PeerLearn Labs

PeerLearn — Project Plan (v1)

Date: 2025-09-25

1. Overview

PeerLearn is a collaborative learning platform where users work in rooms to co-edit documents, generate flashcards, and run quizzes (solo or live). The MVP targets fast, reliable collaboration with clear progression (XP, badges).

Goals

- Enable real-time doc collaboration in rooms.
- Generate flashcards from selected text using an AI provider.
- Deliver self and live quiz flows with scoring and XP events.
- Provide presence, chat, and invites to coordinate collaboration.

Success Criteria

- Room creation/joining < 3s P95.
- Real-time edits propagate < 300ms P95 within a region.
- Quiz latency (host → participants) < 500ms P95.
- Uptime ≥ 99.5% during test period.

2. Scope

In-scope (MVP)
- Auth (email/password via ASP.NET Identity; optional Google OAuth later)
- Rooms: create, invite (email token), membership roles
- Documents: CRDT-based collaborative editing (Yjs + y-websocket), snapshots
- Flashcards: deck creation, manual edit, AI generation from selection
- Quizzes: create from decks/docs; runs (self/live), scoring, results
- Chat: per-room basic messages
- Gamification: XP events; simple badges
- Admin basics: room moderation by owner/cohost

Out-of-scope (MVP)
- Mobile apps; offline-first editing
- Payments/monetization
- Rich whiteboard/canvas tooling (basic note kind only)
- LLM fine-tuning; multi-model routing

3. Deliverables

D1: System context diagram (C4 level 1)
D2: Logical data model (DBML + SQL DDL)
D3: Class diagram (UML)
D4: API surface (OpenAPI/Swagger)
D5: Project plan (this document)
D6: Test plan & QA checklist
D7: Deployment & runbook

4. Milestones & Sprints (8 weeks MVP)

M1 — Foundations (Week 1–2)
- Repo setup; CI; envs (dev/test/prod); infrastructure as code stubs

- Auth & Identity (email/password, force password change on first login)
- Base schema migrations (Postgres); seed admin; health checks

M2 — Rooms & Presence (Week 2–3)
- Rooms CRUD; invitations (email token); membership roles
- SignalR hub for presence & chat; basic room chat UI

M3 — Collaborative Docs (Week 3–4)
- Yjs integration (y-websocket server); document CRUD; selection API
- Snapshot service (optional server-side save)

M4 — Flashcards (Week 4–5)
- Decks CRUD; AI generation from selection; manual editing
- Deck browse & study mode

M5 — Quizzes (Week 5–6)
- Quiz from deck/document; live run flow (host + participants); scoring
- Results view; XP event creation

M6 — Polish & NFRs (Week 7)
- Role-based guards; rate limits; audit logs
- Observability: logs, metrics, traces; error budgets

M7 — QA & Launch (Week 8)
- End-to-end tests; load tests; backup/restore drills
- Launch checklist & handover docs

5. Architecture Summary

Frontend: React SPA (Vite), Auth via cookies (HttpOnly JWT), SignalR client, Yjs client (websocket).
Backend: ASP.NET Core Web API; EF Core → Postgres; MongoDB for content (optional, phase 2).
Realtime: SignalR for presence/chat; y-websocket for CRDT doc sync; WebRTC (phase 2) for voice.
AI Provider: OpenAI/HF via server-side calls with rate limiting.
Storage: Postgres (users, rooms, quizzes, XP); S3 for snapshots (optional); Redis (optional) for presence cache.
Observability: Serilog + OpenTelemetry (OTLP) → Grafana/Loki/Tempo.

6. Data Model (high-level)

Core: users, rooms, memberships, documents, decks/cards, quizzes (questions, runs, answers), XP, badges, messages, invitations, sessions. See DBML & SQL deliverables.

7. API Endpoints (MVP sketch)

- Auth: POST /auth/login, POST /auth/logout, GET /auth/me
- Rooms: POST /rooms, GET /rooms/:id, POST /rooms/:id/invite, POST /rooms/:id/join
- Presence: WS /hub (SignalR)
- Documents: GET/POST /rooms/:id/documents, WS /yjs/:docId
- Flashcards: POST /flashcards/generate, CRUD /decks, /cards
- Quizzes: POST /quizzes, POST /quizzes/:id/start, POST /quiz-runs/:id/answer
- XP: GET /me/xp

- Chat: POST /rooms/:id/messages, GET /rooms/:id/messages

8. Risks & Mitigations

Realtime complexity: Strict versioning & backpressure; test with lossy networks.
Vendor limits (LLM): Circuit breakers, request budgeting per room/user.
Data races: CRDT for docs; for commands use idempotency keys.
Abuse/spam: Rate limiting, invite tokens, CSRF for non-GET, audit logs.
Privacy: Text sent to AI providers only with user action; redact PII.

9. Security & Compliance

- HttpOnly cookies; rotate JWT signing keys; TLS everywhere.
- RBAC: platform role + room_role.
- Secure headers (CSP, HSTS); input validation; dependency scanning.

10. Testing Strategy

Unit tests (services); integration tests (API/DB); contract tests (web ↔ API).
Load tests for y-websocket and SignalR (k6).
E2E with Playwright; synthetic monitors post-deploy.

11. Deployment

Dockerized services; compose for dev; k8s for prod (or ECS).
Migrations on startup (safe); blue/green deploys.
Backups: nightly Postgres; restore runbook.

12. Acceptance Criteria

Complete M1–M7; latency/SLOs met; zero P1 bugs for 7 days post-launch.

13. Glossary

CRDT: Conflict-free Replicated Data Type for merge-free realtime edits.
Yjs: CRDT framework used for documents.
SignalR: ASP.NET real-time communication library.