FINAL PROJECT REPORT

WISEC: Engineering a Cross-Platform Wildfire Preparedness and Community Resilience Platform

Architecture, implementation, validation, and evolution from WISEC-v1

Prepared for	WISEC project stakeholders and future development teams
Prepared by	Emon Sarker and Jason Xu, Solution Scholars
Supervised by	Dr. Adeniyi Asiyanbi and Dr. Ifeoma Adaji
Affiliations	Community Culture and Global Studies (Asiyanbi); Computer Science, Mathematics, Physics & Statistics (Adaji, Sarker, and Xu)
Sponsored by	Solutions Scholar program of the UBC Climate Solutions Research Collective
Repository basis	`wisec-monorepo`, branch `main`, commit `8cc7c3b`
Report date	August 14, 2026
Scope	Web application, Expo mobile application, NestJS API, shared package, deployment assets, and comparison with `WISEC-v1`

This report describes the state of the local repository snapshot inspected on the date above. It distinguishes implemented code, locally verified behavior, and future recommendations.

Executive Summary

Wildfire preparedness is not a single action. It is a continuing cycle in which residents assess risk, improve their homes and plans, locate trustworthy emergency information, respond under time pressure, and recover after disruption. The Canadian emergency-management framework accordingly treats prevention and mitigation, preparedness, response, and recovery as connected responsibilities shared across governments, organizations, communities, and individuals [1, 2]. This need is especially visible after Canada's historic 2023 season, when more than 6,800 wildland fires burned over 14.6 million hectares and more than 230,000 people were evacuated [3].

WISEC is a cross-platform wildfire preparedness and community resilience system designed around that full cycle. The current implementation consolidates a responsive React web application, an Expo/React Native mobile application, a NestJS API, and shared TypeScript contracts in one pnpm monorepo. The platform connects bilingual onboarding, location selection, readiness surveys, personalized action tasks, scheduling and recurrence, emergency and recovery resources, a location-scoped community feed, current-fire visualization, opt-in proximity notifications, and a source-aware conversational assistant. An administrative interface supports user access approval, survey versioning, task and resource management, knowledge-base documents, disclaimers, system settings, and approved-source domains.

This report evaluates the repository as an engineered system rather than claiming field effectiveness that has not yet been measured. Evidence was drawn from the source tree, architecture notes, migrations, route definitions, UI specifications, deployment assets, recent version-control history, reproducible local checks, local Web and Expo Go runtime captures, and public deployment checks. At commit `8cc7c3b`, the main build completed successfully for the shared package, API, and web client. All 32 API test suites invoked by the root test command passed (194 tests total), and the mobile TypeScript project passed `tsc -noEmit`. On August 14, 2026, the public web application at `https://wisec.ca/`, the API readiness route, and a database-backed locations query all returned HTTP 200 through Google-hosted infrastructure [4].

The comparison with `WISEC-v1` shows a substantial architectural and product upgrade. WISEC-v1 was organized as separate React Native and Express applications, used MongoDB and Sanity, called OpenAI directly from the mobile client, and retrieved wildfire events directly from an external API. MongoDB applies only to that legacy generation: the current system uses PostgreSQL through MikroORM, with explicit relations and migrations. It also moves secrets and AI orchestration to the server, adds a ports-and-adapters backend, provides first-party web and admin experiences, shares types across clients, introduces English/French content, persists chat citations and task progress, and operationalizes current-fire ingestion and proximity alerts. A static inventory found roughly 73,700 source lines across the new API, web, mobile, and shared packages versus about 21,400 across the v1 API and mobile source trees. Decorated API operations increased from 29 legacy Express route bindings to 96 NestJS route handlers; these counts indicate scope, not equivalent quality.

The central conclusion is that WISEC has advanced from a mobile prototype into a coherent, publicly deployed multi-surface platform with a credible domain architecture and a demonstrably buildable backend/web baseline. The deployed topology places the web and API services on GCP and the production PostgreSQL database on Neon. Its strongest qualities are the end-to-end readiness workflow, location-aware information model, safety-oriented AI pipeline, and maintainable content operations. Its most important remaining work is equally clear: establish broader web/mobile automated coverage, add end-to-end and load/security validation, resolve documentation drift in the fire-data subsystem, reduce large web bundles, harden authentication and rate limiting, replace in-process scheduling locks with distributed controls where required, perform formal accessibility and bilingual-content reviews, and evaluate the product with residents and emergency-management partners. These steps should precede any claim that the platform improves real-world preparedness outcomes.

Key finding

Overall assessment. The repository supports a strong technical final-project outcome: the target architecture builds, the API has meaningful automated coverage, and the major user journeys are represented on both web and mobile. The next phase should prioritize evidence of reliability and public-safety usability rather than adding more surface area.

Contents

1 Introduction

1.1 Problem context

Wildland fire is both an ecological process and a growing source of risk to people, housing, health, infrastructure, and local economies. Natural Resources Canada describes the 2023 season as historic and contrasts its 14.6 million hectares burned with a long-term annual average of approximately 2.1 million hectares [3]. The practical consequence is not only a need for better suppression capacity. Residents require clear information and repeatable actions before an event, trustworthy local contacts during an event, and navigable support after an event.

The public information environment is fragmented. Preparedness guidance may be national, provincial, municipal, or community-specific. Fire status data may originate from several agencies. Recovery resources vary by location and can change. Social media can provide timely observations but can also mix verified and unverified claims. Generic search and chat systems can answer broadly while obscuring the authority, locality, or date of a source. In this setting, the application problem is one of orchestration: helping a user move from awareness to action while making the provenance and limits of information visible.

FireSmart Canada frames wildfire resilience as shared responsibility and emphasizes practical, science-based actions that households and neighbourhoods can take [5, 6]. Public Safety Canada's resilience strategy similarly calls for accessible risk information, whole-of-society participation, and strong public communication across mitigation, preparedness, response, and recovery [1]. These principles motivate the phase-based organization of WISEC.

1.2 Project objective

The project objective is to provide a single digital environment in which a Canadian resident can:

1. establish language and location context;
2. assess household readiness through configurable surveys;
3. receive and schedule concrete mitigation tasks;
4. monitor current wildfire information near the selected location;
5. find local response and recovery resources with transparent geographic fallback;
6. participate in a location-scoped community information space; and
7. ask wildfire questions through an assistant that exposes sources and can convert advice into personal tasks.

The platform must also be operable by a project team after initial delivery. This creates a second objective: move volatile content and policy settings out of hard-coded client logic and into managed, versioned backend workflows. Survey schemas, task definitions, resources, disclaimers, approved domains, and knowledge-base documents therefore have administrative interfaces and corresponding APIs; comprehensive administrative audit logging remains future work.

1.3 Project team and supervision

The supplied WISEC project poster identifies Emon Sarker and Jason Xu as the Solution Scholars. It identifies Dr. Adeniyi Asiyanbi and Dr. Ifeoma Adaji as the project supervisors. The poster associates Dr. Asiyanbi with the Department of Community Culture and Global Studies, and Dr. Adaji, Emon Sarker, and Jason Xu with the Department of Computer Science, Mathematics, Physics & Statistics. It also credits sponsorship to the Solutions Scholar program of the UBC Climate Solutions Research Collective [7].

1.4 Design principles

Five principles guided the interpretation of the current implementation.

Action over passive information.
A readiness score should lead to a task, a task should be schedulable, and a chat recommendation should be convertible into a journal item.

Location as context.
Fire markers, community posts, and emergency resources are useful only when the interface explains their geographic relevance. The city–province–country hierarchy is therefore a domain primitive rather than a visual filter alone.

Trust should be legible.
Official and curated sources should be distinguishable from unvetted web content or social posts. AI responses should retain citations, and emergency information should never be presented as a replacement for official alerts.

Continuity across devices.
A public-facing web client lowers access barriers, while the mobile client supports on-the-go use. Shared contracts reduce drift between them.

Content can evolve while retaining references.
Survey questions, legal disclaimers, and preparedness resources change. Versioned records associate submissions and consents with specific version identifiers; strict historical immutability still depends on enforcement in the application and database.

These principles align with the Government of Canada's digital guidance on simple interactions, clear language, privacy-aware collection, iterative improvement, and support for both official languages [8, 9]. They also imply a high bar: an emergency-oriented application should be understandable, accessible, and robust under stress, not merely feature-rich.

1.5 Scope of this report

The primary subject is the local `wisec-monorepo` snapshot on branch `main`, commit `8cc7c3b`, last committed July 27, 2026. The analysis covers:

- the monorepo and backend architecture;
- the web and mobile user journeys;
- survey, task, fire, resource, community, chat, identity, and admin subsystems;
- data modelling and deployment configuration;
- a technical comparison with `WISEC-v1`; and
- local build, type-check, automated-test results, runtime mobile inspection, and public deployment checks gathered on August 14, 2026.

The report confirms that the public web application and API were reachable on the review date, but it does *not* claim a service-level availability target, emergency-service certification, usability at population scale, or measured improvement in preparedness. A successful public request and repository configuration cannot substitute for GCP Console records, monitoring history, backup and recovery evidence, security assessment, accessibility conformance, or a user study. This distinction is maintained throughout the evaluation.

Risk and interpretation

Public-safety boundary. Current-fire layers, community posts, and AI answers can support awareness, but they are not authoritative evacuation orders or emergency dispatch. Product copy, testing, and operations should consistently direct users to local authorities and 9-1-1 where appropriate.

2 Project Approach and Evidence Base

2.1 Repository review method

The report was produced as a structured technical review of two local codebases and one reference report. The reference report established the expected scale and tone: a single-column letter-sized report of roughly twenty pages, supported by figures, tables, references, limitations, and appendices. Its subject matter was not reused.

The current repository was then reviewed from four complementary perspectives:

1. **Product structure:** routes, screens, user-flow specifications, translations, and visual design artifacts;
2. **Domain implementation:** NestJS modules, interactors, entities, repositories, migrations, external-service adapters, shared types, and API controllers;
3. **Operations:** deployment, database startup, configuration, health checks, rollback assets, and public endpoint reachability; and
4. **Verification:** source inventory, build commands, automated tests, and type checking.

The `WISEC-v1` review used the same categories where possible. Its React Native client, Express API, route files, MongoDB models, Sanity integration, direct OpenAI client, and EONET client were inspected to identify substantive changes. Because the repositories use different structures and abstractions, quantitative comparisons are presented as scale indicators rather than controlled productivity or quality measurements.

2.2 Evidence hierarchy

Three labels are used to avoid overstating maturity:

Table 1: Interpretation of evidence used in the report.

Label	Meaning	Typical evidence
Verified	Executed successfully in the review environment or directly asserted by a passing test	Build/test output, compiler result, public HTTP response
Observed in code	Implemented or configured in the inspected source, but not exercised end-to-end during this review	Controllers, interactors, UI components, migrations, deployment files
Gap	Missing, incomplete, or not evidenced sufficiently for a stronger claim	Absent user study, documentation mismatch, missing operational proof

This hierarchy is important for features that depend on credentials or external infrastructure. For example, a CIFFC adapter and a fire-alert scheduler are clearly present in code, but the review did not send a real alert to a resident or verify a production cron run. Those capabilities are described as implemented, not operationally proven.

2.3 Static inventory

Source files were counted under the main `src` directories, excluding dependencies, build output, native vendor files, and content exports. Lines were counted as physical text lines. API operation counts were derived from NestJS HTTP method decorators in the current API and Express route bindings in v1. Table 2 summarizes the result.

Table 2: Static source inventory for the reviewed snapshots.

Area	Files	Lines	Test files	Interpretation
Current API	514	23,594	32	Modular backend with the strongest automated coverage
Current Web	402	31,584	1	Broad user/admin surface; production build verified
Current Mobile	126	18,370	0	Cross-platform feature surface; type-check and Expo Go runtime verified
Current Shared	24	196	0	Small contract package used by all applications
WISEC-v1 API	78	5,485	0	Compact Express/MongoDB service
WISEC-v1 Mobile	188	15,891	1	Native-first prototype with one app smoke test

The current system contains 18 controller classes, 96 decorated HTTP operations, 22 migration files, and 29 concrete ORM entity mappings in the inspected source. Some mappings belong to a currently disabled gamification module, so an entity count should not be read as a production table count. The repository's own older data-layer overview still states 15 tables; this is one example of documentation lag identified during the review.

2.4 Reproducible validation

The following commands were executed from the monorepo root:

Table 3: Local validation commands and outcomes, August 14, 2026.

Command	Outcome	Scope
`pnpm build`	Passed	Shared package, NestJS API, and Vite web production build
`pnpm test`	Passed	32 API suites; 194 tests passed
`pnpm -filter @wisec/mobile exec tsc -noEmit`	Passed	Mobile TypeScript compilation
Expo Go runtime session	Completed	Mobile client launched on a Pixel 7 API 36 emulator against the local API and seeded PostgreSQL
Local Web runtime capture	Completed	Vite client rendered representative desktop views against the local Nest API and seeded PostgreSQL
`GET https://wisec.ca/`	Passed	Public Vite application returned HTTP 200 through Google Frontend
`GET api.wisec.ca/health/readiness`	Passed	Public API returned HTTP 200 and `readiness.status=up`
`GET api.wisec.ca/locations?take=1`	Passed	Database-backed public query returned HTTP 200 and location records

The web build also emitted code-size warnings: the main index bundle was approximately 572 kB minified and the admin system-values bundle approximately 1.53 MB, both above Vite's 500 kB warning threshold. These warnings do not invalidate the build, but they are relevant to performance planning.

2.5 Limitations of the method

This was a repository-centered technical evaluation supplemented by unauthenticated checks of the public deployment, a local desktop Web session, and a local Expo Go session. The production database provider was confirmed as Neon and is consistent with the checked-in Cloud Run manifest, Neon-tagged backend image, secret-managed `DATABASE_URL`, and API connection handling for `.neon.tech` hosts. The review did not include GCP or Neon Console access, production telemetry, direct inspection of the running database service, database migration against a clean remote instance, automated browser coverage across devices, production-signed APK/AAB installation testing, push-notification delivery, network fault injection, penetration testing, accessibility tooling, translation review by bilingual domain experts, or interviews with residents and emergency managers. No private credentials were used. The public checks establish reachability and a functioning database-backed API route, but not uptime history, backup status, or recovery readiness. The desktop and Expo Go captures establish that representative Web and mobile screens rendered against a migrated and seeded local database, but do not establish complete journey, cross-browser, production-binary, or device-fleet behavior. External factual context was limited to authoritative government, interagency, standards, and platform sources [3, 10, 11, 12].

The method is therefore well suited to answer, "What has been engineered, how is it organized, and what can be verified locally?" It is not sufficient to answer, "Does WISEC change resident behavior or improve outcomes during an actual wildfire?" That second question requires field evaluation and is included in the roadmap.

3 System Design and Architecture

3.1 Monorepo as the integration boundary

The current repository uses pnpm workspaces to place four primary deliverables under one versioned system: `@wisec/web`, `@wisec/api`, `@wisec/mobile`, and `@wisec/shared`. The shared package exports domain enums and TypeScript interfaces for roles, task status, survey category, location level, fire data, authentication, readiness, disclaimer data, and approved domains. This reduces the risk that web, mobile, and backend silently assign different meanings to the same values.

The monorepo also makes the build dependency explicit: shared contracts are built before the API and web applications. Docker and Cloud Build configurations use the repository root as their build context so that each application can consume the workspace package. This is a practical improvement over copying interfaces across independent repositories.

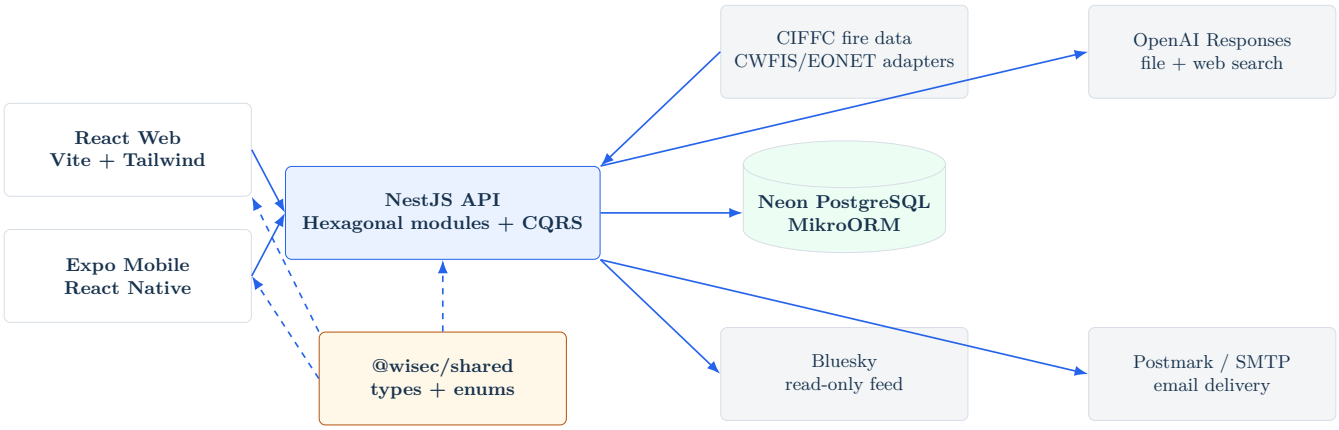


Figure 1: Logical system context reconstructed from the current source tree. External integrations are accessed through the backend except for client-side map rendering and outbound links.

3.2 Backend structure: ports, adapters, and use cases

The API is organized by domain module rather than by generic technical folders. Major modules include identity and access management, surveys, tasks, locations, community, chat, retrieval-augmented generation, fire detection, notifications, disclaimers, news, system values, admin, and health. Within a mature module, the source is split into:

- **application domain objects** that represent business state without HTTP concerns;
- **commands, queries, and interactors** that implement use cases;
- **ports** that define repository or service interfaces;
- **infrastructure adapters** for MikroORM, OpenAI, email, CIFFC, CWFIS, EONET, and Bluesky; and
- **presentation components** for controllers and request/response DTOs.

This pattern supports local testing of business behavior with mocked ports. The passing tests for task transitions, survey submission, user registration, fire-data parsing, chat routing, whitelist matching, community-feed merging, password hashing, and external-client factories demonstrate that the abstraction boundaries are usable rather than decorative.

Cross-cutting controls include a global exception filter, a logging interceptor, validation DTOs, JWT guards, role guards, configuration loading, and optional Swagger/OpenAPI generation. The Swagger interface is enabled outside production and can be explicitly enabled in production. Repository notes report 90 operations at the time of that documentation; the current static scan found 96 decorators, illustrating both continuing development and the need to regenerate authoritative API documentation from source.

3.3 Data architecture

PostgreSQL and MikroORM replace the flexible but less explicitly relational MongoDB model used in v1. The current mappings make ownership and relationships concrete: users have profiles; profiles own task progress, survey

submissions, chat threads, community content, and alert preferences; surveys own numbered versions referenced by submissions; and location resources are attached to a geographic hierarchy. The current administrative API can update version content, so the implementation should not be described as enforcing immutable survey history.

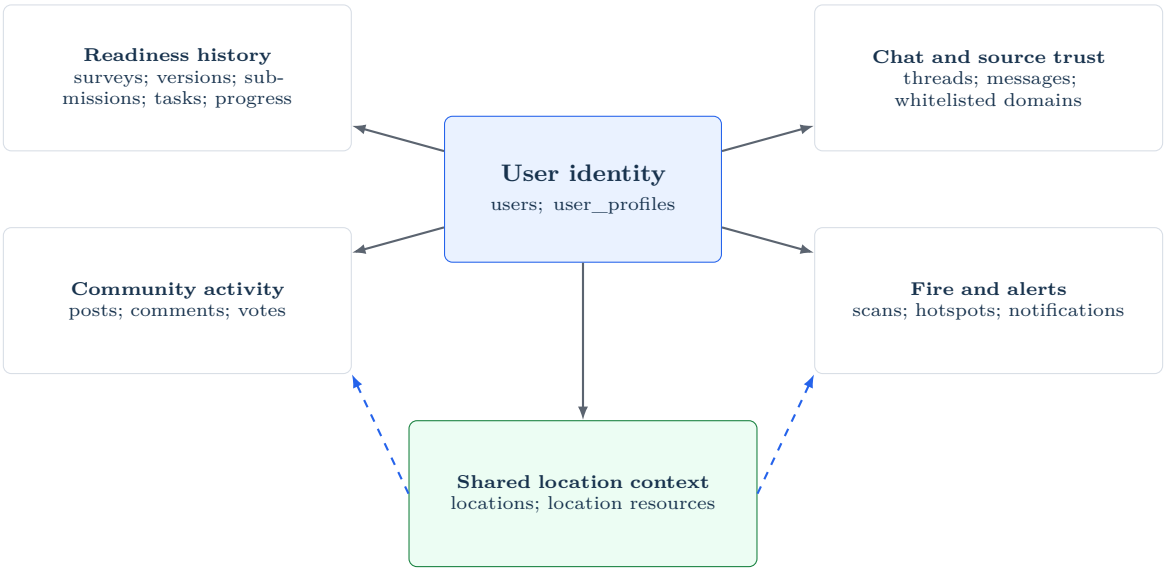


Figure 2: Simplified relationship view. It emphasizes the data needed for readiness history, location context, community activity, source-aware chat, and fire notifications rather than every mapped field.

Twenty-two dated migrations encode the system's progression from the initial schema through location hierarchy, scheduled and recurring tasks, community tables, RAG documents, disclaimer consent, email verification, approved domains, fire-detection records, alert notifications, bilingual task fields, task resource links, and resource helpfulness feedback. This is a material operational upgrade: schema evolution is reproducible and reviewable.

3.4 Location as a shared domain service

Locations are represented as a country–province–city tree with codes and optional coordinates. The same context drives three different forms of relevance:

1. **resources:** if a city has no matching response or recovery resource, the API falls back to the province and then country, returning metadata that tells the UI where the result came from;
2. **community:** the feed can scope internal and Bluesky posts to city, province, or Canada, with automatic fallback when a narrower feed is empty; and
3. **fire information:** the selected coordinates constrain map queries and drive the 100 km proximity calculation used by alerts.

This reuse is more valuable than three independent filters because it gives the platform one explainable answer to the question, "What does near me mean?" The remaining requirement is data governance: city coordinates, resource ownership, expiry dates, and fallback behavior should be monitored as managed reference data rather than seeded once and assumed current.

3.5 Deployment topology

The system has a live GCP deployment. On August 14, 2026, `https://wisec.ca/` served the production web entry point, `https://api.wisec.ca/health/readiness` returned HTTP 200 with `readiness.status=up`, and a public `/locations?take=1` request returned records from the application's persistence layer. The responses included `server: Google Frontend` and Google Cloud trace headers, consistent with the repository's Cloud Build and Cloud Run configuration [4].

The current persistence engine is PostgreSQL through MikroORM; MongoDB/Mongoose belongs to WISEC-v1 and is not the current database stack. The production PostgreSQL service is hosted by Neon and is connected to the

GCP-hosted API through a secret-managed `DATABASE_URL`. This topology is reflected in `backend-neon.yaml`, the deployed backend image tag, and connection code that automatically enables SSL for `.neon.tech` hosts. Earlier Cloud SQL and Cloud Run PostgreSQL-sidecar materials are superseded artifacts, not supported production alternatives; they should be archived or clearly marked as obsolete.

Key finding

Public reachability and the GCP–Neon topology are verified at the configuration level, but this is not proof of a monitored or resilient service. Production assurance still requires environment-specific migration drills, Neon backup/restore evidence, secret rotation, observability history, alerting, capacity testing, and rollback exercises.

4 Implemented Product Capabilities

4.1 A phase-based resident journey

The user-facing information architecture follows the emergency-management cycle: Home, Prepare, Mitigate, Respond, Recover, and Community. Diary, Ask AI, profile, settings, and task catalog functions extend that core. On desktop, a persistent sidebar and larger dashboard grid support scanning; on smaller screens, the same concepts move to a bottom navigation bar, compact header, and drawer. The Expo application mirrors the major views with native navigation stacks.

First-time web users encounter a sequence that establishes context before presenting the dashboard: language choice, versioned disclaimer consent, welcome, location selection, a short prepare assessment, an optional mitigation assessment, and an animated readiness reveal. Guests can complete assessments and store progress locally; authenticated users synchronize submissions and tasks with the backend. This “try before account” design lowers the entry barrier, but it also creates a dual-state system that must be tested carefully when a guest later registers.

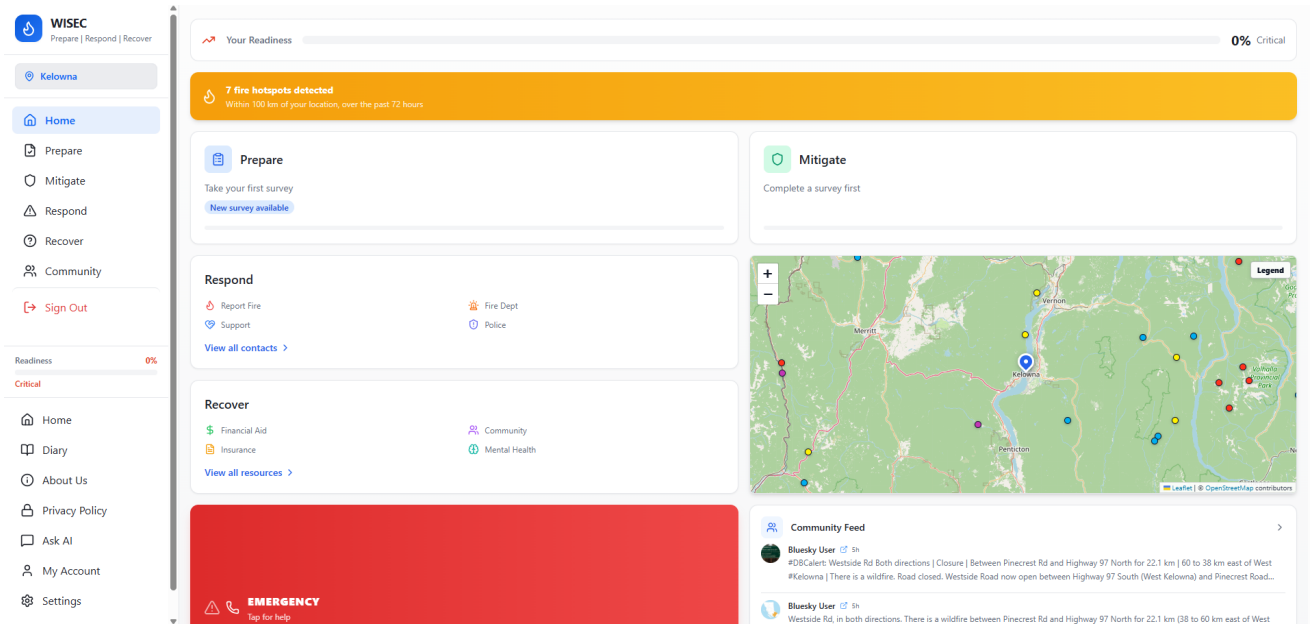
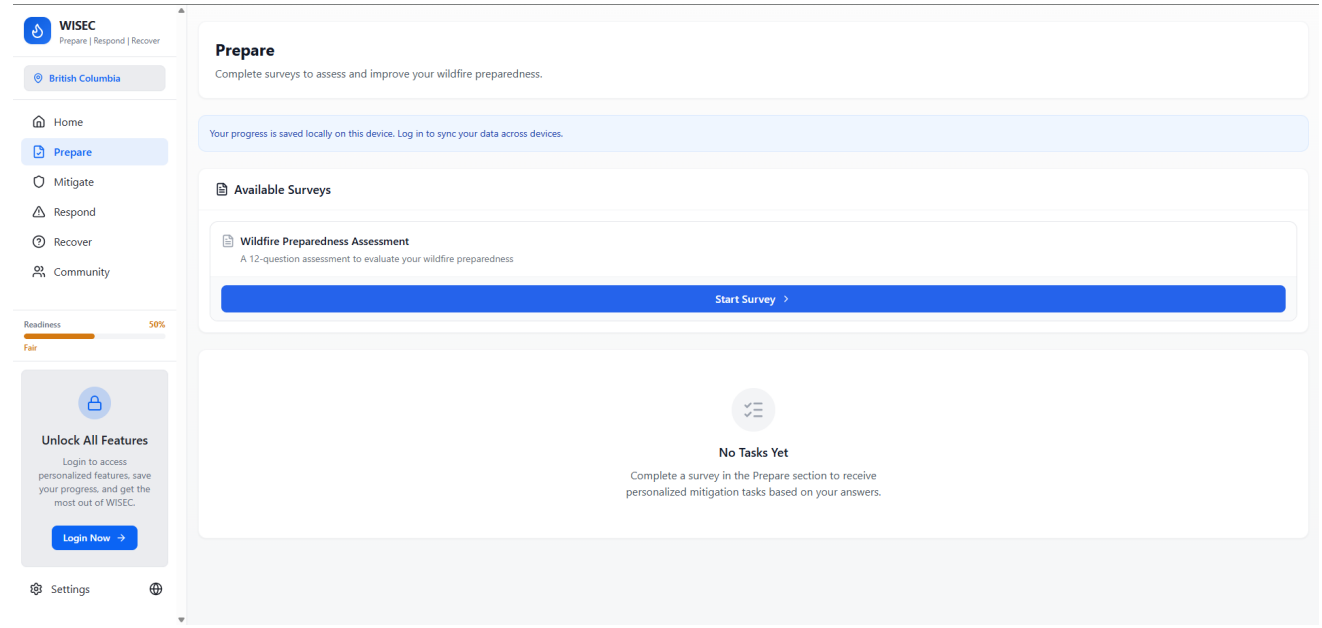
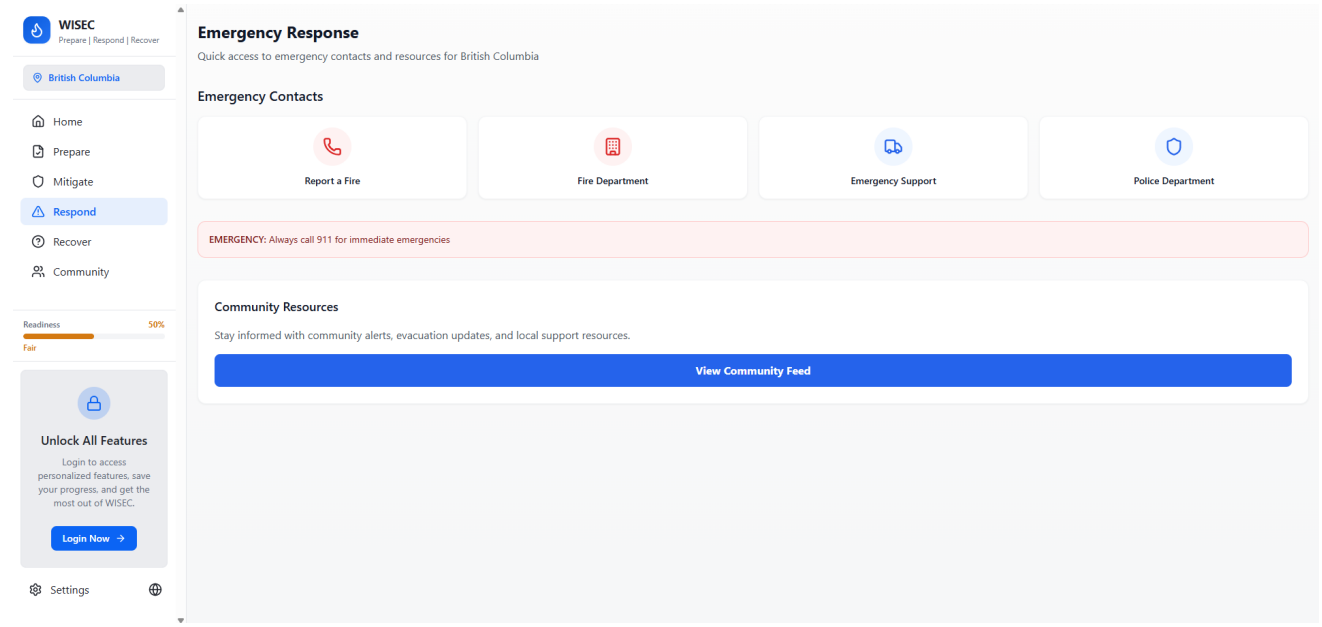


Figure 3: Authenticated Kelowna dashboard captured from the locally running Vite client with the local NestJS API and seeded PostgreSQL database. The view combines readiness, nearby-hotspot status, response and recovery shortcuts, a fire map, and community updates.

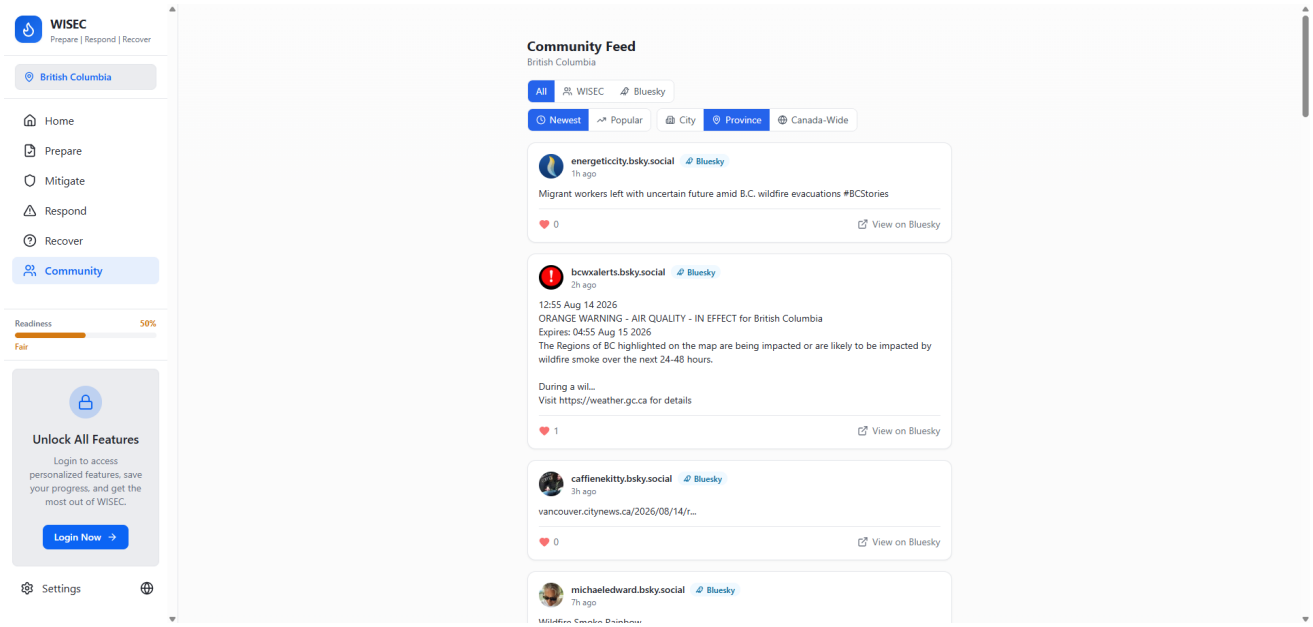


(a) Prepare survey catalog

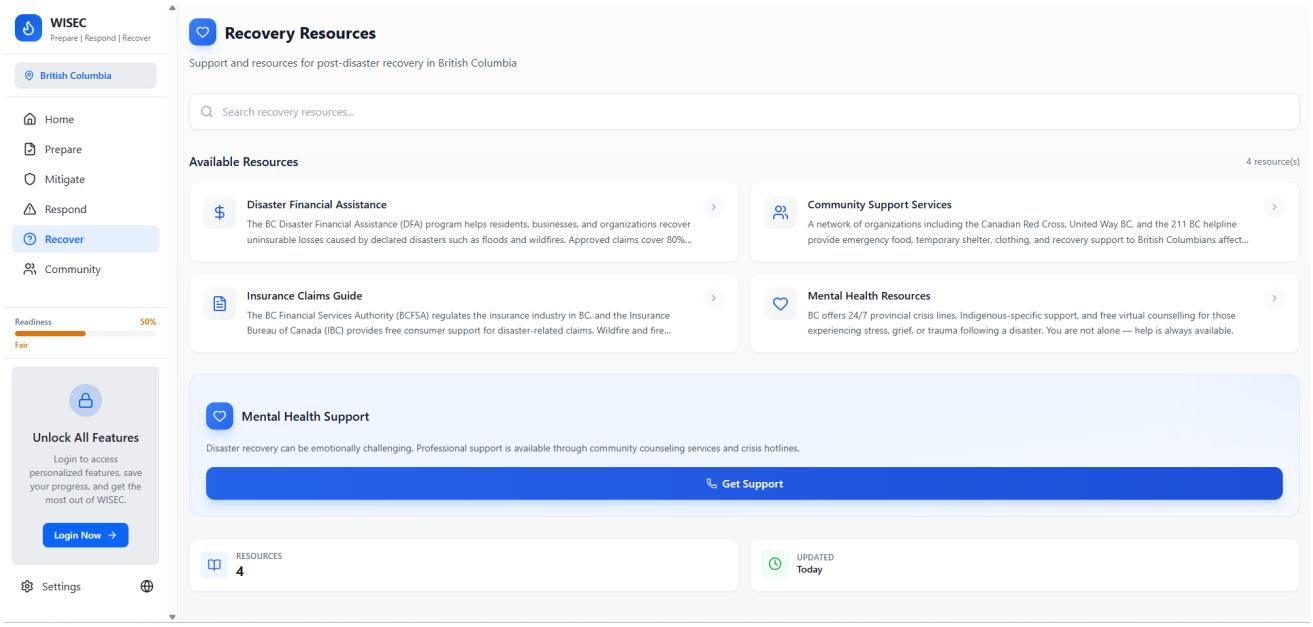


(b) Emergency response hub

Figure 4: Implemented Prepare and Respond views captured from the locally running Web stack. The larger reproduction preserves readable interface labels and controls.

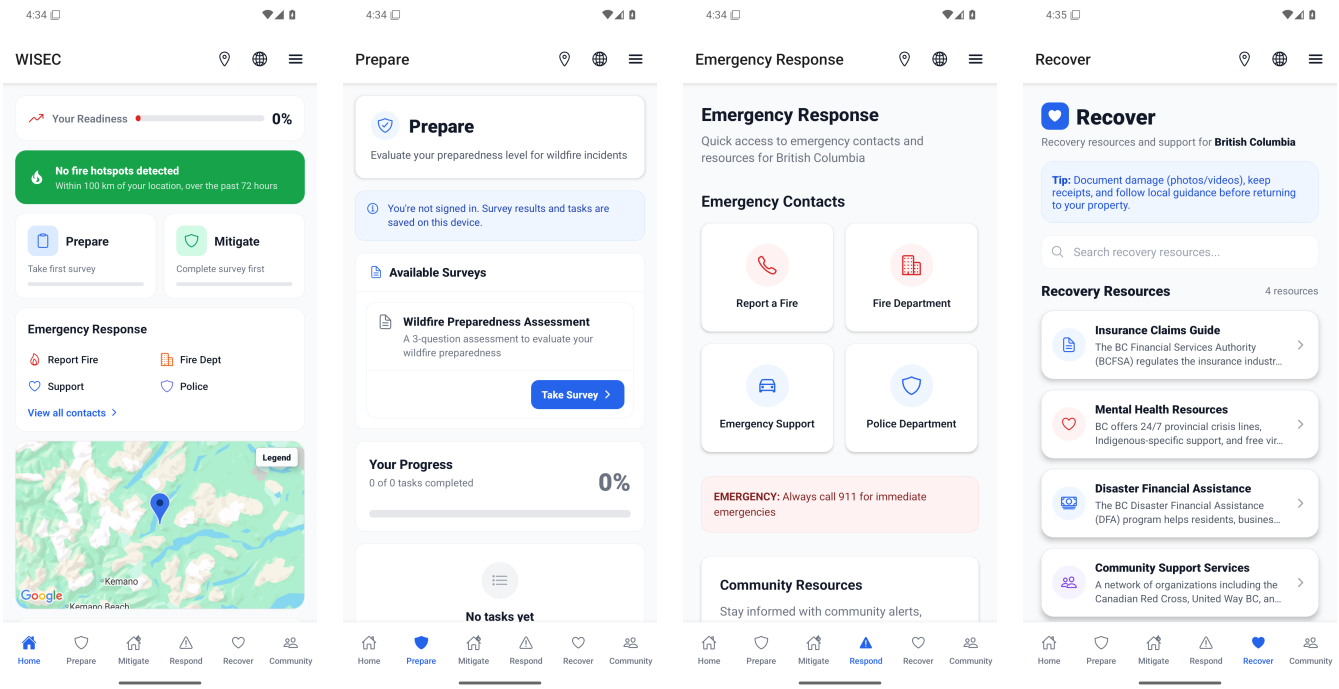


(a) Location-scoped community feed



(b) Recovery resource catalog

Figure 5: Implemented Community and Recover views captured from the same local Web stack. These views, together with Figure 4, show a guest/local-progress session scoped to British Columbia and are not presented as a continuation of the authenticated dashboard session in Figure 3.



(a) Home and readiness **(b)** Prepare assessment **(c)** Response contacts **(d)** Recovery resources

Figure 6: Implemented mobile views captured from the running Expo Go client on a Pixel 7 API 36 emulator. The session used the repository's migrated and seeded local PostgreSQL development database through the Nest API; these are runtime captures rather than design mockups.

4.2 Readiness surveys and versioned content

Surveys are not static client forms. An administrator creates a survey container and one or more JSON-based versions. The activation workflow deactivates the other versions before selecting one version as active, and new submissions store that version's identifier. Earlier submissions therefore retain a version reference, although the current administrative update endpoint means that version content is not strictly immutable. The form schema supports localized labels, multiple field types, validation, conditional behavior, scoring metadata, and task slugs. Both a full-form renderer and an onboarding-oriented stepped renderer consume the schema.

On submission, answers can complete tasks whose criteria are satisfied and assign tasks for unmet readiness actions. The client then calculates or displays a readiness score and, for authenticated users, can immediately offer scheduling modes. This closes a common gap in assessment tools: a low score is not merely reported; it becomes a set of actionable records.

Version references create a basis for longitudinal analysis, but the current update endpoint weakens that guarantee because a referenced version's content can change in place. Reliable research exports would require immutable version snapshots or an equivalent audit history before treating each response as permanently tied to the exact wording originally shown.

4.3 Tasks, diary, and recurrence

System tasks follow an explicit lifecycle: `not_started` → `in_progress` → `completed`. Invalid backward or skipped transitions are rejected. Progress records are user-specific, and custom tasks let residents record actions that are not in the managed catalog.

The Diary view turns the task list into a weekly plan. A resident can distribute newly assigned tasks across weeks, place them on the next weekend, choose dates manually, browse the task catalog, create a custom task, reschedule incomplete work, and configure weekly or monthly recurrence. Daily schedulers reset completed recurring tasks when the next occurrence is due and send reminders for upcoming work.

This workflow is behaviorally stronger than a one-time checklist because it makes preparedness visible over time. Nevertheless, the repository does not contain a behavior-change study. The appropriate conclusion is that the system

supports planning and repeated action, not that it has already increased adherence.

4.4 Current-fire information and proximity alerts

The current implementation's active aggregation service retrieves CIFFC map-layer CSV files, filters active records, applies time and distance windows, deduplicates agency/fire identifiers, and maps results into the shared hotspot representation. CIFFC is the interagency organization owned by Canadian federal, provincial, and territorial wildland-fire agencies and provides national fire information and coordination [10, 13]. Separate CWFIS and NASA EONET service adapters remain in the codebase and have automated tests; EONET itself is a curated near-real-time natural-event metadata API [14, 11].

A scheduled backend job runs every fifteen minutes. It creates a scan, persists normalized hotspots, retains a 72-hour rolling window, and then checks users who have opted in, have a verified/approved account, have coordinates, and have not been notified during the six-hour cooldown. Nearby-fire matching uses a 100 km radius and sends email in batches. The home dashboard refreshes its local view every five minutes and shows a severity-colored summary plus map markers.

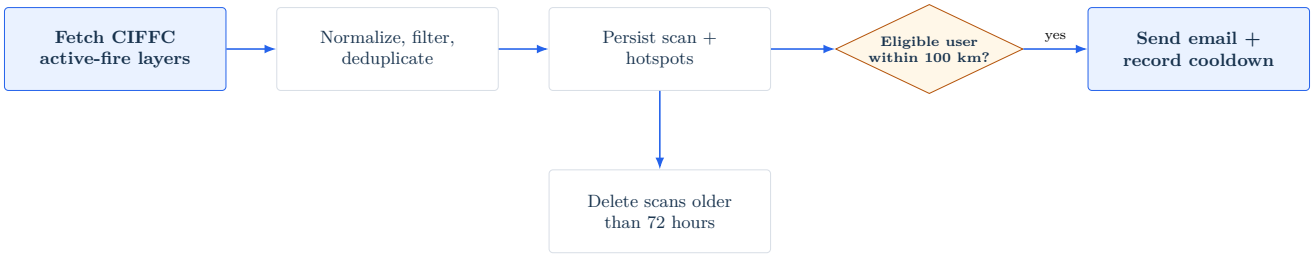


Figure 7: Implemented current-fire ingestion and alert workflow. Production operations should add distributed locking, provider-health monitoring, and explicit stale-data indicators.

The implementation has a documentation mismatch: an older architecture note still describes a GOES-18 Python ingestion path, while the current interactor calls the CIFFC-backed combined service. This does not break the compiled code, but it can mislead operators. The runbook and source-of-truth diagram should be updated before handoff.

4.5 Source-aware conversational assistance

The AI assistant is a major redesign from v1. Messages now pass through the backend, where three gates constrain behavior.

1. A wildfire classifier first performs a fast match against more than eighty domain keywords and uses a language model for ambiguous queries. Errors fail closed.
2. A search router enables the configured vector store and web search for accepted wildfire questions.
3. A system instruction requires calm, plain language, short explanations, numbered actions, source references, and escalation for medical, legal, or emergency matters.

Responses stream to the client using server-sent events. The backend persists user and assistant messages, extracts file and web citations, labels web sources against global or location-specific approved-domain records, and treats managed vector sources as trusted. After the answer completes, a separate short-running extractor identifies up to five actionable tasks; the resident may add a selected suggestion to the Diary.

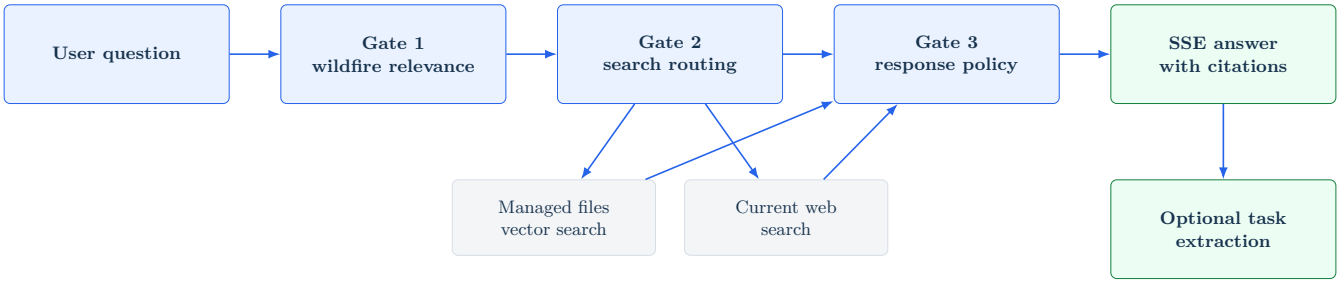


Figure 8: The chat pipeline separates topic control, retrieval choice, response format, citation enrichment, and task generation.

The whitelist is a transparency mechanism, not a guarantee that every retrieved statement is correct. The system should continue to display source links, dates where available, and a clear distinction between official instructions and general guidance. Automated evaluation should test citation completeness, unsupported claims, refusal behavior, prompt injection through retrieved content, and graceful failure when either search channel is unavailable.

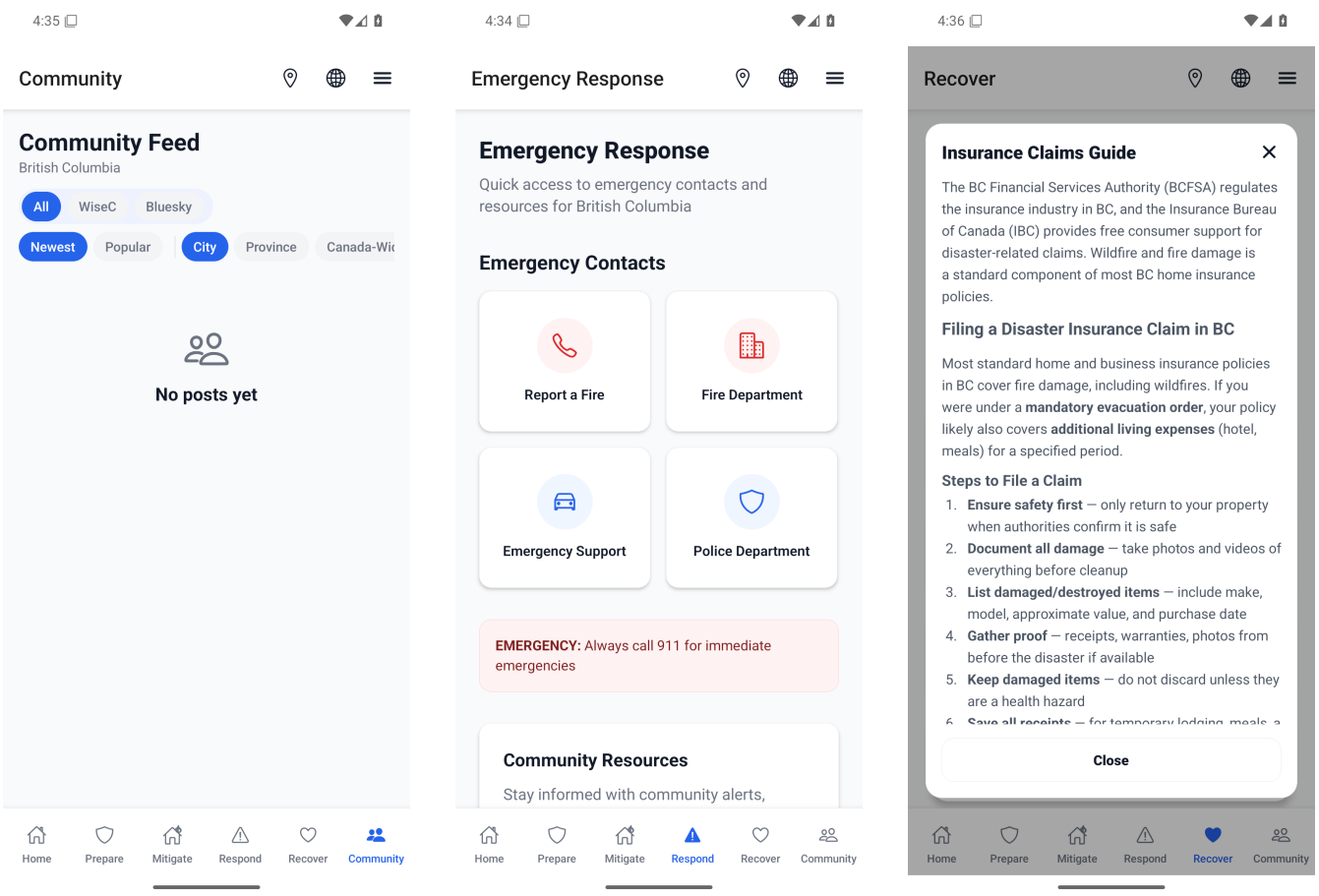
4.6 Response, recovery, and community information

Respond resources emphasize immediate actions such as fire reporting, emergency support, police, evacuation, and official websites. Recover resources cover insurance, rebuilding, financial assistance, and social support. Cards adapt to data type (phone, link, document, address, or structured details), and recovery items include a helpful/not-helpful feedback action. Because the API returns the resolved location, the UI can say when city-level content was unavailable and province-level information is being shown.

The Community module merges two content streams:

- internal WISEC posts with images, comments, deletion by the author, and up/down voting; and
- read-only Bluesky results, labeled by source and linked back to the original network.

Users can filter by source, recency/popularity, and location scope. The design deliberately removes internal interaction controls from Bluesky items. This avoids implying that a vote or comment in WISEC changes the external post. The feed still needs moderation policy, abuse reporting, retention rules, and misinformation-response procedures before broad public launch.



(a) Location-scoped community feed **(b)** Emergency response hub **(c)** Structured recovery guidance

Figure 9: Runtime examples of location-aware emergency, recovery, and community information in the Expo Go session. The empty community state reflects the seeded local dataset and is shown without fabricated posts.

4.7 Administration, consent, and bilingual delivery

The web admin surface is a notable expansion of project operability. Protected routes cover dashboard metrics, users and access approval, surveys, schema construction, system values, response/recovery resources, tasks, RAG documents, disclaimer versions, and approved domains. Content teams can update operational information without rebuilding the mobile client.

Identity uses Argon2 password hashing, signed JWTs, role-based guards, separate user/admin client token keys, and an administrator-controlled access state. The latest registration flow creates a pending account, and an administrator can grant or revoke access from the user table. Disclaimer content and consent are versioned, including anonymous-user support. English and French translation objects are shared conceptually across web and mobile, and localizable fields appear in surveys, tasks, disclaimers, and selected content.

These controls are a strong foundation, but they should be treated as the beginning of governance. The product needs documented data retention, consent withdrawal, account deletion, administrative audit logs, translation ownership, and periodic emergency-resource review. Government digital guidance emphasizes limiting personal information, explaining its use, and offering both official languages with equal quality [8]. W3C recommends WCAG 2.2 as the current shared accessibility target for web and mobile content [12]; conformance has not yet been formally evaluated for WISEC.

5 Evolution from WISEC-v1

5.1 Baseline characteristics of WISEC-v1

WISEC-v1 established the project's essential product idea. Its React Native application already organized content around preparation, mitigation, response, and recovery; included surveys, tasks, a diary, maps, a community feed, and a chatbot; and connected to an Express API backed by MongoDB. Sanity acted as the content-management system, Cloudinary stored media, Bluesky supplied external feed content, and Firebase supported mobile messaging. These choices enabled rapid prototyping and demonstrated a coherent resident journey.

The limitations were primarily structural. The mobile and API code lived as separate applications with independently maintained types. The Express backend concentrated logic in route/controller utilities without consistent use-case boundaries. Survey and task content crossed MongoDB, Sanity, GraphQL-generated types, and client transformations. The mobile client called the OpenAI endpoint with a bundled environment key, placing a server credential and safety policy at the edge. Wildfire events were fetched directly by the client from EONET using a 30-day bounding-box query, with no server-side ingestion history, proximity workflow, or provider abstraction.

These are understandable prototype trade-offs. The purpose of the current architecture is not to criticize them, but to preserve the validated user concept while reducing the cost and risk of adding a web client, administration, versioned content, AI retrieval, location-aware resources, and scheduled alerts.

5.2 Upgrade matrix

Table 4: Major changes from WISEC-v1 to the current monorepo.

Dimension	WISEC-v1	Current WISEC
Repository	Separate mobile and API projects	pnpm monorepo containing web, API, mobile, and shared contracts
Client reach	React Native mobile application	Responsive React web application plus Expo/React Native mobile application
Backend	Express routes and controller utilities	NestJS modules with commands, queries, interactors, ports, adapters, guards, and DTOs
Persistence	MongoDB/Mongoose documents	PostgreSQL relations, MikroORM mappings, and 22 schema migrations
Shared contracts	Duplicated or generated per integration	`@wisec/shared` enums and interfaces consumed across applications
Content operations	Sanity CMS plus webhook/GraphQL flows	First-party admin APIs and pages for surveys, tasks, resources, documents, disclaimers, source whitelist, and system settings
Survey versioning	Survey content stored as documents	Survey containers with sequential versions and version-referenced submissions
Task behavior	Assigned tasks and diary functions	Validated lifecycle, catalog, custom tasks, batch scheduling, reminders, and weekly/monthly recurrence
AI placement	Direct mobile call to OpenAI Chat Completions	Server-side streaming orchestration with classification, file/web retrieval, citations, whitelist labels, and task extraction
Fire data	Direct client EONET query	Server-side CIFFC aggregation, persisted scans, alternate provider adapters, map queries, and opt-in email proximity alerts
Location	Client/user location for map and content	Shared country–province–city hierarchy for resources, community fallback, chat source policy, and fire proximity

Dimension	WISEC-v1	Current WISEC
Community	Internal/Bluesky feed in mobile	Cross-client internal feed, comments/votes, source filters, location scopes, and read-only Bluesky representation
Identity	JWT plus token blacklist/refresh patterns	Argon2, JWT guards, RBAC, account approval, email/consent state, and admin access controls
Language	Predominantly English	English/French translation systems and localizable domain content
API documentation	Manual route understanding	Swagger/OpenAPI generation from controllers and DTO metadata
Deployment	Conventional Node/mobile setup	Live GCP web/API deployment at `wisec.ca` with Neon-hosted PostgreSQL, secret-managed connection URL, Cloud Build, startup migrations/seeders, and health checks
Automated validation	One mobile smoke test; no API tests found	32 passing API suites (194 tests), mobile type-check, Expo Go runtime evidence, and a successful web production build

5.3 Security and trust improvements

The most consequential improvement is the movement of privileged integrations to the backend. In v1, the mobile OpenAI client read `OPENAI_API_KEY` and sent requests directly to the public endpoint. A distributed binary cannot keep such a value secret. The current system keeps the OpenAI client behind the NestJS API and adds user authentication, topic classification, retrieval policy, citation enrichment, response constraints, and server-side task generation.

Trust is also represented in the data model. Approved domains can be global or location-specific; vector-store documents are managed by administrators; web citations retain source metadata; external Bluesky content is visibly distinct; consent is versioned; and fire alerts are opt-in with cooldown records. These mechanisms do not eliminate misinformation or misuse, but they make review and remediation possible.

Authentication has also matured through Argon2 hashing, role guards, protected admin routes, and an approval workflow. Remaining concerns include browser token storage in `localStorage`, a 24-hour default JWT lifetime (overridden to one hour for administrator tokens), limited evidence of rate limiting, and no reviewed administrative audit log. The architecture is safer than v1 but should not be described as security-complete.

5.4 Maintainability and delivery improvements

The move from MongoDB documents and Sanity-managed structures to explicit relational models is well aligned with the platform's need for explicit relationships: a survey submission refers to an identified survey-version record; a task progress record belongs to one user profile and one catalog task; a chat message belongs to one thread; and a notification records which user and fire scan triggered it. Migrations make these changes inspectable across environments. Survey-version immutability would require additional enforcement.

The backend's ports make external dependencies replaceable. Current code includes separate service clients for CIFFC, CWFIS, and EONET, even though the active combined service is CIFFC-backed. Email delivery supports provider abstraction. OpenAI client factories isolate platform APIs from use cases. This separation is especially useful for testing failure paths without live services.

The new `brain` directory is another meaningful addition: data, business, and design knowledge are cross-referenced for handoff. However, documentation is not automatically synchronized with code. Examples found during this review include an outdated 15-table count, an obsolete GOES-18 fire-ingestion narrative, and older admin route/status descriptions. The lesson is to keep the knowledge base, but generate inventories and endpoint references where

possible.

Key finding

WISEC's evolution is best characterized as a platform reconstruction, not a feature port. The resident-facing concept remains recognizable, while the implementation adds web reach, administrative control, shared contracts, explicit version references, safer AI boundaries, and reproducible deployment.

5.5 Trade-offs introduced by the reconstruction

The new system is larger and more capable, but also more complex. It now depends on PostgreSQL, OpenAI services, email delivery, multiple fire/social data sources, scheduled jobs, two client applications, and numerous admin workflows. Guest and authenticated state paths can diverge. Web and mobile implementations can still drift despite shared types because view logic and translations are duplicated. Cloud Run's elastic execution model does not automatically coordinate in-memory cron mutexes across instances. A broad admin surface increases the need for auditability and least-privilege roles.

In other words, the architecture replaces prototype fragility with operational obligations. That is the correct direction for the project, provided the next phase invests in integration tests, observability, performance, security, and content governance rather than continuing to add independent features.

6 Verification and Evaluation

6.1 Build and automated-test results

The local production build is the strongest whole-repository result available in this review. It compiled the shared package in CommonJS and ESM forms, built the NestJS API, type-checked the web client, transformed 3,964 Vite modules, and produced the deployable web bundle. The mobile project independently passed a no-output TypeScript compilation.

Verification evidence

Shared/API/Web build	**Verified**	Exit code 0; all three deliverables compiled
API automated tests	**Verified**	32 suites and 194 tests passed; no snapshots
Mobile type safety	**Verified**	`tsc -noEmit` completed with exit code 0
Web local runtime	**Verified**	Representative desktop views rendered through the local Vite, NestJS, and PostgreSQL stack
Mobile runtime	**Verified**	Expo Go rendered representative flows on a Pixel 7 API 36 emulator against the local API and seeded PostgreSQL
End-to-end operation	Not tested	Public API/database reachability was checked, but no authenticated full journey, third-party workflow, or production mobile binary was exercised

API tests cover more than happy-path compilation. The suite includes invalid task transitions, survey creation/update/submission behavior, user registration, password hashing, IAM controllers, news CRUD interactors, community-feed combination, chat routing and streaming orchestration, approved-domain matching, OpenAI client/factory behavior, email templates, and parsers/filters for CIFFC, CWFIS, and EONET. The classifier test also verifies its fail-closed response to an upstream timeout.

The test evidence is scoped rather than repository-wide. The root `pnpm test` command exercised the API suite. This report therefore makes no automated-test result claim for the web or mobile packages; their verified evidence is limited to the production build, local desktop runtime captures, mobile type-check, and Expo Go runtime session.

6.2 Functional completeness by capability

Table 5 assesses each major capability against the inspected evidence.

Table 5: Capability-level evidence and remaining validation.

Capability	Status	Evidence and interpretation
Web resident experience	Implemented	Production build passes; representative local desktop views show the dashboard, survey catalog, response hub, recovery resources, and community feed. A complete authenticated journey and cross-browser run were not exercised.
Mobile resident experience	Implemented	Expo screens and native navigation cover the primary phases; TypeScript passes, and representative views ran in Expo Go on a Pixel 7 API 36 emulator. A production-signed APK/AAB and push delivery were not exercised.
Identity and approval	Tested in part	Registration, hashing, JWT-related controllers, and approval UI/API are present; unit tests cover key paths. No session, revocation, or abuse test was run.
Survey and task workflow	Strong API evidence	Versioning/submission interactors and task lifecycle have multiple passing suites. End-to-end client synchronization remains untested.
Diary and recurrence	Implemented	Scheduling, custom tasks, reminders, and recurrence code/migrations are present. Scheduler behavior was not observed over real time.
Fire data and alerts	Strong adapter evidence	Three provider adapters have tests; active CIFFC path, persistence, 15-minute cron, distance filtering, and email workflow are present. No live alert was sent.
AI and RAG	Strong unit evidence	Classifier, router, send-message interactor, whitelist matcher, and OpenAI clients are tested. No adversarial or factuality evaluation was run.
Community and Bluesky	Tested in part	Feed-merging interactor is tested; internal CRUD and Bluesky adapter/UI are present. Moderation and live API behavior remain unverified.
Admin/content operations	Implemented	Protected routes and pages exist for users, surveys, schemas, tasks, resources, documents, disclaimers, settings, and whitelist. Role matrix and audit trail need system testing.
Deployment	Live; checked in part	Web/API run on GCP and use Neon-hosted PostgreSQL. Public web, readiness, and database-backed location requests returned HTTP 200. GCP/Neon Console health, uptime history, backups, and recovery were not verified.

6.3 Usability and accessibility assessment

The interface design has several positive characteristics for stressful contexts: phase labels are concrete verbs; emergency phone numbers are visibly separated from informational links; the location bar explains context; cards use concise calls to action; source badges differentiate internal and external content; and the onboarding survey can auto-advance through a small number of questions. Desktop and mobile layouts have explicit responsive designs rather than relying on automatic shrinking.

English/French selection is available in both clients, and localized strings are type-checked against the English translation structure. Form schemas support localized labels, while disclaimers and tasks include bilingual fields. These choices align with Canadian digital-service principles, but completeness cannot be inferred from the existence of translation files. Inline conditionals and duplicated web/mobile dictionaries create opportunities for untranslated or inconsistent text.

Accessibility intent appears in semantic buttons, keyboard handling for stepped forms, source labels for screen readers, alt-text support, Radix UI primitives, and contrast-aware status colors. Formal conformance is still unproven.

WCAG 2.2 evaluation should include keyboard-only navigation, focus order, target size, map alternatives, non-color indicators, zoom/reflow, motion reduction, screen-reader announcements for streaming chat and alerts, and equivalent access to emergency contacts [12].

6.4 Safety, security, privacy, and reliability

6.4.1 Safety

The application has useful safety controls: a wildfire-only classifier that fails closed, concise response instructions, managed knowledge documents, approved-source labeling, versioned disclaimers, opt-in alerts, verified/approved account checks, and explicit emergency-contact content. The strongest remaining risk is authority ambiguity. A map marker or AI citation may be current and credible without being an official order. UI copy should show data timestamps and agency names, explain stale or unavailable data, and keep local authority links one action away.

6.4.2 Security

Server-side secrets, Argon2 hashing, DTO validation, JWT and role guards, production CORS configuration, and secret-manager deployment guidance are clear improvements. However, the reviewed code provides limited evidence for request throttling, account lockout, refresh-token rotation, security headers, content-security policy, malware scanning for uploaded documents/images, or audit logging for administrative changes. Browser tokens stored in `localStorage` increase the consequence of cross-site scripting. These items warrant a threat model and dedicated test plan before public deployment.

6.4.3 Privacy and governance

The system may store email addresses, names, selected location, readiness scores, survey answers, task history, chat content, community contributions, consent records, and alert history. Each field can be legitimate for a feature, but the aggregate profile is sensitive. The project should define purpose, retention, export/deletion, staff access, breach response, and research-use rules. Anonymous onboarding identifiers and later account linking need explicit merge semantics so that consent and locally stored history do not become ambiguous.

6.4.4 Reliability

External dependence is unavoidable but must be visible. CIFFC files, Bluesky, OpenAI, vector stores, email providers, maps, GCP services, and Neon can fail independently. The current code contains several graceful-degradation behaviors, such as fail-closed classification and async task extraction, but there is no repository-level evidence of circuit breakers, provider health dashboards, dead-letter handling, or service-level objectives. The cron job uses an in-memory mutex, which prevents overlap inside one process but not across multiple Cloud Run instances. A PostgreSQL advisory lock or separate scheduled worker would provide safer coordination.

6.5 Performance and scalability

The web build's largest warnings identify a concrete optimization opportunity. The Admin System Values page chunk is roughly 1.53 MB minified, the shared index chunk roughly 572 kB, and the research-tools chunk roughly 474 kB. Admin routes are lazy-loaded, which protects most residents from the largest admin bundle, but route-specific manual chunks and dependency review would reduce download and parse cost further. Map and chart libraries should load only where used.

The backend architecture can scale horizontally for stateless HTTP requests, while Neon-hosted PostgreSQL provides transactional coordination. Scheduled work, file uploads on container-local storage, persistent server-sent-event connections, and database connection limits across autoscaled Cloud Run instances require more careful deployment design. The current static-assets path points at a local `uploads` directory; ephemeral Cloud Run files should be replaced by durable object storage for production community media. Load tests should measure feed queries, chat concurrency, hotspot map payloads, and admin bulk operations.

6.6 Documentation quality

The structured `brain` knowledge base is valuable and covers the main data, business, and design layers. It provides more handoff context than code comments alone. Yet several pages describe earlier states of the implementation. The current fire interactor uses the CIFFC combined service, while the cron document describes a GOES-18 Python script. Cloud SQL and Cloud Run PostgreSQL-sidecar files remain even though the production database path has moved to Neon. The data overview reports 15 tables, while later migrations and modules add many mappings. The admin documentation uses older route names and completion counts.

Documentation drift is an operational risk when it affects data provenance, alert timing, or deployment. The project should generate endpoint and schema references from source, assign an owner to each manual runbook, add "verified against commit" metadata, and fail CI when internal links or generated inventories are stale.

Risk and interpretation

Interpretation of readiness. The platform is technically credible for a controlled pilot, but the evidence does not yet support unattended public-safety operation. A pilot should include partner review, monitored infrastructure, limited geography, explicit fallback to official channels, and rapid content correction.

7 Conclusions and Recommended Next Steps

7.1 Overall conclusion

WISEC has progressed beyond its first-generation mobile prototype into a coherent cross-platform system. The current monorepo integrates a responsive web application, Expo mobile client, modular NestJS API, a relational data layer backed in production by Neon-hosted PostgreSQL, shared contracts, administrative content workflows, source-aware AI, current-fire information, and reproducible GCP deployment assets. The architecture preserves the original vision—help residents prepare for, respond to, and recover from wildfire—while making the system more maintainable and safer to extend.

The local evidence is meaningful. The shared, API, and web production build passed; all 194 API tests passed; and the mobile client type-checked. The repository also contains strong implementation detail in domains that are often left superficial in prototypes: survey version references, task state transitions, recurrence, location fallback, consent versions, citation trust, fire-alert cooldown, and administrator-managed knowledge.

At the same time, the report identifies a clear boundary between implementation and demonstrated impact. The GCP deployment is publicly reachable, but its operational controls were not independently verified. The invoked automated suite covers the API rather than all clients; user and emergency-partner studies are absent; and the system has unresolved security, reliability, accessibility, content-governance, and performance work. These limitations do not negate the project result; they define the correct next phase.

7.2 Prioritized roadmap

Table 6: Recommended next steps, ordered by launch risk and learning value.

Priority	Workstream	Concrete outcome	Exit evidence
P0	Test baseline	Establish web/mobile package test commands and create one authenticated end-to-end journey from onboarding through survey, task scheduling, resource lookup, and chat.	CI runs all packages; smoke suite passes on every change
P0	Safety and data provenance	Update the fire runbook to the CIFFC path, show source/timestamp/staleness in the UI, define official-alert language, and test provider failure.	Partner-approved copy and failure drill
P0	Security hardening	Add rate limits, security headers/CSP, upload validation, token/session review, least-privilege admin roles, and audit logging.	Threat model plus independent security test
P0	Production data handling	Move uploads to object storage; document backups, restores, retention, account deletion, consent withdrawal, and secret rotation.	Successful restore and deletion exercises
P1	Reliability	Replace process-local cron coordination with a distributed lock or dedicated worker; add structured logs, metrics, tracing, provider health, and alerts.	Operational dashboard and runbook exercise
P1	Accessibility and bilingual QA	Audit against WCAG 2.2; conduct keyboard/screen-reader tests; review English/French parity with domain experts.	Issue log closed to an agreed conformance target
P1	Performance	Split the two bundles above 500 kB, defer map/admin dependencies, and test on low-bandwidth mobile connections.	Agreed performance budgets met
P1	State continuity	Specify and test guest-to-account migration, offline behavior, conflict resolution, and cross-device synchronization.	Deterministic migration and sync tests
P2	Pilot evaluation	Run a limited geographic pilot with residents, municipal/First Nations partners, and emergency professionals; measure comprehension, task completion, trust, and resource findability.	Pre-registered study and analyzed results
P2	Product scope	Decide whether to re-enable gamification based on evidence; formalize moderation and misinformation processes for community content.	Governance decision with owners and metrics
P2	Resilience outcomes	Connect readiness measures to validated indicators and longitudinal behavior without over-collecting personal data.	Ethics/privacy review and outcome model

7.3 Recommended pilot posture

A responsible first pilot would be geographically limited, invite-only or administrator-approved, and operated with named content owners. It would use official links as the final authority, display data freshness, monitor every scheduled job and third-party integration, and provide a fast method to correct resources. Participants would receive a plain-language explanation of what WISEC can and cannot do, what data is stored, and how to leave the pilot.

Evaluation should focus on tasks users actually need to complete:

- Can a new user select a location and understand the readiness result?
- Can the user find and schedule one appropriate mitigation action?
- Can the user identify the official emergency contact for the selected city?
- Can the user distinguish an official/approved source, an unvetted web source, and a Bluesky post?
- Can the user understand whether a map is current, stale, or unavailable?

- Can users complete the same essential journey in English and French, with keyboard or assistive technology where required?

These measures would produce more useful evidence than adding another dashboard feature. They would also respect the whole-of-society emphasis of Canadian resilience policy, which treats communication and community capability as central to preparedness [1, 15].

Key finding

Final recommendation. Freeze broad feature expansion for one milestone. Use that milestone to make the existing cross-platform journey observable, testable, accessible, secure, and partner-reviewed. WISEC already has enough product breadth for a meaningful pilot; its next competitive advantage should be trustworthiness.

References

[1] Public Safety Canada. Emergency management strategy for canada: Toward a resilient 2030. https://www.publicsafety.gc.ca/cnt/rsrcs/pblctns/mrgncy-mngmnt-strtgy/, 2019. Accessed August 12, 2026.

[2] Public Safety Canada. Federal policy for emergency management. https://www.publicsafety.gc.ca/cnt/rsrcs/pblctns/plc-mrgnc-mngmnt/index-en.aspx, 2009. Accessed August 12, 2026.

[3] Natural Resources Canada. Forest fires. https://natural-resources.canada.ca/forests-forestry/wildland-fires/forest-fires, 2026. Accessed August 12, 2026.

[4] WISEC Project Team. Wisec public web application and api. https://wisec.ca/ and https://api.wisec.ca/health/readiness, 2026. Public deployment accessed August 14, 2026; web, readiness, and database-backed location requests returned HTTP 200 through Google Frontend.

[5] FireSmart Canada. About firesmart. https://firesmartcanada.ca/about-firesmart-2/, 2026. Accessed August 12, 2026.

[6] FireSmart Canada. The seven firesmart disciplines. https://firesmartcanada.ca/about-firesmart-2/the-seven-firesmart-disciplines/, 2026. Accessed August 12, 2026.

[7] WISEC Project Team. Wisec project poster. Local project poster supplied for this report; identifies Solution Scholars Emon Sarker and Jason Xu, supervised by Dr. Adeniyi Asiyanbi and Dr. Ifeoma Adaji, with sponsorship from the Solutions Scholar program of the UBC Climate Solutions Research Collective.

[8] Treasury Board of Canada Secretariat. Guideline on service and digital. https://www.canada.ca/en/government/system/digital-government/guideline-service-digital.html, 2026. Accessed August 12, 2026.

[9] Government of Canada. Digital standards playbook. https://www.canada.ca/en/government/system/digital-government/government-canada-digital-standards.html, 2026. Accessed August 12, 2026.

[10] Canadian Interagency Forest Fire Centre. Canadian interagency forest fire centre. https://ciffc.ca/, 2026. Accessed August 12, 2026.

[11] National Aeronautics and Space Administration. Eonet version 3 api documentation. https://eonet.gsfc.nasa.gov/docs/v3, 2026. Accessed August 12, 2026.

[12] World Wide Web Consortium. Web content accessibility guidelines (wcag) 2 overview. https://www.w3.org/WAI/standards-guidelines/wcag/, 2026. Accessed August 12, 2026.

[13] Canadian Interagency Forest Fire Centre. Canada reports. https://ciffc.ca/publications/canada-reports/, 2026. Accessed August 12, 2026.

[14] National Aeronautics and Space Administration. Earth observatory natural event tracker. https://eonet.gsfc.nasa.gov/, 2026. Accessed August 12, 2026.

[15] Public Safety Canada. Advancing the federal-provincial-territorial emergency management strategy: Areas for action. https://www.publicsafety.gc.ca/cnt/rsrcs/pblctns/2024-ems-ctn-rs/index-en.aspx, 2024. Accessed August 12, 2026.

A Technical Inventory

A.1 Workspace and technology summary

Workspace	Primary technology	Responsibilities observed in the repository
`@wisec/web`	React 19, TypeScript, Vite 7, Tailwind, Radix/Shadcn, React Query, React Hook Form, Zod, Leaflet	Resident journey, responsive maps, onboarding, surveys, readiness, tasks, diary, resources, community, streaming chat, authentication, and full admin panel
`@wisec/mobile`	Expo 54, React Native 0.81, React Navigation, React Query, React Hook Form, Zod, native maps/notifications	Native/tabbed resident experience, location, language, readiness, tasks, resources, community, chat, diary, profile, settings, and local notifications
`@wisec/api`	NestJS 11, MikroORM 6, Neon-hosted PostgreSQL in production, Passport/JWT, OpenAI SDK, mail providers, Swagger	Domain use cases, persistence, authentication/RBAC, content management, RAG/chat, fire ingestion/alerts, community integration, schedulers, email, migrations, and health
`@wisec/shared`	TypeScript, tsup	Shared roles, task/survey/resource/location/document enums and cross-application interface contracts

A.2 Backend modules

Module	Primary responsibility	Representative controls
IAM	Registration, login, profiles, approval, roles	Argon2, JWT, guards, email/access status
Survey	Survey containers, versions, activation, submissions	Activation workflow; version-referenced submissions
Tasks	Catalog, progress, custom tasks, diary, recurrence	Transition rules, scheduling validation, cron reset
Location	Hierarchy and response/recovery resources	City–province–country fallback
Community	Internal posts/comments/votes plus Bluesky feed	Ownership checks, source labels, location scopes
Chat	Threads, messages, SSE, citations, task suggestions	Relevance gate, retrieval router, source whitelist
RAG	Knowledge bases and document ingestion	Admin-managed vector-store content
Fire detection	Fire records, scan history, nearby queries, alerts	Provider normalization, time/radius filter, cooldown
Notification	Verification, reminder, and proximity email	Template/provider abstraction
Disclaimer	Bilingual versions and consent records	Active version and user/anonymous consent
Admin/system	Users, settings, content operations, health	Role guard, Swagger, health endpoints
Gamification	Achievement entities and controller code	Module currently disabled in `AppModule`

B Verification Record

B.1 Executed commands

Reproduction from the monorepo root

```
pnpm build
pnpm test
pnpm -filter @wisec/mobile exec tsc -noEmit
```

All listed commands passed on August 14, 2026. The root test command exercised 32 API suites and 194 tests. No automated-test result is claimed here for the client packages. Separately, the repository's PostgreSQL container, Nest API at `localhost:3000`, and Vite Web client at `localhost:3012` were started for the local browser captures.

B.2 Measured repository indicators

Table 8: Selected indicators used in the report.

Indicator	Value	Caution
Current source files in reviewed areas	1,066	Excludes tests outside `src`, assets, infrastructure, docs, and dependencies
Current source lines in reviewed areas	73,744	Physical lines; not a maintainability score
WISEC-v1 source files in reviewed areas	266	API and mobile `src` only
WISEC-v1 source lines in reviewed areas	21,376	Physical lines; architecture differs
Current HTTP method decorators	96	Static source count; may include admin/public variations
V1 Express route bindings	29	Static route-binding count, not one-to-one with decorators
Current migrations	22	Dated migration source files
Current concrete entity mappings	29	Includes mappings from the disabled gamification module
Current API test suites / tests	32 / 194	All passed locally
Current mobile test files	0	TypeScript check passed, but this is not behavioral coverage

C Evidence-to-Claim Traceability

Report claim	Repository evidence	Validation level
Monorepo supports web, API, mobile, and shared contracts	Root/workspace package manifests and application directories	Build verified for shared/API/web; Web runtime captured; mobile type-check verified
Backend uses ports-and-adapters/CQRS patterns	Module application, infrastructure, presentation, port, command/query, and interactor directories	Static inspection plus unit tests
Surveys use numbered version records referenced by submissions	Survey/version/submission entities, migrations, activation and submission interactors	Static inspection plus interactor tests; content immutability is not enforced
Tasks support scheduling and recurrence	Progress/custom-task entities, lifecycle methods, controllers, cron services, migrations, diary clients	Static inspection plus core task tests

Report claim	Repository evidence	Validation level
Fire layer uses CIFFC in the active path	`CombinedWildfireService` and `IngestFireScanInteractor`	Static inspection plus provider unit tests
AI uses relevance, retrieval, and response gates	Classifier, search router, send-message interactor, SSE controller, whitelist matcher, task extractor	Static inspection plus targeted unit tests
Sources can be labeled by trust context	Approved-domain entity/admin API, location-aware matcher, source DTO/UI badges	Static inspection plus whitelist tests
Administration covers core content	Web admin routes/pages and matching controllers	Production build; no end-to-end role test
Web/mobile support English and French	Mirrored locale contexts and translation dictionaries; localizable content fields	Static inspection; no linguistic QA
GCP deployment with Neon PostgreSQL is publicly reachable	`wisec.ca`, API readiness/location endpoints, `backend-neon.yaml`, Neon-aware database connection, Dockerfiles, Cloud Build and startup documentation	Public HTTP checks plus configuration inspection; GCP/Neon console operations not observed