.

**HLA — Academic Data Management & AI Analysis Platform**

نسخه: 1.0 — تهیه‌شده برای پیاده‌سازی و تحویل به دپارتمان  
شامل: دیاگرام کلی، شرح کامپوننت به‌صورت دقیق، API contracts کلیدی، داده‌فلوها، الگوهای مقیاس‌پذیری، نیازمندی‌های infra و امنیتی، و checklist عملی برای شروع توسعه.

**1. خلاصهٔ معماری (Executive Summary)**

پلتفرم یک معماری **میكروسرویسِ رویدادمحور (event-driven microservices)** است که از الگوی زیر پیروی می‌کند:

* **Presentation Layer**: وب‌اپ React (و API برای mobile)، UI/UX برای دانشجو/استاد/مدیر
* **API Gateway & Auth**: احراز هویت (OAuth2/JWT)، نرخ‌دهی (rate-limiting)، مسیریابی
* **Core Microservices**: User, Student, Faculty, Course, Enrollment, Scheduling, Recommendation, Analytics/ML
* **Data Layer**: PostgreSQL (OLTP) + MongoDB/Parquet (Feature Store / OLAP) + Redis (cache)
* **Event Bus**: Kafka (یا RabbitMQ) برای انتقال رویدادها بین سرویس‌ها
* **ML Infrastructure**: Kubeflow/Airflow for pipelines, GPU nodes for training, Model Registry (MLflow)
* **Infra & Ops**: Kubernetes cluster, Helm charts, GitHub Actions CI, ArgoCD/Flux CD, Prometheus+Grafana, ELK for logs
* **Security**: TLS, mTLS، RBAC، Vault برای secrets، Audit logs

این معماری هزینه/زمان توسعه را بهینه می‌کند، امکان توسعهٔ موازی تیمی را می‌دهد، و برای پذیرش دانشگاهی و مقالات پژوهشی مناسب است.

**2. دیاگرام HLA (PlantUML)**

(این بلوک را می‌توان در PlantUML اجرا و به PNG/SVG تبدیل کرد.)

```

@startuml

!theme spacelab

title Academic Data Platform - High Level Architecture

package "Users & External Systems" {

actor "Student / Faculty / Admin" as User

rectangle "LMS (Moodle/Canvas)\nSIS, Library, SSO" as External

}

package "Frontend" {

[React App\n(Web & Mobile)] as FE

[Admin Dashboard] as DASH

}

package "Edge" {

[Ingress (NGINX/CloudLB)] as INGRESS

[API Gateway & Auth (Keycloak/FastAPI)] as GATEWAY

}

package "Microservices (Kubernetes)" {

component "User Service\n(FastAPI/Django)" as USER

component "Student Service\n(FastAPI)" as STUD

component "Faculty Service" as FAC

component "Course & Section Service" as COURSE

component "Enrollment Service" as ENR

component "Scheduling Service\n(Optimizer: ILP/GA)" as SCHED

component "Analytics Service\n(Feature extraction)" as AN

component "Recommendation Service" as REC

component "ML Workers / Orchestrator\n(Kubeflow, GPU Pool)" as ML

component "Integration Adapters\n(LMS, SIS connectors)" as ADAPTERS

component "Notification Service\n(Email/SMS/WebSocket)" as NOTIF

}

package "Data & Infra" {

database "Postgres Cluster\n(OLTP)" as PG

database "MongoDB / Feature Store\n(Parquet on S3)" as FS

queue "Kafka / RabbitMQ\n(Event Bus)" as KAFKA

queue "Redis\n(Cache & Session)" as REDIS

storage "Object Storage (S3)\n(Parquet, artifacts)" as S3

rectangle "Monitoring\nPrometheus + Grafana\nELK Stack" as MON

rectangle "CI/CD\nGitHub Actions + ArgoCD" as CICD

rectangle "Secrets\nVault / AWS Secrets Manager" as SECRETS

rectangle "Model Registry\n(MLflow)" as MREG

}

' flows

User --> FE : HTTP/HTTPS

FE --> INGRESS : HTTPS

INGRESS --> GATEWAY : HTTPS

GATEWAY --> USER : REST

GATEWAY --> STUD : REST

GATEWAY --> ENR : REST

GATEWAY --> SCHED : REST

ADAPTERS --> GATEWAY : REST/Webhook

ENR --> KAFKA : publish(EnrollmentCreated)

SCHED --> PG : read/write

AN --> KAFKA : subscribe(GradeUpdated, EnrollmentCreated)

AN --> ML : submitTraining()

ML --> MREG : register(model)

ML --> FS : write(features, artifacts)

AN --> PG : write(risk\_scores)

REC --> PG : read profiles

REC --> FS : read features

NOTIF --> GATEWAY : websocket / push

PG --> REDIS : cache hits

PG --> S3 : export/parquet

@enduml

```

## . شرح ریزکامپوننت‌ها (Component-by-Component)

برای هر کامپوننت: هدف، وظایف اصلی، API نمونه، دیتااستور، مقیاس‌گذاری پیشنهادی، وابستگی‌ها، نکات عملی.

### 3.1 Ingress & API Gateway

* **هدف:** ورود ترافیک، TLS termination، rate-limiting، مسیریابی به میکروسرویس‌ها، احراز هویت و اعمال سیاست‌های RBAC.
* **تکنولوژی پیشنهادی:** NGINX/Cloud Load Balancer + Kong/Traefik یا FastAPI Gateway + Keycloak برای OAuth2.
* **وظایف:** SSL/TLS, JWT validation, request throttling, request logging.
* **API contracts:** همه میکروسرویس‌ها OpenAPI دارند؛ Gateway مسیرها را به سرویس مربوطه پروکسی می‌کند.
* **مقیاس‌پذیری:** scale horizontally behind LB, use autoscaling.

### 3.2 User Service

* **هدف:** مدیریت حساب‌ها، نقش‌ها، احراز هویت (در کنار Keycloak) و پروفایل کاربران.
* **وظایف:** CRUD user, password reset, role management, user search (pg\_trgm).
* **API نمونه:** GET /users/{id}, POST /users, PUT /users/{id}
* **دیتااستور:** PostgreSQL (users table).
* **نکات:** password hashing (bcrypt) + email verification + MFA for admin actions.

### 3.3 Student Service

* **هدف:** عملیات domain-specific برای دانشجویان: ترنسکریپت، تاریخچهٔ ثبت‌نام، وضعیت تحصیلی.
* **وظایف:** generate transcript, compute GPA, export student data.
* **API:** GET /students/{id}/transcript, GET /students/{id}/enrollments.
* **دیتااستور:** PostgreSQL (students, enrollments, grades).
* **نکات:** read-heavy — use read-replicas and Redis cache for hot queries.

### 3.4 Faculty Service

* مشابه User Service اما با فیلدهای پژوهشی، list publications, grants.

### 3.5 Course & Section Service

* **وظیفه:** مدیریت course catalog و section instances (terms).
* **API:** POST /courses, POST /sections, GET /courses?dept=.
* **نکات:** support course equivalence and prerequisites rules.

### 3.6 Enrollment Service

* **هدف:** business rules ثبت‌نام و مدیریت capacity/overrides.
* **وظایف:** enforce prereqs, capacity checks, create enrollment row, publish EnrollmentCreated event to Kafka.
* **API:** POST /enrollments (body: student\_id, section\_id).
* **نکات:** idempotency, transaction patterns (use DB transaction + outbox pattern to reliably publish events).

### 3.7 Scheduling Service (Optimizer)

* **هدف:** تولید برنامهٔ زمان‌بندی بر اساس constraints (rooms, instructor availability, conflicts).
* **وظایف:** accept constraints JSON, run optimizer (ILP using OR-Tools or GA), persist schedule.
* **API:** POST /scheduling/generate (returns job\_id).
* **دیتااستور:** Postgres (schedules, rooms), may use temporary in-memory graph.
* **نکات:** long-running jobs → run as background job (Kubernetes Job) and report status via DB/events.

### 3.8 Analytics Service (Feature Extraction + Orchestration)

* **هدف:** جمع‌آوری و آماده‌سازی features برای مدل‌های ML، اجرای inference و ذخیره risk\_scores.
* **وظایف:** ETL pipelines (Airflow), feature computation, triggering ML training/serving, storing results.
* **API:** POST /analytics/at\_risk/run, GET /analytics/jobs/{id}/status.
* **دیتااستور:** Feature Store (MongoDB / Parquet on S3), Postgres for results.
* **نکات:** maintain feature lineage, reproducibility (dataset hashes), use outbox/event-driven updates for incremental features.

### 3.9 ML Workers / Orchestrator

* **هدف:** اجرای آموزش مدل‌ها و سروینگ.
* **تکنولوژی:** Kubeflow / MLflow / TorchServe / KFServing.
* **وظایف:** train jobs on GPU nodes, model versioning, expose prediction endpoints.
* **MREG:** register model metadata in MLflow + model\_registry table.

### 3.10 Recommendation Service

* **هدف:** تولید توصیه‌های دوره/پروژه به‌صورت زمانبندی‌شده یا on-demand.
* **الگوریتم پایه:** collaborative filtering (matrix factorization) + content-based hybrid.
* **API:** GET /recommendations/{student\_id}.
* **نکات:** A/B testing harness; store rec results in recommendations table.

### 3.11 Integration Adapters

* **هدف:** connector برای LMS, SIS, Library — ingest data (webhooks/polling), normalize payloads, publish events.
* **نکات:** transformation rules, scheduler for batch sync, secure access (API keys/SSO).

### 3.12 Notification Service

* **هدف:** ارسال ایمیل، پیامک، و نوتیف WebSocket.
* **وظایف:** subscribe to events (EnrollmentCreated, GradeUpdated, RiskAlert) and dispatch notifications.
* **نکات:** retry logic, dlq for failed messages.

### 3.13 Observability & CI/CD

* **Monitoring:** Prometheus metrics per service, Grafana dashboards for system & ML metrics.
* **Logging:** centralized ELK stack (Logstash/Elasticsearch/Kibana), structured logs (JSON).
* **Tracing:** OpenTelemetry for distributed tracing (end-to-end request id).
* **CI/CD:** GitHub Actions for CI; ArgoCD/Flux for CD (helm charts).
* **Secrets:** Vault or cloud secret manager.

## 4. جریان داده‌های کلیدی (Data Flows / Sequence Summaries)

در این بخش مهم‌ترین سناریوها با گام‌ها و تعامل سرویس‌ها ذکر شده‌اند.

### 4.1 Enrollment flow (real-time)

1. Student UI -> POST /enrollments at Gateway
2. Gateway -> Enrollment Service (validate JWT, permission)
3. Enrollment Service -> Postgres transaction: check prereq & capacity
4. On success: create enrollment row, commit transaction
5. Enrollment Service -> Kafka publish EnrollmentCreated (outbox pattern)
6. Analytics Service subscribes -> update features for student in Feature Store
7. Notification Service subscribes -> send confirmation to student

### 4.2 Grade submission -> risk recompute (near real-time)

1. Instructor -> POST /grades
2. Grades Service writes grade, publishes GradeUpdated event
3. Analytics Service subscribes -> trigger incremental feature recompute for affected student(s)
4. If threshold crossed -> recompute risk model/inference (ML Serving) -> store risk\_score
5. Notification Service alerts advisor/admin if risk\_score > threshold

### 4.3 At-Risk Batch Prediction

1. Admin triggers POST /analytics/at\_risk/run or scheduled cron triggers job
2. Analytics Service composes dataset from Feature Store, submits job to ML Orchestrator (Kubeflow)
3. ML Worker trains/infers; stores model in Model Registry and results in risk\_scores table
4. Dashboard reads risk\_scores and presents to admin

## 5. API Contracts — نمونه‌ها (OpenAPI-ready)

(باید به صورت OpenAPI YAML/JSON تولید شود؛ این نمونه‌ها برای شروع کافی‌اند.)

### POST /auth/login

* Request:

{ "username": "string", "password": "string" }

Response:

{ "access\_token": "jwt", "refresh\_token": "jwt", "expires\_in": 3600 }

### POST /enrollments

* Request:

{ "student\_id": "uuid", "section\_id": "uuid" }

Response:

{ "enrollment\_id": "uuid", "status": "active" }

* Errors: 400 (prereq), 409 (capacity full), 401 (unauthorized)

### POST /analytics/at\_risk/run

* Request:

{ "scope":"all" | {"program\_id":"uuid"}, "model\_id":"latest" | "uuid", "notify": true }

Response:

{ "job\_id": "uuid", "status": "queued" }

نکته: همه endpointها باید دارای نمونه responses، خطاها، و نقش‌های مجاز در OpenAPI باشند.

## مقیاس‌پذیری، اعتمادپذیری و الگوهای عملیاتی

### 6.1 مقیاس‌پذیری

* **Stateless services** (API, Enrollment, Recommendation) -> HPA (Horizontal Pod Autoscaler) در Kubernetes با معیار CPU/RPS.
* **Stateful services** (Postgres) -> managed RDS/Cloud SQL با read-replicas.
* **Event Bus (Kafka)** -> multi-broker cluster با replication.factor >= 3.
* **ML training** -> node pools با GPU؛ job scheduling توسط Kubeflow.

### 6.2 اعتمادپذیری و در دسترس‌پذیری

* Use **outbox pattern** for transactional event publishing.
* Retries + dead-letter queues for message consumers.
* Backups: daily DB snapshots to S3, WAL archiving.
* Health checks (liveness/readiness) for all pods.

### 6.3 تحمل خطا (Fault Tolerance)

* Circuit Breakers (client-side) برای تماس بین سرویس‌ها (e.g., via Istio/Linkerd).
* Graceful shutdown for pods; draining nodes before update.
* Canary deployments for ML models and APIs.

## 7. امنیت (Security & Compliance)

### 7.1 Authentication & Authorization

* Centralized Identity Provider (Keycloak) for SSO/OAuth2.
* Gateway validates access\_tokens; microservices rely on token + audience claim.
* RBAC for resource authorization; attribute-based checks for sensitive ops (grade override).
* MFA for admin.

### 7.2 Data protection

* Encryption in transit (TLS 1.2+).
* Encryption at rest: Postgres disk encryption + S3 SSE.
* PII handling: redact in logs; separate access controls; data anonymization flows.

### 7.3 Audit & Logging

* All sensitive actions logged with user\_id, timestamp, origin IP.
* Immutable audit logs exported to secure storage; retention policy per compliance.

## 8. Observability & SLOs

* **Metrics:** request latency, error rate, ML model latency, job durations, queue lag.
* **Dashboards:** per-service and cross-cutting dashboards in Grafana.
* **Alerts:** p95 latency > threshold, error rate > threshold, kafka consumer lag > threshold.
* **Tracing:** distributed traces for enrollment & analytics flows (OpenTelemetry).

**Suggested SLOs:**

* API availability: 99.5% monthly
* Enrollment request p95 latency < 500ms
* ML job completion time: configurable; for batch ≤ 4 hours (depends on dataset)

## 9. Deployment Topology (Production)

* Kubernetes Cluster (multi-zone)
  + Namespace per environment: dev / staging / prod
  + Ingress controller + API Gateway + Keycloak (HA)
  + Backend services deployed as Deployments (replicas), ML workers as Jobs/CronJobs
  + Postgres as managed statefulset or cloud-managed RDS with read-replicas
  + Kafka as statefulset cluster or managed (Confluent/MSK)
  + S3 (or S3-compatible) for feature/artifact storage

## 10. Dev & Release Workflow (CI/CD)

* **Branching:** GitFlow (feature/ -> develop -> release -> main)
* **CI:** lint, unit tests, type checks, build docker images, run security scans (Snyk/Trivy)
* **CD:** image pushed to registry -> ArgoCD sync to staging -> smoke tests -> manual promotion to prod.
* **Release process:** release PR includes migration scripts and rollback plan.

## 11. Data Migration & Backwards Compatibility

* Use Alembic for Postgres schema migrations.
* Feature Store schema versioning and model compatibility checks.
* Blue/Green or Canary for DB migration where needed (rolling migrations, avoid breaking changes).

## 12. Testing Strategy (mapping to microservices)

* **Unit tests:** per service (pytest, unittest)
* **Integration tests:** spin up test DB (docker-compose) and run real requests (supertest/requests)
* **Contract tests:** use Pact/OpenAPI contract tests to ensure gateway and services consistent.
* **E2E tests:** Cypress for UI flows.
* **Load tests:** k6 or JMeter for API endpoints and Kafka throughput.

## 13. Cost & Resource Estimates (initial)

* **Dev cluster** (for 4–6 devs): small k8s on cloud — 3 nodes (4 vCPU, 16GB) ~ modest cost.
* **Staging/Prod**: multi-zone k8s + managed Postgres + managed Kafka + S3 + GPU nodepool (1-2 GPUs for experiments) — estimates depend on provider; ask IT for credits.
* **Storage:** initial S3 50–100GB.  
  (سنجش دقیق‌تر پس از تعیین مقیاس کاربران و نرخ ETL انجام شود.)

## 14. Security / Compliance Checklist قبل از Prod