**Exam Mode Rooms**

**Implementation:**

- **Rust WebSocket Server**: Use tokio + axum for managing concurrent room connections
- **Room Codes**: Generate cryptographically secure codes with rand crate, store in Supabase with expiry timestamps
- **Role Management**: Store user-room relationships in Supabase with role enums (Host/Explainer/Participant/Viewer)
- **Auto-lock**: Use Rust's tokio::time::sleep() to trigger room closure and update status in Supabase

**Suggested Changes:**

- Add a "Scheduled Rooms" feature where hosts can pre-create rooms with start times (useful for planned study sessions)
- Implement room templates (e.g., "DBMS Quiz Room", "OS Lab Practice") that auto-configure settings

**Shared Study Canvas**

**Implementation:**

- **CRDT Engine**: Use yrs crate (Rust Yjs port) for collaborative editing
- **Canvas Components**:
    - Markdown editor using pulldown-cmark for parsing
    - Diagrams via tldraw or excalidraw embedded in frontend, sync state through Rust
    - Cursor tracking: broadcast cursor coordinates via WebSocket every 100ms
- **Persistence**: Periodically snapshot CRDT state to Supabase PostgreSQL using postgrest-rs

**Suggested Changes:**

- Add version history (Git-like) so students can revert to earlier canvas states
- Implement @mentions to tag specific people in notes
- Add LaTeX support for math equations (critical for engineering students)

**Smart File Sharing**

**Implementation:**

- **File Upload**: Use Supabase Storage with RLS policies
- **PDF Parsing**:
    - Use pdf-extract or lopdf crates in Rust background workers
    - Extract text with poppler bindings
- **Auto-tagging**:

    - Train a simple keyword extraction model or use regex patterns to detect "Unit X", "Important", "PYQ"
    - Store tags in Supabase JSONB column
- **Inline Preview**: Use pdf.js on frontend, highlight coordinates stored in Supabase

**Suggested Changes:**

- Add OCR for handwritten notes using tesseract-rs (huge for students who write notes by hand)
- Implement "Quick Tags" - let users create custom tags like "Revision", "Hard", "Doubts"
- File versioning: track when students re-upload updated notes

**Explain Mode**

**Implementation:**

- **Audio Signaling**: Use WebRTC with Rust signaling server (webrtc-rs crate)
- **Hand Raise Queue**: Maintain priority queue in Rust, broadcast updates via WebSocket
- **Doubt Queue**: Store doubts in Supabase with status (pending/answered) and timestamps
- **Reactions**: Lightweight emoji broadcasts without DB persistence

**Suggested Changes:**

- Add screen sharing capability (critical for showing code/diagrams)
- Implement "Breakout Rooms" - host can split participants into smaller groups temporarily
- Record audio explanations and store in Supabase Storage for replay

**AI-Assisted Doubt Solver**

**Implementation:**

- **Embeddings**: Use fastembed-rs or call OpenAI/Cohere APIs from Rust
- **Vector Search**:
    - Use Supabase's pgvector extension for storing embeddings
    - Alternative: qdrant in-memory for faster search, periodically sync with Supabase
- **RAG Pipeline**: When doubt is tagged, query vector DB for similar past doubts and relevant notes

**Suggested Changes:**

- Integrate with a local LLM via Ollama (privacy-focused, no API costs)
- Add "Doubt Analytics" - show students which topics they struggle with most
- Implement "Expert Mode" where seniors/TAs can be notified of complex doubts

**Exam-Safe Code Rooms**

**Implementation:**

- **Sandboxing**:
  - Use wasmtime for WASM-based execution (supports Python via PyO3 + WASM)
  - Alternative: Docker containers with strict resource limits (bollard crate for Docker API)
- **Language Support**:
  - Java: Use rustjail or container-based execution
  - Python: WASM or rustpython interpreter
  - C/C++: Compile to WASM with emscripten
- **Resource Limits**: Set CPU time (100ms-5s), memory (128MB-512MB) via WASM config
- **Network Isolation**: WASM has no network access by default; for containers, use --network=none

**Suggested Changes:**

- Add test case validation (students upload test cases, code auto-runs against them)
- Implement "Live Debugging View" - see variables and execution flow in real-time
- **Critical Change**: Support MySQL/SQLite queries for DBMS practice (run in isolated DB instance)

**Replayable Sessions**

**Implementation:**

- **Event Streaming**: Use serde for serialization, store events in binary format with bincode
- **Event Types**: Code edits, cursor moves, chat messages, file uploads
- **Storage**: Store compressed event logs in Supabase Storage, metadata in PostgreSQL
- **Replay Player**: Frontend reads event log and replays at adjustable speed

**Suggested Changes:**

- Add "Key Moments" tagging during session (e.g., "Doubt Solved", "Important Concept")
- Implement search within replay (e.g., "find when we discussed binary trees")
- Export replay as video (using headless browser + ffmpeg in Rust worker)

**Micro-Games for Study Breaks**

**Implementation:**

- **Game Types**:
  - Logic puzzles: Sudoku, 2048-style number games
  - MCQ battles: Pull questions from uploaded notes, real-time scoring

    - Memory games: Flashcard matching using keywords from notes
- **Game Server**: Authoritative server in Rust using bevy_ecs for game logic
- **Multiplayer**: WebSocket-based state sync, 60 tick/sec for smooth gameplay

**Suggested Changes:**

- Add "Study Streak" gamification (daily login bonuses, XP for completed sessions)
- Leaderboards stored in Supabase with monthly resets
- **New Game Idea**: "Code Golf" - shortest code to solve a problem

**Presence That Actually Matters**

**Implementation:**

- **Presence States**: Store enum in Rust memory, persist to Supabase every 30s
- **High-Frequency Updates**: Broadcast presence changes via WebSocket to room members only
- **Bandwidth Optimization**: Send binary presence packets (1-2 bytes per user)

**Suggested Changes:**

- Add "Study Together" matching - connect with random users studying the same subject
- Show time spent in each state (analytics for students)
- Implement "Focus Sessions" - Pomodoro timer integrated with presence

**Anti-Distraction Mode**

**Implementation:**

- **Content Filtering**: Rust middleware intercepts messages, blocks based on regex patterns
- **Mode Enforcement**: Room setting stored in Supabase, enforced server-side
- **Timer**: Use tokio::time::interval() for Pomodoro-style breaks

**Suggested Changes:**

- Add "Website Blocker" (browser extension) that syncs with room mode
- Implement "Focus Score" - reward students for staying in distraction-free mode
- **Critical**: Add parental/teacher controls for enforcing anti-distraction in group sessions

**Cross-Room Intelligence**

**Implementation:**

- **Subject Tagging**: Rooms tagged with subject metadata in Supabase
- **Graph Database**: Use Supabase PostgreSQL with recursive CTEs or add age (Apache AGE) extension for graph queries
- **Room Discovery**: Index rooms by subject+time, show "Similar Active Rooms"

- **Note Sharing**: Read-only access via Supabase RLS policies

**Suggested Changes:**

- Implement "Study Hubs" - persistent subject communities (e.g., "DBMS Hub") beyond individual rooms
- Add room ratings/reviews for quality control
- **New Feature**: "Ask Alumni" - connect current students with seniors who took same courses

---

**Critical Architecture Recommendations**

**Database Schema:**

sql

-- *Supabase PostgreSQL*

rooms (id, code, host_id, subject, **mode**, **status**, expires_at)

room_members (room_id, user_id, role, presence_state, joined_at)

canvas_snapshots (room_id, crdt_state, version, created_at)

files (id, room_id, uploader_id, tags[], storage_path)

events (room_id, **type**, payload, **timestamp**) -- *For replay*

doubts (id, room_id, user_id, content, embedding, resolved)

**Rust Service Split:**

1. **WebSocket Gateway** (axum + tokio-tungstenite) - Handles all real-time connections
2. **CRDT Sync Service** (yrs) - Canvas collaboration
3. **Code Execution Worker** (wasmtime + queue system) - Sandboxed execution
4. **File Processing Worker** (PDF parsing, OCR, embeddings) - Background tasks
5. **Game Server** (optional microservice for games)

**Frontend:**

- Use React + TypeScript for type safety (you're familiar with React)
- @supabase/supabase-js for auth and database queries
- y-websocket or custom WebSocket client to connect to Rust server
- monaco-editor for code editing
- tldraw for canvas diagrams