**I. Nội dung lý thuyết: Tìm hiểu công nghệ sử dụng**

**1. Kiến trúc Microservices**

Kiến trúc Microservices là phương pháp phát triển phần mềm bằng cách chia nhỏ hệ thống thành các dịch vụ độc lập, mỗi dịch vụ đảm nhiệm một chức năng riêng biệt và có thể triển khai riêng rẽ.

* Mỗi vi dịch vụ giao tiếp với nhau qua giao thức HTTP hoặc messaging queue (như RabbitMQ, Kafka).
* Microservices thường đi kèm với Service Discovery (Eureka), API Gateway (Spring Cloud Gateway), và các công cụ DevOps (Docker, Kubernetes).
* Ưu điểm:
  + Dễ mở rộng theo chiều ngang (scale service nào cần).
  + Dễ bảo trì, nâng cấp từng phần mà không ảnh hưởng toàn hệ thống.
  + Độc lập về công nghệ nếu cần (dù trong đề tài này bạn dùng chung Java/Spring Boot).
* Nhược điểm:
  + Phức tạp hơn Monolithic trong triển khai, debug, logging và quản lý dữ liệu phân tán.

**2. Spring Boot, RESTful API và MySQL**

* **Spring Boot**:
  + Là framework Java giúp phát triển ứng dụng nhanh, nhẹ, cấu hình tối giản.
  + Tự động cấu hình nhiều thành phần, hỗ trợ tạo RESTful API nhanh chóng.
* **RESTful API**:
  + Giao tiếp qua HTTP, sử dụng các phương thức GET, POST, PUT, DELETE.
  + Dữ liệu thường trao đổi ở định dạng JSON, dễ tích hợp frontend hoặc mobile app.
* **MySQL**:
  + Hệ quản trị cơ sở dữ liệu quan hệ phổ biến, dùng để lưu dữ liệu của từng microservice riêng biệt (mỗi service quản lý schema riêng).
  + Hỗ trợ hiệu quả các truy vấn phức tạp và mối quan hệ giữa thực thể.

**3. RabbitMQ hoặc Kafka**

* **RabbitMQ**:
  + Dễ cài đặt và tích hợp với Spring Boot (qua Spring AMQP).
  + Dùng mô hình Producer – Exchange – Queue – Consumer.
  + Phù hợp cho các tác vụ như gửi thông báo, xử lý nền.
* **Apache Kafka**:
  + Hiệu năng cao, xử lý lượng lớn dữ liệu theo thời gian thực.
  + Mô hình publish-subscribe, dùng topic để phân phối sự kiện đến nhiều consumer.
  + Phù hợp nếu bạn hướng đến quy mô lớn, nhiều sự kiện phức tạp.

**4. Eureka – Service Discovery**

* Là một phần của hệ sinh thái Spring Cloud, đóng vai trò như **trung tâm đăng ký các vi dịch vụ**.
* Mỗi microservice sẽ **tự động đăng ký** vào Eureka khi khởi động.
* Khi một dịch vụ cần gọi một dịch vụ khác, nó sẽ tra cứu thông tin (IP/port) từ Eureka thay vì hard-code.
* Tăng tính linh hoạt và khả năng tự phục hồi khi dịch vụ thay đổi hoặc chết tạm thời.

**5. Spring Cloud Gateway**

* Là **API Gateway** giúp định tuyến các request từ người dùng đến đúng microservice.
* Tích hợp dễ dàng với Eureka để biết dịch vụ nào đang sống.
* Có thể thêm các chức năng:
  + Xác thực và phân quyền.
  + Giới hạn tốc độ request.
  + Log và giám sát luồng dữ liệu ra/vào hệ thống.

**II. Nội dung thực hành: Xác định đối tượng và chức năng**

**✦ Đối tượng sử dụng**

**1. Ban Quản Lý Chung Cư**

* Quản trị hệ thống, phê duyệt tài khoản cư dân.
* Tạo danh mục các dịch vụ cung cấp cho cư dân lựa chọn.
* Định nghĩa mẫu hợp đồng (các điều khoản, thời hạn).
* Tạo hóa đơn hàng tháng dựa trên hợp đồng.
* Theo dõi và xác nhận thanh toán (có thể tích hợp thanh toán điện tử).

**2. Cư dân sống tại chung cư**

* Tạo tài khoản và chờ xét duyệt.
* Xem và lựa chọn các dịch vụ muốn đăng ký (hợp đồng).
* Có thể cập nhật hoặc chấm dứt hợp đồng.
* Nhận thông báo khi hợp đồng sắp hết hạn, đến hạn thanh toán.
* Tra cứu lại hợp đồng và lịch sử thanh toán.
* Thực hiện thanh toán trực tuyến qua cổng thanh toán.

**✦ Chức năng hệ thống**

**a. Dành cho ban quản lý**

* ✅ Duyệt/Cấp/Hủy tài khoản người dùng.
* ✅ Đăng ký các dịch vụ: mô tả, đơn giá, loại dịch vụ.
* ✅ Tạo mẫu hợp đồng gồm danh sách dịch vụ và điều khoản.
* ✅ Tự động sinh hóa đơn mỗi tháng dựa vào hợp đồng đang hoạt động.
* ✅ Xử lý thanh toán cư dân (ghi nhận trạng thái thanh toán thủ công hoặc qua API thanh toán điện tử).

**b. Dành cho cư dân**

* ✅ Tạo tài khoản, đăng nhập.
* ✅ Lập hợp đồng: chọn dịch vụ, nhập thông tin, thời hạn.
* ✅ Chỉnh sửa hợp đồng: thêm, bớt dịch vụ.
* ✅ Xem thông tin hợp đồng, hóa đơn.
* ✅ Nhận thông báo hệ thống (cảnh báo, hết hạn, nhắc nhở).
* ✅ Thanh toán online: chọn hóa đơn, chọn cổng thanh toán, nhận kết quả.

**III. Xác định các thực thể (chưa thiết kế CSDL)**

Dưới đây là các **thực thể (entities)** cần có trong hệ thống — là cơ sở để bạn chuyển sang thiết kế CSDL sau này:

| **Tên thực thể** | **Thuộc tính tiêu biểu** | **Mô tả** |
| --- | --- | --- |
| **User** | userId, username, password, role | Thông tin đăng nhập và phân quyền |
| **Resident** | residentId, fullName, apartmentNo, email, phone | Thông tin cư dân |
| **Service** | serviceId, name, type, price | Dịch vụ do ban quản lý cung cấp |
| **Contract** | contractId, residentId, createdAt, expiredAt, status | Hợp đồng của cư dân |
| **ContractItem** | itemId, contractId, serviceId, quantity | Mỗi dòng dịch vụ trong hợp đồng |
| **Invoice** | invoiceId, contractId, month, totalAmount, isPaid | Hóa đơn sinh theo hợp đồng mỗi tháng |
| **Payment** | paymentId, invoiceId, method, date, status | Thanh toán cho hóa đơn |
| **Notification** | notiId, userId, title, content, isRead, sentAt | Thông báo hệ thống đến cư dân |
| **Admin** (nếu không gộp chung với User) | adminId, name, contact | Tài khoản ban quản lý |