**8 sơ đồ chi tiết, được tạo riêng cho hệ thống MathBridge, giúp dễ dàng lập trình bằng Java.**

### **1. Sơ đồ Use Case**

Sơ đồ này thể hiện các chức năng chính của hệ thống và cách các đối tượng (người dùng) tương tác với chúng. Nó giúp bạn xác định các module chính cần được phát triển.

### **2. Sơ đồ Hoạt động (Đặt lịch học)**

Sơ đồ này mô tả chi tiết quy trình đặt một buổi học, từ tìm kiếm gia sư đến thanh toán.

### **3. Sơ đồ Lớp (Quản lý hồ sơ gia sư)**

Đây là cấu trúc dữ liệu của các đối tượng trong hệ thống, đặc biệt là hồ sơ gia sư, giúp bạn định nghĩa các lớp Java.

### **4. Sơ đồ Trình tự (Đăng nhập)**

Sơ đồ này minh họa trình tự trao đổi thông điệp giữa các thành phần khi một người dùng đăng nhập.

### **5. Sơ đồ Hoạt động (Quy trình thanh toán)**

Sơ đồ này tập trung vào luồng thanh toán, từ khi người dùng bắt đầu giao dịch đến khi hoàn tất.

### **6. Sơ đồ Thành phần**

Sơ đồ này thể hiện kiến trúc hệ thống ở cấp độ cao, chia nhỏ ứng dụng thành các thành phần độc lập.

### **7. Sơ đồ Triển khai**

Sơ đồ này mô tả môi trường vật lý nơi hệ thống sẽ được triển khai, bao gồm máy chủ và các container.

### **8. Sơ đồ Trạng thái (Trạng thái buổi học)**

Sơ đồ này minh họa các trạng thái khác nhau của một buổi học và các sự kiện gây ra sự thay đổi trạng thái đó.

**Cách thực hiện và mô hình đơn giản hóa nhất của chương trình**

Chắc chắn rồi. Dựa trên đề bài và các sơ đồ bạn đã cung cấp, tôi sẽ phân tích kiến trúc hệ thống MathBridge theo hướng đơn giản, chặt chẽ và dễ dàng code bằng Java. Chúng ta sẽ sử dụng phương pháp Lập trình Hướng đối tượng (OOP) và kiến trúc đa tầng (Multi-tier Architecture).

### **1. Kiến trúc Đa tầng (Multi-tier Architecture)**

Để đảm bảo tính chặt chẽ, dễ bảo trì và mở rộng, hệ thống nên được chia thành 3 tầng chính:

* **Tầng Giao diện (Presentation Layer):** Gồm giao diện người dùng (Frontend) tương tác với người dùng.
  + **Nhiệm vụ:** Hiển thị dữ liệu, nhận yêu cầu từ người dùng (đặt lịch, tìm kiếm, đăng nhập) và gửi đến tầng logic.
  + **Công nghệ Java:** Bạn có thể sử dụng **JavaServer Pages (JSP)**, **Thymeleaf** hoặc các framework frontend hiện đại như **React** (như đề xuất) để tạo giao diện và gọi các API backend.
* **Tầng Logic nghiệp vụ (Business Logic Layer):** Xử lý toàn bộ logic của hệ thống. Đây là nơi code Java của bạn sẽ hoạt động.
  + **Nhiệm vụ:** Xử lý các yêu cầu từ tầng giao diện, thực hiện các quy tắc nghiệp vụ (ví dụ: kiểm tra lịch trống của gia sư, tính toán học phí), và giao tiếp với tầng dữ liệu.
  + **Công nghệ Java:** Sử dụng **Spring Boot** hoặc **Java EE** để xây dựng các RESTful API. Đây là lựa chọn phổ biến và mạnh mẽ.
* **Tầng Truy cập dữ liệu (Data Access Layer - DAL):** Quản lý việc kết nối và thao tác với cơ sở dữ liệu.
  + **Nhiệm vụ:** Tương tác trực tiếp với cơ sở dữ liệu (SQL, NoSQL), thực hiện các truy vấn (CRUD - Create, Read, Update, Delete). Tầng này không chứa logic nghiệp vụ.
  + **Công nghệ Java:** Sử dụng **JPA (Java Persistence API)** với **Hibernate** hoặc **Spring Data JPA**. Điều này giúp bạn thao tác với cơ sở dữ liệu bằng các đối tượng Java (POJO) thay vì viết câu lệnh SQL thủ công.

### **2. Phân tích Module và Lớp (Class) trong Java**

Dựa trên sơ đồ lớp và các yêu cầu chức năng, chúng ta có thể phân tích thành các module và các lớp cụ thể.

#### **Module Quản lý người dùng**

* **Lớp User (lớp cha):**
  + Các thuộc tính chung: userId, email, password, role (vai trò: STUDENT, PARENT, TUTOR, ADMIN).
* **Lớp Student (kế thừa User):**
  + Thuộc tính riêng: studentId, parentInfo, learningHistory.
* **Lớp Tutor (kế thừa User):**
  + Thuộc tính riêng: tutorId, qualifications, experience, rating.
* **Lớp Admin (kế thừa User):**
  + Thuộc tính riêng: adminId.
* **Lớp AuthenticationService:**
  + Nhiệm vụ: Chứa các phương thức login(), register(), logout(). Sử dụng **JWT** để tạo và xác thực token.

#### **Module Quản lý gia sư**

* **Lớp TutorProfile:**
  + Thuộc tính: tutorId, bio, certifications, subjects (danh sách môn học), hourlyRate.
* **Lớp Review:**
  + Thuộc tính: reviewId, tutorId, studentId, rating (điểm), comment.
* **Lớp TutorService:**
  + Nhiệm vụ: Chứa các phương thức searchTutors(criteria), getTutorProfile(tutorId), updateTutorSchedule().

#### **Module Quản lý lịch học**

* **Lớp Session:**
  + Thuộc tính: sessionId, tutorId, studentId, subject, startTime, endTime, mode (online/offline), status (pending, confirmed, completed...).
* **Lớp ScheduleService:**
  + Nhiệm vụ: Chứa các phương thức bookSession(sessionId), confirmSession(sessionId), cancelSession(sessionId).

#### **Module Quản lý thanh toán**

* **Lớp Payment:**
  + Thuộc tính: paymentId, sessionId, amount, paymentDate, status (success, failed, pending).
* **Lớp PaymentService:**
  + Nhiệm vụ: Chứa các phương thức createPayment(sessionId), handlePaymentWebhook(). Lớp này sẽ gọi đến các API của cổng thanh toán (**PayOS**).

### **3. Hướng dẫn Code Đơn giản và Chặt chẽ**

Bạn nên bắt đầu với một dự án **Spring Boot** để tận dụng các tính năng sẵn có.

1. **Cấu trúc dự án:**
   1. Tạo các package tương ứng với các tầng: com.mathbridge.controller, com.mathbridge.service, com.mathbridge.repository, và com.mathbridge.model.
   2. Package model sẽ chứa các lớp (Class) đại diện cho dữ liệu như User, Tutor, Session, Payment.
   3. Package repository sẽ chứa các interface **Spring Data JPA** để thao tác với cơ sở dữ liệu. Ví dụ: TutorRepository extends JpaRepository<Tutor, Long>.
2. **Viết code từng bước (Iterative Development):**
   1. **Bước 1: Xây dựng cơ bản.** Bắt đầu với module người dùng. Tạo các lớp User, Tutor, Student và UserRepository.
   2. **Bước 2: Triển khai REST API.** Tạo một Controller (UserController) sử dụng @RestController và các Annotation như @GetMapping, @PostMapping để tạo các endpoint API cho việc đăng ký và đăng nhập.
   3. **Bước 3: Phát triển logic nghiệp vụ.** Trong lớp UserService, bạn sẽ viết logic để xử lý việc đăng ký, kiểm tra email trùng lặp, mã hóa mật khẩu.
   4. **Bước 4: Tích hợp các module.** Khi module người dùng hoạt động, tiếp tục với module gia sư, lịch học, và thanh toán theo cùng một quy trình. Mỗi module sẽ có một Controller, Service và Repository riêng, đảm bảo tính độc lập.
3. **Sử dụng Dependency Injection:**
   1. Sử dụng @Autowired trong Spring để tự động kết nối các thành phần (ví dụ: kết nối TutorService với TutorRepository), giúp code gọn gàng và dễ kiểm thử.
4. **Bảo mật:**
   1. Sử dụng **Spring Security** để bảo vệ các endpoint API.
   2. Khi tạo lớp UserService, hãy đảm bảo sử dụng **BCryptPasswordEncoder** để mã hóa mật khẩu trước khi lưu vào cơ sở dữ liệu.

Bằng cách tuân thủ kiến trúc và phân tích trên, bạn có thể dễ dàng xây dựng một hệ thống MathBridge chặt chẽ, có cấu trúc rõ ràng và dễ dàng mở rộng trong tương lai.

**Dựa trên toàn bộ tài liệu về hệ thống MathBridge, cách lập trình bằng Java một cách hợp lý và chặt chẽ, theo mô hình kiến trúc hiện đại để đảm bảo hiệu suất và khả năng mở rộng.**

### **1. Lựa chọn Kiến trúc: Microservices**

Thay vì xây dựng một ứng dụng nguyên khối (monolithic), việc sử dụng kiến trúc **Microservices** sẽ giúp bạn phân chia hệ thống thành các dịch vụ độc lập, dễ phát triển và bảo trì. Mỗi dịch vụ sẽ tập trung vào một chức năng nghiệp vụ cụ thể.

Dưới đây là các dịch vụ chính được đề xuất:

* **Dịch vụ Người dùng (User Service):** Quản lý hồ sơ người dùng (học sinh, gia sư, phụ huynh, quản trị viên), xác thực đăng nhập (JWT, OAuth2) và phân quyền.
* **Dịch vụ Đặt lịch (Booking Service):** Xử lý việc tìm kiếm gia sư, quản lý lịch trống, đặt lịch và hủy buổi học.
* **Dịch vụ Thanh toán (Payment Service):** Xử lý các giao dịch tài chính, tích hợp cổng thanh toán và quản lý các gói học phí.
* **Dịch vụ Giao tiếp Thời gian thực (Real-time Service):** Cung cấp các tính năng chat và thông báo tức thì giữa người dùng.

### **2. Công nghệ và Framework cho Java**

Để xây dựng các dịch vụ trên, **Spring Boot** là lựa chọn tốt nhất. Nó đơn giản hóa việc khởi tạo dự án, quản lý dependency và xây dựng các API RESTful.

* **Backend:** **Spring Boot**
* **Cơ sở dữ liệu:** **JPA (Java Persistence API)** với **Hibernate** để tương tác với cơ sở dữ liệu **Microsoft SQL**.
* **Xác thực:** **Spring Security** kết hợp với **JWT (json-io)** cho API và **Spring OAuth2 Client** cho đăng nhập Google.
* **Caching:** **Redis Cache** với thư viện Spring Data Redis.
* **Giao tiếp giữa các dịch vụ:** Sử dụng **RESTful API** (đồng bộ) hoặc **RabbitMQ/Kafka** (bất đồng bộ) cho các giao tiếp liên dịch vụ.
* **Chat:** Mặc dù đề cập đến SignalR, bạn có thể sử dụng **WebSockets** tích hợp sẵn trong Spring hoặc các thư viện như **Netty** để xây dựng dịch vụ chat.

### **3. Hướng dẫn Lập trình chi tiết (theo mô hình DDD)**

Mỗi dịch vụ nên tuân thủ một cấu trúc code chặt chẽ theo mô hình **Domain-Driven Design (DDD)**. Điều này giúp code sạch, dễ hiểu và dễ bảo trì.

#### **Cấu trúc thư mục của một dịch vụ (ví dụ: User Service)**

* com.mathbridge.user.domain: Nơi chứa các lớp cốt lõi đại diện cho nghiệp vụ, chẳng hạn như User, Tutor, Student.
* com.mathbridge.user.repository: Nơi chứa các giao diện (interface) để tương tác với cơ sở dữ liệu (sử dụng **Spring Data JPA**). Ví dụ: TutorRepository extends JpaRepository<Tutor, Long>.
* com.mathbridge.user.service: Nơi chứa logic nghiệp vụ. Các lớp này sẽ gọi các phương thức từ Repository và xử lý yêu cầu. Ví dụ: TutorService.java.
* com.mathbridge.user.controller: Nơi tiếp nhận các yêu cầu từ phía người dùng (Frontend) và trả về phản hồi qua các API RESTful. Ví dụ: TutorController.java.
* com.mathbridge.user.dto: Nơi chứa các lớp **Data Transfer Object (DTO)** để truyền dữ liệu giữa các lớp một cách an toàn và hiệu quả, tránh lộ dữ liệu nhạy cảm.

#### **Các bước triển khai code**

1. **Khởi tạo Dự án:** Bắt đầu bằng cách tạo một dự án **Spring Boot** mới trên start.spring.io với các dependency cần thiết (Web, JPA, Security, SQL Server Driver, Lombok).
2. **Thiết kế Database:** Định nghĩa các bảng (table) cho từng dịch vụ và các mối quan hệ. Đảm bảo sử dụng các kiểu dữ liệu phù hợp.
3. **Viết code cho từng tầng:**
   1. **Repository Layer:** Tạo các giao diện Repository để Spring tự động tạo các phương thức CRUD cơ bản.
   2. **Domain Layer:** Định nghĩa các lớp Tutor, Student, Session... với các thuộc tính và mối quan hệ đã thiết kế. Sử dụng @Entity và @Table để ánh xạ tới database.
   3. **Service Layer:** Viết logic nghiệp vụ trong các lớp Service. Ví dụ: logic tìm kiếm gia sư, xác thực mật khẩu. Sử dụng @Service và @Transactional.
   4. **Controller Layer:** Tạo các API RESTful bằng @RestController. Các phương thức trong Controller sẽ gọi các phương thức từ Service để thực hiện tác vụ và trả về kết quả.
4. **Triển khai Bảo mật:** Tích hợp **Spring Security** để cấu hình các quy tắc xác thực và phân quyền truy cập. Bạn sẽ cần một bộ lọc (filter) để xử lý **JWT Token** và một lớp để xử lý thông tin người dùng.
5. **Tích hợp Giao tiếp:**
   1. **Booking Service** có thể gọi **User Service** qua một API RESTful để lấy thông tin gia sư.
   2. **Payment Service** sẽ nhận các thông báo từ cổng thanh toán thông qua **Webhook** và gửi một sự kiện đến **Booking Service** (qua Message Queue) để cập nhật trạng thái buổi học.

### **4. Hướng dẫn Code Chặt chẽ**

* **Sử dụng Lombok:** Để giảm thiểu code lặp lại, hãy sử dụng **Lombok** để tự động tạo getter, setter, constructor.
* **Validation:** Thêm các annotation @NotNull, @Size, @Email... để xác thực dữ liệu đầu vào ngay tại lớp DTO, đảm bảo dữ liệu hợp lệ trước khi đi vào logic nghiệp vụ.
* **Xử lý Ngoại lệ:** Xây dựng một lớp xử lý lỗi toàn cục (@ControllerAdvice) để bắt các ngoại lệ và trả về các phản hồi HTTP chuẩn xác. Ví dụ: trả về 404 Not Found khi không tìm thấy tài nguyên.
* **Tên biến và phương thức rõ ràng:** Sử dụng tên biến và phương thức dễ hiểu để người khác (và chính bạn sau này) có thể dễ dàng đọc và bảo trì code.

Bằng cách tuân thủ những nguyên tắc này, bạn sẽ có thể xây dựng một hệ thống MathBridge mạnh mẽ, có cấu trúc tốt và sẵn sàng cho việc mở rộng trong tương lai.