TutorLink

Jorge Ramirez: 38883171
Gio Kandelaki: 38883147
Nada Elhaddad: 38883074

Team Roles

Nada

- Compliance with Feature ASRs (Front-end)
- Front-end
  - Graphical User Interface

Jorge

- Compliance with Architectural ASRs
- Compliance with Architectural Style
- Front-end Helper Libraries
  - Authentication
  - HTTP Calls to Backend
- Back-end
  - Gateway Components
  - Authentication Components
- Regression Tests Implementation

Gio

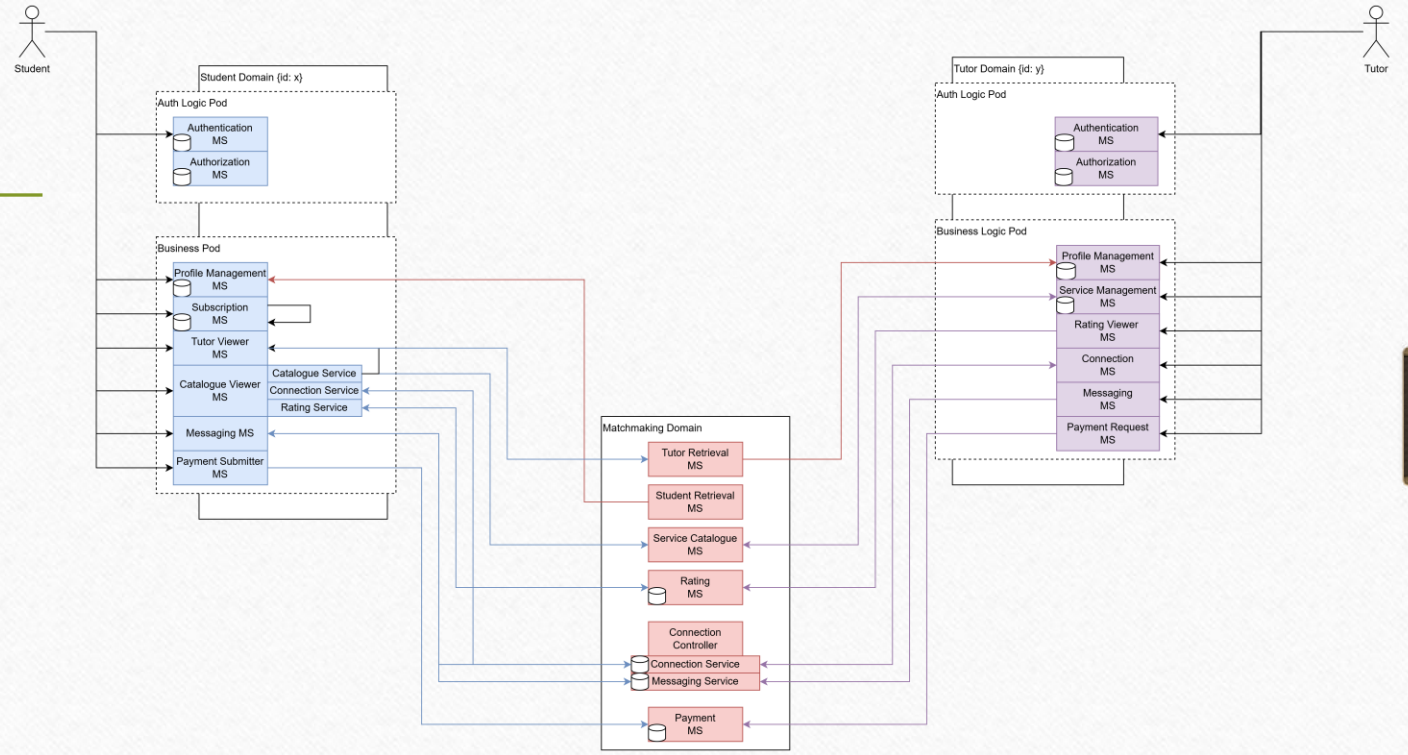
- Domain-Driven Back-end Design
- Compliance with Feature ASRs (Back-end)
- Back-end
  - Tutor Domain Functional and Authentication Components
  - Student Domain Functional and Authentication Components
  - Matchmaking Domain Functional Components

System Demonstration

Architectural Style

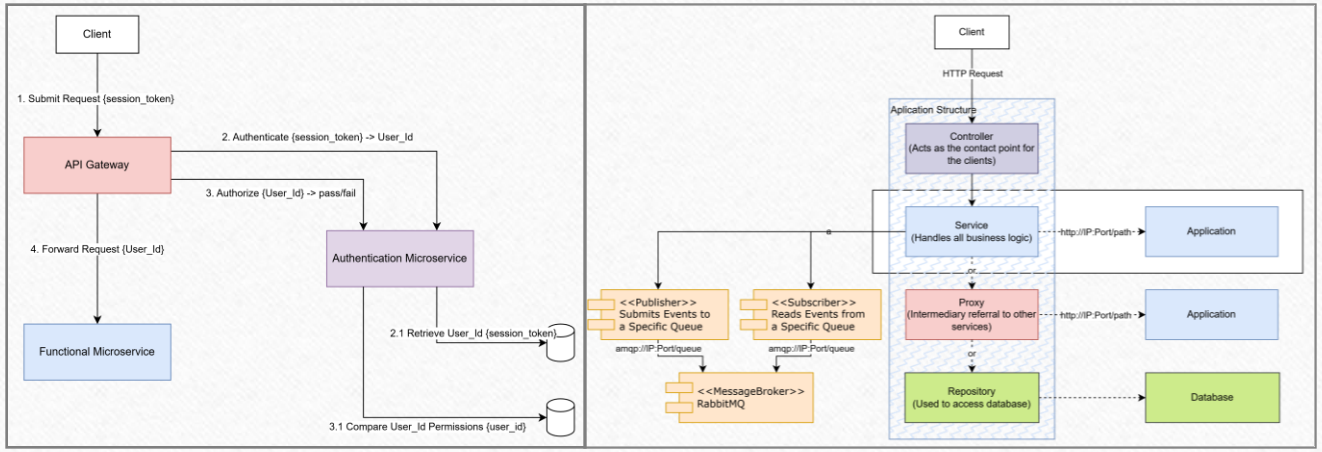
System Architectural Style

- Microservice Architectural Style
- Client-Server Architectural Style
- Multi-Tenant Architectural Style



Architectural Patterns

- Tenant-specific Patterns:
  - API Gateway Architectural Pattern
    - Gateway Aggregation
    - Gateway Routing
    - Gateway Security
- Microservice-specific Patterns:
  - Layered Architectural Pattern
  - Proxy Architectural Pattern
  - Repository Architectural Pattern
  - Event-Driven Architectural Pattern
  - Database-per-service



System Demonstration

Mapping ASRs to Software Architecture

System Demonstration – Mapping to ASRs

- Non-Functional ASRs
  - 2.1 Portability – The system should run on multiple operating systems without requiring extensive changes.
  - 2.2 Maintainability – System Concerns must be separated in separate responsibility layers.
  - 2.3 Reliability – System Components must be fault-tolerant.
- Constraints:
  - 3.1 System must utilize Microservices architectural style.
  - 3.2 GDPR-Compliant User Data Handling (Separation of Concerns, Encryption)

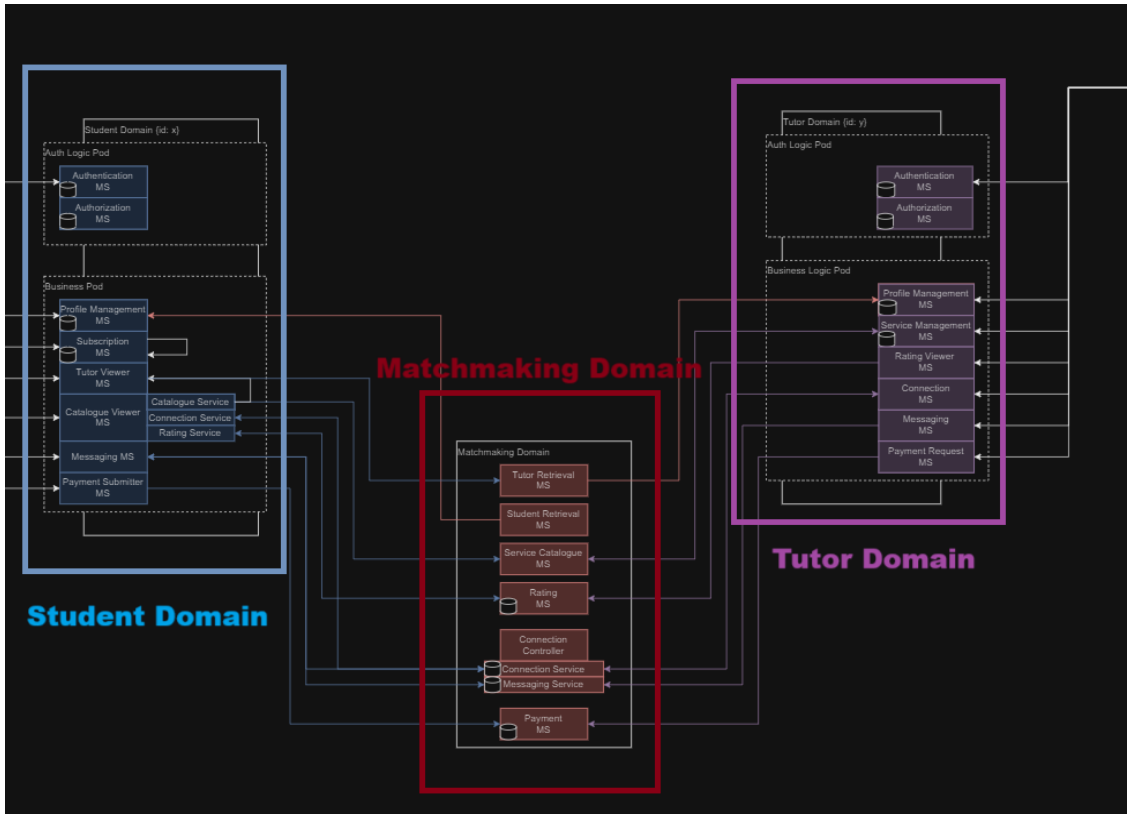
System Demonstration

Functional ASRs

	• 1.1 User Registration for Students/Tutors • 1.2 Tutor information
System Demonstration – Mapping to Functional ASRs	• 1.3 Catalogue Browsing • 1.4 Messaging • 1.5 Student Upgrades Account Tier • 1.6 Course Rating

System Demonstration

Domain-Driven Development



System Demonstration – Domain-Driven Development

Principles:

- Focus on the domain model
- Enable collaboration between domain experts and developers
- Decompose into **bounded contexts** for better modularity

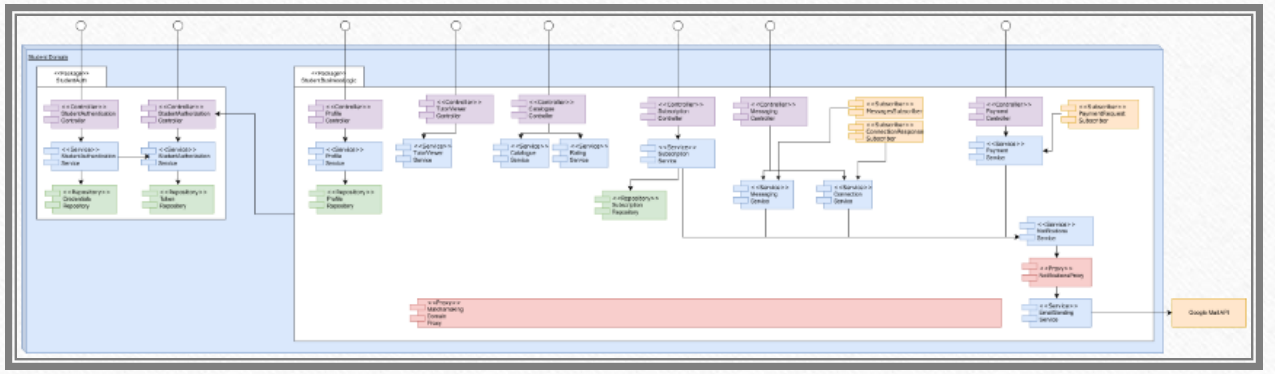
Why DDD?

- **Focus on Business Needs**
- **Improved Team Collaboration**
- **Easier Maintenance and Scaling**
- **Handles Complexity Better**

- Ensures the application solves real-world problems by aligning design with business goals.
- Encourages developers and business experts to speak the same language, reducing misunderstandings.
- Modular domains (like Student, Tutor, Matchmaking) make it simple to add features or scale specific parts of the system.
- Breaks down large systems into smaller, manageable "contexts" that reflect real-world concepts.

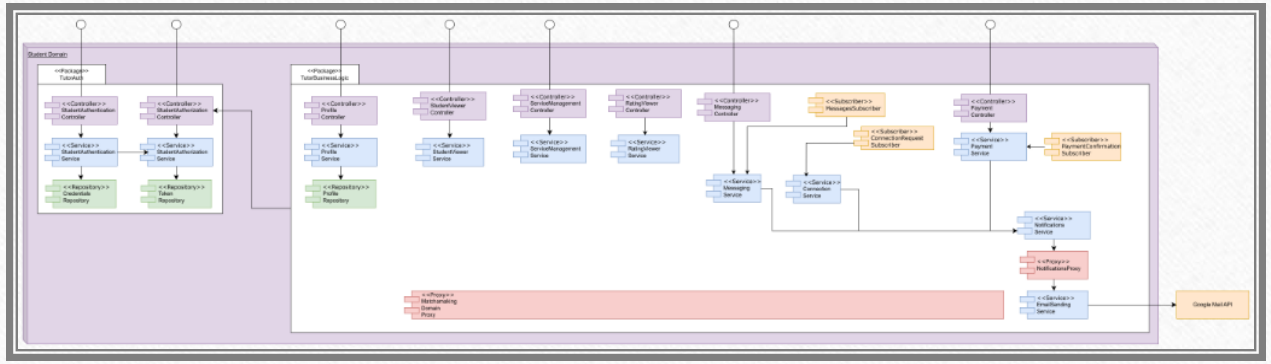
Student Domain

- Create and update student profiles.
- Authenticate and Authorize students.
- View and enroll in courses.



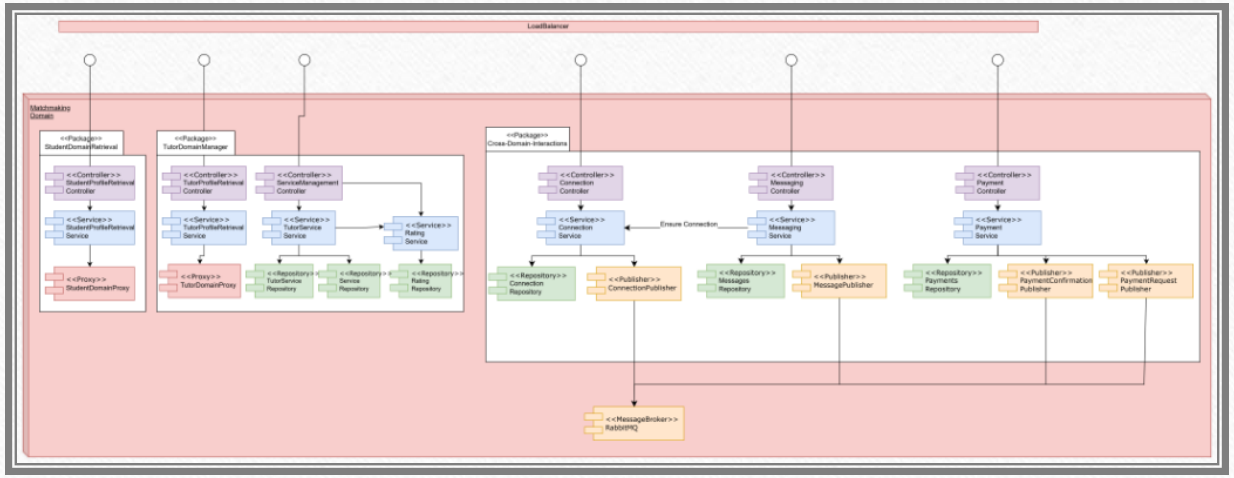
Tutor Domain

- Manage tutor profiles and services.
- Monitor ratings and reviews.



Matchmaking Domain

- Connect students and tutors
- Manage connections
- Manage messaging
- Manage payment



Cross-Domain Features

- Payment handling
- Notifications
- Messaging
- Email integration

Building Around Basic Domain Objects

- Domain Objects as the Foundation:
  - Student: Profile, subscription plan, enrolled courses.
  - Tutor: Profile, offered services, ratings.
  - Course: Name, description, tutor, pricing, location.
  - Connection: Linking students and tutors for sessions.
- Microservices:
  - Each domain object has its own microservice.
  - Microservices interact via proxies and event-driven mechanisms.

Student Auth

Microservice	Endpoint	Description
Student Auth	POST /auth/registration	Register a new student account.
Student Auth	POST /auth/authorize	Authorize a student and create a session token.
Student Auth	GET /auth/validate	Validate a session token.

Student Functional

Microservice	Endpoint	Description
Student Functional	GET /profile/{id}	View a student's profile by ID.
Student Functional	PUT /profile	Update the student's own profile.
Student Functional	GET /profile/subscription	View the student's subscription status.
Student Functional	GET /catalog/courses	View all available courses.
Student Functional	GET /catalog/courses/{courseId}	View details of a specific course.
Student Functional	POST /catalog/courses/{courseId}/enroll	Enroll in a course.
Student Functional	GET /catalog/enrolled	View all courses a student is enrolled in.
Student Functional	POST /messages/send	Send a message between users.
Student Functional	GET /messages/conversation/{userId}/{User2Id}	View all messages in a conversation between two users.

Tutor Auth

Microservice	Endpoint	Description
Tutor Auth	POST /auth/registration	Register a new tutor account.
Tutor Auth	POST /auth/authorize	Authorize a tutor and create a session token.
Tutor Auth	GET /auth/validate	Validate a session token.

Tutor Functional

Microservice	Endpoint	Description
Tutor Functional	GET /tutors/{id}	View a tutor's profile by ID.
Tutor Functional	PUT /tutors/profile	Update the tutor's profile.
Tutor Functional	POST /services	Add a new service offered by the tutor.
Tutor Functional	DELETE /services/{serviceId}	Remove a service offered by the tutor.
Tutor Functional	GET /ratings/{tutorId}	View average ratings for a tutor.
Tutor Functional	POST /ratings	Submit a new rating for a tutor.

Tutor Domain Manager

Microservice	Endpoint	Description
Tutor Domain Manager	GET /tutors/{tutorId}	Retrieve detailed profile information for a specific tutor.
Tutor Domain Manager	GET /tutors	Retrieve a list of all tutors in the system.
Tutor Domain Manager	POST /tutors/fetch-profiles	Request profiles of multiple tutors from the Tutor Domain.

Student Domain Manager

Microservice	Endpoint	Description
Student Domain Manager	GET /students/{studentId}	Retrieve detailed profile information for a specific student.
Student Domain Manager	GET /students	Retrieve a list of all students in the system.
Student Domain Manager	POST /students/fetch-profiles	Request profiles of multiple students from the Student Domain.

Cross Domain Interactions

Microservice	Endpoint	Description
Cross Domain Interactions	POST /connection	Create a new connection between a student and a tutor.
Cross Domain Interactions	GET /connection/student/{studentId}	Retrieve all connections for a specific student.
Cross Domain Interactions	GET /connection/tutor/{tutorId}	Retrieve all connections for a specific tutor.
Cross Domain Interactions	GET /messages/conversation/{userId}/{otherUserId}	Send a message between two users (student and tutor, or student to student).
Cross Domain Interactions	POST /notifications/send	Trigger a notification for a specific event
Cross Domain Interactions	POST /notifications/email	Send an email notification for critical events
Cross Domain Interactions	POST /payment/request	Accept and complete a payment request.
Cross Domain Interactions	POST /payment/{paymentRequestId}/pay	Create a payment request after a tutoring session.
Cross Domain Interactions	POST /payment/{paymentRequestId}/decline	Decline a payment request.

Code Example: Domain-Centric Implementation

```
@Entity
   public class StudentProfile {
       private Long id;
       private String name;
       private String location;
       private String subscriptionPlan;
       // Domain Logic: Profile completion check
       public boolean isComplete() {
           return name != null && location !=
   null;
       }
   }
```

Explanation:

- Core domain logic is encapsulated in the object.
- Data consistency and business rules are enforced within domain models.

System Testing

Unit Tests

- 4 Unit Tests
  - Show Source Code and Demonstrate Execution
    - Execution through GitHub Actions

Integration Tests

- 2 Integration Tests
  - Show Source Code and Demonstrate Execution
    - Execution through Postman

Regression Tests

- 2 Regressions Tests
  - Integration Tests, but on every change (Simple to run)
  - Show Source Code and Demonstrate Execution
    - On GitHub Actions mandatorily on every commit to main
  - Integration Tests:
    - Automatically Attempt Back-end Build
    - Automatically Attempt Front-end Build