and grading operations relevant to workshop assessment flows.

Table 2: Comparative positioning by operational dimension.

Dimension	Codelab tools	Classroom repos	Assessment systems	Open Codelabs
Live facilitator operations	Partial	Partial	Limited	Full
Integrated AI content lifecycle	Limited	Limited	Limited	Full
Bidirectional real-time classroom layer	Limited	Limited	Limited	Full
Certificate-gated completion workflows	Limited	Limited	Partial	Full
Single-system authoring-to-execution loop	Partial	Limited	Limited	Full

Table 3: Architectural comparison on synchronization and control semantics.

System	Primary state model	Real-time instructor view	Role boundary	AI authoring pipeline
ClassCode [63]	Classroom-oriented tutorial state sync with instructor observation focus	Present	Instructor/learner model	Not central
Marmoset [67]	Submission/assessment transaction model	Limited	Course staff/student model	Not central
Open Codelabs	Unified codelab relational state (content, realtime, assessment, AI metadata)	Native WebSocket fan-out for progress/help/chat signals	Facilitator/attendee checks at API and handler layers	Built-in staged Plan→Draft→Review→Revise

2.3 Positioning and Gap Analysis

The closest systems to Open Codelabs usually optimize one axis: either static codelab content, assignment logistics, or assessment automation. Open Codelabs intentionally combines these axes with live facilitation primitives (help queue, DM, progress telemetry, live screen-share status) and AI-assisted content lifecycle management.

This matrix is intentionally descriptive. It documents operational scope differences, but it is not presented as a causal superiority test against those systems.

2.4 Architectural Comparison with ClassCode and Marmoset

To address architectural (not only feature-level) comparison, table 3 summarizes consistency and control-path differences against two commonly cited systems. Likewise, this architectural table is a design-space contrast, not a latency/quality benchmark across independently deployed systems.

3 System Objectives and Scope

3.1 Operational Objectives

The platform targets the following operational objectives [47]:

- Fast setup and low barrier to authoring (Docker, Markdown-first flow).
- Low-latency facilitator visibility during workshops (WebSocket events, live dashboards).
- Portable content operations (import/export and backup/restore).
- Configurable deployment model (self-hosted backend or serverless-oriented frontend modes).
- AI augmentation that remains inspectable and operator-controlled.

3.2 Role Model

Open Codelabs implements a strict two-role model:

- **Facilitator**: manages codelab lifecycle, runs live operations, reviews outcomes, controls AI workflows, and accesses governance features.
- **Attendee**: consumes step content, reports progress, requests help, participates in assessment/-submission/feedback, and optionally uses AI Q&A.

Role separation is enforced at route level and in handler-level checks for codelab-scoped resources [15, 13].

4 Technology Stack Rationale

4.1 Why Rust and Axum for Backend Services

The backend uses Rust with Axum/Tokio for memory-safe concurrency and predictable performance under mixed workloads (HTTP CRUD, WebSocket fan-out, AI SSE proxying) [65, 66]. For this system profile, language-level memory safety lowers production risk in long-running event loops, while async primitives allow one service boundary for both API and real-time channels. A second practical reason is compile-time validation pressure: many classes of schema and type mismatches surface before runtime, which is useful for workshop-critical flows where downtime during events is costly.

Implementation structure follows handler/domain/middleware layering, with explicit modules for auth, security headers, CSRF, and rate limits [15, 51]. SQL access uses SQLx, which enables migration-driven schema evolution while retaining typed query interfaces [7, 27].

4.2 Why Svelte and SvelteKit for Frontend

Open Codelabs uses SvelteKit for file-based routing and a lightweight rendering model suitable for interactive dashboards with frequent state updates [64, 35]. The facilitator console includes many mode-specific surfaces (edit, guide, live, quiz, submissions, monitoring, AI), and Svelte component boundaries map directly to these modes.

SvelteKit also provides convenient routing primitives for role-specific pages and codelab-bound paths, while still allowing one coherent app shell. The platform leverages route-level composition and shared library modules for API access, i18n, theme state, and markdown rendering.

4.3 Why Bun in the Frontend Runtime Toolchain

Bun is used for local development and package/runtime workflows in the frontend project [1, 49]. Within this codebase, the practical value is reduced local startup and dependency management overhead, helping facilitators and contributors iterate rapidly on UI and content tooling. Because workshop organizers frequently need to adapt materials quickly, operational developer velocity is a relevant design criterion, not just a convenience.

4.4 Storage and Deployment Choices

The default persistence layer is SQLite for low-friction local and private deployments, with documented serverless-oriented alternatives for frontend data/runtime choices in Firebase or Supabase modes [49, 47]. This split supports two common deployment realities:

- Workshop-hosted private sessions where a single containerized stack is preferred.
- Cloud-connected scenarios where operators want managed infrastructure at the cost of additional integration complexity.

4.5 Rust Runtime Behavior in Workshop Workloads

Open Codelabs mixes latency-sensitive event traffic and consistency-sensitive transactional paths. Typical workshop sessions generate bursty write patterns (step progress updates, chat/DM persistence, help queue state changes, quiz submissions) while facilitators concurrently execute read-heavy monitoring and authoring operations. Rust with Axum/Tokio is useful here because it allows one process to combine:

- request/response APIs with middleware-heavy security policy,
- long-lived WebSocket fan-out loops,
- upstream SSE proxying for AI responses,
- filesystem and archive operations for workspace management.

The code organization in this repository uses this unification directly rather than splitting into microservices [21, 25, 58].

4.6 Svelte and SvelteKit Route-Level Modularity

The facilitator console is mode-rich and route-centric. In implementation terms, the single codelab admin route hosts editing, preview, guide generation, live classroom operations, feedback, materials, quizzes, submissions, gallery, settings, workspace, raffle, certificate, AI, and monitoring surfaces. SvelteKit's route composition and Svelte component granularity make this feasible without introducing a heavyweight global state orchestration framework [35, 43].

This design has two practical effects:

- **Operational coherence:** facilitator workflows stay in one bounded route context, reducing mode-switch friction during live sessions.
- **Change locality:** feature evolution can remain localized to mode-specific components while sharing common utilities (API adapters, i18n, theming, markdown, AI client transport).

Table 4: End-to-end stack layers used by Open Codelabs.

Layer	Technology set	Responsibility in this project
Frontend framework	Svelte 5 + SvelteKit 2	Route composition, facilitator/attendee UI rendering, client-side state for live workshop flows
Frontend runtime/-tooling	Bun 1 + Vite 7 + TypeScript 5	Local development/build pipeline and production bundle generation
UI/i18n layer	Tailwind CSS 4, bits-ui, svelte-i18n, markdown/highlight toolchain	Design system theming, 21-locale runtime translation, markdown/code rendering
Backend web core	Rust 2021, Axum 0.8.8, Tokio 1, tower-http 0.6.8	REST endpoints, middleware, WebSocket handling, static asset serving
Persistence	SQLx 0.8 + SQLite(default)/Postgres-capable Any pool	Migration-managed relational storage for codelab, realtime, assessment, AI metadata
AI transport	reqwest stream client + Gemini SSE endpoint/proxy	Streaming generation, structured output contracts, model/tool config forwarding
Workspace subsystem	code-server orchestration + filesystem archive/git operations	Per-codelab workspace create/read/update/-download and branch/folder snapshots
Deployment/runtime ops	Docker multi-stage images + docker-compose	Reproducible containerized deployment and environment-variable policy injection

4.7 Bun-Oriented Development Throughput

Frontend development is built around Bun commands in project documentation and local workflows [49, 1]. For this project class, faster dependency/install cycles are not merely ergonomic; they reduce preparation risk when facilitators need to patch workshop content shortly before delivery. The practical target is minimizing "time-to-rehearsal" and "time-to-fix" for small UI/content issues.

4.8 Why This Stack Instead of Common Alternatives

The selected stack should be interpreted as workload-driven rather than trend-driven. For this platform, the governing criteria were:

1. real-time + REST + AI proxy in one service boundary,
2. secure middleware defaults and explicit policy controls,
3. rapid iteration in facilitator-facing UI with many interactive states,
4. low-friction self-hosting for workshops with constrained ops capacity.

Alternative designs (for example, separate realtime gateways, JS-only backend stacks, or heavier frontend state ecosystems) are viable, but would shift complexity either toward runtime safety guarantees, operations overhead, or facilitator UI maintainability.

Table 5: Representative backend crate set and pinned versions.

Crate	Version	Use in Open Codelabs
`axum`	0.8.8	HTTP routing and WebSocket extraction
`tokio`	1.x (full)	Async runtime for API/WS/AI streaming operations
`sqlx`	0.8	Migrations and typed SQL access across SQLite/Postgres-capable runtime
`tower-http`	0.6.8	CORS, tracing, static file services
`reqwest`	0.13.1	Upstream Gemini API calls (JSON + stream)
`axum-extra`, `cookie`	0.12.3, 0.18	Multipart handling and cookie/session plumbing
`jsonwebtoken`	9.3.0	Session token issuance/verification
`aes`, `cbc`, `pbkdf2`, `sha2`, `hmac`	0.8, 0.1, 0.12, 0.10, 0.12	API-key encryption/decryption and credential protection utilities
`bollard`	0.18	Container API integration for workspace infrastructure

Table 6: Representative frontend package stack and roles.

Package	Version	Use in Open Codelabs
`@sveltejs/kit`	2.52.0	SvelteKit app routing/runtime
`svelte`	5.51.3	Component rendering model
`svelte-adapter-bun`	1.0.1	Bun-targeted deployment adapter
`tailwindcss`	4.1.18	Utility-first styling and theme variable layering
`svelte-i18n`	4.0.1	Runtime locale switching and translation key resolution
`marked`, `marked-highlight`, `highlight.js`	17.0.3, 2.2.3, 11.11.1	Markdown and syntax-highlight rendering
`dompurify`	3.3.1	Sanitization for rendered markdown/AI output
`Firebase SDK`, `Supabase SDK`	12.9.0, 2.96.0	Optional serverless backend modes
`KaTeX stack`	0.16.28 / 5.1.7	Formula rendering in markdown content

5 Comprehensive Technology Stack Datasheet

5.1 Stack Layers and Responsibilities

5.2 Backend Dependency Manifest Snapshot

The backend dependency profile is explicitly declared in `Cargo.toml`, enabling deterministic dependency review for security and reproducibility [22].

5.3 Frontend Dependency Manifest Snapshot

The frontend dependency set is declared in `package.json` and split between runtime packages and development-time toolchain packages [39].

5.4 Deployment and Configuration Surface

The container/runtime boundary is defined by backend and frontend Dockerfiles, docker-compose service wiring, and environment-variable controls in `.env.sample` [24, 37, 28, 29]. From a reproducibility perspective, this is a first-class part of the system specification because model allow-lists, rate limits, cookie policies, and CORS behavior are environment controlled.

Table 7: High-impact environment configuration classes.

Class	Representative keys	Operational impact
Auth/session policy	`AUTH_SECRETS`, `ADMIN_SESSION_TTL_SECONDS`, `COOKIE_*`	Session validity, cookie transport semantics, token rotation boundaries
Network/security policy	`CORS_ALLOWED_ORIGINS`, `TRUST_PROXY`, `CSP_HEADER`, `HSTS_HEADER`	Cross-origin behavior and response header hardening
Rate limiting policy	`RATE_LIMIT_GENERAL_*`, `RATE_LIMIT_AI_*`, `RATE_LIMIT_UPLOAD_*`	Burst control for API/AI/upload paths under workshop concurrency
AI model policy	`GEMINI_API_KEY`, `ALLOWED_GEMINI_MODELS`	Upstream model access and backend model-name validation envelope
Frontend mode policy	`VITE_USE_FIREBASE`, `VITE_USE_SUPABASE`, `VITE_API_URL`	Runtime backend selection and API routing behavior

6 Feature-Complete Coverage

6.1 Facilitator Capabilities

Table 8: Facilitator feature coverage with implementation anchors.

Capability	Concrete functions	Implementation anchors
Codelab lifecycle management	Create/update/delete/copy, metadata and visibility control, bulk step save, import/export zip workflows.	Routes and handlers for `/api/codelabs*` [15].
Guide authoring and export	Markdown editing, split preview, PDF/DOC export helpers, AI generation hooks.	Admin route and guide mode [43].
AI codelab generation	Basic run, prompt-only mode, advanced plan/draft/review/revise with structured schema contracts and diff views.	AI generator component [12].
Realtime classroom operation	Attendee presence tracking, step-progress watch, help queue triage and resolution.	WebSocket handler and live mode [58].
Realtime communication	Public chat broadcast and 1:1 DM with history persistence.	WebSocket + chat history route [58, 15].
Inline commenting workflow	Anchor-based inline thread creation, reply, and deletion on step/guide text spans.	Inline comment routes [15].
Assessment authoring	Quiz CRUD, AI quiz draft generation, per-attendee submissions and aggregation.	Quiz routes and admin page prompts [15, 43].
Submission management	Attendee file/link submissions, facilitator listing/filter/delete operations.	Submission routes [15].
Materials distribution	Link/file materials registration and upload pipeline for session prep artifacts.	Material routes [15].
Certificate governance	Completion gating by quiz/feedback/submission toggles, certificate verify route.	Codelab/certificate routes [15].
Raffle orchestration	Certificate-holder-constrained raffle mode for post-session closing events.	Facilitator UI mode in admin stack [32].
Screen-share operations	Facilitator status broadcast and attendee upstream share status over WebSocket signaling.	WebRTC signaling messages in WebSocket handler [58].
Workspace orchestration	Optional code-server workspace creation, branch/folder snapshots, archive download.	Code-server routes and service [15, 26].

Continued on next page

Capability	Concrete functions	Implementation anchors
Audit and resilience	Audit-log browsing plus backup export/inspect/restore for operational continuity.	Admin routes and audit infrastructure [15, 18].

6.2 Attendee Capabilities

Table 9: Attendee feature coverage with implementation anchors.

Capability	Concrete functions	Implementation anchors
Session registration	Name/code-based enrollment with codelab binding and scoped session identity.	Register route and auth middleware [15].
Stepwise learning	Ordered step navigation, markdown/code rendering, saved progress state.	Codelab routes and attendee pages [32].
Progress telemetry	Per-step progress events pushed during session execution.	WebSocket `step_progress` message flow [58].
Help requests	Step-scoped help requests with lifecycle state transitions.	Help request routes [15].
Chat and DM	Public chat participation and facilitator DM receipt.	WebSocket chat/dm messages [58].
AI Q&A (Ask Gemini)	Contextual, streamed question answering from attendee workflow with optional persistence.	Ask Gemini component and AI conversation save route [16, 13].
Quiz participation	Quiz submission and correctness tracking.	Quiz submit routes [15].
Submission upload	File/link assignment submission endpoints by attendee id.	Submission routes [15].
Feedback submission	Difficulty/satisfaction/comment capture for post-session signal collection.	Feedback routes [15].
Certificate retrieval	Completion certificate retrieval and verification endpoint usage.	Certificate routes [15].
Material consumption	Access/download of facilitator-provided links/files and prep artifacts.	Materials routes [15].
Attendee screen-share status	Attendee-side screen-share status notifications for facilitator visibility.	WebSocket `attendee_screen_status` [58].

6.3 Internationalization: 21 Locale Support

Unlike earlier docs that describe limited official support, the current frontend code registers 21 locales and exposes all 21 in the runtime language selector [44, 50]. A translation audit script in the repository reports zero missing keys against the 748-key English base for all non-English locale files at the time of this report [56].

6.4 Theme System and Accessibility Support

Open Codelabs provides explicit theming and accessibility support that should be considered first-class product functionality rather than cosmetic add-ons. The theme state model supports system/-light/dark modes and seven color presets (default, mint, ocean, sunset, forest, berry, slate), plus

Table 10: Registered locale set in current frontend implementation.

Code	Language	RTL	Evidence
en	English	No	Registered and selectable [44, 50]
ko	Korean	No	Registered and selectable [44, 50]
ja	Japanese	No	Registered and selectable [44, 50]
zh	Chinese (Simplified)	No	Registered and selectable [44, 50]
zh-TW	Chinese (Traditional)	No	Registered and selectable [44, 50]
de	German	No	Registered and selectable [44, 50]
es	Spanish	No	Registered and selectable [44, 50]
fr	French	No	Registered and selectable [44, 50]
it	Italian	No	Registered and selectable [44, 50]
pl	Polish	No	Registered and selectable [44, 50]
pt	Portuguese	No	Registered and selectable [44, 50]
vi	Vietnamese	No	Registered and selectable [44, 50]
tr	Turkish	No	Registered and selectable [44, 50]
ru	Russian	No	Registered and selectable [44, 50]
id	Indonesian	No	Registered and selectable [44, 50]
th	Thai	No	Registered and selectable [44, 50]
hi	Hindi	No	Registered and selectable [44, 50]
bn	Bengali	No	Registered and selectable [44, 50]
ar	Arabic	Yes	Registered; document direction switch supported [44, 50]
fa	Persian	Yes	Registered; document direction switch supported [44, 50]
he	Hebrew	Yes	Registered; document direction switch supported [44, 50]

persistent colorblind mode toggling [55]. Global styling declares color variables in both normal and dark contexts and applies dedicated colorblind adjustments for contrast/saturation, link underlines, and button boundaries [42].

Accessibility coverage is implementation-visible across layout and components:

- ARIA labels for high-frequency controls (language/theme toggles, logout, dialog controls).
- Focus trapping for modal/panel flows in AI assistant and consultant interfaces.
- Keyboard-close patterns (Escape) for dialogs and overlays.
- `aria-live` regions in AI output surfaces for incremental updates.
- `focus-visible` ring styles for keyboard navigation confidence.

These are anchored in `+layout.svelte`, `FocusTrap.svelte`, and related UI components [50, 33, 42].

7 System Architecture

7.1 End-to-End Architecture Diagram

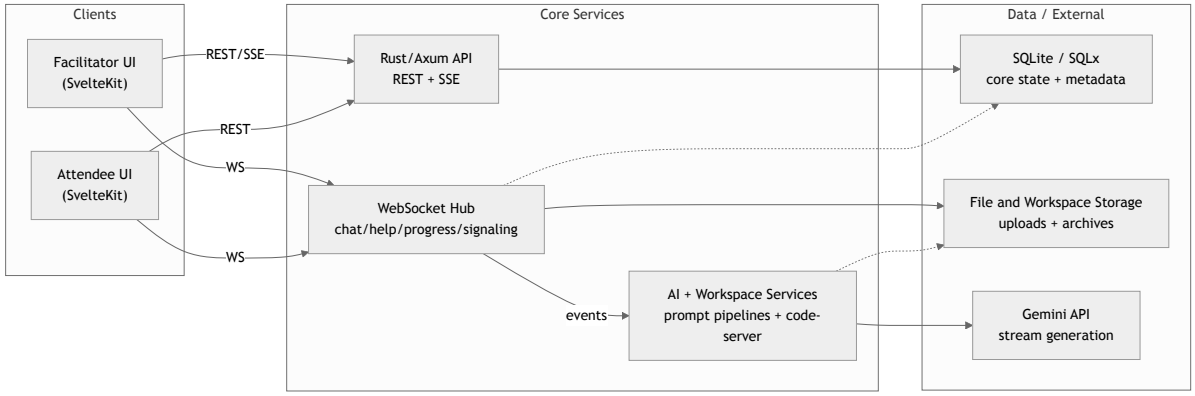

Figure 1: Operational architecture used in Open Codelabs.

7.2 Live Session Signal Flow

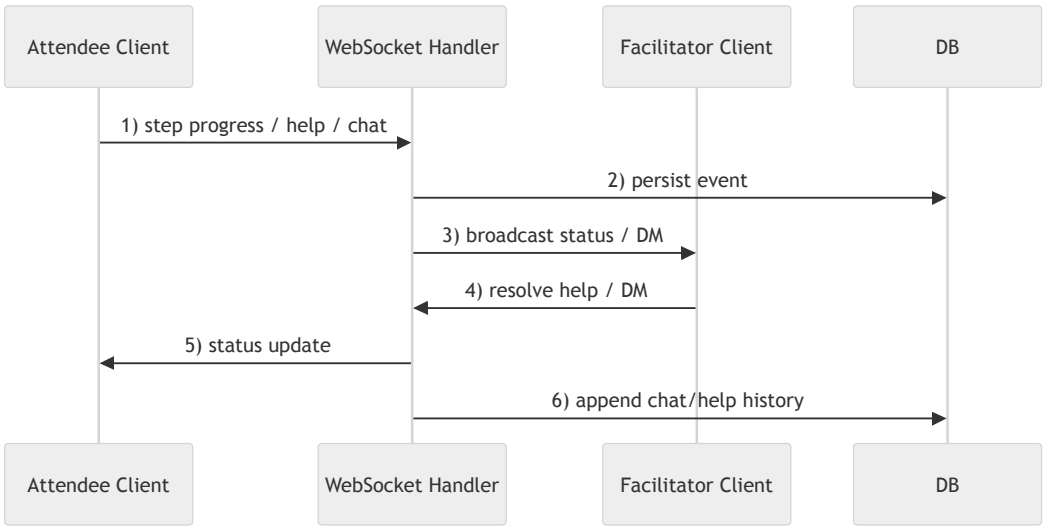

Figure 2: Representative live classroom signal flow over WebSocket.

7.3 Backend Control Surfaces

Route assembly merges authentication, admin controls, codelab operations, uploads, AI services, WebSocket handling, and optional code-server endpoints into one router [15]. Core runtime configuration includes SQL migrations at startup and middleware stacks for rate limit, CSRF, and security headers [25, 51].

7.4 Security and Governance Layers

Security middleware explicitly configures CSP, HSTS, referrer policy, frame restrictions, and permission policy headers, with API-vs-static CSP branching [51]. CSRF checks apply to state-changing methods when a session cookie is present, and rate limits are bucketed by route class (login, AI, upload, general) [51, 48]. AI usage is also audit-recorded with actor and request metadata [13, 18].

8 Operational Protocols and Data Semantics

8.1 Facilitator Mode Surface Model

The facilitator route is not a single-page editor with auxiliary dialogs; it is a multi-mode operational console. The mode state in the admin codelab route exposes 15 explicit operational surfaces in one bounded workspace [43].

8.2 WebSocket Contract Matrix

The WebSocket channel carries heterogeneous event classes: collaboration, progress, and signaling. Server logic enforces message-type-specific behavior (broadcast, direct fan-out, persistence side effects) [58].

Table 12: Live message contract matrix implemented in the WebSocket handler.

Direction	Type	Required fields	Semantics
Client → Server	chat	message	Persist chat row and broadcast to channel
Client → Server	dm	target_id, message	Persist DM and fan-out to all target sessions
Client → Server	step_progress	step_number	Update attendee current step and broadcast progress
Client → Server	webrtc_signal	signal (+ optional target_id)	Direct or broadcast WebRTC signaling payload
Client → Server	screen_share_status	status	Update active facilitator-share map and broadcast status
Client → Server	attendee_screen_status	status	Update attendee share map and notify facilitator sessions
Server → Client	attendee_joined	attendee object	Broadcast attendee arrival with current step snapshot
Server → Client	attendee_left	attendee_id	Broadcast departure when final tab/session closes

Continued on next page

Direction	Type	Required fields	Semantics
Server → Client	screen_share_status	sender_id, status	Class-wide facilitator share state updates
Server → Client	attendee_screen_status	attendee_id, attendee_name, status	Facilitator visibility into attendee share participation
Server → Client	dm/chat/step_progress	message-specific payload	Delivery for persisted communication/progress events

8.3 Data Invariants and Completion Gate Semantics

The schema and handler behavior imply explicit invariants beyond simple CRUD [27, 15, 13]. Representative invariants include:

- at most one workspace row per codelab (`codeserver_workspaces.codelab_id` is unique),
- inline comment thread uniqueness per anchor key within a codelab,
- codelab-scoped ownership checks for attendee submissions and AI conversation writes,
- thread ownership enforcement for facilitator consultant message access.

Completion is controlled by requirement toggles (quiz, feedback, submission). For attendee i in codelab c, define:

$$G_{i,c} = \mathbf{1}(q_c \Rightarrow Q_{i,c})\,\mathbf{1}(f_c \Rightarrow F_{i,c})\,\mathbf{1}(s_c \Rightarrow S_{i,c})\,, \tag{1}$$

where $q_c, f_c, s_c \in \{0,1\}$ are codelab requirement toggles and $Q_{i,c}, F_{i,c}, S_{i,c} \in \{0,1\}$ are attendee fulfillment states. Certificate eligibility is satisfied when $G_{i,c} = 1$.

8.4 Migration and Compatibility Strategy

The backend migration history records iterative expansion from core codelab tables to real-time support, assessment, workspace orchestration, AI thread persistence, and inline comments [27]. This evolution pattern matters operationally:

- feature introduction is append-only at schema level where possible,
- compatibility risk is concentrated around columns that alter completion or message semantics,
- workshop operators can reason about rollout order from migration timestamps.

9 AI Subsystem: Design and Operations

9.1 AI Surface Inventory

Open Codelabs exposes five distinct AI surfaces:

- AI Codelab Generator (basic/prompt-only/advanced staged generation).
- Preparation Guide Generator (single-pass and Pro staged flow).
- Facilitator Consultant (threaded multi-turn advisory chat with search grounding).
- Ask Gemini (attendee-side contextual Q&A).

Table 11: Facilitator operational modes and their dominant responsibilities.

Mode	Primary intent	Representative actions
edit	Authoring core content	Step markdown editing, inline AI improve, structural changes
preview	Render verification	Validate markdown/code rendering before release
guide	Preparation material pipeline	Standard or staged Pro guide generation, save and export
live	Classroom control	Monitor attendees, triage help queue, DM, announce
feedback	Outcome signal review	Difficulty/satisfaction review and qualitative comments
materials	Distribution control	Curate links/files and publish participant resources
quiz	Assessment authoring	Edit questions, run AI quiz generation, inspect submissions
submissions	Assignment review	Inspect uploaded files/links, remove invalid entries
gallery	Visual browsing	Session artifact inspection and image-oriented review
settings	Governance toggles	Completion requirements, visibility, metadata policies
workspace	Code-server management	Branch/folder snapshots, file edits, archive download
raffle	Completion-linked closeout	Participant raffle over verified completion pool
certificate	Credential lifecycle	Verify completion gates and certificate deliverability
ai	Consultant surface	Threaded facilitator advisory with optional grounding
monitoring	Ops telemetry	Session status observation and operational checks

- AI quiz generation and inline markdown improvement.

These surfaces share a common transport and telemetry substrate but use different prompt templates and output contracts [12, 43, 30, 16, 41].

9.2 AI Codelab Generator Pipeline

The codelab generator implements three operating modes:

- **Basic mode:** one-pass structured generation into title/description/steps JSON.
- **Prompt-only mode:** tutorial steps constrained to prompt-centric pedagogy without mandatory code emission.
- **Advanced staged mode:** plan → draft → third-party style review → revision.

Advanced mode is especially relevant for production workshops because it separates structural planning from narrative synthesis and introduces an explicit review checkpoint before final output application [12]. The staged protocol also allows optional Google Search grounding and facilitator comment injection into draft prompts.

Table 13: Pro-mode agent stage contracts in Gemini codelab generation.

Stage	Agent role	Output contract	Tool access	Fallback on parse fail
Planning	Planner	Plan JSON (audience, objectives, env setup, step goals, `search_terms`)	none	`input`
Drafting	Drafter	Codelab JSON (title/description/steps)	optional `googleSearch`, optional `urlContext`	`plan`
Reviewing	Reviewer	Review JSON (summary, issues, missing items, improvements)	none	`draft`
Revising	Reviser	Revised codelab JSON (publishable)	optional `googleSearch`, optional `urlContext`	`draft`

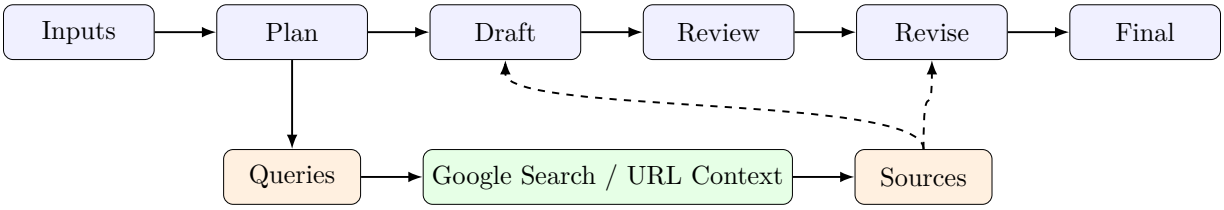

Figure 3: Gemini Pro-mode agent workflow and grounding-tool side loop for codelab generation.

9.3 Gemini Pro Mode: Agentic Workflow for Codelab Synthesis

The Gemini-based Pro mode is implemented as an explicit stateful, multi-agent pipeline rather than a single prompt. In implementation terms, the frontend tracks explicit stage labels: `input -> planning -> plan -> drafting -> draft -> reviewing -> revising -> final`. Planner, drafter, reviewer, and reviser are separated by schema boundaries (plan JSON, draft JSON, review JSON, revised JSON), and each transition is conditioned on structured-output parse success [12, 41]. When parsing fails, the controller falls back to the nearest safe upstream stage (for example, draft parse failure returns to plan; review/revise parse failure returns to draft) instead of auto-publishing partial output.

Pro mode separates artifact roles as follows:

- **Plan agent:** emits objectives, prerequisites, environment setup, per-step verification, and `search_terms`.
- **Draft agent:** converts plan + source context (+ facilitator comments) into tutorial markdown JSON.
- **Review agent:** audits structural and pedagogical defects and emits actionable issue list.
- **Revise agent:** applies review output and finalizes publishable codelab content.

The landing page publicly exposes the same conceptual graph (Inputs → Plan → Draft → Review → Revise → Final, with query-grounding side loop), and this aligns with implementation states and tool routing [46, 12].

Table 14: AI-agent tool exposure by product flow and implementation layer.

Flow	Model path	Exposed tools	Notes
AI Codelab Generator (Pro draft/revise)	`streamGeminiStructuredOutput`	`googleSearch`, optional `urlContext`	Planner/reviewer run without tools; toggles controlled in UI
Preparation Guide (Pro draft/revise)	`streamGeminiStructuredOutput`	`googleSearch`	Plan/review stages run without tool payload
Facilitator Consultant	`streamGeminiChat`	`googleSearch`	Grounding metadata persisted to thread messages
Ask Gemini attendee Q&A	`streamGeminiResponse`	none (default)	Context-question wrapper, no explicit grounding tool injection
Backend AI proxy contract	`/api/ai/stream`	pass-through JSON `tools`	Transport accepts arbitrary tool JSON; integration test includes `codeExecution` payload

9.4 AI Agent Tool Inventory and Exposure Boundaries

Tool availability is not uniform across AI surfaces. The implementation-level rule is flow- and stage-specific allowlisting over `googleSearch`, `urlContext`, and transport-exposed tool payloads. This avoids conflating "AI enabled" with "all tools enabled" and keeps authority boundaries explicit per workflow stage.

The practical implication is that "agent" in Open Codelabs is a staged orchestration pattern over multiple Gemini calls with selective tool activation, not a single monolithic black-box response [12, 43, 30, 16, 41, 13, 20].

9.5 Preparation Guide Generation Pipeline

Guide generation is implemented twice: a standard direct pass and a Pro staged pass. The Pro variant builds a plan JSON (summary, audience, prerequisites, sections, checklist, search terms), drafts markdown, performs dual-perspective review (expert + novice), and revises accordingly [43]. This design reduces one-pass hallucination risk by enforcing intermediate structured artifacts.

Input context is bounded by prompt-budget guards (`MAX_GUIDE_PROMPT_CHARS`, per-step truncation, section-level truncation notes), which makes generation predictable under large codelab payloads while preserving explicit truncation transparency for the operator.

9.6 Consultant and Attendee Q&A Flows

Facilitator Consultant uses multi-turn thread persistence with optional grounding metadata capture; attendee Ask-Gemini uses step-contextual question answering with optional conversation persistence. These two flows share transport primitives but differ in governance:

- facilitator flow is thread-centric and admin-scoped,
- attendee flow is codelab-bound and participant-scoped,
- both parse usage metadata for token accounting.

This split allows high-agency facilitator ideation without overloading attendee UI with workflow complexity [30, 16, 13].

9.7 AI Safety and Governance Controls

The AI proxy enforces prompt validation, role-aware codelab scoping, model-name validation (including optional allow-list environment policy), encrypted API-key forwarding, and audit logging [13]. From an operations perspective, this establishes a governance boundary:

- frontend templates remain inspectable and version-controlled,
- backend enforces policy and persistence constraints,
- per-request metadata supports retrospective incident analysis.

9.8 Transport and Contract Model

The Gemini client helper supports three transport forms:

- `streamGeminiResponseRobust`: context-question wrapper text payload.
- `streamGeminiChat`: role-normalized chat messages with optional `system_instruction`.
- `streamGeminiStructuredOutput`: schema-constrained JSON output with optional thinking-level config.

For backend mode, payloads are proxied through `/api/ai/stream` with encrypted API key forwarding and CSRF headers [41, 13].

9.9 Model Inventory and Effective Selection Rules

The implementation uses a single dominant runtime model family in production code paths: `gemini-3-flash-preview`. This appears both as the frontend default and backend proxy fallback when `model` is omitted [41, 13].

Runtime model resolution follows three concrete branches:

- use request model when present and valid under allow-list/policy checks;
- use backend default (`gemini-3-flash-preview`) when absent;
- reject invalid model names before upstream calls.

For auditing, persisted conversation labels are compared against runtime model usage. In the attendee Ask-Gemini path, a mismatch can appear because persistence still stores a legacy label (`gemini-1.5-flash`) while runtime defaults to `gemini-3-flash-preview` unless overridden [16, 41, 13].

9.10 External AI API and Tool Capability Matrix

Open Codelabs uses the Gemini streaming generation endpoint with both direct (serverless mode) and proxied (backend mode) transport. The implementation exposes advanced capability knobs rather than a minimal text-only call surface.

Table 15: Model usage inventory by AI surface (implementation-level).

Surface	Request assembly path	Effective runtime model	Notes
Ask Gemini (attendee)	`AskGemini.svelte->streamGeminiResponseRobust`	Default `gemini-3-flash-preview`	Saved label may remain `gemini-1.5-flash`
Facilitator Consultant	`FacilitatorConsultant.svelte->streamGeminiChat`	Explicit `gemini-3-flash-preview`	Uses `googleSearch` grounding
AI Codelab Generator	`AiCodelabGenerator.svelte->structuredstream`	Explicit `gemini-3-flash-preview`	Supports high thinking level and optional tools
Guide generation (admin page)	`admin/[id]/+page.svelte helperpath`	Explicit `gemini-3-flash-preview`	Includes staged plan/draft/review/revise variants
Backend proxy fallback	`proxy_gemini_stream(AI handler)`	Default `gemini-3-flash-preview`	Applied when request model is absent

9.11 Prompt Governance and Transparency

Prompt logic is not hidden in backend binaries; it is declared in frontend components and sent through explicit API payload fields. This provides versionable, reviewable prompt evolution at the source level. To align with transparent research reporting, this report publishes all system-level prompt templates in appendix C. For verbatim implementation excerpts with prompt-construction logic and payload assembly, see appendix D.

9.12 AI Cost and Latency Observability

Token usage metadata is parsed from stream events and persisted in conversation/thread records, enabling direct cost accounting and retrospective auditing [41, 13]. Grounding metadata from search-enabled responses is also retained in consultant/thread flows for source traceability [30, 13].

10 API and Deployment Datasheet

10.1 Route-Level Availability Summary

The backend router currently declares 55 distinct API paths and 69 method-path operations. This difference is expected because several paths expose multiple methods (for example GET+POST or GET+PUT+DELETE) [15]. Main route catalog is listed in appendix A; reproducibility-oriented router/DTO/domain source anchors and regeneration commands are provided in appendix E.

10.2 Frontend API Abstraction and Multi-Backend Modes

The frontend exposes one API facade that dispatches by runtime mode: `backend`, `firebase`, or `supabase`. Mode is selected by build-time flags (`VITE_USE_FIREBASE`, `VITE_USE_SUPABASE`) and some operations intentionally degrade to no-op, empty-list, or explicit not-supported errors in serverless paths [34].

Table 16: External AI API capability contracts used by the system.

Capability	Contract field / endpoint	Implementation behavior
Streaming generation	`/v1beta/models/\{model\}:streamGenerateContent?alt=sse`	Parses SSE `data:` events and incremental text chunks
Role-based chat transport	`contents[role,parts]` + optional `system_instruction`	Normalizes assistant role to Gemini `model` role
Structured JSON output	`generationConfig.responseMimeType=application/json`, `responseJsonSchema`	Enforces schema-constrained generation for codelab/guide pipelines
Reasoning budget control	`generationConfig.thinkingConfig.thinkingLevel`	Supports low/medium/high thinking level selection
Grounding tools	`tools=[{googleSearch}, {urlContext}]`	Selective tool activation per feature flow
Governed backend proxy	`/api/ai/stream`	Adds auth/CSRF enforcement, prompt validation, audit metadata, and key handling

10.3 Build Footprint and Minimum System Specification

To make deployment planning concrete, we measured artifact sizes in the current repository workspace after production-style builds (`cargo build -release -bin backend`, `bun run build`) and Docker image inspection on the same host. Table values are implementation-grounded measurements, not theoretical estimates.

For capacity planning, operators should split storage into three practical buckets: runtime images, build/dependency cache, and time-varying data/log volumes. Using measured images, the base container footprint is approximately $536 + 234 = 770$ MB before persistent data and logs.

Because workshop data grows with attendee submissions, chat/help history, and uploaded materials, operators should track persistent data/log growth over time rather than relying only on static image size.

10.4 API Contract Transparency and Reproducibility

To make this report auditable as a systems artifact rather than a marketing summary, API evidence is disclosed at multiple layers:

- route registration source (`routes.rs`),
- DTO payload contracts used by handlers,
- domain model structs for persisted records,
- published API reference document with request/response examples.

These are indexed with representative excerpts and reproducible counting commands in appendix E, and cross-checkable against the endpoint catalog in appendix A [15, 14].

Table 17: Route families and declared path counts.

Route family	Distinct paths	Scope
Authentication	3	Login, logout, session
Admin governance	6	Settings, updates, audit, backup export/inspect/restore
Codelab domain + certificates	28	Lifecycle, attendees, help, feedback, quizzes, submissions, comments, chat, AI conversation list
Upload	2	Image and material file upload
AI core	4	Stream proxy, conversation save, thread CRUD/message
WebSocket	1	Session channel endpoint
Code-server workspace	11	Workspace lifecycle, branch/folder operations, archive/download
Total	55	Path-level declarations in router

Table 18: HTTP method distribution over declared API operations.

Method	Count	Typical usage class
GET	30	Read/list/query operations, WebSocket upgrade endpoint
POST	30	Create/mutate actions, AI streaming trigger, uploads
PUT	3	Bulk update and idempotent codelab/quiz writes
DELETE	6	Resource deletion and cleanup
Total	69	Method-path operation count

Table 19: Capability compatibility across frontend API modes.

Capability cluster	backend mode	firebase mode	supabase mode
Core codelab CRUD and attendee registration	Full	Full	Full
Code-server workspace operations	Full	Not supported	Not supported
Backup inspect/restore and codelab import/export	Full	Not supported	Not supported
Material/submission/quiz admin flows	Full	Partial/limited	Mostly full
AI conversation persistence/list retrieval	Full	save no-op/list empty	save no-op/list empty
Auth provider switching (Google login)	Not supported	Supported	Supported

Table 20: Measured storage footprint of key build and deployment artifacts (2026-02-19 snapshot).

Artifact	Size	Measurement context
Backend release service binary (`target/release/backend`)	23 MB	Native arm64 release binary
Frontend production bundle (`frontend/build`)	14 MB	SvelteKit adapter-bun output
Frontend client output only (`.svelte-kit/output/client`)	6.4 MB	Static client payload subset
Frontend server output only (`.svelte-kit/output/server`)	3.0 MB	SSR/server chunks subset
Local frontend dependencies (`frontend/node_modules`)	505 MB	Developer machine dependency cache
Local backend release target tree (`backend/target/release`)	1.0 GB	Includes binary + Rust build artifacts
Backend data directory (`backend/data`)	1.0 MB	Local snapshot (workload/content dependent; includes active SQLite files)
Docker image: `open-codelabs-frontend`	536 MB	Local image store (`docker images`)
Docker image: `open-codelabs-backend`	234 MB	Local image store (`docker images`)

Table 21: Minimum system specification and operational baseline.

Profile	Requirement	Notes
Documented minimum	2 GB RAM, 1 GB free disk, Linux/macOS/Windows(WSL2)	Official installation baseline [45]
Software baseline (Docker path)	Docker Engine 20.10+, Docker Compose v2.0+	Recommended installation path [45, 49]
Software baseline (local dev path)	Rust 1.75+, Bun 1.0+, SQLite 3.35+	Native development path [45]
Observed practical disk baseline (Docker)	$\geq$ 3 GB free recommended	Inference from 770 MB image footprint + DB/log/backup headroom
Load-tested concurrency evidence	100 stable, 200 maximum-tested users	FAQ operational report; CPU rises near upper bound [31]

11 Database Architecture and Data Datasheet

11.1 Schema Evolution and Integrity Inventory

Database behavior is migration-driven through `backend/migrations` and applied at startup using `sqlx::migrate!`. In current repository state, the migration corpus contains 26 files with 17 table-creation statements and 21 schema-alteration statements [53, 25]. The schema includes 19 explicit non-comment foreign-key clauses and 10 index definitions (9 `CREATE INDEX` + 1 `CREATE UNIQUE INDEX`), plus 3 `UNIQUE` constraints at table/column definition level across operational domains (learning flow, realtime, AI metadata, workspace, and audit).

11.2 ER Diagram of Runtime Schema

11.3 DML Workload Semantics

The backend handler/audit layer embeds a large SQL DML surface with query classes spanning read models, command writes, event logging, and backup restore workflows. A source-line keyword inventory over SQL-bearing lines in `backend/src/api/handlers` and `backend/src/infrastructure/audit.rs` yields 93 `SELECT`, 47 `INSERT INTO`, 10 `UPDATE`, and 44 `DELETE FROM` occurrences in the current snapshot [52].

11.4 Schema Datasheet Synchronization Status

To avoid documentation drift in this revised manuscript, all database metrics reported here and in appendix F are regenerated directly from migration files at the manuscript snapshot commit (`a3510b39`) rather than copied from prose documentation. The public schema document remains useful as a conceptual reference, but migration SQL is treated as source-of-truth for reproducible counting and integrity analysis. To keep this report auditable while maintaining manuscript conciseness, appendix F provides source anchors, representative DDL excerpts, and exact regeneration commands for all reported metrics.

12 Security Architecture and Cryptographic Protocols

12.1 Threat Model and Control Objectives

Open Codelabs is exposed as a stateful web system with authenticated facilitator and attendee sessions, cross-origin browser traffic, WebSocket channels, file uploads, and external AI API interactions. The primary security objectives are:

- prevent unauthorized state changes (auth + CSRF + role checks),
- protect sensitive fields (attendee join codes, admin-provided AI keys),
- bound abusive traffic (bucketed rate limits),
- preserve forensic visibility (audit event capture with request metadata).

Table 22: Database structure metrics from migration corpus.

Metric	Value	Interpretation
Migration files	26	Sequential schema evolution history
CREATE TABLE statements	17	Logical relation count at DDL level
ALTER TABLE statements	21	Incremental evolution pressure on existing entities
Foreign-key clauses	19	Referential constraints for codelab-scoped integrity
Index definitions (`CREATE INDEX`/`CREATE UNIQUE INDEX`)	10	Query acceleration paths over hot access patterns
`UNIQUE` constraints (table/column)	3	Declarative integrity constraints for key tuple/value uniqueness
Data backfill UPDATE statements	1	Explicit historical field migration (`correct_answers`)

Table 23: Secondary explicit and contextual links separated from figure 4 for readability.

Link class	Relations
Secondary explicit FK	`attendees -> help_requests`, `attendees -> feedback`, `attendees -> submissions`, `attendees -> quiz_submissions`, `quizzes -> quiz_submissions`, `inline_comment_threads -> inline_comment_messages`, `ai_threads -> ai_messages`
Contextual/optional linkage	`ai_threads -> codelabs`, `audit_logs -> codelabs`

12.2 Defense-in-Depth Control Inventory

12.3 Cryptographic Construction for Sensitive Fields

Backend-side secret protection is implemented with a password-based envelope: PBKDF2-HMAC-SHA256 key derivation, AES-256-CBC encryption with PKCS#7 padding, and HMAC-SHA256 authentication over payload bytes. The ciphertext format is versioned as `v1:BASE64(salt || iv || ciphertext || tag)` [23]. This construction is used for attendee code storage and for encrypted admin AI-key transport validation [17, 11].

Let password be P, salt S, plaintext M, and random IV be IV. Key derivation is:

$$(K_{enc} \| K_{mac}) = \mathrm{PBKDF2}_{\text{HMAC-SHA256}}(P, S, r, 64), \quad r = 100000. \tag{2}$$

Encryption and authentication are:

$$C = \text{AES-256-CBC}^{\text{PKCS7}}_{K_{enc}}(IV, M), \qquad T = \mathrm{HMAC}_{\text{SHA256}}(K_{mac}, S \| IV \| C). \tag{3}$$

Serialized envelope:

$$E = \texttt{"v1:"} \,\|\, \text{Base64}(S \| IV \| C \| T). \tag{4}$$

On decryption, tag verification is performed before plaintext release and compared in constant time:

$$V_{mac} = \mathbf{1}\left[\text{ct\_eq}\left(T, \widehat{T}\right) = 1\right], \quad \widehat{T} = \mathrm{HMAC}_{\text{SHA256}}(K_{mac}, S \| IV \| C). \tag{5}$$

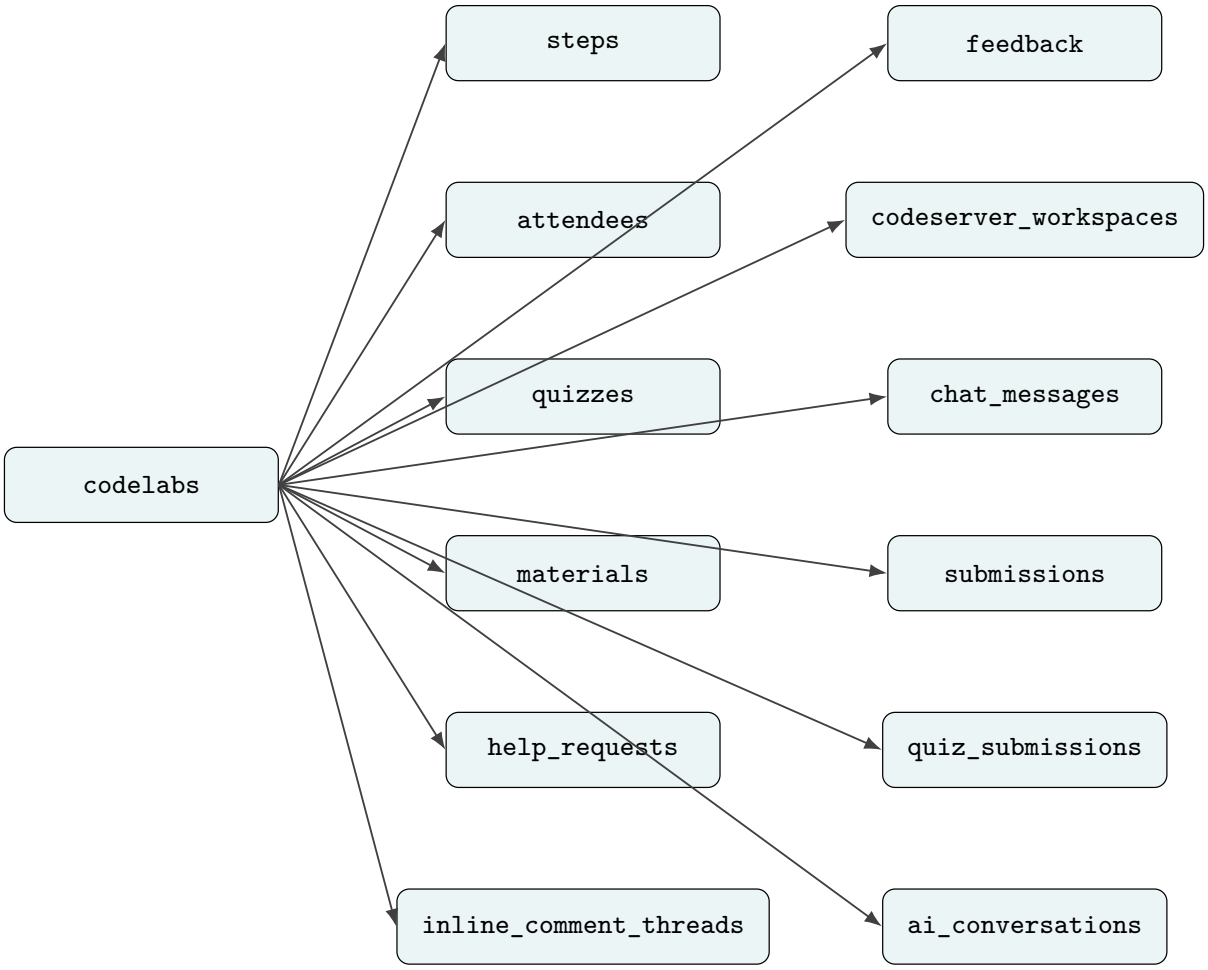

Figure 4: Core database structure derived from migration DDL (first-order codelab-scoped foreign keys).

If $V_{mac} = 0$, ciphertext is rejected.

12.4 Session, Cookie, and CSRF Semantics

Session claims include subject, role, optional codelab scope, issuer/audience, and expiration. Tokens are signed with HS256 and validated against issuer/audience with expiration checks [19]. Cookie policy is environment-driven (`COOKIE_SECURE`, `COOKIE_SAMESITE`, TTL controls), and CSRF cookies are issued on login/registration/session refresh [19, 11, 17].

For request r with method m_r, session-cookie indicator $s_r \in \{0, 1\}$, CSRF cookie token τ_c, and header token τ_h:

$$V_{csrf}(r) = \begin{cases} 1, & m_r \in \{\text{GET}, \text{HEAD}, \text{OPTIONS}\}, \\ 1, & m_r \notin \{\text{GET}, \text{HEAD}, \text{OPTIONS}\} \wedge s_r = 0, \\ \mathbf{1}[\tau_c \neq \varnothing \wedge \tau_c = \tau_h], & \text{otherwise.} \end{cases} \tag{6}$$

State-changing requests are accepted only when $V_{csrf}(r) = 1$ [51, 36, 14].

Table 24: DML class distribution across handler and audit SQL-bearing lines.

DML class	Count	Dominant use pattern
`SELECT`	93	codelab/attendee/material/AI retrieval and admin analytics views
`INSERT INTO`	47	attendee events, AI/chat persistence, backup restore writes
`UPDATE`	10	progress, resolution, status evolution, and timestamp transitions
`DELETE FROM`	44	lifecycle cleanup, codelab deletion cascade orchestration, and backup reset paths

12.5 Rate-Limit Model

Rate limiting uses per-bucket sliding windows keyed by route class and client identity. For bucket k with window W_k, limit L_k, and event timestamps $\{t_i^k\}$:

$$N_k(t) = \sum_i \mathbf{1}[t - W_k < t_i^k \leq t], \qquad \text{allow}_k(t) = \mathbf{1}[N_k(t) < L_k]. \tag{7}$$

Default policy from environment fallbacks is:

- general: 120/min,
- login: 20/5min,
- AI: 30/min,
- upload: 20/min.

[48]

12.6 Security Dataflow Diagram

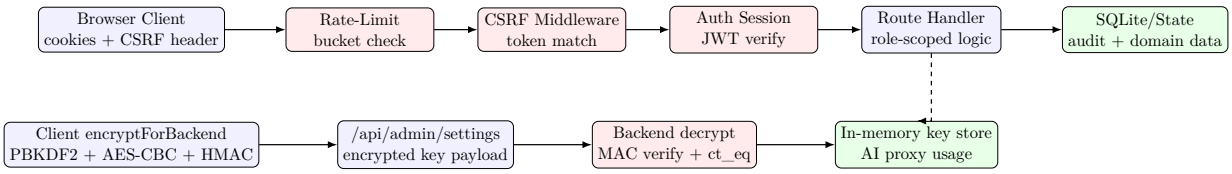

Figure 5: Security-relevant request path and secret-handling path in Open Codelabs.

12.7 Transparent Security Limitations

The implementation intentionally distinguishes between two client-side encryption contexts:

- local browser storage obfuscation (`encrypt`/`decrypt`) uses a static client key and should be interpreted as convenience obfuscation rather than strong at-rest protection,
- backend-compatible encryption (`encryptForBackend`) uses password-derived keys and authenticated encryption envelope semantics.

In addition, if `AUTH_SECRETS` is not set, token signing falls back to `ADMIN_PW`; this is explicitly warned in code and should be treated as a deployment hardening requirement [19, 38].

Table 25: Security control layers and concrete enforcement points.

Control family	Mechanism	Implementation anchors
Session auth	JWT (HS256) with issuer/audience validation, cookie-scoped sessions	`auth.rs`, login/register handlers [19, 11, 17]
CSRF defense	Double-submit cookie validation for unsafe methods (`X-CSRF-Token`)	`csrf_middleware`, frontend CSRF header propagation [51, 36, 14]
Header hardening	CSP, HSTS(HTTPS), X-Frame-Options, nosniff, referrer-policy, permissions-policy	Backend security middleware + frontend hook fallback [51, 40]
Abuse control	Route-class bucket rate limiting (login/AI/upload/general) by client key	`rate_limit.rs`, middleware classifier [48, 51]
Secret protection	Password-based encryption with integrity tag and constant-time verification	`utils/crypto.rs`, client encryption helper [23, 38]
Auditability	Append-oriented audit event recording in normal operations (with backup-restore rewrite path)	Audit infra + admin retrieval route [18, 15]

13 Measured Metric Definitions

13.1 Scope and Interpretation

This section defines only metrics that are directly reported in section 15 or derived from persisted runtime metadata. No closed-form optimization claim is made from these equations. The intent is reproducible measurement semantics, not theoretical superiority proof.

13.2 Completion Metric

For N attendees and completion indicator $z_i \in \{0, 1\}$:

$$C = \frac{1}{N} \sum_{i=1}^{N} z_i. \tag{8}$$

This aligns with requirement-gate enforcement in the runtime model (quiz/feedback/submission toggles).

13.3 Latency and Token-Cost Decomposition

For AI request r:

$$L_r = L_{u \to b}^{net} + L_b^{queue} + L_r^{model} + L_{b \to u}^{stream}. \tag{9}$$

For R AI requests with token counts (x_r, y_r) and prices (p_{in}, p_{out}):

$$C_{ai} = \sum_{r=1}^{R} \left(p_{in} x_r + p_{out} y_r \right). \tag{10}$$

Table 26: Cryptographic parameters used by backend encryption utility.

Parameter	Value	Role
PBKDF2 iteration count r	100,000	Password stretching cost factor
Salt length $\|S\|$	16 bytes	Per-message randomness for key derivation
IV length $\|IV\|$	16 bytes	CBC nonce/initialization vector
Derived key material	64 bytes	Split into K_{enc} and K_{mac}
Encryption key $\|K_{enc}\|$	32 bytes	AES-256-CBC key
MAC key $\|K_{mac}\|$	32 bytes	HMAC-SHA256 key
Tag length $\|T\|$	32 bytes	Integrity/authentication tag

Because usage metadata is captured directly in stream events, equation (10) can be estimated from persisted records without replaying traces [41, 13].

13.4 Instantiated Values in This Revision

Using the internal Basic-vs-Pro ablation artifact (`backend/bench-results/reviewer-20260219/ai-ablation.json`), the generation-only totals are:

- Basic: 14,507 tokens, 0.0177345 USD,
- Pro: 58,000 tokens, 0.0674305 USD.

This corresponds to approximately 3.80× higher generation spend for Pro mode in the reported two-task internal sample (table 36). For latency, equation (9) is instantiated with measured envelopes from table 34, where median p95 remained below the 200 ms classroom target up to 100 concurrent users.

13.5 Localization and Accessibility Reporting Semantics

Localization and accessibility are reported as concrete inventory/checklist evidence (locale key coverage audits, keyboard/focus/ARIA checks, and theme-mode behavior) rather than a single collapsed score. This choice avoids over-compressing heterogeneous UI quality signals into one scalar and keeps operator-facing remediation paths explicit.

14 Verification and Test Evidence

14.1 Implemented Automated Tests

The backend includes integration and module-level tests for critical invariants:

- API/session/cookie/CSRF flow and full handler integration [20].
- Public-private visibility and authorization boundaries [57].
- Workspace file/git/archive service behavior and failure paths [26].
- Rate-limit window behavior and bucket classification assumptions [48, 51].

Table 27: Test coverage map for high-impact subsystems.

Subsystem	Covered behaviors	Evidence
Authentication/session	Login, session cookie flow, CSRF token handling under integration flow	`backend/tests/api_tests.rs` [20]
Authorization visibility	Private codelab denial for public users, admin-visible supersets	`backend/tests/visibility_test.rs` [57]
Workspace service	Workspace create/write/remove, git init/branch/list/read, archive/remove	`backend/src/domain/services/code server.rs` [26]
Rate limiting	Window rollover and bucket classification logic	`backend/src/middleware/rate_limi t.rs`, `security.rs` [48, 51]
Audit logging	Audit row persistence and non-fatal insertion failure behavior	`backend/src/infrastructure/audit .rs` [18]
WebSocket state cleanup	Session cleanup by connection id and attendee-sharing cleanup	`backend/src/api/handlers/websock et.rs` [58]

Table 28: Automated quality metrics from recorded artifact + local refresh run (2026-02-19).

Check	Result	Notes
Backend tests (`cargo test`)	pass (exit 0)	Local refresh run: 31.67 s wall time (167 tests total)
Frontend tests (`bun test -coverage`)	pass (exit 0)	Local refresh run: 52 tests, 8.73 s
Backend coverage (`cargo llvm-cov`, workspace)	lines 74.51%	Regions 71.28%, functions 66.77%
Backend coverage (`cargo llvm-cov`, core service)	lines 95.24%	Excluding `src/bin` and `src/main.rs`: regions 90.54%, functions 88.13%
Backend lint (`cargo clippy`)	pass (exit 0)	54 warnings, 0 errors
Frontend static check (`bun run check`)	pass (exit 0)	Local refresh run: 0 errors, 1 warning (`EditMode.svelte`)
Security audit (`cargo audit`/`bun audit`)	findings present	See archived SWE artifact for advisory details and counts

- Audit persistence semantics and failure-tolerant logging behavior [18].

14.2 Automated Quality Signals from Reproducible SWE Run

Beyond test presence, we report both the archived SWE artifact (`backend/bench-results/all-20260219-184314/swe/swe-metrics.json`) and a local refresh run executed for this manuscript update (2026-02-19).

14.3 Representative Test Pseudocode

14.4 Role-Oriented Verification Matrix

Beyond isolated module tests, workshop reliability depends on multi-role behavioral consistency. table 31 maps high-impact facilitator/attendee behaviors to concrete failure classes that should be validated before release deployments.

Table 29: Pseudo-code: integration-path test for admin login and codelab roundtrip.

Step	Procedure
1	Initialize isolated test app state (ephemeral DB + in-memory server context).
2	Execute admin login and collect session cookie + CSRF token.
3	Construct `CreateCodelab` payload (title, description, author, default requirement flags).
4	Call `POST /api/codelabs` with auth headers and payload; assert HTTP 201.
5	Parse created codelab identifier from response body.
6	Call `GET /api/codelabs/{id}` with same auth context; assert HTTP 200.
7	Verify persisted fields match request payload (`title`, `author`, visibility defaults).

Table 30: Pseudo-code: workspace service test for branch creation and file listing invariants.

Step	Procedure
1	Create temporary workspace root and instantiate `CodeServerManager`.
2	Create workspace named `branch-test`.
3	Initialize git repository for `branch-test`.
4	Write `README.md` with seed content and commit initial changes.
5	Create step branch (`step=1`, name=`start`).
6	List workspace branches and assert the branch set contains `start`.
7	List files on branch `start` and assert `README.md` is present.

15 Empirical Evaluation

15.1 Evaluation Questions

This revision addresses reviewer-raised evidence gaps with four explicit questions:

- **EQ1:** What latency/error envelope is observed on core REST control paths?
- **EQ2:** How does WebSocket roundtrip latency scale with concurrent attendees?
- **EQ3:** Does Pro-mode staged generation improve quality proxies over Basic mode, and at what token/cost overhead?
- **EQ4:** Is the system operationally stable in deployment-style rehearsal runs?

15.2 Measurement Setup and Reproducibility

For stability against run-to-run variance, EQ1 REST control paths were repeated 5 times per attendee cardinality (`local-a10-r1..r5`, `local-a50-r1..r5`) and EQ2 WebSocket runs were repeated 3 times per concurrency point (`ws-u{5,20,50,100}-r1..r3`).

Table 31: Role-oriented verification matrix for release readiness.

Role flow	Failure class	Verification focus
Facilitator login and settings	Session/CSRF mismatch	Cookie issuance, CSRF header requirement, secure logout semantics
Attendee registration and progression	Identity collision / scope leak	Duplicate-name handling, codelab-bound attendee token constraints
Live help triage and DM	Misrouted messaging	Correct target fan-out to multi-tab sessions, state transition pending→resolved
AI assistant usage	Policy bypass / noisy failures	Model allow-list, prompt validation, graceful surfacing of rate/timeout errors
Workspace snapshot operations	Data loss or corruption	Branch/folder file integrity, archive completeness, deletion guard behavior
Completion and certificate path	Incorrect eligibility	Requirement-gate toggles and derived completion consistency

Table 32: Evaluation environment and artifact locations for this manuscript revision.

Item	Value
Repository snapshot	`a3510b39`
Host platform	macOS 15.7.4 (arm64), 10 CPU cores, 16 GB RAM
Backend runtime	Rust release build (`cargo run -release -bin backend`)
Frontend runtime (route benchmark)	SvelteKit/Bun on `http://localhost:5173`
Benchmark harness	`backend/bench/run_all_local.sh`, `run_matrix.sh`, `ai_mode_ablation.py`
Primary result artifacts	`backend/bench-results/reviewer-20260219/*` and `backend/bench-results/all-20260219-184314/*`
Rate-limit policy during load tests	high benchmark envelope (`RATE_LIMIT_{GENERAL,LOGIN,AI,UPLOAD}*=10000`)

15.3 REST Control-Path Latency (EQ1)

Table 33: REST p95 latency summaries under two attendee cardinalities (median [Q1,Q3], 5 repeats each).

Scenario	p95 @10 attendees (ms)	p95 @50 attendees (ms)
`admin_get_attendees`	1420.4 [875.1, 1423.2]	7228.8 [4365.0, 8419.0]
`admin_get_chat_history`	6.1 [3.5, 8.3]	7.9 [6.5, 11.3]
`attendee_post_help`	23.6 [23.2, 37.7]	24.2 [23.7, 24.3]
`attendee_post_submission_link`	21.8 [12.6, 37.8]	24.6 [22.6, 24.6]

All 10 repeated runs (5 at each attendee setting) stayed error-free on these control paths, but `admin_get_attendees` is a clear latency hotspot and dominates p95 under larger attendee sets. This quantitatively identifies where indexing/query-shape optimization should be prioritized before

larger live events.

15.4 WebSocket Scaling (EQ2)

Table 34: WebSocket chat roundtrip summaries across concurrency levels (median [Q1,Q3], 3 repeats each).

Users	p50 (ms)	p95 (ms)	p99 (ms)	Delivery ratio
5	11.9 [11.2, 16.7]	42.2 [34.5, 66.8]	42.9 [39.0, 71.9]	1.000 [1.000, 1.000]
20	13.4 [12.8, 15.6]	65.7 [54.0, 113.0]	96.7 [82.0, 143.6]	1.000 [0.999, 1.000]
50	23.0 [21.2, 23.9]	47.8 [45.5, 54.9]	67.1 [66.5, 79.5]	0.995 [0.992, 0.996]
100	45.1 [44.5, 45.8]	99.3 [94.0, 108.9]	119.8 [109.3, 126.5]	0.999 [0.998, 0.999]

The 100-user case remained below the 200 ms p95 classroom target across repeats (median 99.29 ms, IQR 94.00–108.94 ms). Harness-level disconnect/error counters include teardown resets; delivery ratio remained high (median ≥ 0.995 at all tested loads).

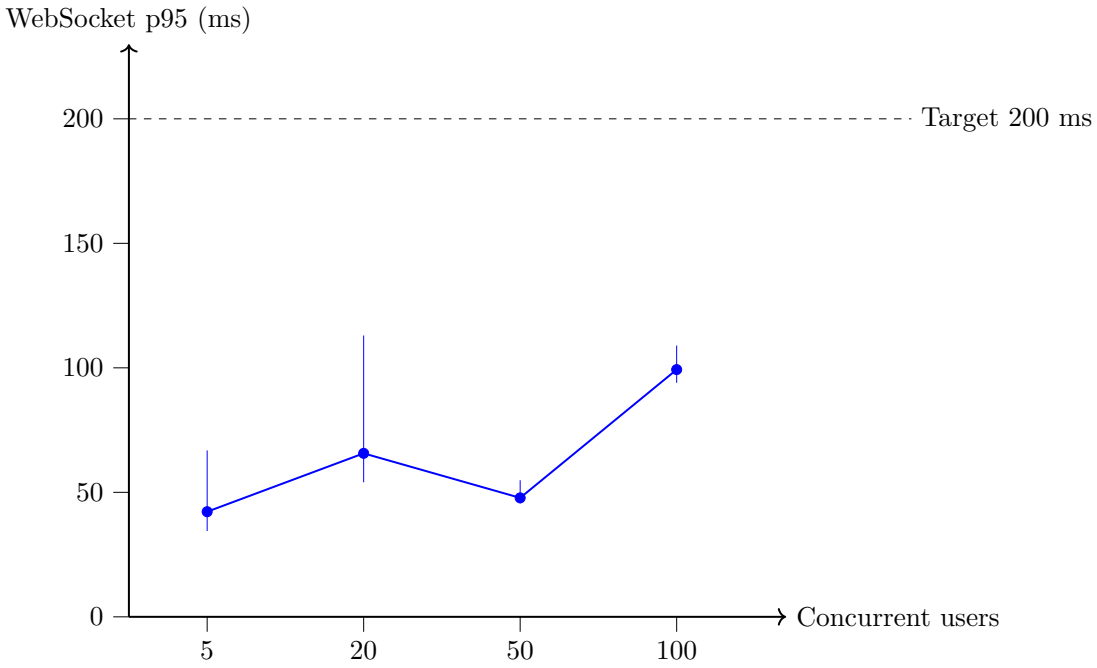

Figure 6: Observed WebSocket p95 latency scaling from 5 to 100 concurrent users (point: median, bar: IQR, 3 repeats each).

15.5 Operational I/O and Frontend Route Results (EQ1, EQ4)

15.6 AI Quality-Cost Ablation: Basic vs Pro (EQ3)

We ran an internal pilot ablation on two backend tasks (`ai.rs`, `websocket.rs`) using the same model family (`gemini-3-flash-preview`), comparing:

- **Basic mode**: single-pass structured generation.

Table 35: Representative operational-path benchmark results.

Scenario class	Metric	Observed value
Upload (1 MB, concurrency 4)	p95 latency	11.13 ms, 12/12 success
Upload (3 MB, concurrency 4)	p95 latency	33.31 ms, 12/12 success
Backup tiny dataset	export / inspect / restore	75.93 / 59.95 / 254.46 ms; all status 200
Backup small dataset	export / inspect / restore	61.75 / 90.79 / 213.79 ms; all status 200
Code-server workspace	create / download	54.81 / 30.67 ms
Code-server updates	branch p95 / folder p95	75.25 / 0.94 ms
Frontend route (`/`, c=4)	p95 latency	104.09 ms, 0.0% error
Frontend route (`/admin`, c=4)	p95 latency	105.10 ms, 0.0% error

Table 36: Basic vs Pro ablation summary (internal pilot, $n = 2$ tasks).

Metric	Basic	Pro
Mean issues per task	5.5	4.5
Mean missing items per task	5.5	4.0
Mean section-hits score	3.0	4.0
Total generation tokens	14,507	58,000
Total generation cost (USD)	0.0177345	0.0674305
Cost per task (USD)	0.0088673	0.0337153

- **Pro mode**: Plan → Draft → Review → Revise.

Both outputs were audited using the same review schema to compute comparable issue/missing-item counts. This is explicitly a directional engineering signal ($n = 2$), not a statistically powered quality study.

Relative to Basic mode, Pro mode reduced mean issue count by 18.18% and missing-item count by 27.27%, while increasing token/cost load by about 4×. With only two tasks, we do not report inferential statistics or generalization claims; this quantifies a local governance trade-off under one internal workload.

15.7 Deployment-Style Pilot Evidence (EQ4)

For operational validation beyond microbenchmarks, we include two deployment-oriented signals:

- soak rehearsal artifact (`backend/bench-results/all-20260219-184314/soak/soak.json`): 1/1 successful cycle, no failed cycles;
- project FAQ field report: stable operation at 100 concurrent users and tested up to 200 users with rising CPU pressure [31].

This is not yet a randomized educational-outcome study, but it provides concrete operations evidence from both instrumented rehearsal and prior deployment practice.

15.8 Evidence Gaps and Next-Phase Evaluation Protocol

To address reviewer concerns on scientific rigor, the next revision is scoped with explicit minimum evidence targets:

1. **Human workshop study:** at least one controlled facilitator-attendee deployment comparing Open Codelabs with a multi-tool baseline, with measured intervention latency and post-session learner/facilitator feedback.

2. **Scaled AI benchmark:** 50–100 codelab-generation tasks with blinded human rating on pedagogy, correctness, and structural completeness (Basic vs Pro vs external one-pass baseline).

3. **Agent-stage ablation:** isolate stage impact (e.g., Basic, Basic+Review, Plan+Draft, full Pro) to identify which stage contributes measurable gains.

4. **Independent review channel:** separate evaluator model and human raters from generator model family to reduce same-model bias.

The current manuscript includes only the operational pilot evidence; the protocol above defines the acceptance-bar expansion path rather than claiming it is already complete.

16 Limitations, Risks, and Future Work

Current constraints include single-node real-time fan-out assumptions, facilitator-centric trust model for sensitive operations, and dependence on external model behavior for AI output quality. From an evaluation perspective, this revision still has important limits:

- the AI ablation sample is small ($n = 2$ tasks) and uses one model family for both generation and schema-based review;
- no external codelab-generator baseline (outside this codebase) is included in the current EQ3 run;
- deployment evidence is rehearsal/field-report oriented, not a randomized controlled educational-outcome trial;
- WebSocket disconnect/error counters in the harness include teardown resets and should not be interpreted as production outage rate.

Even with structured output and staged prompting, generative errors remain possible, especially when source context is incomplete or ambiguous.

Future work priorities:

1. Distributed WebSocket architecture with shared message broker for larger cohorts.

2. Controlled workshop studies with human raters and blinded Basic-vs-Pro scoring for pedagogical quality.

3. Longitudinal learning-outcome analysis beyond session-completion proxies.

4. Adaptive support scheduling with empirically calibrated request-priority features from real classroom traces.

5. Formal prompt/version governance with regression tests over representative codelab corpora and explicit token-budget guardrails.

17 Conclusion

Open Codelabs demonstrates that a modern workshop platform can unify facilitator operations, attendee learning flow, governance controls, and AI-assisted content lifecycle in one architecture. The implementation goes beyond static codelab rendering: it includes live classroom control, structured AI pipelines, optional workspaces, multilingual runtime support, and accessibility-aware UI behavior. This revised report adds empirical evidence for latency scaling, operational-path performance, and Basic-vs-Pro AI quality/cost trade-offs, while explicitly reporting current evaluation limits. It documents capabilities at code-level granularity and provides transparent artifacts for reproducibility, including explicit route maps and prompt templates.

References

[1] Bun Contributors. Bun runtime. https://bun.sh/, 2026. Accessed: 2026-02-18.

[2] CloudCoder Contributors. Cloudcoder: A web-based programming exercise and assessment system. https://github.com/cloudcoderdotorg/CloudCoder, 2026. Accessed: 2026-02-18.

[3] Coder Technologies, Inc. code-server documentation. https://coder.com/docs/code-server, 2026. Accessed: 2026-02-18.

[4] GitHub. Github classroom. https://classroom.github.com/, 2026. Accessed: 2026-02-18.

[5] Google Codelabs Contributors. Google codelabs tools. https://github.com/googlecodelabs/tools, 2026. Accessed: 2026-02-18.

[6] Doyoun Kim, Suin Kim, and Yojan Jo. Knowledge tracing in programming education integrating students' questions. *arXiv preprint arXiv:2502.10408*, 2025.

[7] Launchbadge Contributors. Sqlx. https://github.com/launchbadge/sqlx, 2026. Accessed: 2026-02-18.

[8] Huiyong Li and Boxuan Ma. Design of ai-powered tool for self-regulation support in programming education. *arXiv preprint arXiv:2504.03068*, 2025.

[9] Boxaun Ma, Li Chen, and Shin'ichi Konomi. Enhancing programming education with chatgpt: A case study on student perceptions and interactions in a python course. *arXiv preprint arXiv:2403.15472*, 2024.

[10] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi. Automated grading and feedback tools for programming education: A systematic review. *arXiv preprint arXiv:2306.11722*, 2023.

[11] Open Codelabs Contributors. Open codelabs admin handler. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/api/handlers/admin.rs, 2026. Accessed: 2026-02-18.

[12] Open Codelabs Contributors. Open codelabs ai codelab generator component. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/components/admin/AiCodelabGenerator.svelte, 2026. Accessed: 2026-02-18.

[13] Open Codelabs Contributors. Open codelabs ai handler (proxy, conversations, threads). https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/api/handlers/ai.rs, 2026. Accessed: 2026-02-18.

[14] Open Codelabs Contributors. Open codelabs api reference. https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/en/specification/api-reference.md, 2026. Accessed: 2026-02-18.

[15] Open Codelabs Contributors. Open codelabs api route definitions. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/api/routes.rs, 2026. Accessed: 2026-02-18.

[16] Open Codelabs Contributors. Open codelabs ask gemini component. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/components/codelabs/AskGemini.svelte, 2026. Accessed: 2026-02-18.

[17] Open Codelabs Contributors. Open codelabs attendee handler. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/api/handlers/attendees.rs, 2026. Accessed: 2026-02-18.

[18] Open Codelabs Contributors. Open codelabs audit logging infrastructure and tests. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/infrastructure/audit.rs, 2026. Accessed: 2026-02-18.

[19] Open Codelabs Contributors. Open codelabs authentication middleware. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/middleware/auth.rs, 2026. Accessed: 2026-02-18.

[20] Open Codelabs Contributors. Open codelabs backend api integration tests. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/tests/api_tests.rs, 2026. Accessed: 2026-02-18.

[21] Open Codelabs Contributors. Open codelabs backend architecture document. https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/en/architecture/backend.md, 2026. Accessed: 2026-02-18.

[22] Open Codelabs Contributors. Open codelabs backend cargo manifest. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/Cargo.toml, 2026. Accessed: 2026-02-18.

[23] Open Codelabs Contributors. Open codelabs backend cryptography utility. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/utils/crypto.rs, 2026. Accessed: 2026-02-18.

[24] Open Codelabs Contributors. Open codelabs backend dockerfile. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/Dockerfile, 2026. Accessed: 2026-02-18.

[25] Open Codelabs Contributors. Open codelabs backend entrypoint (main.rs). https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/main.rs, 2026. Accessed: 2026-02-18.

[26] Open Codelabs Contributors. Open codelabs codeserver service and tests. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/domain/services/codeserver.rs, 2026. Accessed: 2026-02-18.

[27] Open Codelabs Contributors. Open codelabs database schema document. https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/en/specification/database-schema.md, 2026. Accessed: 2026-02-18.

[28] Open Codelabs Contributors. Open codelabs docker compose configuration. https://github.com/JAICHANGPARK/open-codelabs/blob/main/docker-compose.yml, 2026. Accessed: 2026-02-18.

[29] Open Codelabs Contributors. Open codelabs environment variable sample. https://github.com/JAICHANGPARK/open-codelabs/blob/main/.env.sample, 2026. Accessed: 2026-02-18.

[30] Open Codelabs Contributors. Open codelabs facilitator consultant component. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/components/admin/FacilitatorConsultant.svelte, 2026. Accessed: 2026-02-18.

[31] Open Codelabs Contributors. Open codelabs faq (english). https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/en/faq.md, 2026. Accessed: 2026-02-19.

[32] Open Codelabs Contributors. Open codelabs feature specification. https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/en/specification/features.md, 2026. Accessed: 2026-02-18.

[33] Open Codelabs Contributors. Open codelabs focus trap component. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/components/FocusTrap.svelte, 2026. Accessed: 2026-02-18.

[34] Open Codelabs Contributors. Open codelabs frontend api mode selector. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/api.ts, 2026. Accessed: 2026-02-18.

[35] Open Codelabs Contributors. Open codelabs frontend architecture document. https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/en/architecture/frontend.md, 2026. Accessed: 2026-02-18.

[36] Open Codelabs Contributors. Open codelabs frontend backend api client. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/api-backend.ts, 2026. Accessed: 2026-02-18.

[37] Open Codelabs Contributors. Open codelabs frontend dockerfile. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/Dockerfile, 2026. Accessed: 2026-02-18.

[38] Open Codelabs Contributors. Open codelabs frontend encryption helper. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/crypto.ts, 2026. Accessed: 2026-02-18.

[39] Open Codelabs Contributors. Open codelabs frontend package manifest. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/package.json, 2026. Accessed: 2026-02-18.

[40] Open Codelabs Contributors. Open codelabs frontend server hooks security header logic. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/hooks.server.ts, 2026. Accessed: 2026-02-18.

[41] Open Codelabs Contributors. Open codelabs gemini client utilities. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/gemini.ts, 2026. Accessed: 2026-02-18.

[42] Open Codelabs Contributors. Open codelabs global styles (theme and accessibility variables). https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/app.css, 2026. Accessed: 2026-02-18.

[43] Open Codelabs Contributors. Open codelabs guide generation pipeline (admin page). https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/routes/admin/[id]/+page.svelte, 2026. Accessed: 2026-02-18.

[44] Open Codelabs Contributors. Open codelabs i18n locale registry. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/i18n/index.ts, 2026. Accessed: 2026-02-18.

[45] Open Codelabs Contributors. Open codelabs installation guide (english). https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/en/getting-started/installation.md, 2026. Accessed: 2026-02-19.

[46] Open Codelabs Contributors. Open codelabs landing page (pro workflow and agent graph). https://github.com/JAICHANGPARK/open-codelabs/blob/main/landing/src/routes/+page.svelte, 2026. Accessed: 2026-02-18.

[47] Open Codelabs Contributors. Open codelabs project overview. https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/specification/overview.md, 2026. Accessed: 2026-02-18.

[48] Open Codelabs Contributors. Open codelabs rate limiting middleware. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/middleware/rate_limit.rs, 2026. Accessed: 2026-02-18.

[49] Open Codelabs Contributors. Open codelabs readme. https://github.com/JAICHANGPARK/open-codelabs/blob/main/README.md, 2026. Accessed: 2026-02-18.

[50] Open Codelabs Contributors. Open codelabs root layout (locale, theme, accessibility controls). https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/routes/+layout.svelte, 2026. Accessed: 2026-02-18.

[51] Open Codelabs Contributors. Open codelabs security middleware (headers, csrf, rate limit). https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/middleware/security.rs, 2026. Accessed: 2026-02-18.

[52] Open Codelabs Contributors. Open codelabs sql dml in api handlers and audit infrastructure. https://github.com/JAICHANGPARK/open-codelabs/tree/main/backend/src/api/handlers, 2026. Accessed: 2026-02-18.

[53] Open Codelabs Contributors. Open codelabs sql migration corpus. https://github.com/JAICHANGPARK/open-codelabs/tree/main/backend/migrations, 2026. Accessed: 2026-02-18.

[54] Open Codelabs Contributors. Open codelabs system architecture document. https://github.com/JAICHANGPARK/open-codelabs/blob/main/docs/en/architecture/system-architecture.md, 2026. Accessed: 2026-02-18.

[55] Open Codelabs Contributors. Open codelabs theme state manager. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/src/lib/theme.svelte.ts, 2026. Accessed: 2026-02-18.

[56] Open Codelabs Contributors. Open codelabs translation audit script. https://github.com/JAICHANGPARK/open-codelabs/blob/main/frontend/scripts/check-i18n.mjs, 2026. Accessed: 2026-02-18.

[57] Open Codelabs Contributors. Open codelabs visibility and authorization tests. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/tests/visibility_test.rs, 2026. Accessed: 2026-02-18.

[58] Open Codelabs Contributors. Open codelabs websocket handler. https://github.com/JAICHANGPARK/open-codelabs/blob/main/backend/src/api/handlers/websocket.rs, 2026. Accessed: 2026-02-18.

[59] Pedro Orvalho, Mikolas Janota, and Vasco Manquinho. Gitseed: A git-backed automated assessment tool for software engineering and programming education. *arXiv preprint arXiv:2409.07362*, 2024.

[60] Tung Phung, Victor-Alexandru Padurean, Jose Cambronero, Sumit Gulwani, Tobias Kohn, Rupak Majumdar, Adish Singla, and Gustavo Soares. Generative ai for programming education: Benchmarking chatgpt, gpt-4, and human tutors. *arXiv preprint arXiv:2306.17156*, 2023.

[61] Griffin Pitts, Anurata Prabha Hridi, and Arun-Balajiee Lekshmi-Narayanan. A survey of llm-based applications in programming education: Balancing automation and human oversight. *arXiv preprint arXiv:2510.03719*, 2025.

[62] Emily Ross, Yuval Kansal, Jake Renzella, Alexandra Vassar, and Andrew Taylor. Supervised fine-tuning llms to behave as pedagogical agents in programming education. *arXiv preprint arXiv:2502.20527*, 2025.

[63] Ryo Suzuki, Jun Kato, and Koji Yatani. Classcode: An interactive teaching and learning environment for programming education in classrooms. *arXiv preprint arXiv:2001.08194*, 2020.

[64] Svelte Team. Sveltekit documentation. https://svelte.dev/docs/kit, 2026. Accessed: 2026-02-18.

[65] Tokio Contributors. Axum web framework. https://github.com/tokio-rs/axum, 2026. Accessed: 2026-02-18.

[66] Tokio Contributors. Tokio asynchronous runtime. https://tokio.rs/, 2026. Accessed: 2026-02-18.

[67] University of Maryland. Marmoset project. https://marmoset.cs.umd.edu/, 2026. Accessed: 2026-02-18.

[68] Meina Zhu, Lanyu Xu, and Barbara Ericson. A systematic review of research on large language models for computer programming education. *arXiv preprint arXiv:2506.21818*, 2025.

A Complete API Route Catalog

Table 37: Route-level API surface (from backend router).

Method	Path	Typical access	Purpose
POST	/api/login	Public	Admin login session creation
POST	/api/logout	Session user	Session termination
GET	/api/session	Session user	Session role introspection
POST	/api/admin/settings	Admin	Update admin-level settings
GET	/api/admin/updates	Admin	Version/update check
GET	/api/admin/audit-logs	Admin	Audit log retrieval
GET	/api/admin/backup/export	Admin	Backup export
POST	/api/admin/backup/inspect	Admin	Backup inspection
POST	/api/admin/backup/restore	Admin	Backup restore
GET	/api/codelabs/reference	Public/Admin	Reference codelab list
GET	/api/codelabs	Public/Admin	List codelabs (visibility-aware)
POST	/api/codelabs	Admin	Create codelab
GET	/api/codelabs/:id	Public/Admin	Get one codelab
PUT	/api/codelabs/:id	Admin	Update codelab metadata
DELETE	/api/codelabs/:id	Admin	Delete codelab
POST	/api/codelabs/:id/copy	Admin	Duplicate codelab
PUT	/api/codelabs/:id/steps	Admin	Bulk update steps
GET	/api/codelabs/:id/export	Admin	Export codelab archive
POST	/api/codelabs/import	Admin	Import codelab archive
POST	/api/codelabs/:id/register	Public	Register attendee
POST	/api/codelabs/:id/complete	Attendee	Mark completion
GET	/api/codelabs/:id/attendees	Admin/Scoped attendee	Attendee list/status
GET	/api/certificates/:id	Public	Certificate retrieval/verify
POST	/api/codelabs/:id/help	Attendee	Create help request
GET	/api/codelabs/:id/help	Admin	List help requests
POST	/api/codelabs/:id/help/:help_id/resolve	Admin	Resolve help request
POST	/api/codelabs/:id/feedback	Attendee	Submit feedback
GET	/api/codelabs/:id/feedback	Admin	View feedback
GET	/api/codelabs/:id/materials	Public/Attendee	List materials
POST	/api/codelabs/:id/materials	Admin	Add material
DELETE	/api/codelabs/:id/materials/:material_id	Admin	Delete material
GET	/api/codelabs/:id/quizzes	Admin/Attendee	List quizzes
PUT	/api/codelabs/:id/quizzes	Admin	Update quizzes
POST	/api/codelabs/:id/quizzes/submit	Attendee	Submit quiz answers
GET	/api/codelabs/:id/quizzes/submissions	Admin	Quiz submission list
GET	/api/codelabs/:id/submissions	Admin	List submissions
POST	/api/codelabs/:id/attendees/:attendee_id/submissions	Attendee	Upload submission file

Continued on next page

Method	Path	Typical access	Purpose
POST	/api/codelabs/:id/attendees/:attendee_id/submissions/link	Attendee	Submit external link
DELETE	/api/codelabs/:id/attendees/:attendee_id/submissions/:submission_id	Admin/Attendee owner	Delete submission
GET	/api/codelabs/:id/chat	Admin/Attendee	Chat history retrieval
GET	/api/codelabs/:id/inline-comments	Admin/Attendee	List inline comment threads
POST	/api/codelabs/:id/inline-comments	Admin/Attendee	Create inline comment thread
POST	/api/codelabs/:id/inline-comments/:thread_id/comments	Admin/Attendee	Reply in thread
DELETE	/api/codelabs/:id/inline-comments/:thread_id/comments/:comment_id	Admin/Attendee	Delete inline comment
GET	/api/codelabs/:id/ai/conversations	Admin	List codelab AI conversations
POST	/api/upload/image	Admin/Attendee	Upload image
POST	/api/upload/material	Admin	Upload material file
POST	/api/ai/stream	Admin/Attendee	Proxy streaming AI generation
POST	/api/ai/conversations	Admin/Attendee	Persist AI Q&A pair
GET	/api/ai/threads	Admin	List facilitator AI threads
POST	/api/ai/threads	Admin	Create AI thread
GET	/api/ai/threads/:thread_id	Admin	Read AI thread messages
POST	/api/ai/threads/:thread_id	Admin	Append message to thread
DELETE	/api/ai/threads/:thread_id	Admin	Delete AI thread
GET	/api/ws/:id	Admin/Attendee scoped	WebSocket endpoint
POST	/api/codeserver	Admin	Create workspace
GET	/api/codeserver/:codelab_id	Admin	Workspace metadata
DELETE	/api/codeserver/:codelab_id	Admin	Delete workspace
POST	/api/codeserver/:codelab_id/branch	Admin	Create branch snapshot
POST	/api/codeserver/:codelab_id/folder	Admin	Create folder snapshot
GET	/api/codeserver/:codelab_id/download	Admin	Download workspace archive
GET	/api/codeserver/:codelab_id/branches	Admin	List branches
GET	/api/codeserver/:codelab_id/branches/:branch/files	Admin	List files in branch
POST	/api/codeserver/:codelab_id/branches/:branch/files	Admin	Update files in branch
GET	/api/codeserver/:codelab_id/branches/:branch/file	Admin	Read single branch file
GET	/api/codeserver/:codelab_id/folders	Admin	List folders
GET	/api/codeserver/:codelab_id/folders/:folder/files	Admin	List files in folder snapshot
POST	/api/codeserver/:codelab_id/folders/:folder/files	Admin	Update files in folder snapshot
GET	/api/codeserver/:codelab_id/folders/:folder/file	Admin	Read single folder file

B Internationalization, Theme, and Accessibility Details

B.1 Locale Runtime Behavior

Locale registration is lazy-loaded from locale JSON bundles and initialized from browser locale detection with fallback to English [44]. At runtime, document direction is switched to RTL for Arabic, Persian, and Hebrew in the root layout [50].

B.2 Theme and Colorblind Modes

Theme behavior includes:

- theme mode selection: system/light/dark,
- seven preset palettes,
- persistent colorblind mode toggling,
- CSS variable overlays for dark and preset variants.

Implementation lives in `theme.svelte.ts` and `app.css` [55, 42].

B.3 Accessibility Checklist Snapshot

Table 38: Accessibility features implemented in UI surfaces.

Feature	Evidence
Dialog semantics	Widespread `role=dialog`, `aria-modal`, close controls with labels [50, 16, 30]
Focus trapping	Dedicated focus-trap component for modal keyboard containment [33]
Keyboard escape behavior	Escape handlers for modal closure and overlays [16, 30, 50]
ARIA labeling	Language/theme/accessibility toggles and editor controls include ARIA labels [50]
Live regions	AI response panels include `aria-live` support for streamed updates [16, 12]
Focus visibility styling	`focus-visible` ring styles in root layout controls and buttons [50, 42]

C Prompt Transparency Appendix

This appendix defines the disclosure policy for AI prompts in Open Codelabs. Rather than publishing partial pseudocode, we expose prompt templates at three levels:

1. operator-facing template families (what each surface asks the model to do),
2. exact implementation excerpts that build prompt strings and request payloads,
3. schema contracts that constrain structured output.

Table 39: Prompt families disclosed in this report.

Surface	Prompt family	Appendix evidence
AI Codelab Generator	SYSTEM, PLAN, REVIEW, staged plan/draft/review/revise templates	appendix D
Guide Generator	Standard direct template, Pro staged plan/draft/review/revise templates	appendix D
Facilitator Consultant	Base system prompt + dynamic reference-list extension	appendix D
Attendee Ask Gemini	Context-question wrapper composition and stream transport template	appendix D
Admin editing/quiz assist	Inline markdown improve templates and quiz generation templates	appendix D
Transport layer	Backend/serverless payload envelopes and structured generation config	appendix D

C.1 Verbatim Implementation Disclosure

All prompt strings, dynamic placeholder patterns, and payload-construction snippets are disclosed verbatim from source files in appendix D. This includes staged prompt assembly (plan/draft/review/revise), grounding-tool usage hints, truncation-notice mechanics, and role-specific system prompts. The intent is full auditability: a reviewer can map each prompt family directly to concrete implementation lines without reverse engineering.

D Prompt Source Excerpts

This appendix includes only model-facing prompt texts (system/user template strings). Implementation logic and non-prompt code paths are intentionally omitted.

D.1 AI Codelab Generator (AiCodelabGenerator.svelte)

D.1.1 System Prompt

Listing 1: SYSTEM_PROMPT

```
You are a world-class Technical Content Engineer and Developer Advocate and a very strong reasoner
     and planner.
Use these critical instructions to structure your plans, thoughts, and responses.
Your mission is to transform raw source code into a high-quality, professional "Hands-on Codelab"
    that ensures a seamless developer experience.

analyzing two types of inputs:
1. [Reference Codelab]: An existing codelab document used as a structural and stylistic template.
2. [Source Code/New Task]: The actual technical content or code that needs to be converted into a
    new codelab.
```

```
Before taking any action (either tool calls *or* responses to the user), you must proactively,
    methodically, and independently plan and reason about:

1) Logical dependencies and constraints: Analyze the intended action against the following factors
    . Resolve conflicts in order of importance:
    1.1) Policy-based rules, mandatory prerequisites, and constraints.
    1.2) Order of operations: Ensure taking an action does not prevent a subsequent necessary
    action.
        1.2.1) The user may request actions in a random order, but you may need to reorder
    operations to maximize successful completion of the task.
    1.3) Other prerequisites (information and/or actions needed).
    1.4) Explicit user constraints or preferences.

2) Risk assessment: What are the consequences of taking the action? Will the new state cause any
    future issues?
    2.1) For exploratory tasks (like searches), missing *optional* parameters is a LOW risk. **
    Prefer calling the tool with the available information over asking the user, unless** your ‘
    Rule 1‘ (Logical Dependencies) reasoning determines that optional information is required for
     a later step in your plan.

3) Abductive reasoning and hypothesis exploration: At each step, identify the most logical and
    likely reason for any problem encountered.
    3.1) Look beyond immediate or obvious causes. The most likely reason may not be the simplest
    and may require deeper inference.
    3.2) Hypotheses may require additional research. Each hypothesis may take multiple steps to
    test.
    3.3) Prioritize hypotheses based on likelihood, but do not discard less likely ones
    prematurely. A low-probability event may still be the root cause.

4) Outcome evaluation and adaptability: Does the previous observation require any changes to your
    plan?
    4.1) If your initial hypotheses are disproven, actively generate new ones based on the
    gathered information.

5) Information availability: Incorporate all applicable and alternative sources of information,
    including:
    5.1) Using available tools and their capabilities
    5.2) All policies, rules, checklists, and constraints
    5.3) Previous observations and conversation history
    5.4) Information only available by asking the user

6) Precision and Grounding: Ensure your reasoning is extremely precise and relevant to each exact
    ongoing situation.
    6.1) Verify your claims by quoting the exact applicable information (including policies) when
    referring to them.

7) Completeness: Ensure that all requirements, constraints, options, and preferences are
    exhaustively incorporated into your plan.
    7.1) Resolve conflicts using the order of importance in #1.
    7.2) Avoid premature conclusions: There may be multiple relevant options for a given situation
    .
        7.2.1) To check for whether an option is relevant, reason about all information sources
    from #5.
        7.2.2) You may need to consult the user to even know whether something is applicable. Do
    not assume it is not applicable without checking.
    7.3) Review applicable sources of information from #5 to confirm which are relevant to the
    current state.

8) Persistence and patience: Do not give up unless all the reasoning above is exhausted.
```

```
    8.1) Don't be dissuaded by time taken or user frustration.
    8.2) This persistence must be intelligent: On *transient* errors (e.g. please try again), you
    *must* retry **unless an explicit retry limit (e.g., max x tries) has been reached**. If such
     a limit is hit, you *must* stop. On *other* errors, you must change your strategy or
    arguments, not repeat the same failed call.

9) Inhibit your response: only take an action after all the above reasoning is completed. Once you
    've taken an action, you cannot take it back.

Follow these strict guidelines to create the content:

1. STRUCTURE & HIERARCHY:
- Title: Engaging and clear.
- Overview: What will be built and what are the key learning objectives?
- Prerequisites: Detailed system requirements (Language versions, CLI tools).
- Environment Setup:
    * System configurations (Environment variables, OS-specific notes).
    * IDE Recommendation & Configuration (VS Code, IntelliJ, etc.).
    * Required/Recommended Plugins/Extensions (e.g., Prettier, ESLint, Language-specific plugins).
- Step-by-Step Implementation: Logical progression from boilerplate to advanced logic.
- Verification: How to test if each step was successful.
- Conclusion & Next Steps: Summary and challenge for the reader.
- Final Summary: A comprehensive recap of the technical concepts and architecture mastered in this
     codelab.

2. DEPTH OF CONTENT:
- "The Why before the How": Explain the architectural decisions or why a specific configuration is
     needed.
- IDE Integration: Don't just show code; tell the user how the IDE can help (e.g., "Use 'Cmd+Shift
    +P' to run this command").
- Error Prevention: Add "Pro-tips" or "Note" boxes for common pitfalls in system setup.

3. TECHNICAL PRECISION:
- Use clear Markdown headings and syntax highlighting.
- Provide shell commands for installation (e.g., npm install, brew install).
- If specific IDE settings (settings.json) or plugin IDs are relevant, include them.
- For every code block: include inline comments for each logical line AND add a numbered line-by-
    line explanation right after the block in the same language as the content.
- Before each code block, state the filename/path being edited.

4. TONE & STYLE:
- Professional, encouraging, and action-oriented.
- Use the "Instruction -> Code -> Explanation -> Verification" loop for every step.

5. ANALYZE & REPLICATE STRUCTURE:
- Use the [Reference Codelab] as a template for tone, formatting, and flow (e.g., Summary,
    Duration, Step numbering, "What you'll learn" sections).
- Maintain the "Introduction -> Setup -> Step-by-Step implementation -> Verification -> Conclusion
    " sequence.

6. MANDATORY ENVIRONMENT & IDE SETUP (Crucial):
- Create a dedicated "Environment Setup" section even if the source code doesn't explicitly
    mention it.
- IDE Recommendations: Suggest the best IDE for the project (e.g., VS Code, IntelliJ).
- Required Plugins: List specific extensions/plugins that will help the learner (e.g., "Install
    the 'ESLint' and 'Prettier' extensions in VS Code for code quality").
- System Config: Include OS-specific requirements, Node.js/Python versions, and Environment
    Variables (.env setup).
```

```
7. STEP-BY-STEP CONTENT GENERATION:
- Each step must follow this loop:
    a. Step Title & Estimated Duration.
    b. Concept: Why are we doing this? (The logic).
    c. Action: Clear instructions on which file to open/create.
    d. Code Block: Provide the exact code with comments like "<!-- CODELAB: Add this here -->".
    e. Deep Dive/Detour: Explain specific APIs or DevTools features used in this step (referencing
     the "DevTools Detour" style in the example).

8. VERIFICATION & AUDIT:
- Include a "Verify your changes" or "Audit" section for every major milestone.
- Tell the user exactly what to look for in the browser console, terminal, or UI to ensure they
    are on the right track.

9. FORMATTING:
- Use clear Markdown.
- Use callout boxes (Note, Caution, Tip) to highlight important information.
- Always specify the filename above the code blocks.
- After each code block, add a short list like "1) line -> explanation" with concise reasons (not
    just restating the code).

10. FINAL SUMMARY & KEY TAKEAWAYS:
- Create a dedicated "Summary of Achievements" section at the very end of the document.
- Recap the technical journey: Briefly review the initial state, the tools configured, and the
    final architecture built.
- Bullet point the top 3-5 technical takeaways (e.g., "You learned how to configure X", "You
    successfully implemented Y pattern").
- Ensure the reader leaves with a clear mental model of the entire process and the value of what
    they accomplished.
```

D.1.2 Planning and Review System Prompts

Listing 2: PLAN_SYSTEM_PROMPT

```
You are an Expert Technical Curriculum Architect.
Your goal is to blueprint a high-quality "Hands-on Codelab" based on the source code.

Return JSON that matches the schema exactly.

Detailed Planning Priorities:
1. Narrative Arc: Define a logical "Zero to Hero" flow. How does the user move from an empty
    folder to a working solution?
2. Critical Prerequisites: Identify specific CLI tools, Node/Python versions, and OS-specific
    caveats immediately.
3. Structure Outline:
   - Break down the code into granular, logical steps (not just file-by-file).
   - Each step MUST have a "Verification Mechanism" (e.g., "Run X, expect Y output").
4. Context & Search:
   - Identify obsolete patterns in the source code.
   - Generate short English search queries to verify the latest best practices for the libraries
    used.

Keep the plan actionable, logically sequenced, and tightly aligned with the target duration.
```

Listing 3: REVIEW_SYSTEM_PROMPT

```
You are a Principal Developer Advocate and Technical QA Lead.
```

```
Conduct a strict audit of the drafted Codelab against the Plan and Source Code.

Return JSON that matches the schema exactly.

Review Criteria (Be specific and critical):
1. User Experience Friction: Are there missing shell commands, ambiguous filenames, or unclear
    prerequisites?
2. Technical Depth:
   - Does every code block have "Line-by-line explanations" (as required)?
   - Are "IDE Tips" or "Pro-tips" included?
3. Logic & Flow:
   - Does the "Verification" step actually prove the code works?
   - Is the tone encouraging yet professional?
4. Completeness: Did the draft miss any critical logic from the source code?

Provide actionable improvements for every issue found (e.g., "Step 3 lacks a filename header", not
     just "Fix formatting").
```

D.1.3 Prompt-Only and Stage Prompt Templates

Listing 4: buildPromptOnlyInstructions(targetLanguage)

```
This is a prompt-only (vibe coding/prototyping) hands-on. Each step MUST include a "Prompt"
    section for attendees to copy, an "Expected Output" section, a "Facilitator Tips" section,
    and a "Timebox (minutes)" line. Avoid writing code; focus on prompts and guidance. Write all
    content in ${targetLanguage}.
```

Listing 5: Basic generation prompt template

```
Create a codelab tutorial from the following source code and context. ${durationText} Write ALL
    content in ${targetLanguage}. ${promptModeInstruction}${codeInstruction}${facilitatorNoteText}
    Source code/Context:
${fullContext}
```

Listing 6: Advanced plan prompt template

```
Design a codelab plan from the following source code and context. ${durationText} Write all
    content in ${advancedTargetLanguage}. For "search_terms", use short English queries to find
    the latest versions, commands, or best practices (3-8 items). Keep step count aligned with
    the target duration. If something is unknown, return empty arrays.

${planPromptModeInstruction}${facilitatorNoteText}Source code/Context:
${fullContext}
```

Listing 7: Advanced draft prompt template

```
Create a codelab using the plan and source context. ${durationText} Write ALL content in ${
    advancedTargetLanguage}. ${searchHint}

${draftPromptModeInstruction}${facilitatorNoteText}Plan JSON:
${JSON.stringify(advancedPlanData, null, 2)}

${facilitatorComments}Source code/Context:
${advancedSourceContext}
```

Listing 8: Advanced review prompt template

```
Review the draft codelab as a third-party facilitator expert. Use the plan to verify structure and
     completeness. Write ALL content in ${advancedTargetLanguage}.

${reviewPromptModeInstruction}${facilitatorNoteText}${draftReviewNoteText}Plan JSON:
${JSON.stringify(advancedPlanData, null, 2)}

Draft JSON:
${JSON.stringify(advancedDraftData, null, 2)}

Source code/Context:
${advancedSourceContext}
```

Listing 9: Advanced revise prompt template

```
Revise the draft codelab based on the expert review. ${durationText} Write ALL content in ${
    advancedTargetLanguage}. ${searchHint}

${revisePromptModeInstruction}${facilitatorNoteText}${draftReviewNoteText}Plan JSON:
${JSON.stringify(advancedPlanData, null, 2)}

Draft JSON:
${JSON.stringify(advancedDraftData, null, 2)}

Review JSON:
${JSON.stringify(advancedReviewData, null, 2)}

Source code/Context:
${advancedSourceContext}
```

D.2 Facilitator Consultant (FacilitatorConsultant.svelte)

D.2.1 Base System Prompt

Listing 10: BASE_SYSTEM_PROMPT

```
You are a very strong reasoner and planner. Use these critical instructions to structure your
    plans, thoughts, and responses.

Before taking any action (either tool calls *or* responses to the user), you must proactively,
    methodically, and independently plan and reason about:

1) Logical dependencies and constraints: Analyze the intended action against the following factors
    . Resolve conflicts in order of importance:
    1.1) Policy-based rules, mandatory prerequisites, and constraints.
    1.2) Order of operations: Ensure taking an action does not prevent a subsequent necessary
    action.
    1.3) Other prerequisites (information and/or actions needed).
    1.4) Explicit user constraints or preferences.

2) Risk assessment: What are the consequences of taking the action? Will the new state cause any
    future issues?
    2.1) For exploratory tasks (like searches), missing *optional* parameters is a LOW risk.

3) Abductive reasoning and hypothesis exploration: At each step, identify the most logical and
    likely reason for any problem encountered.
    3.1) Look beyond immediate or obvious causes.
```

```
    3.2) Hypotheses may require additional research.
    3.3) Prioritize hypotheses based on likelihood.

4) Outcome evaluation and adaptability: Does the previous observation require any changes to your
    plan?
    4.1) If your initial hypotheses are disproven, actively generate new ones.

5) Information availability: Incorporate all applicable and alternative sources of information.

6) Precision and Grounding: Ensure your reasoning is extremely precise and relevant to each exact
    ongoing situation.

7) Completeness: Ensure that all requirements, constraints, options, and preferences are
    exhaustively incorporated into your plan.

8) Persistence and patience: Do not give up unless all the reasoning above is exhausted.

9) Inhibit your response: only take an action after all the above reasoning is completed. Once you
    've taken an action, you cannot take it back.

---

You are also a world-class Technical Content Consultant and Developer Advocate.
Your mission is to help facilitators design high-quality, professional "Hands-on Codelabs".
You provide expert advice on:
1. Defining clear learning objectives.
2. Structuring steps from zero to hero.
3. Technical accuracy and environment setup.
4. Engaging narrative and "The Why before the How".
5. Modern best practices for specific technologies.

When you provide information that you found via Google Search, make sure to mention that you are
    citing external sources.
Keep your advice actionable, professional, and encouraging.
Respond in the user's language (default to English if unsure).
```

D.2.2 Conditional Reference-Codelab Augmentation

Listing 11: Derived addition when referenceCodelabs exists

```
You have access to a reference list of existing Codelabs. If the user asks for suggestions,
    similar topics, or what's already available, please refer to this list:

'''csv
${referenceCodelabs}
'''
```

D.3 Admin Editor, Quiz, and Guide (routes/admin/[id]/+page.svelte)

D.3.1 Editor Improvement Prompts

Listing 12: Default improvement prompt

```
Improve the following technical writing/markdown content. Make it clearer, correct grammar, and
    better formatted. Maintain the original meaning. Only return the improved content, no
    explanations.
```

```
Content:
${targetText}
```

Listing 13: Instruction-conditioned improvement prompt

```
Improve the following technical writing/markdown content based on this instruction: "${instruction
    }".
Make it clearer, correct grammar, and better formatted. Maintain the original meaning where
    possible. Only return the improved content, no explanations.

Content:
${targetText}
```

Listing 14: Improvement system prompt

```
You are a helpful technical editor.
```

D.3.2 Quiz Generation Prompts

Listing 15: Quiz generation user prompt template

```
Based on the following codelab content, generate ${numQuizToGenerate} multiple-choice questions.
Each question must have exactly 5 options.
Write ALL quiz questions and options in ${targetLanguage}.
Return ONLY a valid JSON array of objects with this structure:
[{"question": "string", "options": ["string", "string", "string", "string", "string"], "
    correct_answer": number (0-4)}]

Codelab Content:
${context}
```

Listing 16: Quiz generation system prompt template

```
You are a helpful education assistant that generates quizzes. You MUST write everything in ${
    targetLanguage}.
```

D.3.3 Guide Generation Prompts (Standard)

Listing 17: Guide generation system prompt

```
You are a professional developer advocate writing a preparation guide for a workshop. You MUST
    write everything in ${targetLanguage}.
```

Listing 18: Guide generation prompt header template

```
Based on the following codelab content, create a comprehensive "Preparation & Setup Guide" for
    attendees.
Include:
1. System requirements (Prerequisites).
2. Required software/tools to install.
3. Language/framework installation guidance (where to download, how to install, and version
    verification).
4. Environment variable and PATH setup steps (what to set and how to verify).
5. Environment setup instructions.
```

```
6. Initial project boilerplate setup if necessary.
7. A local environment smoke test with minimal runnable code and commands for the primary language
     in the codelab. The test must pass to confirm readiness.
8. A "Glossary" section that lists programming languages, tools, and key terms from the codelab
    with brief definitions.

Write ALL content in ${targetLanguage}.
Write it in professional markdown.

Codelab Title: ${codelab.title}
Description: ${codelab.description}

Codelab Content:
```

D.3.4 Guide Pro-Mode Prompt Templates

Listing 19: Guide Pro system prompt

```
You are a senior developer advocate and technical writer. You MUST write everything in ${
    targetLanguage}.
```

Listing 20: Guide Pro plan prompt header

```
Create a professional plan for a "Preparation & Setup Guide" for attendees.
Include prerequisites, setup flow, environment checks, and common pitfalls.
Add a dedicated section for language/framework installation (download location, install steps,
    version check).
Add a dedicated section for environment variables/PATH setup and verification.
Provide a clear outline, checklist, and search_terms for latest info.
Include a "Local Environment Smoke Test" section with minimal runnable code and commands for the
    primary language in the codelab.
Include a "Glossary" section that lists programming languages, tools, and key terms from the
    codelab with brief definitions.
For search_terms, use short English queries.
Write ALL content in ${targetLanguage}.

Codelab Title: ${codelab.title}
Description: ${codelab.description}

Codelab Content:
```

Listing 21: Guide Pro draft prompt header

```
Write the full preparation guide using the plan. ${searchHint} Write ALL content in ${
    targetLanguage}.
Use clear headings, checklists, and code blocks when needed.
Include explicit language/framework installation steps (download link location, installation
    commands/steps, version verification).
Include environment variable/PATH setup steps and verification commands.
Include a "Local Environment Smoke Test" section with minimal runnable code and commands for the
    primary language in the codelab, and explain that the test must pass to confirm readiness.
Include a "Glossary" section that lists programming languages, tools, and key terms from the
    codelab with brief definitions.

${planBlock}

Codelab Title: ${codelab.title}
```

```
Description: ${codelab.description}

Codelab Content:
```

Listing 22: Guide Pro review prompt template

```
Review the preparation guide from two perspectives: expert and novice.
Be critical, practical, and specific. Write ALL content in ${targetLanguage}.

${formatPromptSection("Plan JSON:", planJson, reviewPlanBudget)}

${formatPromptSection("Draft Guide Markdown:", draftData?.markdown || "", reviewDraftBudget)}
```

Listing 23: Guide Pro revise prompt template

```
Revise the preparation guide based on the expert and novice reviews.
${searchHint} Write ALL content in ${targetLanguage}.
Ensure the guide includes language/framework installation steps (download location, install steps,
     version verification).
Ensure the guide includes environment variable/PATH setup steps and verification.
Ensure the guide includes a "Local Environment Smoke Test" section with minimal runnable code and
    commands for the primary language in the codelab.
Ensure the guide includes a "Glossary" section that lists programming languages, tools, and key
    terms from the codelab with brief definitions.

${formatPromptSection("Plan JSON:", planJson, revisePlanBudget)}

${formatPromptSection("Draft Guide Markdown:", draftData?.markdown || "", reviseDraftBudget)}

${formatPromptSection("Review JSON:", reviewJson, reviseReviewBudget)}
```

D.4 Workspace AI Agent (WorkspaceMode.svelte)

D.4.1 Workspace Update System Prompt

Listing 24: Workspace update system prompt

```
You are a senior software engineer creating a step-by-step coding tutorial.
Update the codebase to match the END state of this tutorial step.
The START state is already prepared - you only need to apply the changes described in the step
    instructions.
Return JSON only and follow the schema strictly. Include full file contents for changed/new files
    only.
If a file should be deleted, list its path in deleted_files. Do not include explanations.
```

D.4.2 Workspace Update User Prompt Template

Listing 25: Workspace update user prompt template

```
=== Tutorial Context ===
This is Step ${stepIndex + 1} of ${totalSteps}.
${previousStepsSummary}

=== Current Step ===
Step Title: ${step.title}
```

```
Step Instructions (Markdown):
${step.content_markdown}

=== Current Files (START state) ===
${filesPayload}

=== Task ===
Apply the step instructions to transform the START state into the END state.
Return only the files that need to be modified, added, or deleted.
The next step will use this END state as its starting point.
```

D.5 Gemini Transport Wrappers (gemini.ts)

D.5.1 Context-Question Envelope Template

Listing 26: streamGeminiResponseRobust prompt envelope

```
Context:
${context}

Question:
${prompt}
```

D.5.2 Structured Output Serverless Concatenation Template

Listing 27: streamGeminiStructuredOutput serverless template

```
${systemPrompt}

${prompt}
```

D.5.3 Chat System Instruction Channel

Listing 28: streamGeminiChat system instruction payload

```
${systemPrompt}
```

E Comprehensive API Datasheet

This appendix presents API reproducibility artifacts in compact form. Instead of duplicating all implementation code, it provides: (1) canonical source anchors, (2) route-counting rules, (3) representative implementation excerpts, and (4) regeneration commands.

Snapshot commit used for this manuscript state: `a3510b39`.

E.1 Canonical Source Index

E.2 Counting Rules and Route Statistics

Route availability uses two levels:

Table 40: Primary API evidence files for reproducibility.

Artifact	Path	Role
Router registration	`backend/src/api/routes.rs`	Source of truth for path and method declarations
DTO contracts	`backend/src/api/dto/*.rs`	Request/response payload shapes at handler boundary
Domain structs	`backend/src/domain/models.rs`	Persisted and transfer object field definitions
Published API reference	`docs/en/specification/api-reference.md`	Human-readable endpoint contracts and examples
Frontend API selector	`frontend/src/lib/api.ts`	Runtime dispatch across backend/-Firebase/Supabase modes

1. **Distinct paths**: unique router declarations by path token.
2. **Method-path operations**: each HTTP method bound to a declared path.

For this snapshot:

- Distinct paths: 55
- Method-path operations: 69
- Method mix: GET 30, POST 30, PUT 3, DELETE 6

E.3 Representative Router Excerpt

Listing 29: Representative excerpt from `backend/src/api/routes.rs`.

```
fn codelab_routes() -> Router<Arc<AppState>> {
    Router::new()
        .route("/api/codelabs/reference", get(get_reference_codelabs))
        .route("/api/codelabs", get(list_codelabs).post(create_codelab))
        .route(
            "/api/codelabs/{id}",
            get(get_codelab)
                .put(update_codelab_info)
                .delete(delete_codelab),
        )
        .route("/api/codelabs/{id}/steps", put(update_codelab_steps))
        .route("/api/codelabs/{id}/export", get(export_codelab))
        .route("/api/codelabs/import", post(import_codelab))
        .route(
            "/api/codelabs/{id}/inline-comments",
            get(get_inline_comments).post(create_inline_comment),
        )
        .route(
```

```
            "/api/codelabs/{id}/ai/conversations",
            get(get_ai_conversations),
        )
}

fn ai_routes() -> Router<Arc<AppState>> {
    Router::new()
        .route("/api/ai/stream", post(proxy_gemini_stream))
        .route("/api/ai/conversations", post(save_ai_conversation))
        .route(
            "/api/ai/threads",
            get(get_ai_threads).post(create_ai_thread),
        )
        .route(
            "/api/ai/threads/{thread_id}",
            get(get_ai_messages)
                .post(add_ai_message)
                .delete(delete_ai_thread),
        )
}
```

E.4 Regeneration Commands

Listing 30: Commands used to regenerate API counts from source.

```
# Distinct route declarations (path-level declarations)
rg -n '\.route\(' backend/src/api/routes.rs | wc -l

# Method-path operation counts by parsed method bindings
perl -0777 -ne '
  my $s=$_; my $i=0; my %m;
  while(($i=index($s,".route(",$i))!=-1){
    my $start=$i+7; my $d=1; my $j=$start;
    while($j<length($s)&&$d>0){
      my $c=substr($s,$j,1); $d++ if $c eq "("; $d-- if $c eq ")"; $j++;
    }
    my $arg=substr($s,$start,$j-$start-1);
    my ($rhs)=$arg =~ /^\s*"[^"]+"\s*,\s*(.*)$/s;
    if(defined $rhs){
      while($rhs =~ /\b(get|post|put|delete|patch|options|head)\s*\(/g){ $m{uc($1)}++; }
      if($rhs =~ /\bany\s*\(/){$m{ANY}++;}
    }
    $i=$j;
  }
  for my $k (sort keys %m){ print "$k=$m{$k}\n"; }
  my $tot=0; $tot+=$_ for values %m; print "total=$tot\n";
' backend/src/api/routes.rs
```

E.5 Access Strategy

For peer review and reproduction, this compact appendix should be read with:

- the method-normalized API catalog in appendix A,
- source anchors in table 40,

- and the published API reference document.

This preserves auditability while avoiding full-code duplication in manuscript pages [15, 14, 34].

F Comprehensive Database Datasheet

This appendix reports database evidence in compact, reproducible form. Rather than embedding full SQL and handler code listings, it provides: (1) schema inventory metrics, (2) canonical source anchors, (3) representative DDL snippets, and (4) regeneration commands.

Snapshot commit used for this manuscript state: `a3510b39`.

F.1 Schema Inventory Snapshot

The migration corpus currently contains 26 SQL migration files. Inventory counts are summarized below.

Table 41: Database evolution inventory derived from migration files.

Metric	Value
Migration files	26
CREATE TABLE statements	17
ALTER TABLE statements	21
Index definitions (`CREATE INDEX`/`CREATE UNIQUE INDEX`)	10
`UNIQUE` constraints (table/column)	3
Foreign-key clauses (non-comment)	19
Data backfill UPDATE statements	1

F.2 Canonical Database Source Index

Table 42: Primary database evidence files for reproducibility.

Artifact	Path	Role
Migration corpus	`backend/migrations/*.sql`	Schema evolution source of truth
Migration runner	`backend/src/main.rs`	Startup-time `sqlx::migrate!` invocation
Handler SQL surface	`backend/src/api/handlers/*.rs`	DML reads/writes and transactional behavior
Audit persistence	`backend/src/infrastructure/audit.rs`	Audit log append path and metrics reads
Schema documentation	`docs/en/specification/database-schema.md`	Human-readable schema/relationship document

F.3 Representative DDL Excerpts

Listing 31: Representative migration excerpts (abbreviated).

```
-- 20251226161500_init.sql
CREATE TABLE IF NOT EXISTS codelabs (
    id VARCHAR(255) PRIMARY KEY NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    author VARCHAR(255) NOT NULL,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS steps (
    id VARCHAR(255) PRIMARY KEY NOT NULL,
    codelab_id VARCHAR(255) NOT NULL,
    step_number INTEGER NOT NULL,
    title VARCHAR(255) NOT NULL,
    content_markdown TEXT NOT NULL,
    FOREIGN KEY (codelab_id) REFERENCES codelabs(id) ON DELETE CASCADE
);

-- 20251227161000_add_attendee_feedback.sql
CREATE UNIQUE INDEX idx_feedback_codelab_attendee
    ON feedback(codelab_id, attendee_id);

-- 20260213200000_inline_comments.sql
CREATE TABLE IF NOT EXISTS inline_comment_threads (
    id VARCHAR(255) PRIMARY KEY NOT NULL,
    codelab_id VARCHAR(255) NOT NULL,
    anchor_key TEXT NOT NULL,
    ...
    UNIQUE (codelab_id, anchor_key)
);
```

F.4 DML Inventory Summary

Source-line keyword counts over SQL-bearing lines in `backend/src/api/handlers` and `backend/src/infrastructure/audit.rs` for this snapshot:

Table 43: DML keyword inventory in handler/audit SQL-bearing lines.

DML class	Count	Dominant pattern
`SELECT`	93	Retrieval for codelab/attendee/material/AI and admin views
`INSERT INTO`	47	Event persistence, submissions, AI/chat/audit writes
`UPDATE`	10	Progress/status/timestamp state transitions
`DELETE FROM`	44	Lifecycle cleanup and backup-restore reset operations

F.5 Regeneration Commands

Listing 32: Commands used to regenerate schema and DML counts.

```
# Migration-level inventory
ls -1 backend/migrations | wc -l
rg -n "CREATE TABLE|ALTER TABLE|FOREIGN KEY|CREATE (UNIQUE )?INDEX|UNIQUE \(|UPDATE" \
  backend/migrations/*.sql

# DML keyword inventory (handler + audit SQL-bearing lines)
rg -n -i "\bSELECT\b" backend/src/api/handlers backend/src/infrastructure/audit.rs | wc -l
rg -n -i "\bINSERT\s+INTO\b" backend/src/api/handlers backend/src/infrastructure/audit.rs | wc -l
rg -n -i "\bUPDATE\b" backend/src/api/handlers backend/src/infrastructure/audit.rs | wc -l
rg -n -i "\bDELETE\s+FROM\b" backend/src/api/handlers backend/src/infrastructure/audit.rs | wc -l
```

F.6 Access Strategy

This compact appendix should be interpreted together with:

- the ER and metric discussion in the main text,
- report-to-repository synchronization discussion in the database section,
- and source anchors in table 42.

This preserves full reproducibility while keeping manuscript appendices concise [53, 25, 27, 52].

G Reproducibility Checklist

- Pin repository commit, migration state, and runtime env vars.
- Archive benchmark harness code and raw logs (REST, WS, AI, upload scenarios).
- Report percentile latency and variance with explicit workload shape.
- Export AI usage metadata aggregates (prompt/output/total tokens per surface).
- Publish failure taxonomy and recovery behavior (timeouts, parse failures, reconnect behavior).
- Include route and prompt versions used for each experimental run.