# EduGame AI — Architecture Document

**Role detected:** Teacher / Project Owner — this architecture is written for engineering handoff and stakeholder review.

---

# 1. Executive summary

EduGame AI is a role-aware learning platform providing AI-powered features for **Students** and **Teachers**: the Doubt Solver (chat), Lecture Summariser, Quiz Engine, Assignments & Evaluator, gamified Quiz Arena, **Explain My Mistake** and **AI Lesson Planner**.
This document specifies a production-ready architecture that is scalable, privacy-aware, and modular so AI capabilities can evolve independently.

Key non-functional targets

- Typical AI call latency: **< 5–10s** (where possible)
- Availability: **99.5%** for classroom hours
- Security: TLS in transit, encryption at rest, RBAC, auditable AI outputs
- Deployment flexibility: Cloud, On-prem, Hybrid

---

# 2. High-level architecture (summary)

```
[Browser / Mobile]
       |
   (HTTPS)
       |
   Frontend SPA (React)
       |
   API Gateway / Backend (FastAPI / Node.js)
       |
  ------------------------------------------------
  |                  |             |              |
Auth Service   Orchestration    AI Services     Data Services
(SSO/JWT)     (job queue)     (Doubt, Summ,    (Postgres, pgvector,
                                 Quiz, etc)      Object Store, Redis)
  ------------------------------------------------
       |
 Observability (Prometheus, Grafana, ELK)
```

Components:

- Frontend SPA (React + TypeScript + Tailwind)
- API Layer (FastAPI / Node.js) + WebSocket gateway for real-time features
- AI Microservices (containerized): Doubt Solver, Summarizer, Quiz Engine, Lesson Planner, Mistake Analysis, Evaluator
- Message Broker / Job Queue (RabbitMQ / Redis Streams / Kafka)
- Data Stores: PostgreSQL (transactional), `pgvector` for embeddings, Object Store (S3-compatible) for files
- Cache: Redis
- Orchestration: Kubernetes for production; Docker Compose for dev
- Monitoring & Logging: Prometheus + Grafana, ELK (or Loki) + Fluentd
- CI/CD: GitHub Actions / GitLab CI; images in private registry

---

# 3. Component detail

## 3.1 Frontend

- SPA with route-based role views:
  - `/auth` — login / SSO
  - `/student` — doubt chat, summaries, quiz arena, assignments
  - `/teacher` — class management, quiz creation, lesson planner, analytics
- Real-time via WebSocket for:
  - live chat (Doubt Solver)
  - leaderboards and streak updates
  - job progress notifications (e.g., long summarization)
- Static assets served via CDN

## 3.2 API Gateway / Backend

- Responsibilities:
  - Authentication & Authorization (SSO integration, JWT)
  - RBAC enforcement (student / teacher / admin)
  - Request routing & orchestration
  - Input validation & rate limiting
  - Persist requests, enqueue AI jobs, fetch results
- Tech choices:
  - FastAPI (Python) for quick integration with ML services or
  - Node.js (Express / NestJS) if team prefers JS/TS

## 3.3 AI Microservices

Each is a small service with its own container, autoscaling rules, and endpoint.

## A. Doubt-Solver Service

- Chat orchestration, session management, context windowing, RAG calls
- Stores conversational history per session
- Exposes streaming API for incremental responses
- Returns: answer text, steps, sources (citation pointers), confidence score

## B. Summarizer Service

- Ingests PDFs / text, runs extraction pipeline (e.g., PyMuPDF), chunking, embedding
- Generates summary, key-points, flashcards, and short quizzes
- Handles large documents using chunking + RAG

## C. Quiz Engine

- Generates questions given topic/difficulty using LLM + templates
- Validates question quality (sanity checks)
- Supports MCQ, short answer, coded response; returns metadata and answer keys
- Auto-grading logic for objective questions

## D. Lesson Planner

- Input: topic, class level, duration
- Uses RAG (teacher materials + global prompts) → outputs day-wise plans, objectives, activities, expected mistakes, mini-quizzes

## E. Mistake Analysis (Explain My Mistake)

- Input: problem statement, student answer (optional: model answer)
- Pipeline:
    - Normalize inputs
    - Run LLM analysis to: identify mistakes, map incorrect step(s), propose corrections, suggest practice problems
    - Optionally run code/math step validator (e.g., SymPy or small test harness for program outputs)
- Output: structured JSON with sections (mistakes[], correction_steps[], practice_questions[])

## F. Evaluator Service

- Uses rubrics / templated grading logic

- Runs auto-evaluation for assignments (first-level); flags subjective answers for teacher review.

## 3.4 Message Broker & Orchestration

- Use RabbitMQ or Redis Streams for:
  - enqueueing AI jobs (summaries, lesson plans, mistake checks)
  - asynchronous task processing with retries, DLQ (dead-letter queue)
- Jobs include metadata for traceability (user id, class id, request id, prompt version)

## 3.5 Data Layer

- PostgreSQL as single source of truth (users, classes, quizzes, attempts, lesson_plans, mistake_checks, analytics)
- `pgvector` extension for embeddings (or external milvus/pinecone if high scale)
- Object store (S3-compatible) for PDFs/uploads
- Redis for caching session info, rate-limits, leaderboards, short-term storage

---

# 4. Data flow (typical scenarios)

## 4.1 Explain My Mistake — data flow

1. Student POST `/api/student/mistake-check` → payload saved to DB, file stored in S3 (if provided)
2. API enqueues job to queue with request id
3. Mistake Analysis service consumes job, fetches context (RAG if needed), processes LLM
4. Service stores result JSON in DB (mistake_checks) and embeddings/metadata
5. API notifies front-end via WebSocket; student displays result

## 4.2 Lecture Summariser — data flow

1. Student uploads PDF → File stored in S3 → API triggers Summarizer job
2. Summarizer extracts text, chunks, computes embeddings, stores chunks + vectors
3. Summarizer generates summary, flashcards, quiz Qs; stores outputs and returns an asset link
4. Student views summary; teacher can publish to class

---

# 5. RAG and Prompt Management

## 5.1 RAG pipeline

- Chunking: text split into ~500–1500 token chunks with overlap
- Embedding: use model-specific embeddings (match vector dimension)
- Search: top-K semantic retrieval via `pgvector`
- Prompting: retrieved top context chunks appended to LLM prompt with clear instruction template

## 5.2 Prompt/versioning

- Maintain a Prompt Store (git-backed or DB) with:
  - Prompt ID, version, template, creator, changelog
- All AI calls embed prompt_version + model_id in logs for reproducibility and audits

---

# 6. Models & hosting

- Use locally-hosted/open-source LLMs for privacy-sensitive deployments (Llama3-family or equivalent)
- Optionally a hosted model for non-sensitive clients
- Model hosting options:
  - On-prem GPU nodes (NVIDIA A100/RTX6000/RTX4090 depending on scale)
  - Cloud GPU instances (AWS/GCP/Azure) inside VPCs
- Consider model orchestration via:
  - Ollama / Hugging Face Inference Endpoints (self-hosted)
  - Triton inference for optimized serving

---

# 7. Scalability & performance patterns

## 7.1 Horizontal scaling

- Stateless microservices scaled by Kubernetes HPA (CPU/GPU utilization)
- Redis + Postgres scale via managed offerings or sharding/replica-read patterns

## 7.2 Batch & async processing

- Heavy tasks (document summarization, lesson planner generation) processed asynchronously with job queue
- Keep user-facing operations fast by returning a job id and streaming progress

## 7.3 Caching & precomputation

- Cache frequently requested summaries, lesson templates
- Precompute embeddings for frequently-used notes/teacher resources

## 7.4 Limits

- Enforce per-user rate limits for LLM calls and uploads
- Implement quotas for free vs paid tiers

---

# 8. Security, privacy & compliance

## 8.1 Authentication & Authorization

- Support SSO (SAML / OIDC) for institutions
- JWT tokens for client sessions
- RBAC enforced at API & DB levels

## 8.2 Data protection

- TLS for transport
- Encryption at rest (DB fields with pgcrypto for sensitive text)
- Server-side encryption for object store
- Audit trails for all AI outputs and data accesses

## 8.3 Privacy features

- Option for on-prem deployments (no PHI/PII leaves network)
- Data retention & deletion endpoints for compliance (GDPR/DPDP)
- Masking/anonymization tools for datasets used to fine-tune models

## 8.4 Operational security

- Secrets in vault (HashiCorp Vault / AWS Secrets Manager)
- Minimal service account permissions
- Regular vulnerability scans & pen tests

---

# 9. Observability & SLOs

## 9.1 Monitoring

- Metrics: request latency, model latency, job queue lengths, error rates, CPU/GPU usage
- Tools: Prometheus + Grafana

## 9.2 Logging and tracing

- Structured logs (JSON); centralize to ELK or Loki
- Distributed tracing (OpenTelemetry) across API and services
- Store AI call context (prompt_version, model_id, response_time) for audits

## 9.3 Alerts & SLOs

- SLO example: 95% of chat responses < 7s
- Alert: queue backlog > threshold, error rate spike > 5% in 5m

---

# 10. Backup, recovery & maintenance

- Daily DB backups, retained per org policy (30/90/365 days)
- Object store lifecycle rules (cold storage for older assets)
- Infrastructure IaC (Terraform) with versioned state
- Disaster recovery runbooks for primary regions

---

# 11. Deployment blueprint & environment matrix

## 11.1 Environments

- Local dev (Docker Compose), feature branches
- Staging (mirror of prod, smaller capacity)
- Production (Kubernetes with autoscaling)

## 11.2 K8s primitives (suggested)

- Namespaces: `dev`, `staging`, `prod`
- Deployments: frontend, api-gateway, ai-<service>, worker
- StatefulSets: postgres, pgvector (or managed DB)
- PV persistent volumes for object store gateway (S3-compatible)
- GPU nodepool for AI services using device plugin

## 11.3 CI/CD

- Build & test → push image to registry → deploy via Helm chart / ArgoCD
- Canary release for AI model/ prompt changes

---

# 12. Hardware & cost considerations (baseline)

## Small pilot (school)

- 1-2 GPU nodes (e.g., 1x NVIDIA A40 or equivalent per heavy model) OR CPU-only with smaller LLMs
- 4 vCPU app server
- Managed Postgres (or on-prem VM) 4–8GB RAM
- 100–500GB object store

## Medium / multi-school

- GPU cluster (2–4 GPUs)
- Auto-scaling GPU pool
- Postgres HA with read replicas
- Redis cluster

---

# 13. APIs (representative list)

- `POST /api/auth/login` — SSO/JWT login
- `GET /api/users/me` — fetch user profile
- `POST /api/classes` — create class
- `POST /api/classes/:id/join` — join class via code
- `POST /api/student/doubt` — submit doubt (returns chat id)
- `GET /api/student/doubt/:id/stream` — stream chat responses
- `POST /api/student/mistake-check` — submit Explain My Mistake job
- `GET /api/student/mistake-check/:id` — fetch results
- `POST /api/summaries/upload` — upload and summarize PDF
- `POST /api/teacher/lesson-plans/generate` — generate lesson plan
- `POST /api/teacher/quizzes` — generate quiz
- `GET /api/analytics/class/:id` — class analytics snapshot

All endpoints must log prompt_version & model_id for AI-related calls.

---

# 14. Governance: prompts & model change management

- Any prompt or model change is versioned and requires:
  - PR with rationale
  - Test dataset results
  - Approval by product owner / education SME
- Release notes recorded (prompt changes, model versions, timeout changes)

---

# 15. Appendix: sample sequence (Lecture Summariser)

1. Student uploads `lecture.pdf` → `POST /api/summaries/upload`
2. API stores file in S3 and writes job to queue
3. Worker: Extract text (PyMuPDF) → chunk → embed → store chunks in `embeddings`
4. Summarizer constructs prompt with top-K retrieved chunks → calls model
5. Model returns summary + flashcards + quiz Qs → worker stores outputs in DB
6. API notifies student that summary is ready

# 16. Next steps & deliverables to produce

- Helm charts & Kubernetes manifests for the architecture above
- OpenAPI specification for all endpoints (Swagger)
- DB migration scripts for the described schema (DDL)
- Prompt library repo & versioning system
- Infrastructure IaC (Terraform) for cloud/on-prem bootstrap
- Figma high-fidelity wireframes for the wireframes described