UML Diagrams — Academic Data Management & AI Platform

این سند شامل مجموعه کاملی از دیاگرام‌های ERD و UML است که معماری، کلاس‌ها، کامپوننت‌ها، دپلویمنت و سناریوهای کلیدی سیستم مدیریت و تحلیل داده‌های دانشگاهی را با جزییات دقیق نشان می‌دهد. هر بخش شامل توضیح مختصر و کد PlantUML برای تولید دیاگرام است (قابل تبدیل به PNG/SVG توسط ابزارهای PlantUML).

فهرست دیاگرام‌ها

1. ERD (نمودار موجودیت-رابطه)
2. Class Diagram (دیتا مدل‌ها و سرویس‌ها)
3. Component Diagram (ماژول‌ها و وابستگی‌ها)
4. Sequence Diagram — Use Case: پیش‌بینی دانشجوی در معرض افت
5. Sequence Diagram — Use Case: زمان‌بندی کلاس (Scheduling)
6. Deployment Diagram (محیط اجرا و زیرساخت)
7. Data Flow / ML Pipeline Diagram

توضیحات کلی قبل از دیاگرام‌ها

- نهایی انتخاب شده‌اند RFP نام‌های موجود در دیاگرام‌ها بر اساس.
- هر PlantUML block با علامت ```plantuml قرار داده شده است؛ اگر بخواهید می‌توانید آن‌ها را به تصویر تبدیل کنید سایت‌هایی مثل https://www.plantuml.com/plantuml/png.
- من می‌تونم آماده‌شون کنم، یا توی گیت‌هاب قرار بدیم (PNG/SVG/PDF) در صورت نیاز به خروجی‌های تصویری.

---

1) ERD — موجودیت‌های اصلی و رابطه‌ها

توضیح

این ERD کاربران (User)، دانشجو (Student)، کارمند/استاد (Faculty)، موجودیت‌های اصلی سیستم را نشان می‌دهد: همکاری (ResearchPaper)، مقاله (Project)، پروژه/پایان‌نامه (Grade)، نمره (Enrollment)، ثبت‌نام (Course)، درس پژوهشی (Collaboration)، کلاس (Classroom/Schedule) و لاگ‌ها.

```
@startuml
!theme spacelab
entity "User" as U {
  * user_id : UUID
  * username : string
```

```
  * email : string
  * password_hash : string
  * role : enum
  * created_at : timestamp
}

entity "Student" as S {
  * student_id : UUID
  * user_id : UUID
  * admission_year : int
  * program_id : UUID
  * status : enum
}

entity "Faculty" as F {
  * faculty_id : UUID
  * user_id : UUID
  * department_id : UUID
  * title : string
}

entity "Program" as P {
  * program_id : UUID
  * name : string
  * degree : enum
}

entity "Course" as C {
  * course_id : UUID
  * code : string
  * title : string
  * credits : int
  * department_id : UUID
}

entity "Section" as Sec {
  * section_id : UUID
  * course_id : UUID
  * term : string
  * year : int
  * instructor_id : UUID
}

entity "Enrollment" as E {
  * enrollment_id : UUID
  * student_id : UUID
  * section_id : UUID
  * status : enum
```

```
}

entity "Grade" as G {
  * grade_id : UUID
  * enrollment_id : UUID
  * grade_value : float
  * graded_at : timestamp
}

entity "ResearchPaper" as RP {
  * paper_id : UUID
  * title : string
  * doi : string
  * year : int
}

entity "AuthLog" as L {
  * log_id : UUID
  * user_id : UUID
  * action : string
  * ip : string
  * timestamp : timestamp
}

U ||--o{ S : has
U ||--o{ F : has
S }o--|| P : enrolled_in
F }o--|| P : teaches_in
P ||--o{ C : contains
C ||--o{ Sec : "has sections"
Sec ||--o{ E : "students"
E ||--o{ G : "has grades"
F ||--o{ Sec : "instructor"
RP }o--o{ F : "authored by"
U ||--o{ L : "logs"

@enduml
```

---

2) Class Diagram — Domain Models & Services

توضیح

این نمودار کلاس‌ها (مدل‌های داده و سرویس‌های اصلی) را نشان می‌دهد و متدهای کلیدی و ارتباطات بین سرویس‌ها را مشخص می‌کند.

```
@startuml
!theme cerulean
package "Domain Models" {
  class User {
    +user_id: UUID
    +username: string
    +email: string
    +role: Role
    +authenticate(password): Token
  }
  class Student {
    +student_id: UUID
    +getTranscript(): Transcript
  }
  class Faculty {
    +faculty_id: UUID
    +getPublications(): List<ResearchPaper>
  }
  class Course { +course_id: UUID }
  class Section { +section_id: UUID }
}

package "Services" {
  class AuthService {
    +login(credentials): Token
    +refresh(token): Token
    +authorize(user, action): bool
  }
  class UserService { +createUser(u): User }
  class StudentService { +getAtRiskStudents(): List<Student> }
  class SchedulingService {
    +generateSchedule(constraints): Schedule
    +optimize(schedule): Schedule
  }
  class AnalyticsService {
    +trainModel(data): Model
    +predict(student): RiskScore
  }
  class MLService {
    +runTrainingJob(config): JobId
    +serveModel(modelId): Endpoint
  }
  class NotificationService { +notify(user, msg) }
  class ReportService { +exportPDF(report): File }
}

' Relationships
```

```
AuthService -- UserService : uses
UserService -- StudentService : manages
StudentService .. AnalyticsService : provides data to
AnalyticsService .. MLService : delegates training
SchedulingService .. AnalyticsService : uses predictions
NotificationService .. UserService : sends to
ReportService .. AnalyticsService : generates reports from

@enduml
```

---

3) Component Diagram — ماژول‌های سیستم و ارتباط‌ها

توضیح

نشان‌دهنده‌ی ماژول‌های سطح بالا، پیاده‌سازی میکروسرویس‌ها، و تعامل با سرویس‌های خارجی (LMS, Library, SSO) و سرویس‌های زیرساختی (Message Broker, Object Storage).

```
@startuml
!theme sketchy
component "Frontend (React)" as FE
component "API Gateway / Auth (FastAPI)" as API
component "User Service (Django)" as US
component "Student Service (Django)" as SS
component "Analytics Service (Python)" as AS
component "ML Worker (PyTorch)" as ML
component "Scheduling Service (Golang)" as SCH
component "Message Broker (Kafka/RabbitMQ)" as MB
component "PostgreSQL Cluster" as PG
component "MongoDB" as MDB
component "Redis Cache" as RED
component "Object Storage (S3)" as S3
component "Monitoring (Prometheus/Grafana)" as MON

FE --> API : REST/WebSocket
API --> US : REST
API --> SS : REST
API --> SCH : REST
AS --> ML : submit job
SS --> PG : CRUD
US --> PG : CRUD
AS --> MDB : analytics store
AS --> RED : cache
SCH --> MB : publish/consume
MB --> ML : job queue
```

```
API --> MON : metrics

@enduml
```

---

4) Sequence Diagram — پیش‌بینی دانشجوی در معرض افت (At-Risk Prediction)

شرح سناریو

- رویداد: پایان ترم نزدیک است، سیستم تحلیل اجرای مدل را برای همه دانشجویان انجام می‌دهد و به مسئول آموزشی لیست دانشجویان در ریسک بالا را ارسال می‌کند.

```
@startuml
actor "Scheduler (CRON)" as Cron
participant "API Gateway" as API
participant "AnalyticsService" as AS
participant "MLService" as ML
participant "StudentService" as SS
participant "NotificationService" as NS
participant "DB (Postgres)" as PG

Cron -> API : POST /analytics/run_at_risk
API -> AS : validate + enqueue job
AS -> SS : query students + features
SS -> PG : SELECT features...
PG --> SS : features
SS --> AS : features
AS -> ML : submit training/prediction job
ML --> AS : predictions
AS -> PG : write risk_scores
AS -> NS : send report to admin
NS -> API : push notification
API -> Cron : 200 OK

@enduml
```

---

5) Sequence Diagram — زمان‌بندی کلاس (Scheduling)

```
@startuml
actor "Scheduler UI" as UI
participant "API Gateway" as API
```

```
participant "SchedulingService" as SCH
participant "Optimization Engine" as OPT
participant "DB (Postgres)" as PG

UI -> API : POST /schedules/generate {constraints}
API -> SCH : create job
SCH -> PG : fetch courses, rooms, instructors, constraints
PG --> SCH : data
SCH -> OPT : run optimizer (ILP / GA)
OPT --> SCH : schedule
SCH -> PG : persist schedule
SCH -> API : return schedule_id
API -> UI : 202 Accepted + schedule_id

@enduml
```

---

6) Deployment Diagram — اجزا و نحوه دپلوی

توضیح

نشان می‌دهد که هر سرویس چگونه در کلاستر Kubernetes، نودهای محاسباتی (GPU برای ML)، و دیتابیس‌ها و سرویس‌های کمکی مستقر می‌شود.

```
@startuml
!theme default
node "Kubernetes Cluster (prod)" {
  folder "Ingress" {
    [NGINX Ingress]
  }
  folder "Frontend Pods" {
    [FE ReplicaSet]
  }
  folder "API Pods" {
    [API Gateway Pod]
    [Auth Pod]
  }
  folder "Service Pods" {
    [UserService Pod]
    [StudentService Pod]
    [SchedulingService Pod]
    [AnalyticsService Pod]
  }
  folder "Worker Nodes" {
    [ML Worker GPU Pool]
```

```
    [Background Worker Pool]
  }
}

node "Managed DB" {
  [Postgres Cluster (Managed)]
  [MongoDB ReplicaSet]
}
node "Cache & Queue" {
  [Redis]
  [Kafka]
}
node "Observability" {
  [Prometheus]
  [Grafana]
  [ELK Stack]
}
node "Object Storage" {
  [S3-Compatible Storage]
}

[API Gateway Pod] --> [UserService Pod]
[API Gateway Pod] --> [StudentService Pod]
[AnalyticsService Pod] --> [ML Worker GPU Pool]
[ML Worker GPU Pool] --> [Postgres Cluster (Managed)]
[Background Worker Pool] --> [Kafka]
[Frontend Pods] --> [API Gateway Pod]
[Prometheus] --> [Grafana]

@enduml
```

---

7) ML Pipeline — Data Flow for Training & Serving

شرح

.این دیاگرام جریان داده و مراحل پردازش برای آماده‌سازی داده، آموزش مدل، ارزیابی، و سروینگ مدل را نشان می‌دهد

```
@startuml
!theme plain
start
:Collect features from Postgres / LMS / Logs;
:Preprocessing (cleaning, imputation, feature eng.);
:Store features in Feature Store (MongoDB/Parquet);
:Split train/val/test;
```

```
:Train model (PyTorch) on GPU;
:Evaluate model (metrics: AUC, precision, recall);
if (Good?) then (yes)
  :Register model in Model Registry;
  :Deploy model to Serving Endpoint (KFServing / TorchServe);
else (no)
  :Tune hyperparams / go back to Train;
endif
:Expose prediction API;
stop
@enduml
```

پیوست — نکات طراحی دقیق و توضیحات مهندسی

1. **شناسه‌ها:** برای تسهیل ریپلیکیشن و مانیتورینگ در چندنود و جلوگیری از UUID همه‌ی شناسه‌ها از نوع collision.
2. **مکانیسم تراز زمانی (Timestamps):** همه جداول اصلی دارای created_at و updated_at با timezone-aware timestamps.
3. **Versioning برای مدل‌ها:** تمام مدل‌های ML در Model Registry ثبت شوند با ورژن و متادیتا (git-hash، training dataset hash، hyperparams).
4. **Feature Store:** در یک Feature Store (MongoDB یا فایل‌های Parquet در S3) داده‌های پردازش‌شده برای آموزش و inference ذخیره شوند.
5. **Event-driven architecture:** برای عملیات پس‌زمینه (Training jobs، Notifications، Audit logs) از Message Broker (Kafka) استفاده شود.
6. **Security:** ارتباطات بین میکروسرویس‌ها با mTLS و احراز هویت با OAuth2/JWT.
7. **Auditing & GDPR-like compliance:** AuditLog برای رخدادهای مهم و مکان ذخیره امن برای داده‌های حساس.
8. **Backup & Recovery:** پشتیبان‌گیری روزانه از Postgres و Snapshot برای MongoDB و S3.

راه بعدی

- اگر تمایل داشته باشی، می‌تونم همین PlantUML‌ها رو اکسپورت کنم به PNG/SVG و فایل زیپ آمادهٔ بارگذاری در GitHub یا ارائه‌ات بسازم.
- همچنین می‌تونم ERD را به SQL DDL (Postgres) تبدیل کنم و اسکیماهای اولیه را بسازم.

پایان سند *UML*