Maintainer Guide

Extensions and Maintenance

EscapeCircuit

A logic-circuit puzzle web application

Group Members:
Dor Steinlauf, Noam Yosef, Mendy Dishon, and Yuval Zarmi

Advisor: Niv Gilboa

Clients: Gera Weiss, Oded Margalit

Document version 1.0

Table of Contents

Table of Contents **2**
1. Introduction **5**
1.1 Project Overview 5
1.2 Purpose of this Guide 5
1.3 Intended Audience 5
2. System Overview **6**
2.1 Main Features 6
2.2 User Roles 6
2.3 How the Application Works (High Level) 6
3. Architecture Overview **7**
3.1 Layered Backend 7
3.2 Frontend Composition 7
3.3 Key Design Decisions 7
3.4 Textual Architecture Diagram 7
4. Repository Structure **9**
4.1 Top-Level Layout 9
4.2 Backend Folder (src/Backend/) 9
4.3 Frontend Folder (apps/nextjs-app/src/) 10
4.4 Files Maintainers Most Often Edit 10
5. Local Development Setup **11**
5.1 Prerequisites 11
5.2 First-Time Installation 11
5.3 Environment Variables 11
5.4 Running the App 11
5.5 Running Tests 12
5.6 Common Setup Issues 12
6. Maintenance Guide **13**
6.1 Regular Maintenance 13
6.2 Updating Dependencies Safely 13
6.3 Logs and Errors 13
6.4 Backups and Data Protection 13
6.5 Verifying the System After Changes 13
6.6 Maintenance Checklist 14
7. Extension Guide **15**
7.1 Adding a New Frontend Page 15
7.2 Adding a New API Endpoint 15
7.3 Adding or Modifying a Domain Model 15

7.4 Adding a New Puzzle (Riddle)....16
7.5 Adding a New Gate Type....16
7.6 Adding or Modifying Validations....17
7.7 Adding Tests....17
7.8 Updating UI Behavior....17
8. Testing and Quality Assurance....18
8.1 Test Layout....18
8.2 Test Types....18
8.3 Running Everything....18
8.4 Adding Tests for New Features....18
8.5 Coverage Expectations....19
8.6 Manual Testing Checklist for Lecturers/Evaluators....19
9. Configuration and Constants....20
9.1 Where to Look....20
9.2 Changing Constants Safely....20
9.3 What Must Not Be Hardcoded....20
10. Data and Persistence....21
10.1 Storage Model....21
10.2 Key Tables (Conceptual)....21
10.3 Backups....21
10.4 Schema Changes....21
11. Security and Access Control....23
11.1 Authentication....23
11.2 Authorization....23
11.3 Common Risks for Maintainers to Avoid....23
12. Deployment and Production Notes....24
12.1 Production Topology....24
12.2 Refreshing the Server (Operator Procedure)....24
Backend-only changes (Python, schema, business logic)....24
Frontend changes (Next.js, UI, components)....24
Changes in both....24
12.3 deploy.sh (committed, backend-only)....24
12.4 Local vs Production....25
12.5 Deployment Checklist....25
12.6 Rollback....25
13. Common Maintenance Scenarios....27
13.1 Fixing a Bug....27
13.2 Adding a New Feature....27
13.3 Updating Dependencies....27

13.4 Changing a UI Screen..... 27
13.5 Adding a New Test Case to an Existing Puzzle..... 27
13.6 Debugging Failed Tests..... 27
13.7 Modifying Puzzle Logic..... 28
14. Known Limitations and Future Improvements..... 29
14.1 Current Limitations..... 29
14.2 Recommendations (Not Yet Implemented)..... 29
15. Appendix..... 30
15.1 Glossary..... 30
15.2 Useful Commands..... 30
15.3 Useful File Paths..... 31
15.4 Troubleshooting Table..... 31
16. Assumptions and Unverified Items..... 33

1. Introduction

1.1 Project Overview

EscapeCircuit ("Wire your way out.") is a full-stack web application for creating and solving logic-circuit puzzles. Students wire logic gates (AND, OR, NOT, NAND, NOR, XOR, XNOR, D flip-flops, and higher-level components) to solve riddles such as binary adders, comparators, palindrome detectors, and synchronous counters. The application validates each solution by simulating the circuit against creator-defined test cases.

The system is delivered as a monorepo: a Next.js 14 frontend in apps/nextjs-app and a layered FastAPI backend in src/Backend backed by a SQLite database (WAL mode).

1.2 Purpose of this Guide

This guide is intended to give future maintainers everything they need to keep the system running, fix issues, and extend it. It documents the architecture, repository layout, local setup, day-to-day maintenance tasks, and concrete recipes for adding new features (pages, endpoints, domain entities, puzzles, validations and tests). All content is derived from the actual code in this repository.

1.3 Intended Audience

- Future student developers who continue the project after the original team.
- Lecturers and evaluators who need to verify and run the system.
- Anyone deploying or operating the production server.

Some familiarity with TypeScript, React, Python and SQL is assumed. The guide deliberately avoids generic advice and points at concrete files, classes, scripts and commands.

2. System Overview

2.1 Main Features

- Circuit designer with drag-and-drop gates, wire routing, real-time simulation, cost tracking, and undo/redo.
- Puzzle system: creators define inputs, outputs, budget, time limit, difficulty, allowed gate set, and test cases.
- Test case validation: black-box (input/output), white-box (gate-count or gate-type limits), sequential streams, and latency constraints.
- Arsenal: solvers can save custom circuits as reusable components, gated by XP level.
- Community: forum discussions per category and per puzzle, replies, reactions, accepted-solutions, ratings, leaderboards.
- Admin tools: user management, role assignment, moderation, audit log, content reports.
- Authentication: username/password and Google OAuth login.

2.2 User Roles

Roles are defined in the UserRole enum (src/Backend/DomainLayer/Enums.py):

Role	Capabilities
SOLVER	Default role. Can solve puzzles, rate them, post in discussions.
PENDING_CREATOR	User requested creator status; can create drafts but not publish.
CREATOR	Solver + can create and publish puzzles up to a level-based quota.
ADMIN	Full access: moderation, user/puzzle management, audit logs, role changes.

2.3 How the Application Works (High Level)

1. The frontend (Next.js, port 3000) serves the UI and calls the backend via a base URL configured in NEXT_PUBLIC_API_URL.
2. The backend (FastAPI + Uvicorn, port 8080) exposes REST routers — note that the FastAPI app itself has no global /api prefix. Each router defines its own prefix: /users, /puzzles, /circuits, /arsenal, /ratings, /admin, /debug, /discussions, /reports. The frontend env (apps/nextjs-app/src/config/env.ts) strips a trailing /api from NEXT_PUBLIC_API_URL before issuing requests, so http://localhost:8080/api becomes http://localhost:8080 at call time.
3. Controllers delegate to services in src/Backend/ServiceLayer/, which orchestrate domain objects and repositories.
4. Repositories in src/Backend/PersistantLayer/ wrap SQLite (escape_circuit.db at repo root) using WAL mode.
5. Authenticated requests carry an in-memory session token (issued by AuthService at login). Tokens are not JWTs and are cleared when the backend restarts.
6. When a user submits a circuit, SolvingService runs it through logicEngineService against the puzzle's test cases and records the attempt; XPService awards XP on success.

3. Architecture Overview

3.1 Layered Backend

The backend follows a strict four-layer architecture. Each layer only depends on the layer beneath it. Controllers must not bypass services, and services must not bypass repositories to talk to SQLite directly.

Layer	Folder	Responsibility
APILayer	src/Backend/APILayer/	FastAPI routers and HTTP concerns: parse requests, call services, shape responses, propagate auth tokens.
ServiceLayer	src/Backend/ServiceLayer/	Business logic: orchestrates repos, enforces rules (publishing limits, XP rewards, rating weights, role checks).
DomainLayer	src/Backend/DomainLayer/	Pure domain models (dataclasses) and enums. No I/O.
PersistantLayer	src/Backend/PersistantLayer/	SQLite repositories. Schema creation, queries, transactions. Thread-local connections.

3.2 Frontend Composition

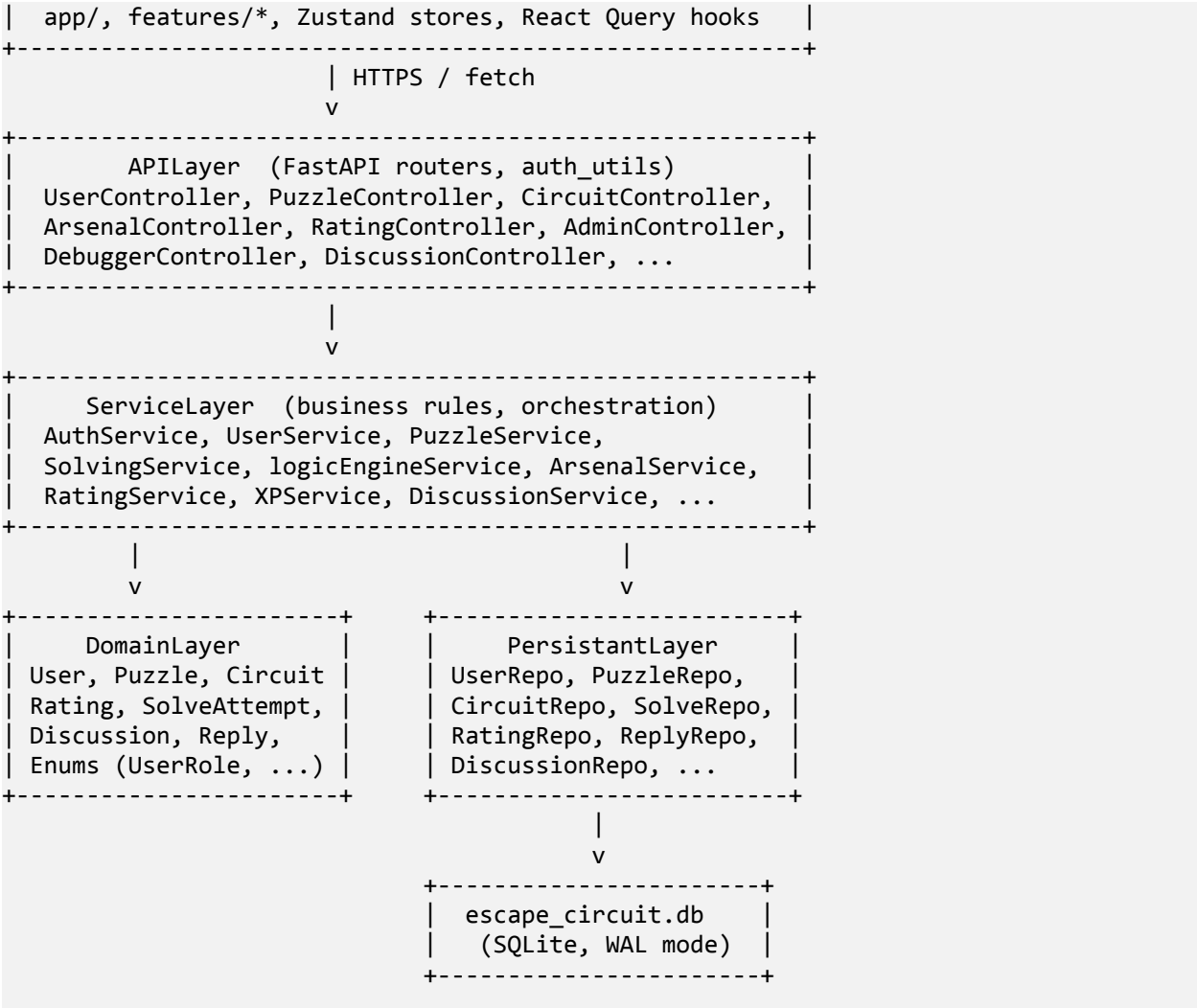
- App Router under apps/nextjs-app/src/app/ groups pages by area: app/ (authenticated workspace), auth/ (login/register/google), public/ (unauthenticated views).
- Feature folders under apps/nextjs-app/src/features/ (admin, arsenal, auth, comments, discussions, notifications, puzzles, ratings, teams, users). Most contain api/ and components/; some (puzzles) also have hooks/ and utils/. There is no enforced store/ or types/ convention — client state lives in colocated Zustand stores where needed.
- State and data: Zustand for client state, TanStack React Query for server state. Do not introduce a second library for either (see CLAUDE.md).
- UI primitives: Radix UI + Tailwind CSS. Forms use React Hook Form with Zod schemas.

3.3 Key Design Decisions

- SQLite + WAL: enables a single deployable file with safe concurrent reads while writers serialize. The deploy script never uses kill -9 because a graceful SIGTERM lets SQLite checkpoint the WAL.
- In-memory sessions: AuthService stores tokens in a Python dict with a 24-hour sliding TTL. Trade-off: simple, no extra infra, but every backend restart logs all users out.
- Repository pattern: each entity has its own repo with explicit SQL. There is no ORM; schema is created on demand by _ensure_schema methods invoked at startup.
- Layered controllers: controllers are kept thin; large logic (puzzle creation, solving, ratings) lives in services that are unit-testable in isolation.
- Frontend feature folders: keep cross-cutting concerns (api hooks, stores, types) close to the UI that consumes them rather than in global folders.

3.4 Textual Architecture Diagram

```
+-------------------------------------------------------+
|                  Browser (Next.js UI)                 |
```


app/, features/*, Zustand stores, React Query hooks
HTTPS / fetch
APILayer (FastAPI routers, auth_utils)
UserController, PuzzleController, CircuitController,
ArsenalController, RatingController, AdminController,
DebuggerController, DiscussionController, ...
ServiceLayer (business rules, orchestration)
AuthService, UserService, PuzzleService,
SolvingService, logicEngineService, ArsenalService,
RatingService, XPService, DiscussionService, ...
DomainLayer
User, Puzzle, Circuit
Rating, SolveAttempt,
Discussion, Reply,
Enums (UserRole, ...)
PersistantLayer
UserRepo, PuzzleRepo,
CircuitRepo, SolveRepo,
RatingRepo, ReplyRepo,
DiscussionRepo, ...
escape_circuit.db
(SQLite, WAL mode)

4. Repository Structure

4.1 Top-Level Layout

Path	What it is
README.md	Project tagline, requirements, list of seeded puzzles.
HOWTORUN.md	Three ways to start the app locally; Google OAuth setup.
CLAUDE.md	Project conventions: tech stack, layering rules, critical do-nots.
docs/SETUP.md, docs/FEATURES.md	Dev setup and feature catalog.
init_env.py	Loads NEXT_PUBLIC_GOOGLE_CLIENT_ID from frontend .env, starts both servers via npx concurrently.
run_server.sh / run_server.bat	One-shot launchers: init DB, seed riddles, seed admin, install frontend, start servers.
deploy.sh	Production deploy: graceful SIGTERM, WAL-safe backup, git pull, deps, migrations, restart, health check.
reset_db_hard.sh / .bat	Drop and recreate the DB (destructive).
requirements.txt	Backend Python dependencies.
package.json (root)	Workspace-level helpers; main package is apps/nextjs-app.
conftest.py, pytest.ini	Pytest configuration; adds src/ to sys.path, patches Mock comparison.
run_coverage.py	Helper for running test coverage.
escape_circuit.db	Dev SQLite database (root). Never commit. WAL sidecar files: .db-wal, .db-shm.
riddles/	Twelve puzzle definitions used as seed data.
src/	Backend source plus seed/init scripts.
apps/nextjs-app/	Frontend application.
backend_class_diagram.mmd	Mermaid class diagram of the backend.

4.2 Backend Folder (src/Backend/)

Subfolder / File	Purpose
main.py	App factory: registers routers, repos, services; configures CORS; mounts ContentionMonitorMiddleware.
settings.py	All tunables: XP rewards, rating weights, publishing quotas, CORS origins, DB timeouts.
APILayer/	Controllers: UserController, PuzzleController, CircuitController, ArsenalController, RatingController, AdminController, DebuggerController, DiscussionController; plus auth_utils.py, middleware.py, _deps.py.

Subfolder / File	Purpose
ServiceLayer/	AuthService, UserService, PuzzleService, SolvingService, logicEngineService, ArsenalService, RatingService, XPService, CircuitService, CluesService, DiscussionService, ReplyService, AdminService, NotificationService.
DomainLayer/	User, Puzzle, Circuit, PuzzleTestCase, SolveAttempt, Rating, Discussion, Reply, Enums, Exceptions, CluePenalty.
PersistantLayer/	_db.py (connection helper, WAL pragma), UserRepo, PuzzleRepo, CircuitRepo, SolveRepo, RatingRepo, CluesRepo, DiscussionRepo, ReplyRepo, EngagementRepo, NotificationRepo, AuditLogRepo, ReportRepo.

4.3 Frontend Folder (apps/nextjs-app/src/)

Subfolder	Purpose
app/	Next.js App Router segments: app/* (authed workspace), auth/* (login, register, complete-google), public/* (public discussion views).
features/	Feature modules: admin, arsenal, auth, comments, discussions, notifications, puzzles, ratings, teams, users.
components/	Shared UI: layouts/, errors/, ui/ (Radix primitives).
hooks/	Reusable hooks not tied to a single feature.
lib/	api-client.ts (fetch wrapper, token injection), auth.tsx, authorization.ts, react-query.ts, cache-storage.ts.
context/	React context providers (theme, auth, notifications).
config/	App-level configuration constants.
types/	Shared TypeScript types.
utils/	Helper functions.
styles/	Global CSS / Tailwind base.
middleware.ts	Next.js edge middleware (route guards / redirects).

4.4 Files Maintainers Most Often Edit

- src/Backend/settings.py — to tune XP, quotas, CORS origins, timeouts.
- src/Backend/ServiceLayer/PuzzleService.py and SolvingService.py — the largest services; new puzzle behavior goes here.
- src/Backend/ServiceLayer/logicEngineService.py — add new gate types or simulation semantics here.
- src/Backend/APILayer/<X>Controller.py — to expose a new endpoint.
- apps/nextjs-app/src/features/<area>/ — to add or modify a frontend feature.
- riddles/riddle_NN_*/ — to add a new puzzle (config JSON, instructions TeX, sample solution JSON).

5. Local Development Setup

5.1 Prerequisites

- Python 3.10 or higher.
- Node.js 20 or higher.
- Yarn 1.22 or higher (project standard) or npm.
- Git.
- A Google OAuth Client ID if you want to test Google login (otherwise leave the value empty and use username/password login).

5.2 First-Time Installation

1. Clone the repository and enter it.
2. Install Python dependencies: pip install -r requirements.txt (use a virtualenv).
3. Copy apps/nextjs-app/.env.example to apps/nextjs-app/.env and set NEXT_PUBLIC_GOOGLE_CLIENT_ID if needed.
4. Install frontend deps: cd apps/nextjs-app && yarn install (or npm install).

5.3 Environment Variables

All frontend public variables live in apps/nextjs-app/.env. The backend reads GOOGLE_CLIENT_ID; init_env.py propagates the value from the frontend .env so both processes see the same ID.

Variable	Where	Default / Notes
NEXT_PUBLIC_API_URL	frontend .env	http://localhost:8080/api
NEXT_PUBLIC_URL	frontend .env	http://localhost:3000
NEXT_PUBLIC_GOOGLE_CLIENT_ID	frontend .env	Required for Google login. Empty disables Google login.
NEXT_PUBLIC_ENABLE_API_MOCKING	frontend .env	false. Set true to use MSW mocks for frontend dev.
GOOGLE_CLIENT_ID	backend env	Set by init_env.py from the frontend value. Required when verifying Google ID tokens.

5.4 Running the App

Three equivalent ways are supported. From the repo root:

```
# Cross-platform single command (recommended)
python init_env.py

# Or, on Windows
run_server.bat

# Or, on macOS/Linux
bash run_server.sh
```

When started this way, the launcher initializes the DB, seeds the 12 riddles, creates the admin user (admin / password123), installs frontend dependencies, then launches the backend on http://localhost:8080 and the frontend on http://localhost:3000.

To run servers separately:

```
# Backend
cd src
python -m uvicorn Backend.main:app --reload --host 127.0.0.1 --port 8080

# Frontend
cd apps/nextjs-app
yarn dev
```

5.5 Running Tests

```
# Backend tests (pytest, from repo root)
pytest                          # all tests
pytest src/tests/ServiceLayerTests
pytest -k "test_solve"          # filter by name
python run_coverage.py          # with coverage

# Frontend static checks (from apps/nextjs-app)
yarn check-types                # tsc --noEmit
yarn lint                       # next lint
yarn build                      # production build
```

Note: at the time of writing, apps/nextjs-app/package.json does not commit test or test-e2e scripts, and no vitest/playwright config files are checked in. Frontend automated tests are therefore not runnable out-of-the-box; the project documents Vitest and Playwright as intended tools, but reintroducing them requires re-adding scripts (e.g. "test": "vitest", "test-e2e": "playwright test") and the matching config files.

5.6 Common Setup Issues

Symptom	Likely cause	Fix
Frontend cannot reach backend	Backend not running or wrong NEXT_PUBLIC_API_URL	Confirm http://localhost:8080/ responds; align the env value.
Google login button does nothing	NEXT_PUBLIC_GOOGLE_CLIENT_ID not set	Add it in apps/nextjs-app/.env, restart both processes.
"database is locked" errors	Long-running write or stuck process holding the DB	Stop all uvicorn instances with SIGTERM (not -9); inspect logs for slow handlers.
Stale schema after edits	DB created before a new column was added	Run python src/reinit_db.py or use reset_db_hard.sh (destructive).
Yarn install fails on Node 18	Some dev deps may need Node 20	Upgrade Node, re-install.
Tests cannot import Backend.*	src/ not on sys.path	Run pytest from repo root so conftest.py applies.

6. Maintenance Guide

6.1 Regular Maintenance

- Pull the main branch periodically and re-run the full test suite locally before deploying.
- Inspect the production logs/uvicorn.log file weekly for ERROR or contention warnings (logged when a request waits more than 3s for a DB write lock).
- Verify scheduled backups exist under backups/ on the production server; deploy.sh keeps backups for 30 days and prunes older ones.
- Check the audit log via the admin UI for any unexpected role changes or moderation events.

6.2 Updating Dependencies Safely

Backend: requirements.txt currently pins fastapi, uvicorn, pydantic, pytest, python-multipart, google-auth, requests. To upgrade:

1. Bump a single line, run pip install -r requirements.txt in a clean venv.
2. Run pytest. Pay special attention to AuthService (depends on google-auth) and PuzzleController (depends on FastAPI request parsing).
3. Bring up the app locally and click through login, puzzle solve, discussion post.
4. Commit and open a PR; never bundle a backend dependency bump with a feature change.

Frontend: edit apps/nextjs-app/package.json one cluster at a time (e.g. React + Next, or Radix UI only). After yarn install, run yarn check-types, yarn lint, yarn build before merging. (If Vitest/Playwright scripts have been restored — see section 16 — also run them.)

6.3 Logs and Errors

- Backend logs to stdout by default. In production (deploy.sh), output is redirected to logs/uvicorn.log.
- ContentionMonitorMiddleware emits INFO at >1s and WARNING at >3s, both with the path and elapsed time. Use these to find slow handlers.
- The global exception handler returns a JSON {detail: ...} body and a 4xx/5xx code; check the traceback in the log for the failing layer.
- Frontend errors surface in the browser console and in components/errors/. React Query devtools (if enabled) show network state for each query.

6.4 Backups and Data Protection

- escape_circuit.db is the only stateful artifact. It must be backed up while the writer is stopped or quiesced — deploy.sh handles this by SIGTERM-then-copy.
- Never delete *.db-wal or *.db-shm manually — SQLite needs them to recover. If you must move the DB file, copy all three together.
- Avoid kill -9 against uvicorn; use SIGTERM so WAL contents are checkpointed into the main DB file.

6.5 Verifying the System After Changes

1. yarn check-types in apps/nextjs-app.
2. yarn lint in apps/nextjs-app.
3. Frontend automated tests (Vitest/Playwright) — only if those scripts and configs have been restored to apps/nextjs-app/ (see section 16). Otherwise verify via yarn build and a manual click-through.
4. yarn build in apps/nextjs-app (catches App Router issues that dev does not).
5. pytest at repo root.
6. Manual smoke: login as admin, solve a seeded puzzle, post a discussion, open http://localhost:8080/docs (FastAPI default; no /api prefix in the backend itself) for the OpenAPI page.

6.6 Maintenance Checklist

Cadence	Task
Every deploy	Run all type/lint/unit/e2e tests; verify backup file exists; tail logs for 2 minutes after restart.
Weekly	Inspect backups/ directory; check disk usage; review WARNING contention logs.
Monthly	Update non-breaking dependencies (patch versions); re-run full test suite.
Per semester	Review admin user list; rotate the admin password if leaked; archive completed audit logs.
Per academic year	Re-evaluate major framework upgrades (Next.js minor, FastAPI minor).

7. Extension Guide

This section gives concrete recipes for extending the system. Each recipe references actual files; copy the nearest existing example rather than starting from scratch.

7.1 Adding a New Frontend Page

1. Pick the right route group: protected pages go under apps/nextjs-app/src/app/app/, login flows under auth/, unauthenticated views under public/.
2. Create a new folder named after the segment, e.g. app/app/badges/, and add a page.tsx exporting a default React component.
3. If the page needs server data, add or reuse a hook under apps/nextjs-app/src/features/<area>/api/ that wraps useQuery with the existing api-client.
4. Add a route constant in apps/nextjs-app/src/config/paths.ts and a navigation entry in apps/nextjs-app/src/app/app/_components/dashboard-layout.tsx so users can reach the page from the authenticated workspace.
5. Add a Vitest test next to the component (e.g. page.test.tsx) and a Playwright happy-path test under the e2e folder.

7.2 Adding a New API Endpoint

Endpoints are added across four layers. Example: add GET /users/{id}/badges (no global /api prefix — UserController already mounts /users as the router prefix).

1. DomainLayer: if a new entity is needed, add a dataclass under src/Backend/DomainLayer/ and any new enum values in Enums.py.
2. PersistantLayer: add a method on the relevant repo. Keep schema creation in _ensure_schema and use parameterised SQL with the cursor() / commit() pattern used in UserRepo.
3. ServiceLayer: add a method on the service that owns the entity (e.g. UserService.list_badges(user_id, token)). Validate inputs, raise ValidationError on bad data.
4. APILayer: in the matching controller, add the route. Use verify_token from auth_utils.py to authenticate, call the service, and return its result.
5. Tests: add a service unit test in src/tests/ServiceLayerTests, a repo test in src/tests/PersistantLayerTests, and — if the endpoint is end-user-visible — an integration test in src/tests/IntegrationTests.

Do not skip layers (e.g. controllers writing SQL directly). The four-layer separation is enforced in code reviews.

7.3 Adding or Modifying a Domain Model

1. Edit the dataclass in DomainLayer; add the field with a default to keep existing callers working.
2. Update the corresponding repo: add the column in CREATE TABLE inside _ensure_schema, then add a small migration in init_db.py (e.g. ALTER TABLE ... ADD COLUMN ... IF NOT EXISTS via PRAGMA-checked logic).
3. Update insert/update SQL in the repo to read/write the new column.
4. Plumb the new field through any service method that constructs the dataclass.

5. Run pytest — broken repo tests will guide you to the missing changes.

7.4 Adding a New Puzzle (Riddle)

1. Create a new folder under riddles/ named riddle_NN_<short_name>/ where NN is the next two-digit index.
2. Add three files (use any existing riddle as a template):

- riddle_NN_<name>_config.json — puzzle metadata and test cases (see schema below).
- riddle_NN_<name>_instructions.tex — LaTeX instructions; the seeder converts a small subset (sections, textbf, textit, texttt, lists) to HTML.
- riddle_NN_<name>_sample_solution.json — a valid circuit graph; useful for sanity-checking test cases.

1. Run python src/insert_riddles.py to seed the new puzzle into the dev DB.
2. Log in as admin and verify the puzzle appears in /app/puzzles.

Config JSON skeleton:

```
{
  "puzzle": {
    "name": "My Puzzle",
    "description": "Short description shown in the browser.",
    "budget": 20,
    "creator_budget": 12,
    "time_limit_seconds": null,
    "difficulty": "MEDIUM",                  // EASY | MEDIUM | HARD
    "default_gate_set": ["AND", "OR", "NOT", "XOR"],
    "inputs": ["A", "B"],
    "outputs": ["Y"],
    "total_gate_count": null,
    "min_cycles": null,
    "max_cycles": null,
    "allow_arsenal": false,
    "riddle_base_name": "riddle_13_my_puzzle",
    "clues": ["First hint", "Second hint"]
  },
  "test_cases": [
    { "kind": "blackbox", "inputs": {"A": 0, "B": 1}, "expected_outputs": {"Y": 1} },
    { "kind": "gate_count_limit", "max_gate_count": 6 }
  ]
}
```

7.5 Adding a New Gate Type

1. Add an entry to GateType in src/Backend/DomainLayer/Enums.py.
2. Implement the gate semantics in logicEngineService.py alongside the existing gate evaluation switch.
3. If the gate has a non-trivial cost, update the cost-calculation code in the same service.
4. Add a UI component under apps/nextjs-app/src/features/puzzles/components/ so users can place the gate on the canvas; the supporting hooks live in apps/nextjs-app/src/features/puzzles/hooks/. Optionally add a Storybook story (yarn storybook) for preview.

5. Add unit tests in src/tests/ServiceLayerTests covering truth-table evaluation for the new gate.

7.6 Adding or Modifying Validations

Backend validation lives in services and raises ValidationError, which the FastAPI exception handler converts to HTTP 400 with a JSON {detail: ...} body. Front-end validation uses Zod schemas in apps/nextjs-app/src/features/<area>/. Mirror the rule on both sides:

- Backend: validate first in the service. Never trust the controller-level Pydantic schema alone for business rules.
- Frontend: extend the Zod schema and surface the error in the form via React Hook Form.
- Add tests for both happy and invalid paths.

7.7 Adding Tests

- Backend unit test for a service: place in src/tests/ServiceLayerTests/test_<service>.py. Use the patterns in test_solving_service.py: instantiate fake repos with Mock, inject into the service constructor, assert behavior.
- Backend integration test: in src/tests/IntegrationTests, wire real repos against a temporary SQLite file.
- Frontend unit test: place <Component>.test.tsx next to the component. Use @testing-library/react and render with the existing test wrappers (QueryClientProvider, etc.).
- Frontend e2e: under the Playwright tests folder; spin up the backend first.

7.8 Updating UI Behavior

- Look first in apps/nextjs-app/src/features/<area>/. Each feature owns its components, hooks, and store; change behavior there before touching shared primitives in components/ui/.
- For visual changes, prefer adjusting Tailwind classes over editing global CSS in styles/.
- For new shared UI primitives, add them under components/ui/ and document with a Storybook story (yarn storybook on port 6006).

8. Testing and Quality Assurance

8.1 Test Layout

Folder	What it covers
src/tests/DomainUnitTests	Domain dataclasses and enums (Circuit, Puzzle, PuzzleTestCase, Rating, SolveAttempt, User, Discussion, Reply, CluePenalty, Enums, Exceptions).
src/tests/PersistantLayerTests	Repository CRUD, schema, search, thread-local connections.
src/tests/ServiceLayerTests	Business logic: AuthService, UserService, PuzzleService, SolvingService, ArsenalService, RatingService, CircuitService, DiscussionService, ReplyService.
src/tests/IntegrationTests	Cross-layer flows (e.g. solving a puzzle end-to-end with real repos against a temp DB).
src/tests/SystemTests	Full-app flows (e.g. discussion creation -> reply -> acceptance).
apps/nextjs-app (Vitest)	Component unit tests next to the component, plus tests under src/features/<area>/__tests__.
apps/nextjs-app (Playwright)	End-to-end browser flows; require the backend to be running.

8.2 Test Types

- Unit tests — most numerous; isolate a single class. Backend uses pytest with Mock; frontend uses Vitest with @testing-library/react.
- Integration tests — wire several backend layers against a temporary SQLite file.
- System / E2E tests — simulate a real user; Playwright on the frontend, pytest SystemTests on the backend.

8.3 Running Everything

```
# Backend (from repo root)
pytest                                  # all backend tests
pytest src/tests/IntegrationTests       # one folder
python run_coverage.py                  # with coverage

# Frontend static checks (from apps/nextjs-app)
yarn check-types && yarn lint && yarn build
```

Frontend Vitest / Playwright commands are intentionally absent from this block. As of the current repository state, no test or test-e2e script is committed in apps/nextjs-app/package.json, and no Vitest or Playwright config files are present under apps/nextjs-app/. The pieces listed under 8.1 describe the intended structure if/when those scripts and configs are restored; treat the frontend test rows there as documentation of intent until then.

8.4 Adding Tests for New Features

For each new feature, add at minimum: one happy-path unit test in the layer where the rule lives, one failure-path test for the validation, and one integration or e2e test that exercises the new endpoint or screen end-to-end. Reuse fixtures from conftest.py rather than rebuilding mocks.

8.5 Coverage Expectations

Recent commit messages on main explicitly raise backend coverage (e.g. "raise backend coverage to 86%"). The 85%+ target referenced here is taken from those commit messages and is not codified in any committed policy file or CI workflow — treat it as aspirational guidance, not a hard gate. Run python run_coverage.py before merging non-trivial backend changes and confirm the new code is exercised.

8.6 Manual Testing Checklist for Lecturers/Evaluators

Area	Check
Startup	python init_env.py brings both servers up; http://localhost:8080/docs (FastAPI default — no /api prefix) renders the OpenAPI page; http://localhost:3000/ loads the landing page.
Auth	Register a new user; log in; log out; admin / password123 grants admin access.
Solve	Open a seeded puzzle, place gates, hit Run, submit a passing solution, see XP increment.
Create	As a CREATOR, create a draft, fill inputs/outputs/test cases, publish, then solve from another account.
Discussions	Create a thread, post a reply, react to a reply, accept a solution as the OP.
Admin	Open /app/admin; assign creator role to a SOLVER; ban a user from discussions; check the audit log.
Robustness	Restart backend mid-session: existing sessions should expire (re-login required), DB should reopen cleanly.

9. Configuration and Constants

9.1 Where to Look

- src/Backend/settings.py — XP rewards, level formula, difficulty tier thresholds, arsenal slot tiers, publishing quotas, rating weights, discussion XP, pagination, CORS origins, DB timeouts, contention thresholds.
- apps/nextjs-app/.env — NEXT_PUBLIC_API_URL, NEXT_PUBLIC_URL, NEXT_PUBLIC_GOOGLE_CLIENT_ID, NEXT_PUBLIC_ENABLE_API_MOCKING, NEXT_PUBLIC_MOCK_API_PORT.
- apps/nextjs-app/src/config/ — frontend-side constants (e.g. route paths, UI defaults).
- pytest.ini — test discovery rules.

9.2 Changing Constants Safely

1. Open src/Backend/settings.py and make the change in a single commit.
2. Run pytest — unit tests that hard-code thresholds (e.g. XP rewards) will fail and tell you what to update.
3. If a constant is consumed by the frontend (rare — most are server-side), check that any matching frontend literal also moves.
4. For runtime tuning (timeouts, contention thresholds) consider whether the change should be deployment-time only, not committed.

9.3 What Must Not Be Hardcoded

- Secrets: Google Client ID, JWT secret (if a future JWT layer is added), DB connection paths in production.
- Origins: production hostnames belong in settings.py, but the dev admin credentials (admin / password123) are dev-only and must never be shipped enabled. Rotate or remove for production.
- Test data: do not embed test users or puzzles into the production seed scripts beyond the canonical 12 riddles.

10. Data and Persistence

10.1 Storage Model

All persistent state is in a single SQLite file, escape_circuit.db, at the repo root. SQLite runs in WAL mode for concurrent read/write safety. The schema is created lazily by each repository's _ensure_schema method when the repo is first instantiated, so a fresh checkout becomes a usable database after running python src/init_db.py.

10.2 Key Tables (Conceptual)

Table (repo)	Holds
UserRepo	Users: id, username, email, hashed password (PBKDF2), role, xp, bio, avatar.
PuzzleRepo	Puzzles: metadata, board size, budget, status, difficulty, allowed gates, clues, test cases.
CircuitRepo	Saved circuits and arsenal pieces.
SolveRepo	Solve attempts (passed/failed, time, cost, fail_reason); XP earned per puzzle.
RatingRepo	Per-user ratings (difficulty/fun/clearness) with experience flag.
DiscussionRepo / ReplyRepo / EngagementRepo	Forum threads, replies, votes, reactions, bookmarks.
NotificationRepo	User notifications (rating received, reply received, etc.).
AuditLogRepo	Admin actions (role changes, bans, deletions).
ReportRepo	Moderation reports against discussions or replies.

10.3 Backups

- deploy.sh automatically backs up the DB on every deploy to backups/escape_circuit_<timestamp>.db; sidecars (-wal, -shm) are copied only if non-empty.
- Backups older than 30 days are pruned.
- For manual backup outside of deploy.sh, stop the backend gracefully first (SIGTERM, not -9), then copy all three .db, .db-wal, .db-shm files together.

10.4 Schema Changes

Because the schema lives inside _ensure_schema methods, there is no formal migrations folder. To add a column safely:

1. Add the column to the CREATE TABLE statement in the repo (this only affects new databases).
2. For existing databases, run an idempotent ALTER TABLE — either by extending init_db.py or by writing a one-shot migration script under src/.
3. Run init_db.py in production via deploy.sh before users hit the new endpoints.

4. Keep changes additive when possible. Renames and type changes are painful in SQLite — do them only with a clear plan.

11. Security and Access Control

11.1 Authentication

- Username and password: AuthService.login(username, password) verifies a PBKDF2 hash stored by UserRepo and issues an in-memory, URL-safe 32-byte token.
- Google OAuth: the frontend obtains an ID token via @react-oauth/google, posts it to the backend, which verifies it with google-auth and calls AuthService.login_external.
- Tokens are NOT JWTs. They live in a Python dict inside the running process with a 24-hour sliding TTL. A backend restart logs everyone out — this is by design.

11.2 Authorization

- Every protected endpoint extracts the token via auth_utils.verify_token and resolves it through AuthService.require_user_id.
- Role checks happen in the service layer: user.role == UserRole.ADMIN for admin endpoints; quota checks (max_published_puzzles) inside PuzzleService.
- PENDING_CREATOR users can create drafts but cannot publish until an admin upgrades them.

11.3 Common Risks for Maintainers to Avoid

- Do not bypass the service layer from a controller. SQL embedded in controllers escapes the layering rules and is unreviewable.
- Do not ship the dev admin credentials (admin / password123) to production. Replace or disable that account during deploy.
- Do not relax CORS origins in settings.py without a clear reason. The defaults intentionally only allow localhost:3000/3001.
- Do not interpolate user input into SQL strings. All repos use parameterised queries — keep it that way.
- Be careful with file-based hints/instructions. The LaTeX-to-HTML conversion in insert_riddles.py is intentionally limited; do not extend it to evaluate arbitrary HTML or scripts.
- Sessions live in memory; never serialize raw tokens to a log or share them across processes.

12. Deployment and Production Notes

12.1 Production Topology

Operator-provided (not derived from the repo, see section 16). The production server is reached over SSH at `admin@132.73.84.76` (the password is held by Noam). The application lives at `/var/www/EscapeCircuit`. On this VM the backend and frontend are supervised by pm2 under the process names `fastapi-backend` and `nextjs-frontend`. The site is served from `https://escapecircuit.online` with the backend mounted at `https://escapecircuit.online/server`. Note: the pm2 setup is part of the production VM configuration; nothing in the repository creates or manages pm2 processes. The committed deploy.sh script uses nohup uvicorn instead (see 12.3).

12.2 Refreshing the Server (Operator Procedure)

Operator-provided. Use this procedure to pick up the latest main on the pm2-managed production VM:

1. Connect to the server:

```
ssh admin@132.73.84.76        # password held by Noam
```

2. Pull the latest code from main:

```
cd /var/www/EscapeCircuit
git pull origin main
```

3. Apply the changes. After git pull the running processes still hold the old code in memory. Choose the steps that match what changed:

Backend-only changes (Python, schema, business logic)

A restart of the FastAPI process is enough so it reloads the new code:

```
pm2 restart fastapi-backend
```

Frontend changes (Next.js, UI, components)

Next.js needs a production rebuild before the new code is served. Build and then restart the frontend process:

```
cd /var/www/EscapeCircuit/apps/nextjs-app
NEXT_PUBLIC_API_URL=https://escapecircuit.online/server npx next build --no-lint
pm2 restart nextjs-frontend
```

Changes in both

Run the backend restart first, then the frontend build and restart. The order matters because the new frontend may rely on new backend endpoints.

12.3 deploy.sh (committed, backend-only)

Independent of the pm2 procedure above, the repository ships a deploy.sh script that handles the BACKEND only. It does not touch pm2 and does not rebuild the frontend. Useful when running the backend under plain uvicorn (the way deploy.sh expects). From the repo root:

1. Send SIGTERM (via pkill -f 'uvicorn Backend.main:app') to the running uvicorn so SQLite checkpoints WAL; escalates to SIGKILL only if uvicorn does not exit within 15s.

2. Copy escape_circuit.db (and sidecars if non-empty) to backups/ with a timestamp.
3. Prune backups older than 30 days.
4. git pull origin main and pip3 install -r requirements.txt.
5. Run init_db.py and seed_admin.py to apply additive schema changes.
6. Restart uvicorn in the background with nohup python3 -m uvicorn Backend.main:app, redirecting to logs/uvicorn.log. (deploy.sh does not invoke pm2; the pm2 supervision in 12.1/12.2 is an alternative path managed by the operator.)
7. curl the backend root and assert HTTP 200; abort with a rollback note if not.

12.4 Local vs Production

Concern	Local	Production
Backend host:port	127.0.0.1:8080	Reverse-proxied behind https://escapecircuit.online/server
Frontend	yarn dev on :3000	next build + pm2 (nextjs-frontend)
Process manager	Foreground or init_env.py	Two paths: pm2 (operator-provided, used on the current VM) OR nohup uvicorn via deploy.sh (committed in repo, backend only)
Admin user	admin / password123	Must be rotated; never ship the dev default
Logs	stdout	logs/uvicorn.log (deploy.sh); additionally pm2 logs <name> when supervised by pm2
DB backups	Manual	Automatic on every deploy.sh run (under backups/, 30-day retention)

12.5 Deployment Checklist

1. Open a PR; merge to main only after all type/lint/test/build commands pass.
2. SSH to the server.
3. git pull origin main.
4. Restart the relevant process(es): pm2 restart fastapi-backend / nextjs-frontend on the pm2-supervised VM, or run bash deploy.sh on a uvicorn/nohup setup. Rebuild the frontend if UI or Next.js code changed.
5. Tail pm2 logs fastapi-backend and pm2 logs nextjs-frontend (or logs/uvicorn.log under deploy.sh) for at least 60 seconds.
6. Open https://escapecircuit.online, log in, solve one seeded puzzle.
7. Confirm the backup file for today exists under backups/ on the server.

12.6 Rollback

- If a deploy breaks the backend: pm2 stop fastapi-backend, git checkout <previous-good-commit>, pm2 restart fastapi-backend.
- If a deploy corrupted the DB: pm2 stop fastapi-backend, copy the most recent backups/escape_circuit_<ts>.db over escape_circuit.db (also restore .db-wal and .db-shm if they were backed up), then pm2 restart fastapi-backend.
- If a deploy breaks the frontend build: cd /var/www/EscapeCircuit && git checkout <previous-good-commit> -- apps/nextjs-app, then re-run the frontend build (NEXT_PUBLIC_API_URL=https://escapecircuit.online/server npx next build --no-lint) and pm2 restart nextjs-frontend.

13. Common Maintenance Scenarios

13.1 Fixing a Bug

1. Reproduce locally. Get a failing test or curl command first.
2. Identify the owning layer (controller for HTTP shape, service for business rules, repo for SQL, frontend for rendering).
3. Write a failing unit test in that layer.
4. Fix the smallest code change that turns the test green.
5. Run the full pytest suite and frontend static checks (yarn check-types, yarn lint, yarn build); open a PR referencing the issue number.

13.2 Adding a New Feature

1. Sketch the feature touchpoints across the four backend layers and the frontend feature folder.
2. Add domain types / enum values, then repo + schema, then service rule, then controller endpoint, then frontend hook, then UI.
3. Add tests at every layer you touched.
4. Update settings.py for any tunable constant.

13.3 Updating Dependencies

See section 6.2. In short: one cluster at a time, tests must pass, no behavior changes bundled in.

13.4 Changing a UI Screen

1. Find the screen under apps/nextjs-app/src/app/.
2. Modify the page or extract a smaller component under features/<area>/components/.
3. Add or update a Vitest test; check yarn check-types and yarn lint.
4. Run yarn dev and click through to verify the screen.

13.5 Adding a New Test Case to an Existing Puzzle

1. Edit riddles/riddle_NN_<name>/riddle_NN_<name>_config.json and append to the test_cases array.
2. Run python src/insert_riddles.py — it re-inserts the puzzle definition.
3. Solve the puzzle with the seeded sample solution to confirm the new test still passes.

13.6 Debugging Failed Tests

- Re-run the single failing test with -k and inspect the short traceback (-v --tb=short are default).
- If the failure is in PersistantLayerTests, check that you ran from the repo root so conftest.py applied.
- If the failure is in ServiceLayerTests, check that you wired the new dependency into the service constructor in main.py.

- If the failure is in a frontend Vitest run, inspect the rendered DOM via screen.debug() and confirm the React Query mock is configured.

13.7 Modifying Puzzle Logic

1. Decide whether the change belongs in the engine (logicEngineService) or in the rules layer (SolvingService, PuzzleService).
2. Engine changes (new gate, new evaluation rule) must come with unit tests over truth tables.
3. Rules changes (e.g. allow stream tests of length > N) must come with both a service test and an integration test that submits a solution.
4. Update at least one sample puzzle to exercise the new behavior so manual evaluators can see it working.

14. Known Limitations and Future Improvements

14.1 Current Limitations

- Sessions are kept in process memory only. A backend restart logs every user out, and horizontal scaling would require an external session store.
- Database is a single SQLite file. Fine for the academic deployment; not built for high concurrent write throughput.
- There is no formal migrations framework. Schema changes rely on additive ALTER TABLE in init_db.py and on _ensure_schema running at startup.
- There is no automated CI workflow in .github/workflows. Tests must be run manually before pushing.
- LaTeX-to-HTML conversion in the riddle seeder supports only a small subset (sections, textbf, textit, texttt, itemize, enumerate).
- Dev admin credentials (admin / password123) live in the seeding script and must be rotated manually for production.

14.2 Recommendations (Not Yet Implemented)

- Add a GitHub Actions workflow that runs pytest, yarn check-types, yarn lint, yarn test, yarn build on every PR.
- Replace in-memory sessions with persistent JWTs or a Redis-backed session store if horizontal scaling becomes a need.
- Introduce a thin migrations folder (numbered SQL files) to make schema evolution auditable.
- Add structured logging (JSON) and ship logs to a remote sink to keep history beyond the production VM.
- Extend the Storybook coverage to all primitives under components/ui/ so UI regressions are easier to spot.
- Add an admin-only screen to view live contention warnings rather than tailing the file.

These are suggestions, not commitments. The current code does not implement them — keep that line clear in any future report.

15. Appendix

15.1 Glossary

Term	Meaning in this project
Riddle / Puzzle	A creator-defined logic-circuit problem with inputs, outputs, budget, test cases, clues.
Arsenal	A solver-owned library of saved sub-circuits that can be reused as components in later puzzles.
Test case kind	blackbox, whitebox, stream, gate_limit, gate_count_limit, latency_limit (TestCaseKind enum).
XP / Level	User experience points; level = xp // 100 + 1; gates publishing capacity and arsenal slot count.
WAL	SQLite Write-Ahead Logging mode; produces .db-wal and .db-shm sidecars.
Session token	URL-safe 32-byte string issued by AuthService and stored in memory.
Pending creator	User who requested creator status; can create drafts but cannot publish.
Engagement	Vote/reaction/bookmark/follow records on discussions or replies.

15.2 Useful Commands

Goal	Command
Run both servers locally	python init_env.py
Backend only	cd src && python -m uvicorn Backend.main:app --reload --host 127.0.0.1 --port 8080
Frontend only	cd apps/nextjs-app && yarn dev
Reset DB (destructive)	bash reset_db_hard.sh | reset_db_hard.bat
Seed riddles	python src/insert_riddles.py
Create / refresh admin	python src/seed_admin.py
Reset admin password	python src/reset_admin.py
Backend tests	pytest
Coverage report	python run_coverage.py
Frontend tests	Not committed — no test/test-e2e script in package.json today. Restore vitest/playwright config and scripts to enable.
Type-check + lint + build	yarn check-types && yarn lint && yarn build
Storybook	cd apps/nextjs-app && yarn storybook (port 6006)
Production deploy	bash deploy.sh (on the server, in /var/www/EscapeCircuit)

Goal	Command
Restart backend (prod)	pm2 restart fastapi-backend
Rebuild + restart frontend (prod)	cd /var/www/EscapeCircuit/apps/nextjs-app && NEXT_PUBLIC_API_URL=https://escapecircuit.online/server npx next build --no-lint && pm2 restart nextjs-frontend

15.3 Useful File Paths

File	Purpose
src/Backend/main.py	Backend entry; router registration and DI wiring.
src/Backend/settings.py	All tunable constants.
src/Backend/ServiceLayer/SolvingService.py	Largest service; submit/simulate/validate.
src/Backend/ServiceLayer/logicEngineService.py	Gate evaluation, cost calculation.
src/Backend/ServiceLayer/PuzzleService.py	Puzzle CRUD, publishing rules.
src/Backend/DomainLayer/Enums.py	UserRole, PuzzleStatus, PuzzleDifficulty, TestCaseKind, GateType.
src/Backend/PersistantLayer/_db.py	Connection helper, WAL mode.
apps/nextjs-app/src/lib/api-client.ts	Frontend fetch wrapper and token injection.
apps/nextjs-app/src/lib/auth.tsx	Frontend auth context.
apps/nextjs-app/src/app/	Next.js App Router segments.
riddles/riddle_NN_*/	Seed puzzle definitions.
init_env.py	Loads .env, starts both servers.
deploy.sh	Production deploy steps.
backend_class_diagram.mmd	Mermaid class diagram.

15.4 Troubleshooting Table

Symptom	Likely Cause	Action
HTTP 401 from /users/me (or /api/users/me through the frontend proxy)	Token expired or backend restarted	Log in again from the frontend.
HTTP 400 with detail message	Service-layer ValidationError	Read the detail; adjust input or fix the service rule.
HTTP 500 in logs	Unhandled exception	Read the traceback in uvicorn.log; failing layer is usually two frames above FastAPI.
"database is locked"	Long-running write holding the WAL	Stop other writers; inspect ContentionMonitor INFO/WARNING entries.

Symptom	Likely Cause	Action
Empty puzzle list after fresh checkout	DB not seeded	Run python src/init_db.py then python src/insert_riddles.py.
Google login fails silently	Mismatched or missing GOOGLE_CLIENT_ID	Set the same ID in apps/nextjs-app/.env and ensure init_env.py was used to start the backend.
pm2 restart succeeds but UI still old	Frontend not rebuilt	Re-run next build before pm2 restart nextjs-frontend.
Tests cannot find Backend modules	Run from wrong directory	Run pytest from the repo root.
Frontend build fails on type errors	TypeScript drift	Run yarn check-types and fix at the source; do not silence with @ts-ignore.

16. Assumptions and Unverified Items

The following items in this guide are best-effort interpretations or could not be fully verified from the repository alone. Future maintainers should confirm them in context:

- Production SSH host (132.73.84.76), the deployment path /var/www/EscapeCircuit, and the pm2 process names (fastapi-backend, nextjs-frontend) come from the operational instructions provided to this document, not from a script committed in the repository. Update the values here if production topology changes.
- Some XP and rating thresholds (e.g. publishing capacity bonuses at levels 10–15) are taken from settings.py; if those constants change later, this guide and any user-facing docs must move together.
- The LaTeX-to-HTML conversion subset documented in section 7.4 was derived by inspecting insert_riddles.py; uncommon TeX commands may not render and should be tested before relying on them.
- There is no CI workflow in .github/workflows at the time of writing. If one is added later, mention it in section 8 (Testing).
- Frontend automated tests are documented as intent only. apps/nextjs-app/package.json contains no test or test-e2e script today, and no vitest or playwright config is committed under apps/nextjs-app/. Any reference in this guide to running yarn test or yarn test-e2e assumes those scripts and configs will be re-added; until they are, frontend correctness must be verified via yarn check-types, yarn lint, yarn build, and manual smoke tests.
- The "85%+ backend coverage" goal mentioned in section 8.5 is taken from commit messages on main (e.g. "raise backend coverage to 86%"), not from a committed policy file or enforced threshold in CI. It is aspirational guidance only.
- Public production URL (https://escapecircuit.online) and the reverse-proxy mount point /server are operator-provided; the repo contains no committed reverse-proxy config or production hostname.