

Chess-in-One AI Coach - Comprehensive Architecture & Design Document

Status: IN PROGRESS

Last Updated: January 3, 2026

Purpose: Complete end-to-end architectural and design reference

Table of Contents

1. [Executive Summary & System Overview](#)
 2. [System Architecture \(High-Level\)](#)
 3. [Detailed Component Design](#)
 4. [Data Flow & State Management](#)
 5. [Security Architecture](#)
 6. [AI Integration Architecture](#)
 7. [API Design](#)
 8. [Deployment Architecture](#)
 9. [Error Handling & Recovery](#)
 10. [Performance & Scalability](#)
 11. [Compliance & Governance](#)
 12. [Implementation Details](#)
 13. [Diagrams & Visualizations](#)
 14. [Appendices](#)
-

1. Executive Summary & System Overview

1.1 Product Intent

Chess-in-One AI Coach is a **post-game chess coaching application** designed to improve **thinking habits** rather than move accuracy. The system uses a Socratic questioning approach to guide junior competitive chess players through reflective analysis of their games.

Core Objective: The system captures the player's thinking first, identifies missing thinking dimensions, and uses structured Socratic questioning to provide minimal, calm, reflective guidance.

Non-Goals:

- No real-time coaching during games
- No move evaluation or scoring
- No chat-style interaction
- No public cloud or multi-tenant service

1.2 System Scope

Supported Mode: Strictly post-game analysis

Explicit Exclusions:

- Live boards
- Clock-aware behavior
- In-game move suggestions

- Real-time coaching
- Chat-style interaction
- Engine-first analysis

1.3 Target Users

- **Primary:** Junior competitive chess players (fast, intuitive decision-makers, tactically strong)
- **Secondary:** Parents (governance and approval)

1.4 Non-Negotiable Constraints

1. **No Calculation:** Frontend NEVER calculates chess moves - relies 100% on backend EngineTruth
 2. **Backend Authority:** Backend is Single Source of Truth
 3. **No Chat:** Structured Socratic state machine, NOT a chatbot
 4. **Parent Approval:** Required for non-default AI tiers
 5. **AI Hallucination Prevention:** All AI prompts include EngineTruth; AI output validated against engine facts
-

2. System Architecture (High-Level)

2.1 Architectural Patterns

Hybrid Intelligence Architecture

The system follows a strict Hybrid Intelligence model:

- **Chess engines** (Stockfish/Chess API providers) are the sole source of chess truth
- **AI Coach** is a pedagogical interpreter, not a chess reasoner
- AI MUST trust engine-provided facts as authoritative
- AI MUST NOT independently evaluate positions
- AI MUST NOT search for best moves or tactics

Zero-Trust Client Model

- Clients (Web, Android) are untrusted
- All validation and business logic in backend
- Frontend is stateless/dumb
- Backend enforces all state transitions

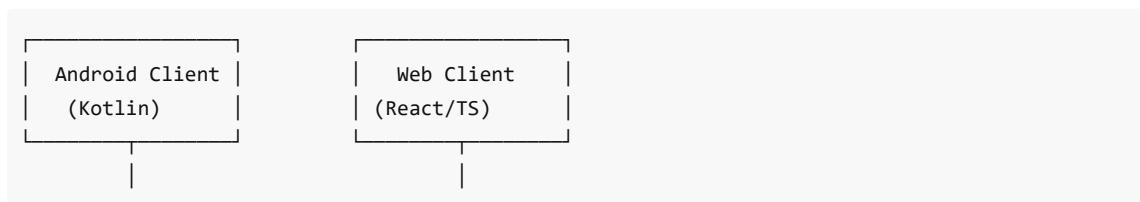
State Machine Pattern

- Strict one-way state transitions: EDITABLE → SUBMITTED → COACHING → COMPLETED
- State transitions are irreversible
- Backend is the only authority for state changes

Event-Driven Pipeline Architecture

- AI pipeline runs asynchronously after game submission
- Background tasks handle long-running operations
- Redis locks prevent concurrent pipeline execution

2.2 System Topology





Trust Boundaries:

- Clients are zero-trust (outside trust boundary)
- Backend services are trusted (inside trust boundary)
- External APIs are untrusted (outbound calls only)

2.3 Component Architecture

Backend Services

- **FastAPI Application** (`backend/api/main.py`): Main API server
- **PostgreSQL**: Primary database (SQLite in development)
- **Redis**: Optional distributed locking and caching
- **AI Providers**: OpenAI, Google Vertex AI (Gemini)
- **Chess Engine Providers**: Stockfish.online, Chess-API.com

Frontend Clients

- **Web Client** (`web/`): React + TypeScript, Tailwind CSS
- **Android Client** (`android/`): Kotlin (structure exists)

Parent Control Interface

- **Web-only** interface for parent governance

- AI tier management
 - Approval workflows
 - Usage visibility
-

3. Detailed Component Design

3.1 Backend Core Components

API Layer

Main Application (`backend/api/main.py`):

- FastAPI app initialization
- Middleware stack:
 - CORS middleware (before auth)
 - Authentication middleware (`AuthMiddleware`)
 - Request logging middleware
- Exception handlers for `ChessCoachError`
- Static file serving for PCI UI
- Health check endpoint

Router Organization:

- `/games` - Game management endpoints
- `/questions` - Question answering endpoints
- `/pci` - Parent Control Interface endpoints

Game Lifecycle Management

Game Service (`backend/api/games/game_service.py`):

- `create_game()` : Creates new game in EDITABLE state
- `get_game()` : Retrieves game with annotations
- `get_games()` : Lists all games
- `delete_game()` : Deletes game
- `_extract_annotations_from_pgn()` : Extracts annotations from PGN

Submission Service (`backend/api/games/submission_service.py`):

- `submit_game()` : Transitions game from EDITABLE to SUBMITTED
- `_requires_approval()` : Checks if parent approval needed
- `_get_valid_approval()` : Retrieves valid approval record
- Enforces state machine rules
- Freezes annotations on submission

Annotation Service (`backend/api/games/annotation_service.py`):

- `add_or_update_annotation()` : Adds/updates annotations
- Enforces EDITABLE state requirement
- Handles annotation freezing

AI Orchestration Pipeline

AI Orchestrator (`backend/api/ai/orchestrator.py`):

- `run_pipeline()` : Main pipeline entry point with Redis locking

- `_execute_pipeline()` : Core pipeline logic
- `_run_analyzer()` : Position analysis and key position selection
- `_generate_socratic_questions()` : Question generation per key position
- Handles graceful degradation if Redis unavailable

Pipeline Stages:

1. Chess Situation Analyzer (internal)
2. Persist 3-5 key positions
3. Socratic Question Loop (per position)
4. Reflection Generation (after all questions answered)

Position Analysis

Position Analyzer (`backend/api/ai/position_analyzer.py`):

- `analyze_position()` : Comprehensive position analysis
- `_calculate_material_balance()` : Material evaluation
- `_detect_tactical_patterns()` : Fork, pin, skewer detection
- `_assess_king_safety()` : King safety scoring
- `_assess_piece_activity()` : Piece activity scoring
- `_analyze_move_quality()` : Move quality vs best move
- `_calculate_criticality_score()` : Multi-factor criticality (0-100)
- `_determine_reason_code()` : OPP_INTENT, THREAT_AWARENESS, TRANSITION
- `select_key_positions()` : Selects 3-5 most critical positions

PositionAnalysis Dataclass:

- Stores comprehensive position data
- Includes engine analysis, positional features, criticality metrics
- Game phase detection (opening/middlegame/endgame)

Question Selection

Intelligent Question Selector (`backend/api/ai/question_selector.py`):

- `select_questions()` : Dynamically selects relevant questions
- Analyzes position characteristics
- Prioritizes questions based on:
 - Threats detected
 - Tactical patterns
 - Evaluation imbalances
 - Move quality
 - Game phase
- Returns 3-5 questions per position (not always 6)
- Always includes REFLECTION at the end

Question Categories:

- `OPP_INTENT` : Opponent's intentions
- `THREAT` : Threat awareness
- `CHANGE` : Position changes
- `WORST PIECE` : Piece activity
- `ALTERNATIVES` : Alternative moves
- `REFLECTION` : Thinking reflection

AI Providers

Chess Engine Provider (backend/api/ai/providers/engine.py):

- `analyze_position()` : Fetches engine analysis
- Priority: Stockfish.online → Chess-API.com v2 → Chess-API.com v1
- Retry logic with exponential backoff
- Fallback values if all APIs fail
- Returns: `{score, best_move, threats, depth}`

Socratic Questioner Provider (backend/api/ai/providers/socratic_questioner.py):

- `generate_question()` : Generates Socratic question
- Supports OpenAI and Google Vertex AI
- Template fallback if no AI configured
- `_build_question_prompt()` : Constructs prompt with EngineTruth
- Enforces contract: neutral, non-evaluative, age-appropriate

Reflection Generator Provider (backend/api/ai/providers/reflection_generator.py):

- `generate_reflection()` : Generates final reflection
- Input: answers by category, player color, question counts
- Output: `{thinking_patterns, missing_elements, habits}` (1-2 habits only)
- Template fallback if no AI configured
- Validates output format

Validators

Orchestrator Validator (backend/api/ai/validators/orchestrator_validator.py):

- `validate_analyzer_output()` : Validates analyzer output
- Checks: key_positions count (1-5), required fields, EngineTruth presence
- Throws on first violation (no auto-repair)

Authentication & Authorization

Auth Middleware (backend/api/auth/middleware.py):

- JWT token validation
- Device binding enforcement
- Role-based access control (CHILD vs PARENT)
- Public routes: `/health`, `/docs`, `/pci-gui`, `/pci-ui`, `/dev/token`
- PCI routes require PARENT role
- Game/question routes require CHILD role

JWT Claims:

```
{
  "sub": "user_id",
  "role": "CHILD | PARENT",
  "device_id": "uuid",
  "exp": 123456
}
```

Parent Control Interface

PCI Router (backend/api/pci/router.py):

- /pci/settings : Get/update AI provider settings
- /pci/approvals : Create approval requests
- /pci/approvals/{id}/decision : Approve/deny requests
- /pci/usage : View AI usage history
- /pci/available-models : List available AI models
- /pci/debug-settings : Debug endpoint for settings

Settings Management:

- Reads from .env file
- Writes updates to .env file
- Masks sensitive values for display
- Supports OpenAI and Google Vertex AI configuration

3.2 Database Design

Schema (backend/api/common/models.py):

Game Model:

```
- id: Integer (PK)
- user_id: String (indexed)
- state: Enum(EDITABLE, SUBMITTED, COACHING, COMPLETED)
- player_color: String
- opponent_name: String
- event: String
- date: DateTime
- time_control: String
- pgn: String
- created_at: DateTime
- updated_at: DateTime
- reflection: JSON (AI-generated)
```

Annotation Model:

```
- id: Integer (PK)
- game_id: Integer (FK → Game.id)
- move_number: Integer
- content: String
- frozen: Boolean (default=False)
```

KeyPosition Model:

```
- id: Integer (PK)
- game_id: Integer (FK → Game.id)
- fen: String
- reason_code: String
- engine_truth: JSON (evaluation, best_move, threats, position_analysis)
- order: Integer
```

Question Model:

```
- id: Integer (PK)
- key_position_id: Integer (FK → KeyPosition.id)
- category: String (OPP_INTENT, THREAT, CHANGE, etc.)
- question_text: String
- answer_text: String (nullable)
- skipped: Boolean (default=False)
- order: Integer
```

ParentApproval Model:

```
- id: Integer (PK)
- game_id: Integer (FK → Game.id)
- tier: String (STANDARD, ADVANCED)
- approved: Boolean (default=False)
- expires_at: DateTime
- used: Boolean (default=False)
- created_at: DateTime
```

SystemSetting Model:

```
- id: Integer (PK)
- key: String (unique, indexed)
- value: String
- updated_at: DateTime
```

State Enforcement Rules:

- `Annotation.frozen = true` iff `Game.state != EDITABLE`
- `Answer` inserts forbidden unless `Game.state == COACHING`
- All state transitions server-validated only

Persistence Layer:

- SQLAlchemy async patterns
- AsyncSession for all database operations
- Transaction management with `async with db.begin()`
- Eager loading with `selectinload()` for relationships

3.3 Frontend Architecture

Web Client Structure (web/src/):

Views:

- `App.tsx` : Main application router
- `GameList.tsx` : List of games
- `CreateGame.tsx` : Game creation
- `GameEntry.tsx` : Game editing and annotation
- `AIProcessing.tsx` : Processing/waiting page
- `GuidedQuestioning.tsx` : Question answering interface

- `FinalReflection.tsx` : Reflection display
- `ParentControlInterface.tsx` : Parent control dashboard

Components:

- `ChessBoard.tsx` : Chess board display
- `MoveNavigator.tsx` : Move navigation controls
- `MoveTree.tsx` : Move tree visualization
- `AnnotationPanel.tsx` : Annotation input
- `GameSubmission.tsx` : Game submission UI

Services:

- `api.ts` : HTTP client with error handling
- `games.ts` : Game API service methods

State Management:

- React hooks for local state
- API service layer for backend communication
- No global state management (stateless frontend)

4. Data Flow & State Management

4.1 Game Lifecycle State Machine

States:

- `EDITABLE` : Moves & annotations allowed
- `SUBMITTED` : Point of no return, annotations frozen, AI triggered
- `COACHING` : Socratic questioning loop active
- `COMPLETED` : Final reflection available

State Transitions:

```
EDITABLE → SUBMITTED → COACHING → COMPLETED
```

Transition Rules:

- Only backend may change state
- Transitions are single-direction and irreversible
- Invalid transitions return HTTP 409
- Submit endpoint is idempotent

Enforcement:

- State checks in all mutation endpoints
- Database constraints (where applicable)
- Service layer validation

4.2 AI Pipeline Data Flow

Input Data:

- PGN string (complete game)
- Annotations (player's thinking)

- Player color
- Game metadata

Pipeline Flow:

1. **Position Collection:** Parse PGN, collect positions after move 10
2. **Engine Analysis:** Analyze each position with chess engine
3. **Position Analysis:** Calculate criticality scores
4. **Key Position Selection:** Select 3-5 most critical positions
5. **Question Generation:** Generate questions per key position
6. **Question Persistence:** Store questions in database
7. **State Transition:** Game state → COACHING

EngineTruth Generation:

- Fetched from Stockfish.online or Chess-API.com
- Includes: evaluation score, best move, threats, depth
- Stored in `KeyPosition.engine_truth` JSON field
- Passed to AI providers for question generation

Question Flow:

- Questions generated per key position
- Dynamically selected based on position characteristics
- Ordered by priority/relevance
- Stored with category and order

Answer Collection:

- Answers submitted via `/questions/{id}/answer`
- Stored in `Question.answer_text`
- Skip option available
- Duplicate answers rejected

Reflection Generation:

- Triggered when all questions answered
- Input: answers by category
- Generated by AI provider
- Stored in `Game.reflection` JSON field
- Game state → COMPLETED

4.3 Question Flow State Machine

Per-Position Flow:

- Questions presented one at a time
- Order determined by intelligent selector
- Skip advances to next question
- Resume continues from last unanswered question

Question Sequence:

- Not fixed order (dynamic selection)
- 3-5 questions per position
- REFLECTION always at end
- Adapts to game phase

Answer Handling:

- One answer per question
 - Skip recorded as `skipped = true`
 - Duplicate answers rejected (HTTP 409)
 - All answers required before reflection
-

5. Security Architecture

5.1 Authentication & Authorization

JWT Token Structure:

- Claims: `sub`, `role`, `device_id`, `exp`
- Algorithm: HS256 (configurable)
- Secret: Loaded from environment

Device Binding:

- `device_id` claim required
- Prevents token reuse on different devices
- Enforced in middleware

Role-Based Access Control:

- `CHILD` role: Game/question endpoints
- `PARENT` role: PCI endpoints only
- Role checked in middleware before route handler

Token Validation:

- Signature verification
- Expiration check
- Required claims validation
- Invalid tokens return HTTP 401

5.2 Network Security

Tailscale Funnel:

- TLS termination at edge
- No direct public IP exposure
- Private network isolation
- Clients never join tailnet

Backend Binding:

- Services bind to `127.0.0.1` only
- No public ports exposed
- Access only through Tailscale

Outbound Connections:

- HTTPS only for external APIs
- No inbound connections accepted
- Firewall rules enforce isolation

5.3 Data Security

AI Credential Management:

- Stored in `.env` file (not in database)
- Never logged or exposed to clients
- Masked in PCI UI display
- Encrypted at rest (file system permissions)

PII Handling:

- No personal data sent to AI providers
- Only chess positions and annotations
- No user identification in AI prompts

Audit Logging:

- State transitions logged
- AI invocation metadata logged (no content)
- Approval grants/denials logged
- Never logged: AI prompts, AI responses, child answers

5.4 Threat Model

STRIDE Analysis:

Spoofing:

- Mitigation: JWT + device binding, parent approval required

Tampering:

- Mitigation: Immutable state machine, server-side enforcement

Repudiation:

- Mitigation: Backend-logged approval events, usage counters

Information Disclosure:

- Mitigation: AI keys encrypted, no PII in AI prompts

Denial of Service:

- Mitigation: Per-game invocation caps, explicit parent approval gates

Elevation of Privilege:

- Mitigation: Separate auth roles, web-only PCI, no parent features in child UI
-

6. AI Integration Architecture

6.1 Hybrid Intelligence Model

EngineTruth Contract:

- Mandatory for all AI invocations
- Includes: evaluation score, best move, threats
- AI MUST trust engine facts as authoritative

- AI MUST NOT independently evaluate positions

AI Role Separation:

- **Chess Situation Analyzer:** Internal only, JSON output
- **Socratic Questioner:** Child-facing, questions only
- **Reflection Generator:** Final output, 1-2 habits only

Hallucination Prevention:

- All AI prompts include EngineTruth
- AI outputs validated against engine facts
- Validators reject contradictions
- No auto-repair or retries

6.2 AI Provider Abstraction

Provider Registry:

- Backend-only mapping: Tier → Provider → Model → Secret
- Clients never see provider names
- Abstracted as "STANDARD", "ADVANCED" tiers

Secret Management:

- Stored in `.env` file
- Loaded on backend startup
- Never exposed to clients
- Masked in PCI UI

Fallback Mechanisms:

- OpenAI → Google Vertex AI → Template
- Template questions/reflections if no AI configured
- Graceful degradation

6.3 AI Contracts

Analyzer Contract:

- Input: PGN + annotations
- Output: JSON with 3-5 key positions
- No natural language
- No evaluation

Socratic Questioner Contract:

- Input: FEN + EngineTruth + category
- Output: Single question string
- Neutral, non-evaluative
- Age-appropriate

Reflection Generator Contract:

- Input: Answers by category
- Output: `{thinking_patterns, missing_elements, habits}`
- 1-2 habits only
- Non-evaluative language

Validator Contracts:

- Output validation before persistence
 - Throws on first violation
 - No auto-repair
-

7. API Design

7.1 REST API Structure

Base URL: `http://localhost:8080` (or Tailscale Funnel URL)

Authentication: Bearer token in Authorization header

Response Format: JSON

Error Handling:

- HTTP status codes
- Error details in response body
- Consistent error format

7.2 Child-Facing APIs

Game Management:

- `POST /games` - Create game
- `GET /games` - List games
- `GET /games/{id}` - Get game
- `DELETE /games/{id}` - Delete game
- `POST /games/{id}/submit` - Submit game for AI processing
- `POST /games/{id}/annotations` - Add/update annotation

Question Endpoints:

- `GET /games/{id}/next-question` - Get next unanswered question
- `POST /questions/{id}/answer` - Answer question

Reflection Endpoints:

- `GET /games/{id}/reflection` - Get final reflection

7.3 Parent Control Interface APIs

Settings:

- `GET /pci/settings` - Get AI provider settings
- `POST /pci/settings` - Update AI provider settings
- `GET /pci/available-models` - List available models
- `GET /pci/debug-settings` - Debug settings

Approvals:

- `POST /pci/approvals` - Create approval request
- `POST /pci/approvals/{id}/decision` - Approve/deny request

Usage:

- `GET /pci/usage` - View AI usage history
-

8. Deployment Architecture

8.1 Infrastructure

Host: Mac Mini (dedicated hardware)

Container Runtime: Docker Desktop

Services:

- FastAPI (Python) - Port 8080 (localhost only)
- PostgreSQL 16 - Internal only
- Redis 7 - Internal only (optional)

Docker Compose (`backend/docker-compose.yml`):

- Service definitions
- Volume mounts
- Environment variables
- Network configuration

8.2 Network Configuration

Tailscale Setup:

- Funnel enabled for public access
- TLS termination at Tailscale edge
- Backend services on private IP
- No public ports exposed

Port Binding:

- API: `127.0.0.1:8080`
- Database: Internal only
- Redis: Internal only

8.3 Environment Configuration

Environment Variables (`.env` file):

- `DATABASE_URL` : Database connection string
- `REDIS_URL` : Redis connection string
- `JWT_SECRET` : JWT signing secret
- `JWT_ALGORITHM` : JWT algorithm (HS256)
- `OPENAI_API_KEY` : OpenAI API key
- `OPENAI_MODEL_NAME` : OpenAI model name
- `GOOGLE_CLOUD_PROJECT` : Google Cloud project ID
- `GOOGLE_CLOUD_LOCATION` : Google Cloud location
- `GOOGLE_APPLICATION_CREDENTIALS_JSON` : Google service account JSON
- `AI_MODEL_NAME` : AI model name

Configuration Loading:

- Pydantic Settings with `.env` file support

- Loaded on backend startup
 - Environment variables override file values
-

9. Error Handling & Recovery

9.1 Error Categories

Validation Errors:

- Invalid state transitions (HTTP 409)
- Missing required fields (HTTP 400)
- Invalid data format (HTTP 422)

State Transition Errors:

- Invalid transition from current state (HTTP 409)
- Game not found (HTTP 404)
- Already in target state (HTTP 409)

AI Provider Errors:

- API failures (logged, fallback to template)
- Timeout errors (logged, fallback)
- Invalid output (validator throws)

Network Errors:

- Redis connection failures (graceful degradation)
- External API timeouts (retry with backoff)
- Database connection errors (HTTP 500)

9.2 Recovery Mechanisms

Partial Progress Preservation:

- Completed answers persist
- Key positions saved even if pipeline fails
- Questions generated persist

Resume from Last Checkpoint:

- Resume from last unanswered question
- State preserved in database
- No data loss on failure

Fallback Strategies:

- Template questions if AI fails
- Template reflection if AI fails
- Pipeline continues without Redis lock

User Notification:

- Child sees neutral interruption message
 - Parent receives failure notification
 - Errors logged for debugging
-

10. Performance & Scalability

10.1 Performance Considerations

Database Query Optimization:

- Eager loading with `selectinload()`
- Indexed foreign keys
- Efficient queries with proper joins

AI API Rate Limiting:

- Sequential position analysis (not parallel)
- Timeout protection (60s per position)
- Retry with exponential backoff

Caching Strategies:

- Redis for distributed locks (optional)
- No response caching (always fresh data)

Async Processing:

- Background tasks for AI pipeline
- Async database operations
- Non-blocking API calls

10.2 Scalability Limits

Single-Instance Deployment:

- Designed for single Mac Mini
- No horizontal scaling
- Vertical scaling only

Resource Constraints:

- Limited by Mac Mini hardware
- Database size (SQLite/PostgreSQL)
- Memory for concurrent requests

AI API Quotas:

- OpenAI rate limits
- Google Vertex AI quotas
- Stockfish.online rate limits

11. Compliance & Governance

11.1 Parent Approval System

Approval Triggers:

- Non-default tier usage
- Second or later AI run on same game

Approval Lifecycle:

- Request created
- Parent approves/denies
- Approval expires (time-bound)
- Approval marked as used (single-use)

Usage Limits:

- Per-game invocation caps
- Explicit parent approval gates
- No automatic retries

11.2 Compliance Rules

AI Visibility Rules:

- AI invisible before submission
- No hints or suggestions pre-submission
- AI only visible after SUBMITTED state

Question Ordering:

- Dynamic selection (not fixed)
- REFLECTION always at end
- Order based on position characteristics

State Machine Enforcement:

- Backend-only state transitions
- Immutable transitions
- No rollback

Audit Requirements:

- State transitions logged
- Approval events logged
- AI invocation metadata logged
- Never logged: prompts, responses, answers

12. Implementation Details

12.1 Key Algorithms

Position Criticality Scoring (`backend/api/ai/position_analyzer.py:416-463`):

- Multi-factor scoring (0-100)
- Factors: eval swing, tactical patterns, threats, king safety, move quality
- Weighted combination

Question Selection Algorithm (`backend/api/ai/question_selector.py:48-228`):

- Priority calculation per category
- Factors: threats, tactics, eval imbalance, move quality, game phase
- Top 3-4 questions selected, REFLECTION always included

EngineTruth Generation (`backend/api/ai/providers/engine.py:19-280`):

- Priority: Stockfish.online → Chess-API.com v2 → Chess-API.com v1

- Retry logic with exponential backoff
- Fallback values if all fail

12.2 Code Patterns

Async/Await Patterns:

- All database operations async
- AI provider calls async
- Background tasks async

Error Handling Patterns:

- Try/except with specific exceptions
- Logging before raising
- Graceful degradation

Logging Patterns:

- Structured logging with context
- Log levels: DEBUG, INFO, WARNING, ERROR
- Request/response logging

Testing Patterns:

- Unit tests for services
- Integration tests for API endpoints
- Test fixtures in `conftest.py`

13. Diagrams & Visualizations

13.1 System Architecture Diagram

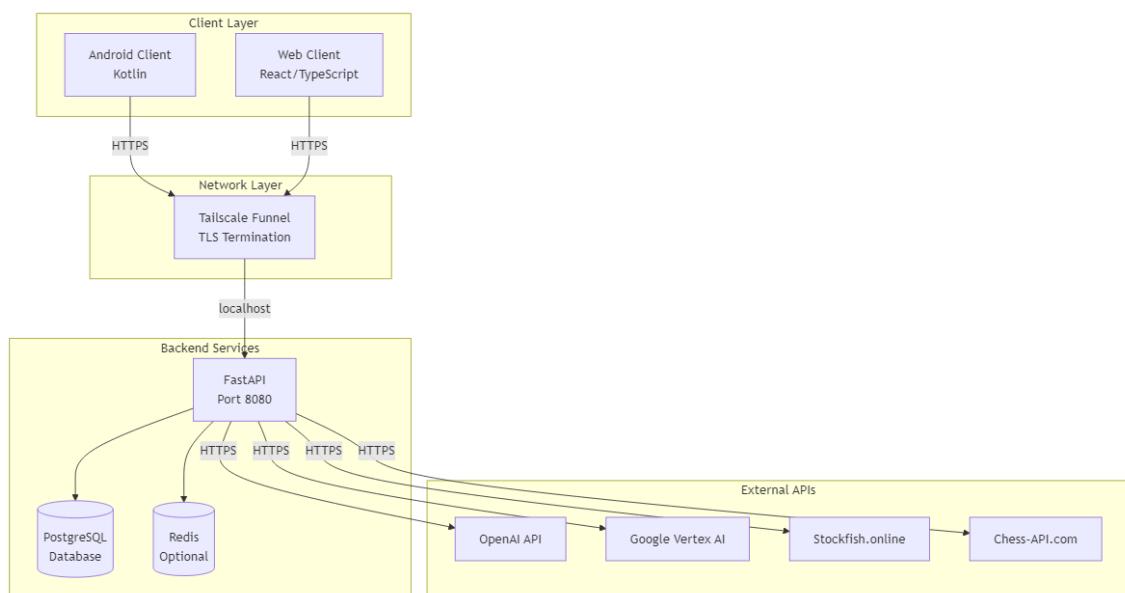


Diagram 1 rendered from Mermaid code

13.2 Game Lifecycle State Machine

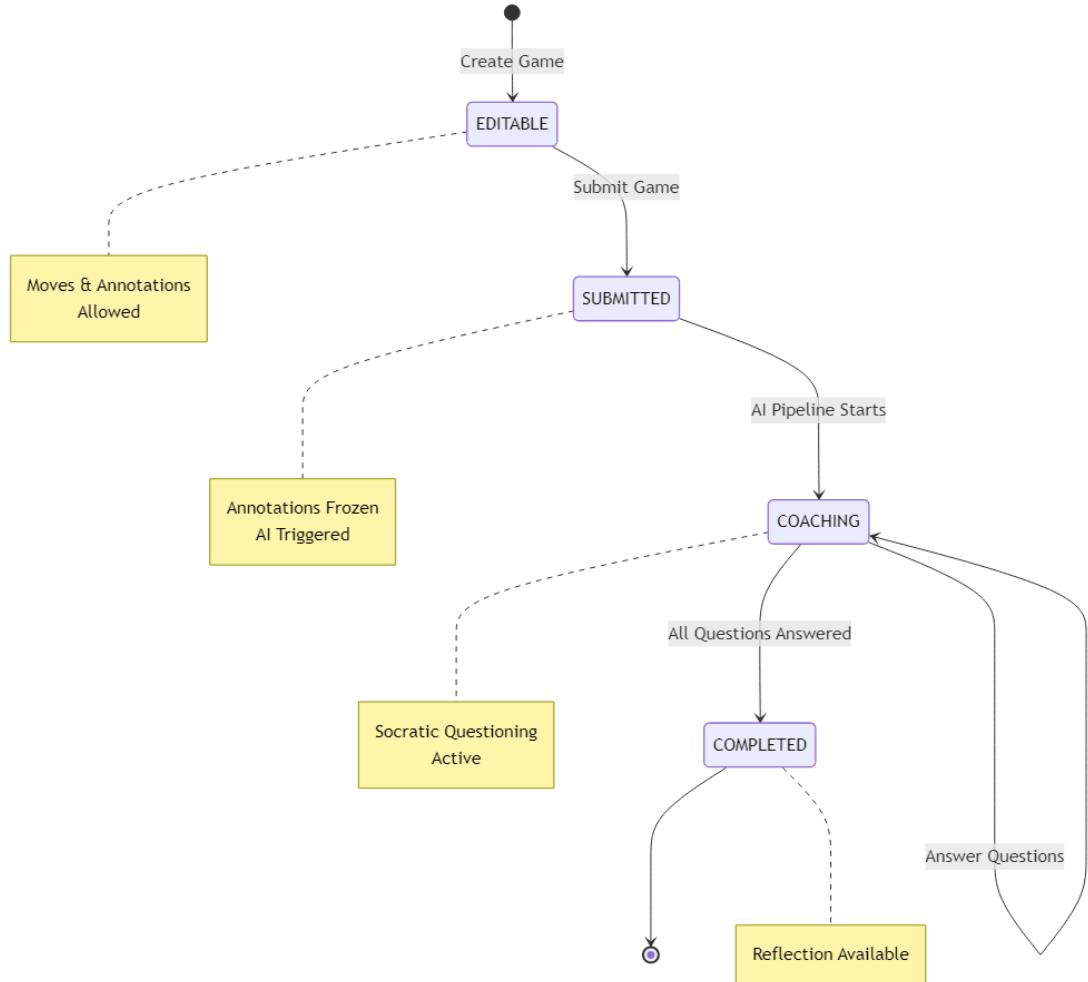


Diagram 2 rendered from Mermaid code

13.3 AI Pipeline Flow

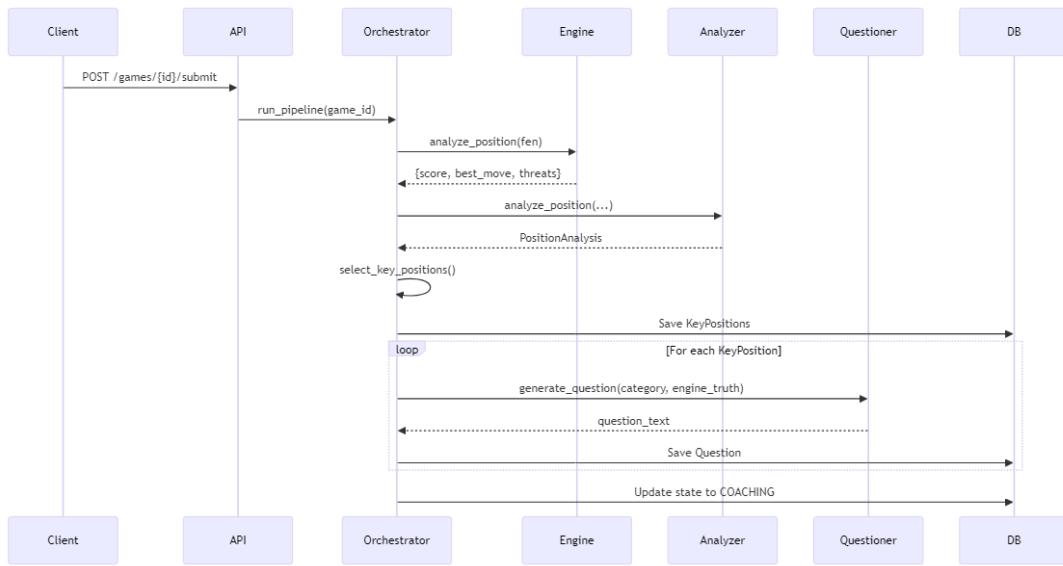


Diagram 3 rendered from Mermaid code

13.4 Question Answering Flow

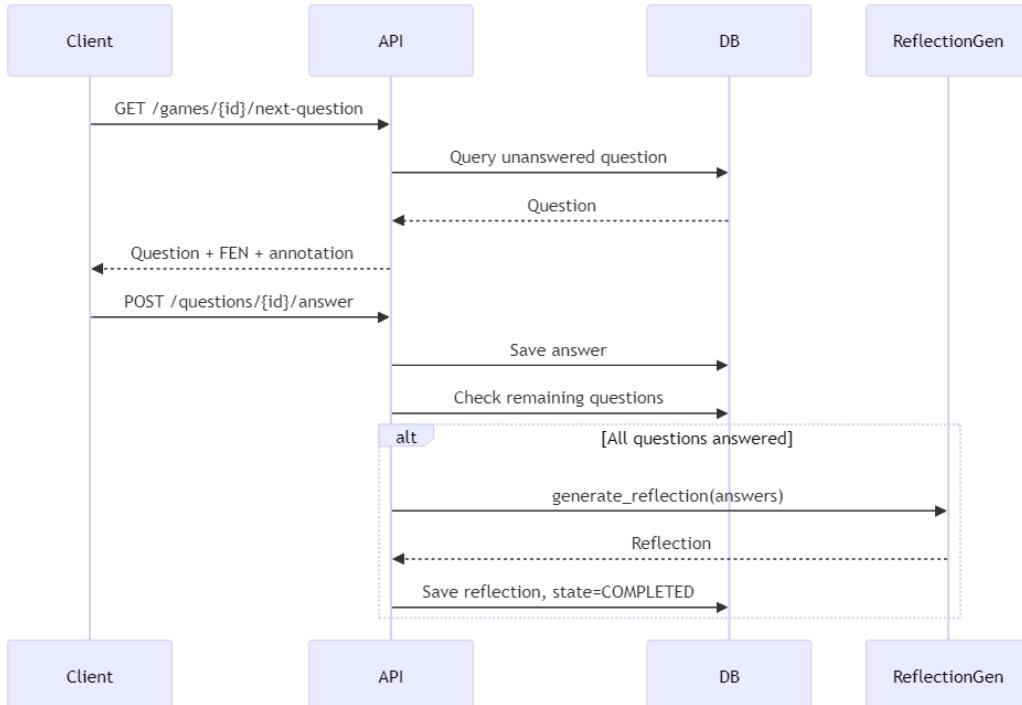


Diagram 4 rendered from Mermaid code

13.5 Game Creation and Annotation Flow

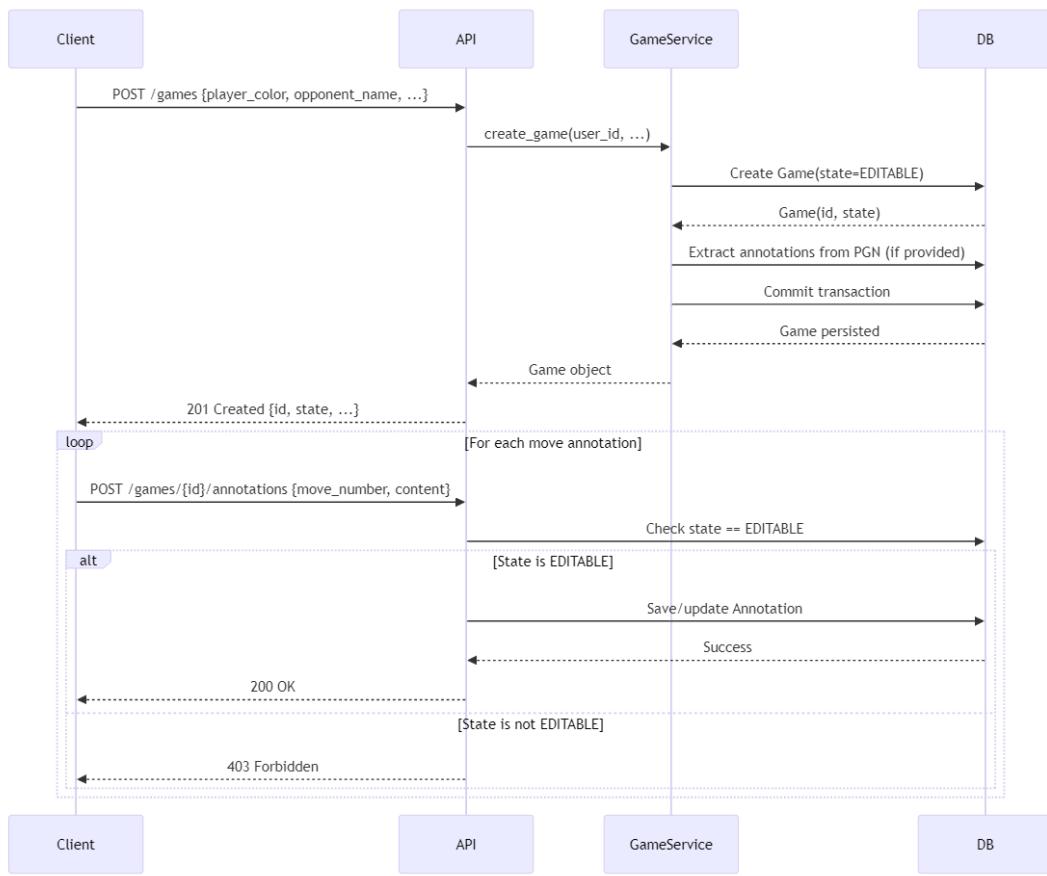


Diagram 5 rendered from Mermaid code

13.6 Parent Approval Workflow

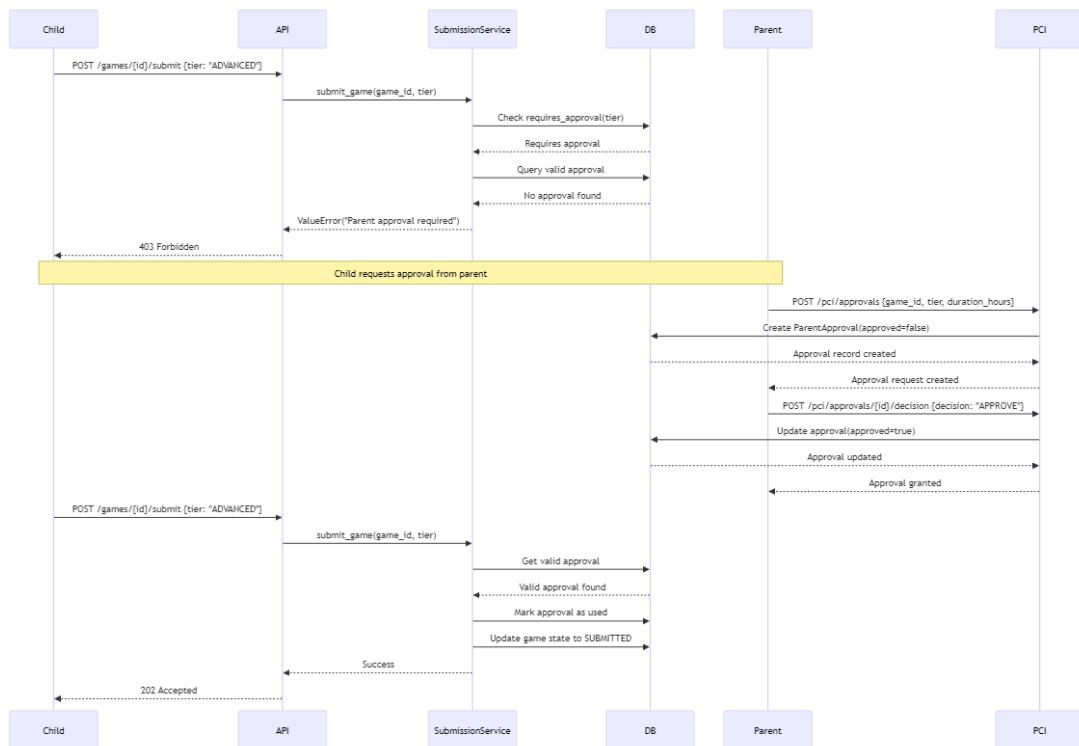


Diagram 6 rendered from Mermaid code

13.7 Error Recovery and Fallback Flow

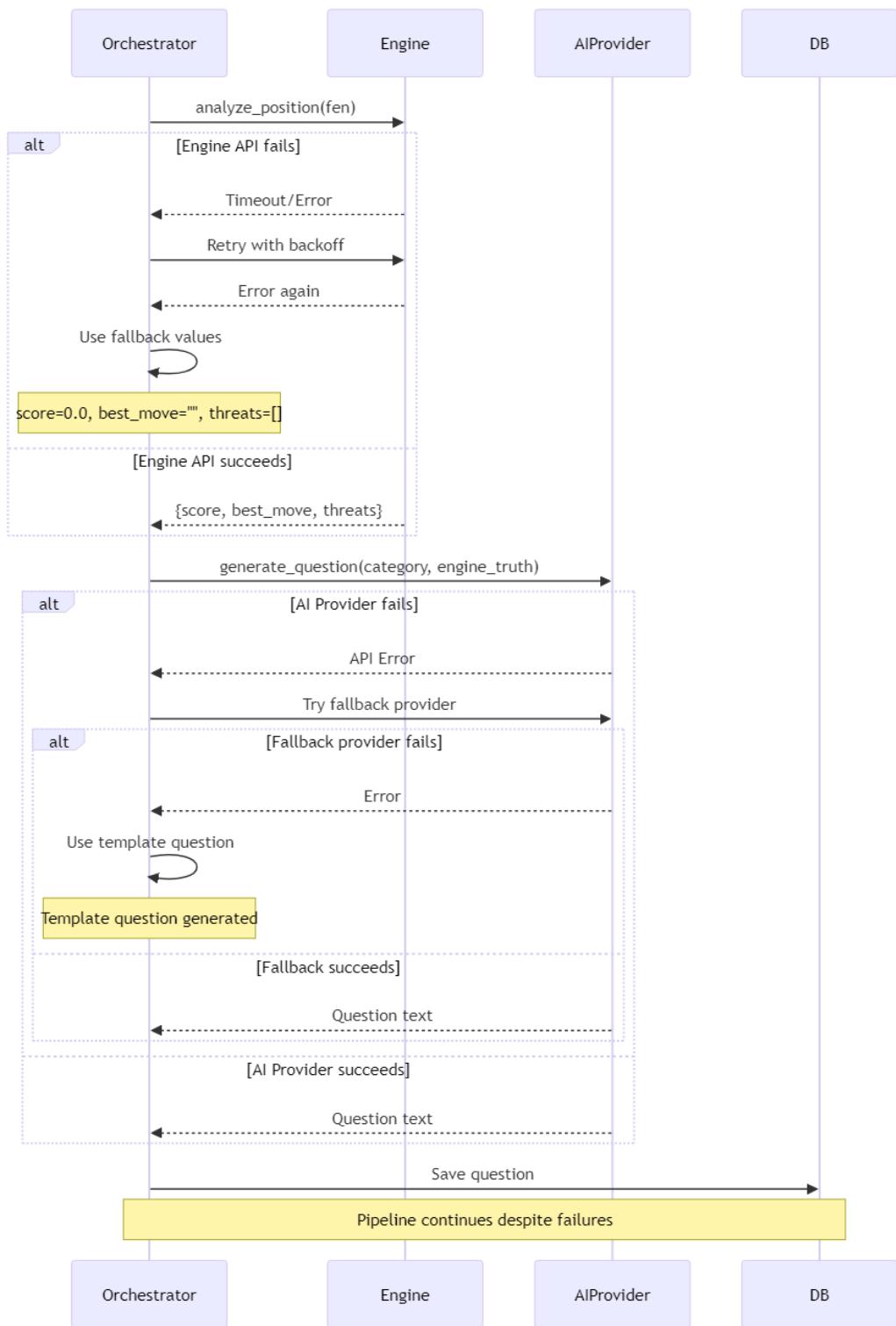


Diagram 7 rendered from Mermaid code

13.8 Reflection Generation Sequence

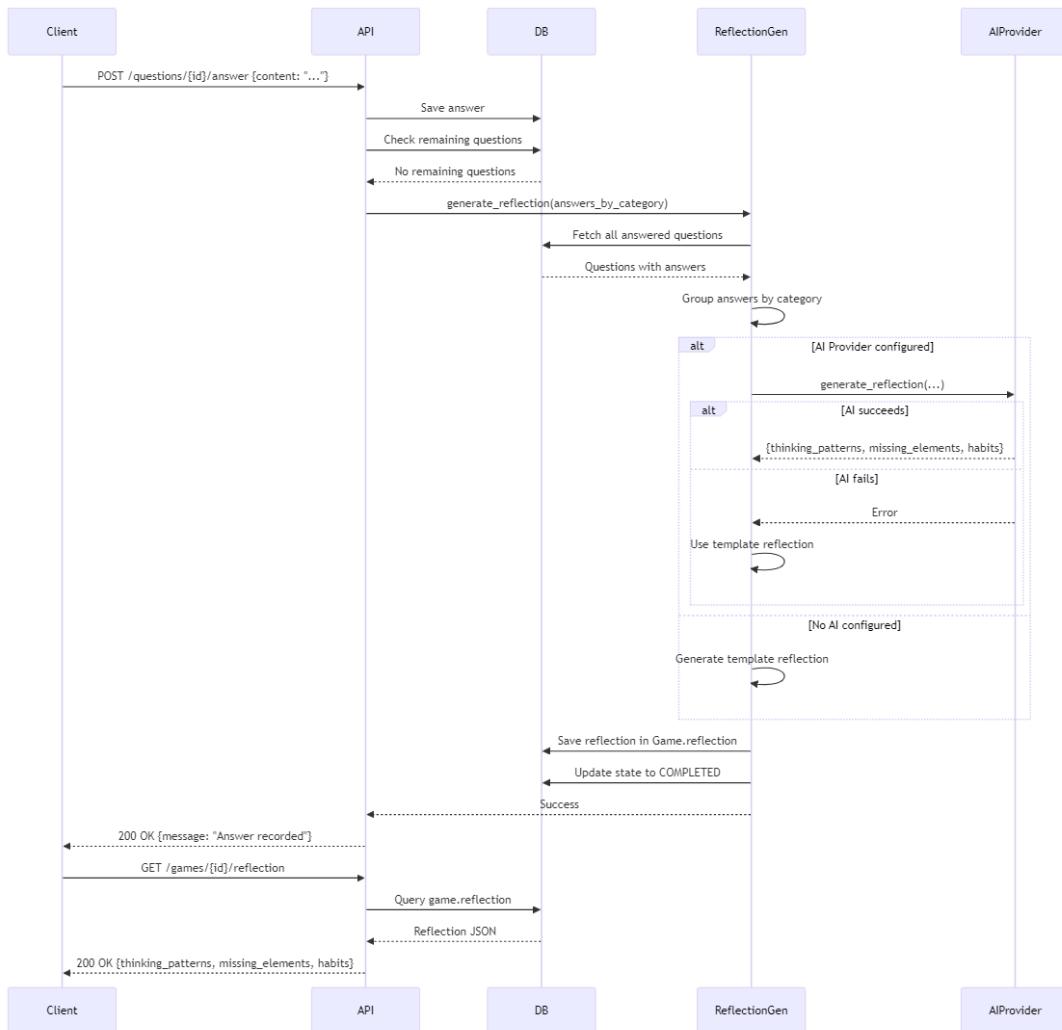


Diagram 8 rendered from Mermaid code

13.9 Component Interaction Diagram

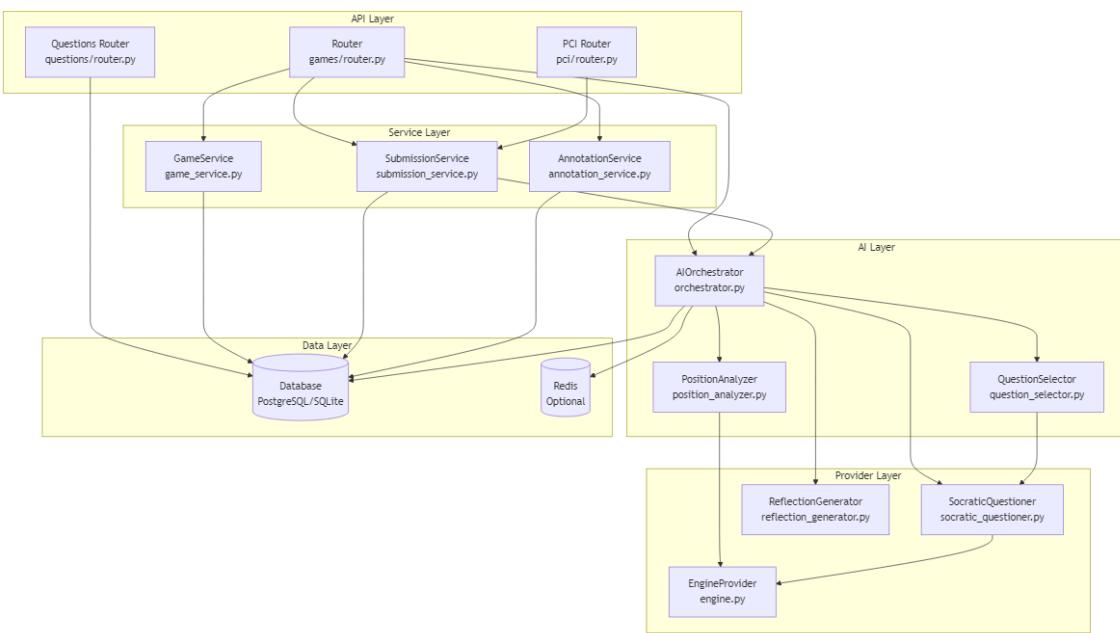
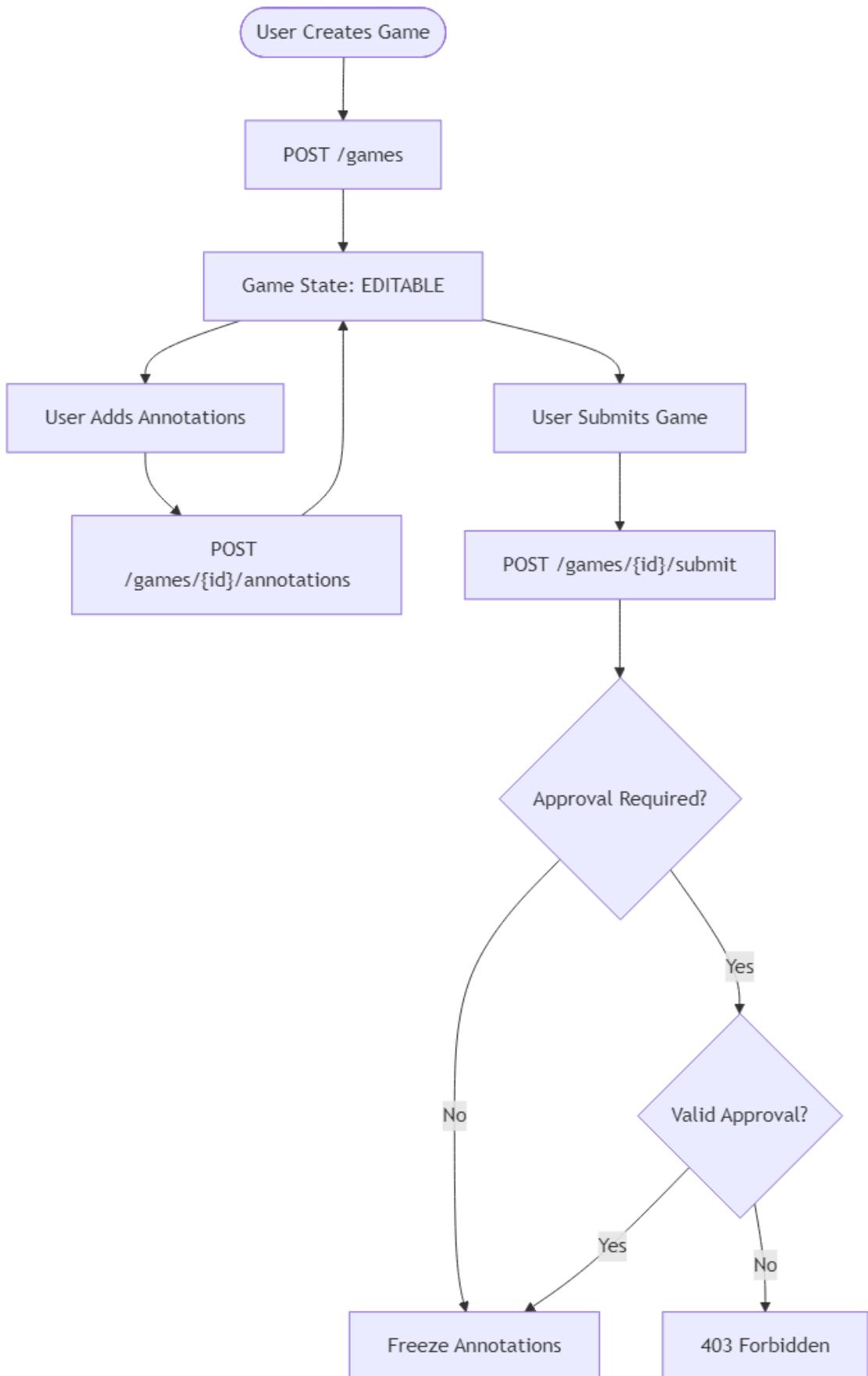
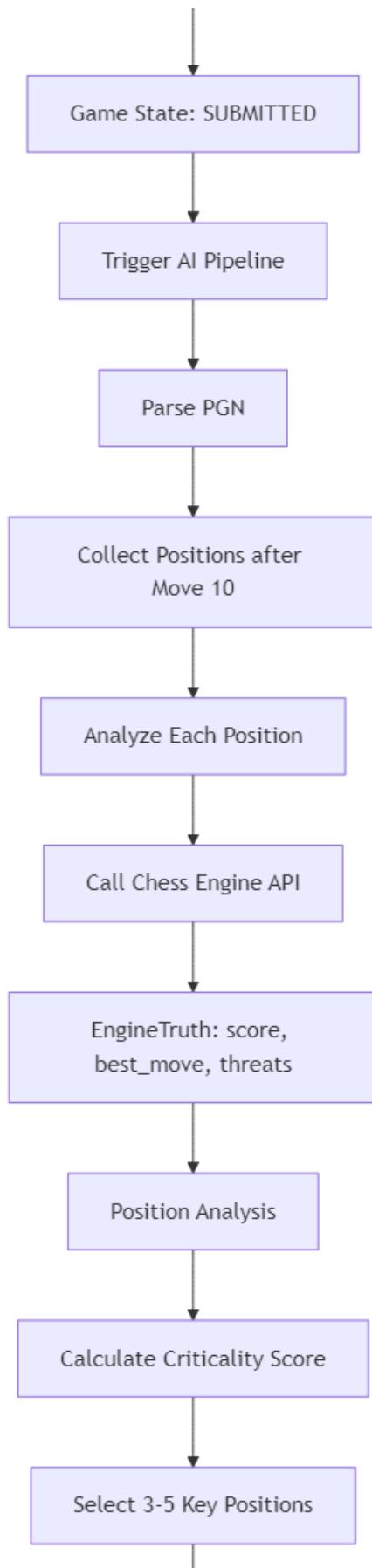
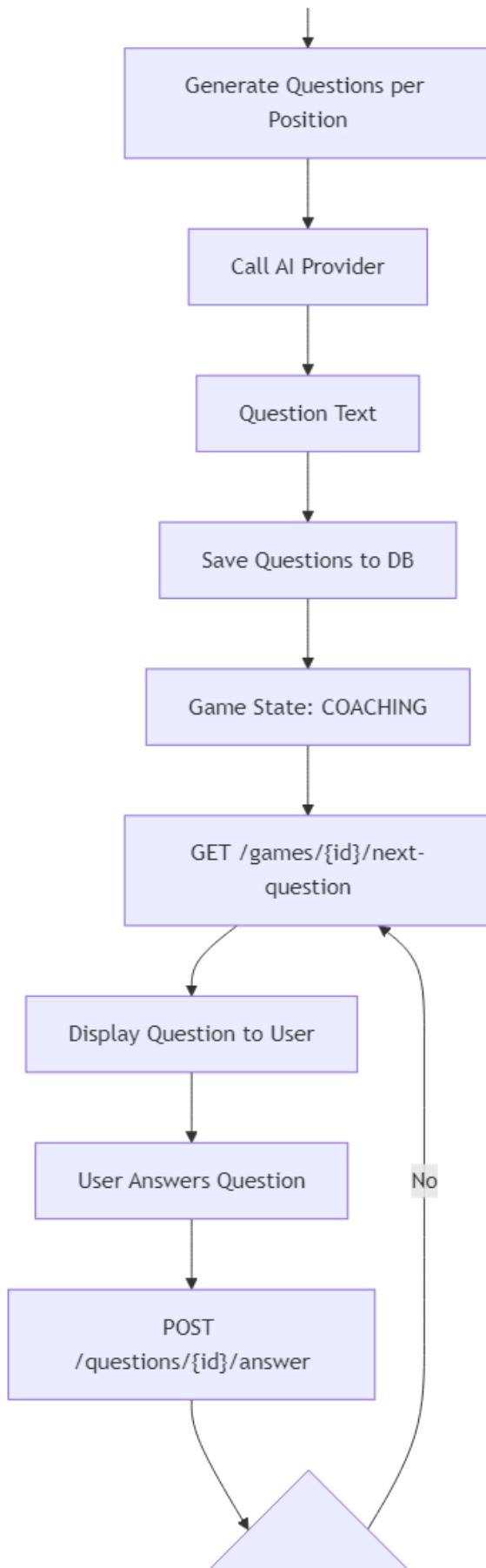


Diagram 9 rendered from Mermaid code

13.10 Data Flow Diagram







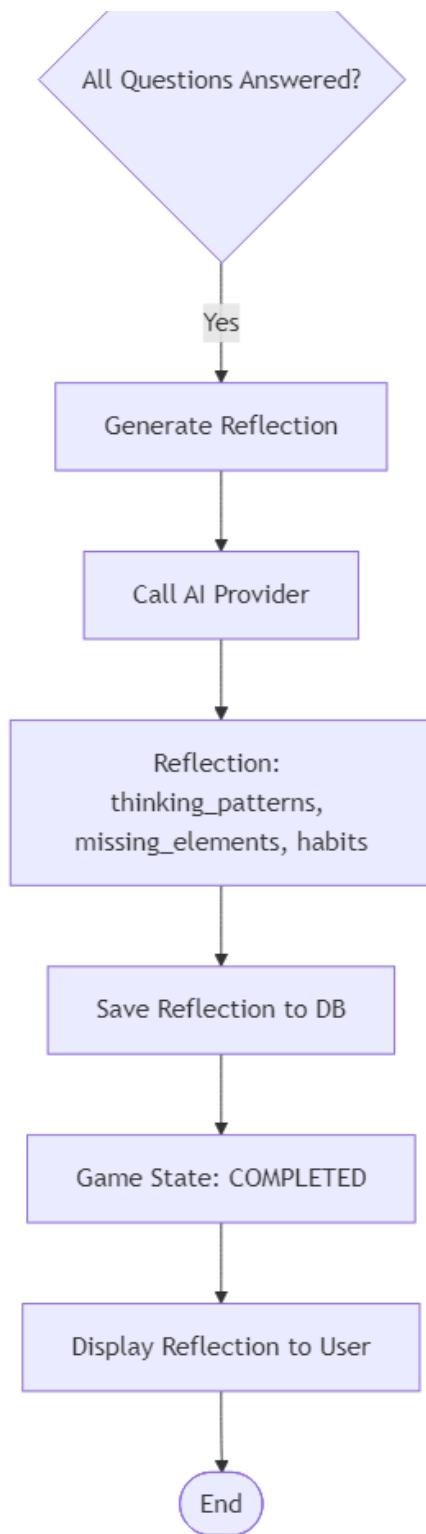


Diagram 10 rendered from Mermaid code

13.11 Deployment Diagram

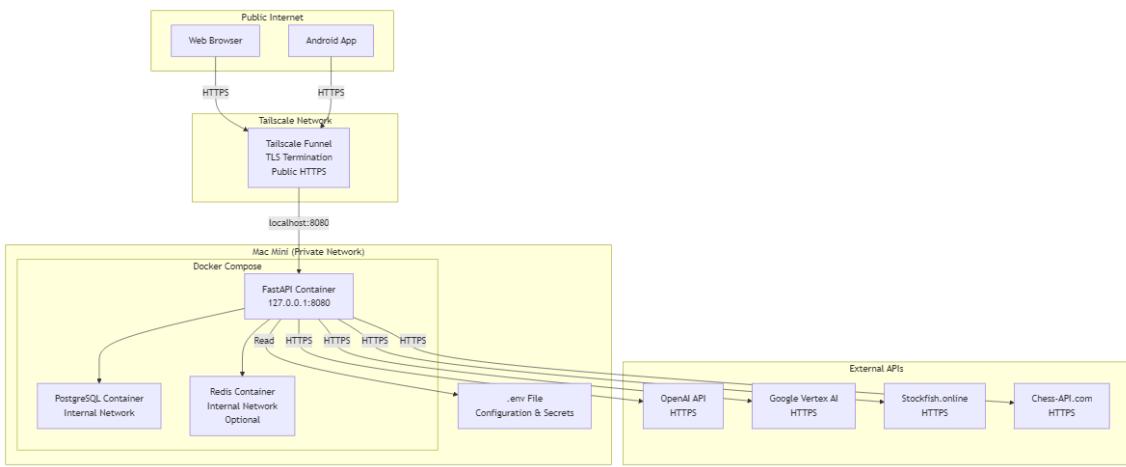


Diagram 11 rendered from Mermaid code

13.12 Security Boundary Diagram

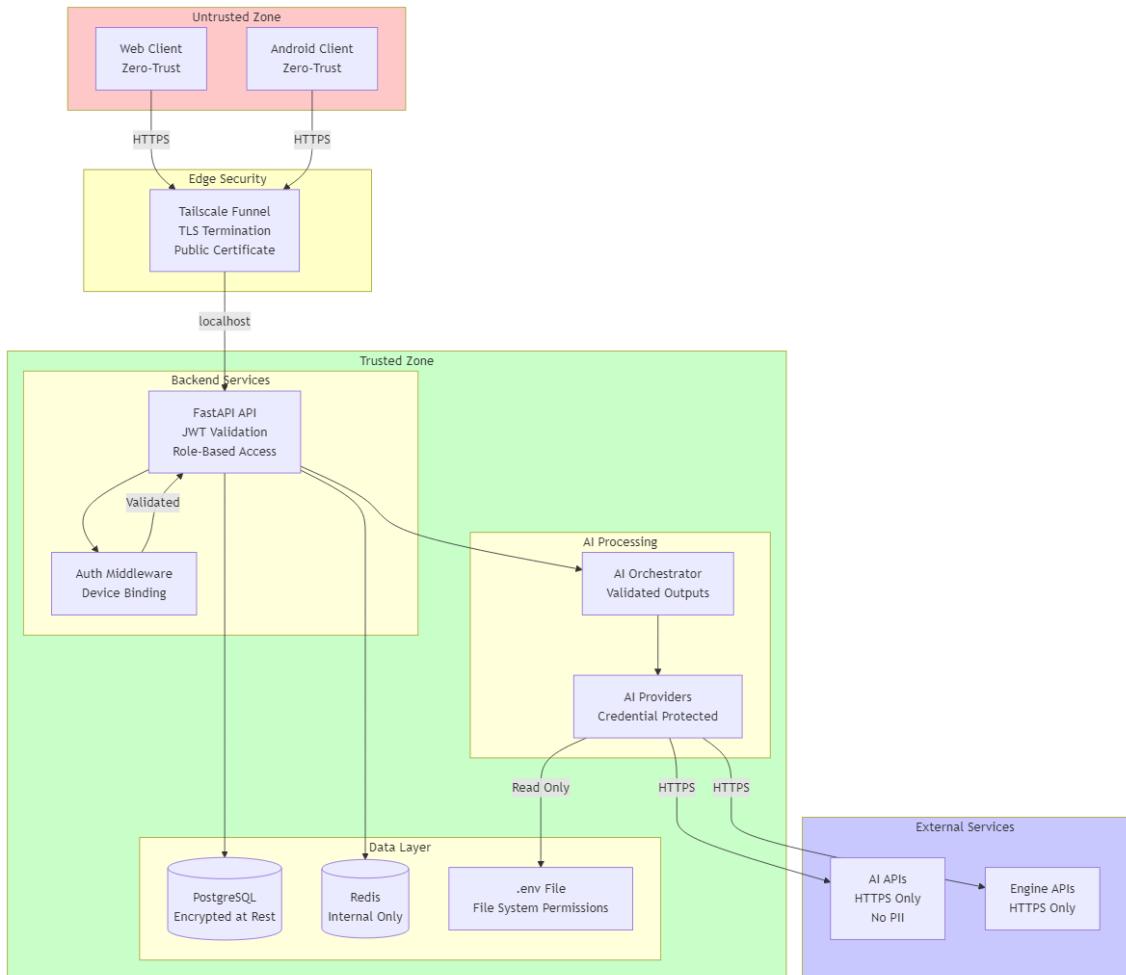


Diagram 12 rendered from Mermaid code

14. Appendices

14.1 Complete API Reference

14.1.1 Game Management Endpoints

POST /games - Create a new game

Request:

```
{  
  "player_color": "WHITE",  
  "opponent_name": "John Doe",  
  "event": "Local Tournament",  
  "date": "2026-01-03T10:00:00Z",  
  "time_control": "15+10",  
  "pgn": "1. e4 e5 2. Nf3 Nc6 ..."}
```

Response (201 Created):

```
{  
  "id": 1,  
  "user_id": "user123",  
  "state": "EDITABLE",  
  "player_color": "WHITE",  
  "opponent_name": "John Doe",  
  "event": "Local Tournament",  
  "date": "2026-01-03T10:00:00Z",  
  "time_control": "15+10",  
  "pgn": "1. e4 e5 2. Nf3 Nc6 ...",  
  "created_at": "2026-01-03T10:00:00Z",  
  "updated_at": "2026-01-03T10:00:00Z"}
```

GET /games - List all games

Response (200 OK):

```
[  
  {  
    "id": 1,  
    "opponent_name": "John Doe",  
    "state": "EDITABLE",  
    "created_at": "2026-01-03T10:00:00Z"  
  },  
  {  
    "id": 2,  
    "opponent_name": "Jane Smith",  
    "state": "COACHING",  
    "created_at": "2026-01-02T15:30:00Z"
```

```
    }
]
```

GET /games/{id} - Get game details

Response (200 OK):

```
{
  "id": 1,
  "user_id": "user123",
  "state": "EDITABLE",
  "player_color": "WHITE",
  "opponent_name": "John Doe",
  "annotations": [
    {
      "id": 1,
      "move_number": 10,
      "content": "I was looking for tactics here",
      "frozen": false
    }
  ]
}
```

POST /games/{id}/submit - Submit game for AI processing

Request:

```
{
  "pgn": "1. e4 e5 2. Nf3 Nc6 ...",
  "tier": "STANDARD"
}
```

Response (202 Accepted):

```
{
  "message": "Game submitted successfully",
  "state": "SUBMITTED"
}
```

Error Response (403 Forbidden) - Parent approval required:

```
{
  "detail": "Parent approval required for ADVANCED tier or repeat run"
}
```

Error Response (409 Conflict) - Invalid state:

```
{
  "detail": "Invalid transition from COACHING to SUBMITTED"
}
```

```
}
```

POST /games/{id}/annotations - Add or update annotation

Request:

```
{
  "move_number": 10,
  "content": "I was looking for tactics here"
}
```

Response (200 OK):

```
{
  "message": "Annotation saved"
}
```

Error Response (403 Forbidden) - Game not editable:

```
{
  "detail": "Cannot add annotations to non-editable game"
}
```

14.1.2 Question Endpoints

GET /games/{id}/next-question - Get next unanswered question

Response (200 OK):

```
{
  "id": 5,
  "key_position_id": 2,
  "category": "THREAT",
  "question_text": "What threats do you see in this position?",
  "order": 0,
  "fen": "rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1",
  "original_annotation": "I was looking for tactics here"
}
```

Response (204 No Content) - All questions completed:

```
{
  "message": "All questions completed"
}
```

POST /questions/{id}/answer - Answer a question

Request:

```
{  
  "content": "I see a fork threat on my knight",  
  "skipped": false  
}
```

Or skip:

```
{  
  "content": "",  
  "skipped": true  
}
```

Response (200 OK):

```
{  
  "message": "Answer recorded"  
}
```

Error Response (409 Conflict) - Already answered:

```
{  
  "detail": "Question already answered"  
}
```

14.1.3 Reflection Endpoints

GET /games/{id}/reflection - Get final reflection

Response (200 OK):

```
{  
  "thinking_patterns": [  
    "You focused on tactical opportunities in several positions",  
    "You recognized threats when they were immediate"  
  ],  
  "missing_elements": [  
    "Long-term strategic planning",  
    "Opponent's positional intentions"  
  ],  
  "habits": [  
    "Before each move, ask: 'What is my opponent trying to accomplish?'",  
    "Scan the entire board for piece activity, not just the center"  
  ]  
}
```

Error Response (400 Bad Request) - Not ready:

```
{  
  "detail": "Reflection not available yet. The game analysis is still in progress."  
}
```

```
}
```

14.1.4 Parent Control Interface Endpoints

GET /pci/settings - Get AI provider settings

Response (200 OK):

```
{
  "settings": {
    "GOOGLE_CLOUD_PROJECT": "gen-lang-client-0397559410",
    "GOOGLE_CLOUD_LOCATION": "us-central1",
    "AI_MODEL_NAME": "gemini-1.5-flash"
  },
  "masked_settings": {
    "OPENAI_API_KEY": "sk-...abcd",
    "GOOGLE_APPLICATION_CREDENTIALS_JSON": ""
  },
  "type": "service_account",
  "project_id": "...",
  "client_email": "...",
  "private_key": "",
  "configured": {
    "GOOGLE_CLOUD_PROJECT": true,
    "OPENAI_API_KEY": false,
    "GOOGLE_APPLICATION_CREDENTIALS_JSON": true
  }
}
```

POST /pci/settings - Update AI provider settings

Request:

```
{
  "settings": {
    "OPENAI_API_KEY": "sk-...",
    "OPENAI_MODEL_NAME": "gpt-4o-mini"
  }
}
```

Response (200 OK):

```
{
  "message": "Settings updated in .env file. Restart backend to apply changes."
}
```

POST /pci/approvals - Create approval request

Request:

```
{
  "game_id": 1,
  "tier": "ADVANCED",
```

```
        "duration_hours": 24
    }
```

Response (201 Created):

```
{
  "id": 1,
  "game_id": 1,
  "tier": "ADVANCED",
  "approved": false,
  "expires_at": "2026-01-04T10:00:00Z",
  "used": false,
  "created_at": "2026-01-03T10:00:00Z"
}
```

POST /pci/approvals/{id}/decision - Approve or deny request

Request:

```
{
  "decision": "APPROVE"
}
```

Response (200 OK):

```
{
  "message": "Approval approved"
}
```

GET /pci/usage - View AI usage history

Response (200 OK):

```
{
  "usage": [
    {
      "id": 1,
      "game_id": 1,
      "tier": "STANDARD",
      "approved": true,
      "used": true,
      "expires_at": "2027-01-03T10:00:00Z",
      "created_at": "2026-01-03T10:00:00Z"
    }
  ]
}
```

14.1.5 Authentication

All endpoints (except public routes) require authentication via Bearer token:

Header:

```
Authorization: Bearer <JWT_TOKEN>
```

JWT Token Structure:

```
{
  "sub": "user123",
  "role": "CHILD",
  "device_id": "device-uuid-1234",
  "exp": 1735920000
}
```

Error Response (401 Unauthorized):

```
{
  "detail": "Missing or invalid token"
}
```

Error Response (403 Forbidden) - Wrong role:

```
{
  "detail": "Parent access required"
}
```

14.2 Database Schema Reference

14.2.1 Entity Relationship Diagram

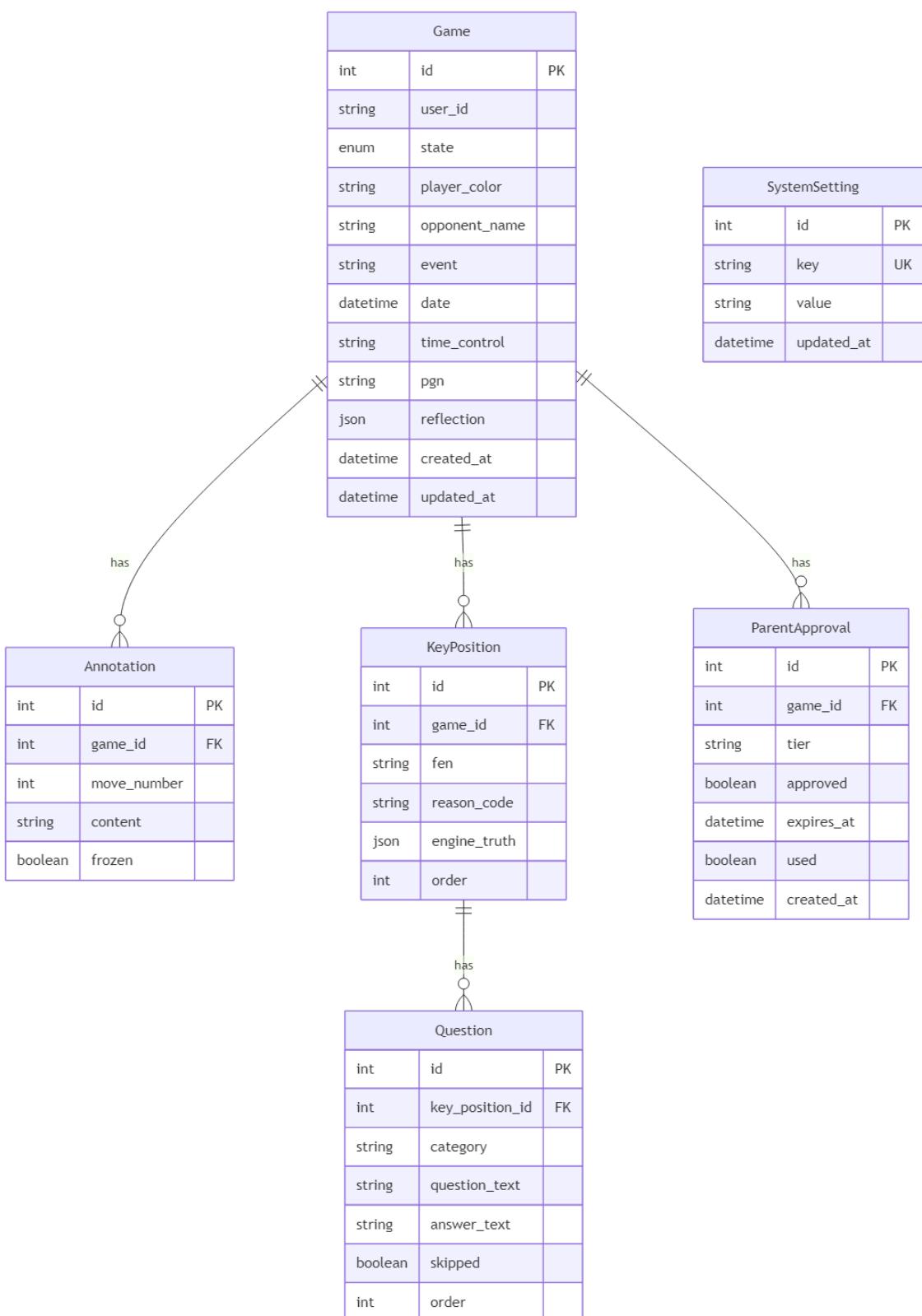


Diagram 13 rendered from Mermaid code

14.2.2 Table Relationships

- **Game → Annotation:** One-to-many (cascade delete)
- **Game → KeyPosition:** One-to-many (cascade delete)
- **Game → ParentApproval:** One-to-many
- **KeyPosition → Question:** One-to-many (cascade delete)

14.2.3 Indexes

- Game.user_id : Indexed for user queries
- Game.state : Used for state filtering
- Annotation.game_id : Foreign key index
- Annotation.frozen : Used for state enforcement
- KeyPosition.game_id : Foreign key index
- Question.key_position_id : Foreign key index
- Question.answer_text : Used for finding unanswered questions
- SystemSetting.key : Unique index

14.3 Configuration Reference

14.3.1 Sample .env File

```
# Database Configuration
DATABASE_URL=sqlite+aiosqlite:///./chess_coach.db
# For PostgreSQL: DATABASE_URL=postgresql+asyncpg://user:password@localhost/chess_coach

# Redis Configuration (Optional)
REDIS_URL=redis://localhost:6379/0

# Authentication
JWT_SECRET=your-secret-key-here-change-in-production
JWT_ALGORITHM=HS256
ALLOW_DEV_TOKEN_ENDPOINT=0

# OpenAI Configuration (Optional)
OPENAI_API_KEY=sk-...
OPENAI_MODEL_NAME=gpt-4o-mini

# Google Vertex AI Configuration (Optional)
GOOGLE_CLOUD_PROJECT=your-project-id
GOOGLE_CLOUD_LOCATION=us-central1
GOOGLE_APPLICATION_CREDENTIALS_JSON=
>{"type":"service_account","project_id":"...","private_key_id":"...","private_key":"...","client
AI_MODEL_NAME=gemini-1.5-flash

# Alternative: Use file path instead of JSON
# GOOGLE_APPLICATION_CREDENTIALS=/path/to/service-account.json
```

14.3.2 Environment-Specific Configuration

Development:

```
DATABASE_URL=sqlite+aiosqlite:///./chess_coach.db
REDIS_URL=redis://localhost:6379/0
ALLOW_DEV_TOKEN_ENDPOINT=1
JWT_SECRET=dev-secret-key
```

Production:

```
DATABASE_URL=postgresql+asyncpg://user:password@localhost/chess_coach
REDIS_URL=redis://localhost:6379/0
ALLOW_DEV_TOKEN_ENDPOINT=0
JWT_SECRET=<strong-random-secret>
```

14.3.3 Tailscale Setup Guide

1. **Install Tailscale** on Mac Mini
2. **Enable Funnel:**

```
tailscale funnel 443
```

3. **Configure Backend:**

- o Backend binds to 127.0.0.1:8080
- o Tailscale forwards public HTTPS to localhost:8080

4. **Access:**

- o Public URL: https://your-machine.tailscale.ts.net
- o Clients connect via HTTPS
- o TLS terminates at Tailscale edge

14.4 Deployment Step-by-Step Guide

14.4.1 Prerequisites

- Mac Mini with macOS
- Docker Desktop installed
- Tailscale account and client
- Python 3.10+ (for local development)
- Node.js 18+ (for frontend build)

14.4.2 Installation Steps

1. **Clone Repository:**

```
git clone <repository-url>
cd chess-in-one-ai-coach
```

2. **Backend Setup:**

```
cd backend
cp .env.example .env
# Edit .env with your configuration
```

3. **Install Dependencies:**

```
pip install -r requirements.txt
```

4. Initialize Database:

```
python scripts/init_db.py
```

5. Frontend Build:

```
cd ../web  
npm install  
npm run build
```

6. Start Services (Development):

```
cd ../backend  
uvicorn api.main:app --reload --port 8080
```

7. Start Services (Production with Docker):

```
cd backend  
docker-compose up -d
```

14.4.3 Configuration Walkthrough

1. Configure Database:

- Edit `DATABASE_URL` in `.env`
- For production, use PostgreSQL

2. Configure AI Providers:

- Add OpenAI API key OR
- Add Google Vertex AI credentials
- Set model names

3. Configure Authentication:

- Set strong `JWT_SECRET`
- Set `ALLOW_DEV_TOKEN_ENDPOINT=0` in production

4. Configure Tailscale:

- Install Tailscale client
- Enable funnel: `tailscale funnel 443`
- Note the public URL

14.4.4 Verification and Testing

1. Health Check:

```
curl http://localhost:8080/health
```

```
# Should return: {"status": "healthy"}
```

2. **Generate Dev Token** (dev only):

```
curl http://localhost:8080/dev/token?role=CHILD
```

3. **Test Game Creation:**

```
curl -X POST http://localhost:8080/games \
-H "Authorization: Bearer <token>" \
-H "Content-Type: application/json" \
-d '{"player_color": "WHITE", "opponent_name": "Test"}'
```

4. **Check Logs:**

```
tail -f docs/debug/backend.log
```

14.5 Code Examples

14.5.1 Position Criticality Scoring

Location: backend/api/ai/position_analyzer.py:416-463

```
def _calculate_criticality_score(
    self,
    eval_score: float,
    previous_eval: Optional[float],
    material_balance: float,
    tactical_patterns: List[str],
    king_safety_score: float,
    move_quality_score: float,
    threats: List[str],
    depth: int
) -> float:
    """
    Calculate overall criticality score (0-100).
    Higher score = more critical position for coaching.
    """
    score = 0.0

    # 1. Evaluation swing (transition) - up to 30 points
    if previous_eval is not None:
        eval_change = abs(eval_score - previous_eval)
        if eval_change > 0.5:
            score += min(30.0, eval_change * 15.0)

    # 2. Tactical patterns - up to 25 points
    if tactical_patterns:
        score += min(25.0, len(tactical_patterns) * 8.0)
```

```

# 3. Threats detected - up to 20 points
if threats:
    score += min(20.0, len(threats) * 5.0)

# 4. King safety issues - up to 15 points
if king_safety_score < 50.0:
    score += (50.0 - king_safety_score) / 50.0 * 15.0

# 5. Move quality (suboptimal moves are more critical) - up to 10 points
if move_quality_score < 0.7:
    score += (0.7 - move_quality_score) / 0.7 * 10.0

# 6. Large evaluation imbalance - up to 10 points
if abs(eval_score) > 1.0:
    score += min(10.0, (abs(eval_score) - 1.0) * 5.0)

return min(100.0, score)

```

14.5.2 State Machine Enforcement

Location: backend/api/games/submission_service.py:13-87

```

async def submit_game(self, game_id: int, pgn: str, tier: str) -> dict:
    """Submit a game for AI processing."""
    # Fetch game
    result = await self.db.execute(select(Game).where(Game.id == game_id))
    game = result.scalar_one_or_none()
    if not game:
        raise ValueError("Game not found")

    # Check idempotency
    if game.state == GameState.SUBMITTED:
        return {"message": "Game already submitted", "state": game.state}

    # Validate transition - ENFORCEMENT POINT
    if game.state != GameState.EDITABLE:
        raise ValueError(f"Invalid transition from {game.state} to SUBMITTED")

    # Check parent approval requirement
    requires_approval = await self._requires_approval(game_id, tier)
    if requires_approval:
        approval = await self._get_valid_approval(game_id, tier)
        if not approval:
            raise ValueError(f"Parent approval required for {tier} tier or repeat run")
        approval.used = True

    # Update PGN and freeze annotations
    game.pgn = pgn
    await self.db.execute(
        update(Annotation)
        .where(Annotation.game_id == game_id)
        .values(frozen=True)
    )

```

```

    )

# Transition to SUBMITTED - IMMUTABLE TRANSITION
game.state = GameState.SUBMITTED
await self.db.commit()

return {"message": "Game submitted successfully", "state": game.state}

```

14.5.3 AI Provider Integration with Fallback

Location: backend/api/ai/providers/socratic_questioner.py:33-81

```

async def generate_question(
    self,
    category: str,
    engine_truth: Dict[str, Any],
    student_annotation: Optional[str] = None,
    reason_code: Optional[str] = None
) -> str:
    """
    Generate a Socratic question for a specific category.
    Implements fallback chain: OpenAI → Vertex AI → Template
    """

    # Try OpenAI first if configured
    if self.openai_api_key:
        try:
            return await self._generate_with_openai(
                category, engine_truth, student_annotation, reason_code
            )
        except Exception as e:
            logger.warning(f"OpenAI question generation failed: {e}")

    # Fall back to Google Vertex AI if configured
    if self.google_credentials and self.google_project:
        try:
            return await self._generate_with_vertex(
                category, engine_truth, student_annotation, reason_code
            )
        except Exception as e:
            logger.warning(f"Vertex AI question generation failed: {e}")

    # Fallback to template if no AI configured
    logger.warning("No AI provider configured, using template question")
    return self._generate_template_question(
        category, engine_truth, student_annotation, reason_code
    )

```

14.5.4 Validation Logic

Location: backend/api/ai/validators/orchestrator_validator.py:1-23

```

def validate_analyzer_output(output: dict):
    """
    Implements PHASE 4: AI Role Contracts & Validators.
    Validators MUST throw on first violation.
    """

    if "key_positions" not in output:
        raise ValueError("Missing key_positions in analyzer output")

    kp_count = len(output["key_positions"])
    if not (1 <= kp_count <= 5):
        raise ValueError(f"Invalid number of key positions: {kp_count}")

    for kp in output["key_positions"]:
        required = ["fen", "reason_code", "engine_truth"]
        for field in required:
            if field not in kp:
                raise ValueError(f"Missing field {field} in key position")

    # Implementation Spec 9.1: Accept EngineTruth as immutable input
    engine_truth = kp["engine_truth"]
    if "best_move" not in engine_truth or "score" not in engine_truth:
        raise ValueError("EngineTruth missing best_move or score")

```

14.6 Troubleshooting Guide

Common Issues:

1. **AI Provider Not Configured:** Check `.env` file, restart backend
2. **Frontend Stuck:** Rebuild frontend, check browser console
3. **Pipeline Not Completing:** Check logs, verify Redis connection
4. **State Transition Errors:** Verify game state, check logs

Debug Endpoints:

- `/health` : Health check
- `/pci/debug-settings` : Settings debug
- `/dev/token` : Dev token generation (dev only)

14.7 Known Limitations

1. **Single-Instance:** No horizontal scaling
2. **Redis Optional:** System works without Redis (no locking)
3. **AI Fallback:** Template questions/reflections if AI fails
4. **Rate Limits:** Subject to external API rate limits
5. **SQLite in Dev:** PostgreSQL required for production

15. Remaining Work & Enhancements

15.1 Diagrams Section (Complete)

Completed:

- System architecture diagram (Section 13.1)

- Game lifecycle state machine (Section 13.2)
- AI pipeline flow (Section 13.3)
- Question answering flow (Section 13.4)
- Game creation and annotation flow (Section 13.5)
- Parent approval workflow (Section 13.6)
- Error recovery and fallback flow (Section 13.7)
- Reflection generation sequence (Section 13.8)
- Component interaction diagram (Section 13.9)
- Data flow diagram (Section 13.10)
- Deployment diagram (Section 13.11)
- Security boundary diagram (Section 13.12)

15.2 Appendices (Complete)

Completed:

- Complete API reference with request/response examples (Section 14.1)
- Database schema with ERD diagram (Section 14.2)
- Configuration examples and Tailscale setup (Section 14.3)
- Deployment step-by-step guide (Section 14.4)
- Code examples for key algorithms (Section 14.5)
- Troubleshooting guide (Section 14.6)
- Known limitations (Section 14.7)

15.3 Code References (Complete)

Completed:

- File paths referenced throughout document
- Function names documented
- Algorithm references with line numbers (Section 12.1)
- Code snippets for key algorithms (Section 14.5)
- State machine enforcement code examples
- AI provider integration examples
- Validation logic examples

16. Testing Strategy

16.1 Overview

The system implements a comprehensive testing strategy with multiple layers to ensure reliability, security, and maintainability. See `docs/test/TEST_STRATEGY.md` for complete details.

Testing Objectives:

- Reliability: Ensure core business logic works correctly
- Security: Validate parent approval workflows and authentication
- Maintainability: Provide regression protection
- Performance: Identify bottlenecks in AI pipeline and database operations

16.2 Test Pyramid

Distribution:

- **Unit Tests:** 75% of test suite

- **Integration Tests:** 20% of test suite
- **End-to-End Tests:** 5% of test suite

16.3 Testing Frameworks

Backend (backend/tests/):

- **Framework:** pytest with pytest-asyncio
- **Coverage Tool:** pytest-cov (target: 80%+ coverage)
- **Mocking:** unittest.mock
- **Database:** SQLite in-memory for tests
- **Test Files:**
 - test_games.py : Game state machine, CRUD operations
 - test_parent_approval.py : Approval workflow validation
 - test_ai_orchestrator.py : AI pipeline orchestration
 - conftest.py : Shared fixtures and test database setup

Frontend (web/src/):

- **Framework:** Jest (with create-react-app)
- **Testing Library:** @testing-library/react, @testing-library/jest-dom
- **Mocking:** jest.mock for API calls
- **Test Files:** Component tests (.test.tsx files)

16.4 Test Categories

Unit Tests (75%)

Backend Unit Tests:

- Individual functions and methods
- Business logic validation
- Error condition handling
- Edge cases and boundary conditions

Example (backend/tests/test_games.py:11-22):

```
@pytest.mark.asyncio
async def test_create_game_sets_editable_state(self, db_session):
    """Test that new games are created in EDITABLE state."""
    game = Game(
        user_id="test_user",
        state=GameState.EDITABLE,
        player_color="WHITE"
    )
    db_session.add(game)
    await db_session.commit()
    assert game.state == GameState.EDITABLE
```

Frontend Unit Tests:

- React component rendering
- User interaction simulation
- State management

- Form validation

Integration Tests (20%)

Database Integration:

- Database transactions
- Service layer interactions
- Data persistence and retrieval
- Foreign key relationships

API Integration:

- FastAPI route testing
- Request/response validation
- Authentication middleware
- Error response formatting

End-to-End Tests (5%)

Critical User Journeys:

- Complete game creation to completion workflow
- Parent approval process
- AI coaching pipeline execution

16.5 Test Execution

Local Development:

```
# Backend tests
cd backend
pytest tests/ -v --cov=api --cov-report=html

# Frontend tests
cd web
npm test -- --coverage --watchAll=false
```

Coverage Requirements:

- Backend: 80%+ line coverage
- Business Logic: 90%+ coverage for services
- API Routes: 100% coverage for route handlers
- Frontend: 80%+ component coverage

16.6 Test Data Management

Test Fixtures (`backend/tests/conftest.py`):

- `test_engine` : Creates test database engine
- `db_session` : Provides test database session
- In-memory SQLite for fast, isolated tests

Test Factories:

- `GameFactory`: Creates test games
- `ParentApprovalFactory`: Creates test approvals

- Consistent test data across tests

16.7 Critical Test Cases

1. **Parent Approval Bypass:** Multiple test cases ensuring approval requirements
2. **Game State Corruption:** State machine validation tests
3. **AI Pipeline Failure:** Orchestrator error handling tests
4. **Authentication Bypass:** JWT validation tests
5. **Data Loss:** Database transaction tests

Reference: See `docs/test/TEST_STRATEGY.md` for complete testing strategy and `docs/test/TEST_RESULTS.md` for test execution results.

17. Monitoring and Observability

17.1 Logging Strategy

Logging Configuration (`backend/api/main.py:19-29`):

```
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S'
)
```

Log Levels:

- **DEBUG:** Detailed diagnostic information (pipeline steps, lock acquisition)
- **INFO:** General informational messages (pipeline start, state changes)
- **WARNING:** Warning messages (missing data, fallbacks, expected errors)
- **ERROR:** Error conditions (pipeline failures, unexpected exceptions)

Log Filtering:

- SQLAlchemy engine logs filtered to WARNING and above
- Health check requests not logged
- Expected 400 errors logged as DEBUG (reflection not ready)

Log Locations:

- Development: Console output
- Production: `docs/debug/backend.log` (configurable)
- Log rotation: Manual (consider automated rotation for production)

17.2 Structured Logging

Log Context:

- Request IDs (not currently implemented, recommended for production)
- User IDs (from JWT claims)
- Game IDs (in relevant log messages)
- Operation context (e.g., `[PIPELINE]` , `[SUBMIT_GAME]`)

Log Format:

```
2026-01-03 10:00:00 - api.games.router - INFO - [SUBMIT_GAME] Starting submission for game_id=1, tier=STANDARD
```

17.3 Key Logging Points

Game Lifecycle:

- Game creation ([CREATE_GAME])
- Game submission ([SUBMIT_GAME])
- State transitions
- Annotation freezing

AI Pipeline:

- Pipeline start/completion ([PIPELINE])
- Position analysis ([ANALYZER])
- Question generation ([GENERATE_QUESTIONS])
- Reflection generation ([GENERATE_REFLECTION])

Error Conditions:

- AI provider failures
- Redis connection failures
- Database errors
- Validation failures

17.4 Metrics and Monitoring

Current State: Basic logging only, no metrics collection

Recommended Metrics (Future Enhancement):

- **Request Metrics:** Request count, latency, error rate per endpoint
- **Pipeline Metrics:** Pipeline execution time, success rate, failure reasons
- **AI Metrics:** AI provider response times, success rates, fallback usage
- **Database Metrics:** Query performance, connection pool usage
- **System Metrics:** CPU, memory, disk usage

Monitoring Tools (Recommended):

- **Application Metrics:** Prometheus + Grafana
- **Log Aggregation:** ELK Stack (Elasticsearch, Logstash, Kibana) or Loki
- **APM:** Application Performance Monitoring tool
- **Uptime Monitoring:** External health check service

17.5 Alerting Configuration

Current State: No automated alerting

Recommended Alerts (Future Enhancement):

- **Critical Errors:** Pipeline failures, database connection errors
- **Performance Degradation:** High latency, slow queries
- **Resource Exhaustion:** High memory/CPU usage
- **Security Events:** Authentication failures, suspicious activity
- **Availability:** Service downtime, health check failures

Alert Channels (Recommended):

- Email notifications
- Slack/Discord webhooks
- PagerDuty for critical alerts
- SMS for production incidents

17.6 Performance Monitoring

Key Performance Indicators:

- API response times (p50, p95, p99)
- AI pipeline execution time
- Database query performance
- External API latency (Stockfish, OpenAI, Gemini)

Monitoring Approach:

- Log timing information for critical operations
- Track slow queries
- Monitor external API response times
- Profile CPU/memory usage during peak loads

18. Backup and Recovery Procedures

18.1 Database Backup Strategy

PostgreSQL Backup (Production):

Automated Backups:

```
# Daily full backup
pg_dump -h localhost -U postgres chess_coach > backup_$(date +%Y%m%d).sql

# Or using pg_dump with compression
pg_dump -h localhost -U postgres -Fc chess_coach > backup_$(date +%Y%m%d).dump
```

Backup Schedule:

- **Full Backup:** Daily at 2 AM
- **Retention:** 30 days of daily backups
- **Weekly Archive:** Keep weekly backups for 12 weeks
- **Monthly Archive:** Keep monthly backups for 12 months

Backup Storage:

- Local: Mac Mini external drive
- Remote: Cloud storage (encrypted)
- Verification: Test restore monthly

SQLite Backup (Development):

```
# Simple file copy
cp backend/chess_coach.db backend/chess_coach.db.backup
```

18.2 Configuration Backup

Critical Files to Backup:

- .env file (encrypted backup, no secrets in version control)
- docker-compose.yml
- Database migration files (backend/db/migrations/)
- Tailscale configuration

Backup Procedure:

```
# Backup configuration
tar -czf config_backup_${date +%Y%m%d}.tar.gz \
  backend/.env \
  backend/docker-compose.yml \
  backend/db/migrations/
```

Storage: Encrypted archive in secure location

18.3 Disaster Recovery Plan

Recovery Scenarios:

1. Database Corruption:

- Restore from most recent backup
- Verify data integrity
- Replay any transactions since backup

2. Complete System Failure:

- Restore database from backup
- Restore configuration files
- Reconfigure Tailscale
- Verify service health

3. Data Loss:

- Identify last known good backup
- Restore database
- Notify users of potential data loss
- Document recovery actions

Recovery Time Objectives (RTO):

- **Critical:** 4 hours (database corruption)
- **Major:** 24 hours (complete system failure)
- **Minor:** 48 hours (configuration issues)

Recovery Point Objectives (RPO):

- **Maximum Data Loss:** 24 hours (daily backups)
- **Target RPO:** 1 hour (with transaction logs)

18.4 Data Retention Policies

Game Data:

- **Active Games:** Retained indefinitely
- **Completed Games:** Retained for 2 years
- **Archived Games:** Compressed storage after 1 year

Logs:

- **Application Logs:** 90 days
- **Error Logs:** 1 year
- **Audit Logs:** 7 years (compliance)

Backups:

- **Daily Backups:** 30 days
- **Weekly Backups:** 12 weeks
- **Monthly Backups:** 12 months

Cleanup Procedures:

```
# Archive old games (manual or scheduled)
# Delete old logs (log rotation)
# Remove expired backups
```

18.5 Backup Verification

Verification Steps:

1. Monthly restore test to separate environment
2. Verify data integrity after restore
3. Test application functionality with restored data
4. Document verification results

Automated Verification (Recommended):

- Checksum verification of backup files
- Automated restore tests
- Data integrity checks

19. Performance Tuning Guide

19.1 Database Optimization

Query Optimization:

Indexes (Current):

- Game.user_id : Indexed for user queries
- Game.state : Used for state filtering
- Annotation.game_id : Foreign key index
- KeyPosition.game_id : Foreign key index
- Question.key_position_id : Foreign key index

Additional Indexes (Recommended):

```
-- For finding unanswered questions
CREATE INDEX idx_question_unanswered ON questions(key_position_id, answer_text, skipped)
```

```

WHERE answer_text IS NULL AND skipped = FALSE;

-- For game state queries
CREATE INDEX idx_game_state_user ON games(state, user_id);

-- For approval lookups
CREATE INDEX idx_approval_valid ON parent_approvals(game_id, tier, approved, used,
expires_at);

```

Query Patterns:

- Use `selectinload()` for eager loading relationships
- Avoid N+1 queries
- Use database transactions efficiently
- Batch operations where possible

Connection Pooling:

- SQLAlchemy connection pool configured in `database.py`
- Default pool size: 5 connections
- Adjust based on load: `pool_size=10, max_overflow=20`

19.2 API Performance Tuning

Async Operations:

- All database operations use `async/await`
- Background tasks for long-running operations (AI pipeline)
- Non-blocking API calls

Response Caching (Future Enhancement):

- Cache game list responses (short TTL)
- Cache reflection responses (longer TTL)
- Invalidate on updates

Request Optimization:

- Minimize payload sizes
- Use pagination for large lists
- Compress responses (`gzip`)

Rate Limiting (Recommended):

- Per-user rate limits
- Per-endpoint rate limits
- Protect against abuse

19.3 AI Pipeline Optimization

Position Analysis:

- Current: Sequential analysis (safe, avoids rate limits)
- Optimization: Batch analysis with rate limiting
- Cache engine results for identical positions

Question Generation:

- Generate questions in parallel per key position
- Cache template questions
- Batch AI provider calls (if supported)

Timeout Management:

- Position analysis: 60s timeout per position
- AI provider calls: 30s timeout
- Overall pipeline: 10 minute timeout

19.4 Caching Strategies

Current State: No caching implemented

Recommended Caching:

Redis Caching (If Redis Available):

- Cache engine analysis results (key: FEN string)
- Cache game metadata (short TTL)
- Cache reflection responses (longer TTL)

Application-Level Caching:

- In-memory cache for frequently accessed data
- LRU cache for position analysis
- TTL-based expiration

Cache Invalidation:

- Invalidate on game updates
- Invalidate on state transitions
- Manual cache clear endpoint for admin

19.5 Resource Optimization

Memory Management:

- Monitor memory usage during pipeline execution
- Clear large objects after use
- Use generators for large datasets

CPU Optimization:

- Profile CPU-intensive operations
- Optimize position analysis algorithms
- Consider multiprocessing for parallel analysis

Network Optimization:

- Connection pooling for external APIs
- Retry with exponential backoff
- Circuit breaker pattern for failing services

19.6 Performance Monitoring

Key Metrics to Track:

- API response times (p50, p95, p99)

- Database query execution times
- AI pipeline execution time
- External API latency
- Memory and CPU usage

Profiling Tools:

- Python: cProfile, py-spy
- Database: EXPLAIN ANALYZE for queries
- Application: APM tools

Performance Baselines:

- Document current performance
- Set performance targets
- Track performance regressions

20. Security Hardening Checklist

20.1 Authentication & Authorization

JWT Security:

- Strong secret key (not placeholder)
- Token expiration enforced
- Device binding implemented
- Role-based access control
- **⚠️ TODO:** Implement token refresh mechanism
- **⚠️ TODO:** Implement token revocation list

Password Security (If Added):

- Minimum 12 characters
- Complexity requirements
- Password hashing (bcrypt, Argon2)
- Account lockout after failed attempts

20.2 Network Security

Tailscale Configuration:

- TLS termination at edge
- No public IP exposure
- Private network isolation
- **⚠️ TODO:** Enable Tailscale ACLs for fine-grained access
- **⚠️ TODO:** Regular Tailscale key rotation

Firewall Rules:

- Backend binds to localhost only
- No inbound public ports
- **⚠️ TODO:** Configure macOS firewall rules
- **⚠️ TODO:** Disable unnecessary services

20.3 Data Security

Encryption:

- HTTPS for all external communications
- TLS for database connections (if remote)
- **⚠️ TODO:** Encrypt sensitive data at rest
- **⚠️ TODO:** Encrypt backup files

Secret Management:

- Secrets in `.env` file (not in code)
- `.env` file excluded from version control
- **⚠️ TODO:** Use secret management service (HashiCorp Vault, AWS Secrets Manager)
- **⚠️ TODO:** Rotate API keys regularly

PII Handling:

- No PII in AI prompts
- No PII in logs
- **⚠️ TODO:** Data anonymization for analytics
- **⚠️ TODO:** GDPR compliance measures (if applicable)

20.4 Application Security

Input Validation:

- Pydantic models for request validation
- SQL injection prevention (SQLAlchemy ORM)
- State machine enforcement
- **⚠️ TODO:** Rate limiting per user
- **⚠️ TODO:** Request size limits

Output Sanitization:

- JSON responses (no XSS risk)
- No sensitive data in error messages
- **⚠️ TODO:** Content Security Policy headers

Error Handling:

- Generic error messages to clients
- Detailed errors logged server-side
- No stack traces exposed to clients

20.5 Dependency Security

Dependency Management:

- `requirements.txt` with pinned versions
- **⚠️ TODO:** Regular dependency updates
- **⚠️ TODO:** Automated vulnerability scanning (Dependabot, Snyk)
- **⚠️ TODO:** Security advisories monitoring

Container Security:

- Docker images from trusted sources
- **⚠️ TODO:** Scan Docker images for vulnerabilities
- **⚠️ TODO:** Use minimal base images

- **TODO:** Run containers as non-root user

20.6 Compliance & Audit

Audit Logging:

- State transitions logged
- Approval events logged
- AI invocation metadata logged
- **TODO:** Comprehensive audit trail
- **TODO:** Immutable audit logs

Access Control:

- Role-based access control
- Parent approval enforcement
- **TODO:** Access logging and monitoring
- **TODO:** Regular access reviews

Data Retention:

- Log retention policy (90 days)
- **TODO:** Data deletion procedures
- **TODO:** User data export capability
- **TODO:** Right to be forgotten implementation

20.7 Security Testing

Vulnerability Assessment:

- **TODO:** Regular security scans
- **TODO:** Penetration testing
- **TODO:** Dependency vulnerability scanning
- **TODO:** OWASP Top 10 compliance check

Security Headers:

- **TODO:** Implement security headers:
 - Content-Security-Policy
 - X-Frame-Options
 - X-Content-Type-Options
 - Strict-Transport-Security

20.8 Incident Response

Security Incident Plan:

- **TODO:** Define incident response procedures
- **TODO:** Designate security contact
- **TODO:** Document breach notification procedures
- **TODO:** Regular security drills

Backup Security:

- **TODO:** Encrypt backup files
- **TODO:** Secure backup storage
- **TODO:** Test backup restoration

- **TODO:** Off-site backup copies

20.9 Security Checklist Summary

Critical (Must Have):

- Strong JWT secret
- HTTPS/TLS for all communications
- Secrets not in version control
- Input validation
- Role-based access control

Important (Should Have):

- Token refresh mechanism
- Rate limiting
- Dependency vulnerability scanning
- Encrypted backups
- Security headers

Recommended (Nice to Have):

- Secret management service
- Automated security scanning
- Penetration testing
- Security monitoring and alerting

Document Status: COMPLETE - All core sections and optional enhancements documented

Current State:

- All 20 major sections documented
- 12 Mermaid diagrams included
- Complete API reference with examples
- Database schema with ERD
- Configuration examples and deployment guide
- Code examples for key algorithms
- Testing strategy documented
- Monitoring and observability guide
- Backup and recovery procedures
- Performance tuning guide
- Security hardening checklist
- Comprehensive and production-ready