HƯỚNG DẪN THỰC HÀNH SỐ 01

Học phần: Một số công nghệ phát triển phần mềm

Chủ đề: Phát triển phần mềm hướng dịch vụ

Nội dung:

- Kiến trúc hướng dịch vụ SOA (Service Oriented Architecture)
- Kiến trúc vi dịch vụ MSA (Microservice Architecture)

Kiến thức:

SOA - Webservice: SOAP, RESTful

- Sử dụng thư viện JAX-WS, JAX- RS
- Sử dụng Spring Boot

MSA:

- Phương pháp thiết kế: DDD, Clean / Onion/Hexagonal Architecture
- Mô hình: API Gateway, CQRS (Command Query Responsibility Segregation), Event source.
- Công nghệ phát triển dịch vụ: Spring Boot (Java), NodeJS
- Trienr khai: Docker/GitLab CI/CD/ Kubernetes
- Môi trường phát triển: Eclipse, .NET, PyCharm

PHẦN 1: SOA và Webservice

[Sinh viên tự nghiên cứu, tìm hiểu và thực hành ví dụ sử dụng công cụ AI bài 1, 2,]

Bài 1: Nghiên cứu thư viện JAX-WS:

- Các Annotation của JAX-WS
- Các lớp/giao diện quan trọng của JAX-WS
- Kỹ thuật lập trình webservice kiểu SOAP với JAX-WS.

Bài 2: Nghiên cứu thư viện JAX-RS:

- Các Annotation của JAX-RS
- Các lớp/giao diện quan trọng của JAX-RS
- Kỹ thuật (các bước) lập trình webservice kiểu RESTful với JAX-RS.

- Xây dựng REST API và test với công cụ Postman.

Bài 3: Lập trình với Spring Boot (Hello world – hướng dẫn PHỤ LUC-1)

Bài 4: Lập trình Webservice với SOAP sử dụng Spring Boot (Hello world- PHỤ LỤC-2)

Bài 5: Lập trình Webservice với RESTful sử dụng Spring Boot (Hello world – PHỤ LUC -3)

Bài 6: Xây dựng REST API và test với Postman (CRUD) thông qua ví dụ Thương mại điện tử.

PHẦN 2: Thiết kế, lập trình và triển khai ứng dụng Microservice

A. Nghiên cứu lý thuyết

Tìm hiểu các vấn đề sau:

- Phương pháp thiết kế DDD, các kiến trúc: Clean/Onion/Hexagonal Architecture và sự tích hợp DDD với các kiến trúc này.
- Mô hình API Gateway và kết hợp với các mô hình CQRS (Command Query Responsibility Segregation), Event sourrce. Các công cụ nguồn mở dùng làm gateway (KONG, …). Tìm hiểu về KONG.
- Tìm hiểu các công nghệ và kỹ thuật phát triển Microservice:
  + Node.js + MongoDB
  + Spring Boot + PostgreSQL
  + .NET Core + Redis
  + Python Flask + MySQL
- Các phương pháp truyền thông:
  + Đồng bộ: REST API, gRPC
  + Không đồng bộ: Kafka, RabitMQ
  ⇨ Cài đặt, kỹ thuật lập trình.
- Tìm hiểu các công cụ triển khai:
  + Docker, GitLab CI/CD, Kubernetes (K8S), Docker Swarm.

+ Cài đặt và sử dụng

- Các công cụ nguồn mở khác: Zookeeper, Eureka cloud, …

B. Thực hành:

Xây dựng ứng dụng thương mại điện tử với Microservice. Yêu cầu:

- Thiết kế với DDD
- Mô hình API Gateway
- Sử dụng Spring Boot
- Triển khai với Docker, Gitlab CI/CD, Kubernetes.

Với bài toán thương mại điện tử đơn giản với các chức năng nghiệp vụ sau:

- **Catalog Context**: Quản lý sản phẩm, danh mục, tìm kiếm.
- **Ordering Context**: Xử lý giỏ hàng, đặt hàng, xác nhận đơn hàng.
- **Payment Context**: Tích hợp thanh toán (ví dụ: Stripe, PayPal). **S**
- **hipping Context**: Theo dõi vận chuyển, tích hợp với nhà cung cấp như UPS.
- **User Context**: Quản lý tài khoản, xác thực (OAuth, JWT).
- **Inventory Context**: Quản lý kho hàng, cập nhật tồn kho.

Hướng dẫn:

Bước 1: Phân tích Domain và Xác định Bounded Contexts

DDD bắt đầu bằng việc hiểu miền nghiệp vụ của ứng dụng thương mại điện tử. Chúng ta cần xác định các Bounded Contexts (BCs) để phân chia hệ thống thành các microservices.

1. **Hiểu Domain qua Event Storming**:
    - Làm việc với các bên liên quan (domain experts) để xác định các sự kiện domain (Domain Events) như OrderPlaced, PaymentProcessed, ProductAddedToCart.

- Sử dụng Ubiquitous Language để đảm bảo người phát triển (developer) và người kinh doanh (business) sử dụng chung từ vựng (ví dụ: "Order" là đơn hàng, không phải "Cart").
- Kết quả: Một danh sách các sự kiện và hành động (commands) như CreateOrder, AddItemToCart.

2. **Xác định Bounded Contexts**: Dựa trên domain e-commerce, ta phân chia thành các BC sau:
   - **Catalog Context**: Quản lý sản phẩm, danh mục, tìm kiếm.
   - **Ordering Context**: Xử lý giỏ hàng, đặt hàng, trạng thái đơn hàng.
   - **Payment Context**: Xử lý thanh toán (tích hợp Stripe, PayPal).
   - **User Context**: Quản lý tài khoản, xác thực (OAuth2, JWT).
   - **Inventory Context**: Quản lý kho, cập nhật tồn kho.
   - **Shipping Context**: Quản lý vận chuyển, tích hợp với nhà cung cấp logistics.
3. **Mô hình hóa Domain**: Mỗi BC sẽ là một microservice. Trong hướng dẫn này sẽ tập trung vào **Ordering Context** để minh họa cách áp dụng DDD.

Bước 2: Thiết kế Ordering Microservice với DDD

Ordering Microservice chịu trách nhiệm xử lý việc tạo, cập nhật, và quản lý đơn hàng. Chúng ta sẽ áp dụng các khái niệm DDD như Aggregate, Entity, Value Object, Repository, Domain Event, và Application Service.

2.1. Mô hình hóa Domain

- **Aggregate**: Order là Aggregate Root, chứa các OrderItem (Entity) và Address (Value Object).
- **Domain Event**: OrderCreated, OrderItemAdded để thông báo các microservices khác.
- **Repository**: OrderRepository để lưu trữ và truy xuất Order từ database (PostgreSQL).

- **Application Service**: OrderService để xử lý logic nghiệp vụ từ API.

2.2. Cấu trúc Project trong Eclipse

Tạo một dự án Maven trong Eclipse với cấu trúc thư mục theo DDD:

text

```
ordering-service/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── com.example.ordering/
│   │   │   │   ├── domain/         # Domain logic (Entities, Value Objects, Domain Services)
│   │   │   │   ├── application/      # Application Services, DTOs, Commands
│   │   │   │   ├── infrastructure/   # Repositories, DB, Messaging
│   │   │   │   ├── interfaces/       # REST Controllers, API endpoints
│   │   ├── resources/
│   │       ├── application.yml       # Spring Boot configuration
│   ├── test/
├── pom.xml                    # Maven dependencies
├── Dockerfile                 # Docker configuration
```

2.3. Viết mã Java cho Ordering Service

Dưới đây là ví dụ mã cho Ordering Microservice, sử dụng Spring Boot.

//Order.java

```
package com.example.ordering.domain;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
```

```
public class Order {
    private String id;
    private List<OrderItem> items;
    private Address shippingAddress;
    private BigDecimal total;
    public Order(String id, Address shippingAddress) {
        this.id = id;
        this.items = new ArrayList<>();
        this.shippingAddress = shippingAddress;
        this.total = BigDecimal.ZERO;
    }
    public void addItem(OrderItem item) {
        this.items.add(item);
        this.total = this.total.add(item.getPrice());
    }
    public String getId() {
        return id;
    }
    public List<OrderItem> getItems() {
        return items;
    }
    public Address getShippingAddress() {
        return shippingAddress;
    }

    public BigDecimal getTotal() {
        return total;
```

```
    }
}
```

//OrderItem.java

```
package com.example.ordering.domain;
import java.math.BigDecimal;
public class OrderItem {
    private String productId;
    private int quantity;
    private BigDecimal price;
    public OrderItem(String productId, int quantity, BigDecimal price) {
        this.productId = productId;
        this.quantity = quantity;
        this.price = price;
    }
    public String getProductId() {
        return productId;
    }

    public int getQuantity() {
        return quantity;
    }
    public BigDecimal getPrice() {
        return price;
    }
}
```

// Address.java

```
package com.example.ordering.domain;
```

```
public class Address {
    private String street;
    private String city;
    private String postalCode;
    public Address(String street, String city, String postalCode) {
        this.street = street;
        this.city = city;
        this.postalCode = postalCode;
    }
    // Getters
    public String getStreet() {
        return street;
    }
    public String getCity() {
        return city;
    }
    public String getPostalCode() {
        return postalCode;
    }
}
```

//OrderRepository.java

```
package com.example.ordering.infrastructure;
import com.example.ordering.domain.Order;
public interface OrderRepository {
    void save(Order order);
    Order findById(String id);
}
```

//OrderService.java

```
package com.example.ordering.application;
import com.example.ordering.domain.Address;
import com.example.ordering.domain.Order;
import com.example.ordering.domain.OrderItem;
import com.example.ordering.infrastructure.OrderRepository;
import org.springframework.stereotype.Service;
import java.math.BigDecimal;
import java.util.UUID;
@Service
public class OrderService {
   private final OrderRepository orderRepository;
   public OrderService(OrderRepository orderRepository) {
      this.orderRepository = orderRepository;
   }
   public String createOrder(CreateOrderCommand command) {
      Order order = new Order(
         UUID.randomUUID().toString(),
         new Address(command.getStreet(), command.getCity(),
command.getPostalCode())
      );
      for (CreateOrderItemCommand item : command.getItems()) {
         order.addItem(new OrderItem(item.getProductId(), item.getQuantity(),
item.getPrice()));
      }
      orderRepository.save(order);
      // Publish OrderCreated event (e.g., to Kafka)
```

```
        return order.getId();
    }
}
```

// CreateOrderCommand.java

```
package com.example.ordering.application;
import java.math.BigDecimal;
import java.util.List;
public class CreateOrderCommand {
    private String street;
    private String city;
    private String postalCode;
    private List<CreateOrderItemCommand> items;
    public CreateOrderCommand(String street, String city, String postalCode,
List<CreateOrderItemCommand> items) {
        this.street = street;
        this.city = city;
        this.postalCode = postalCode;
        this.items = items;
    }

    public String getStreet() {
        return street;
    }
    public String getCity() {
        return city;
    }
    public String getPostalCode() {
```

```
        return postalCode;
    }
    public List<CreateOrderItemCommand> getItems() {
        return items;
    }
}

class CreateOrderItemCommand {
    private String productId;
    private int quantity;
    private BigDecimal price;
    public CreateOrderItemCommand(String productId, int quantity, BigDecimal price) {
        this.productId = productId;
        this.quantity = quantity;
        this.price = price;
    }
    public String getProductId() {
        return productId;
    }
    public int getQuantity() {
        return quantity;
    }
    public BigDecimal getPrice() {
        return price;
    }
}
```

// OrderController.java

```
package com.example.ordering.interfaces;
import com.example.ordering.application.CreateOrderCommand;
import com.example.ordering.application.OrderService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/api/orders")
public class OrderController {
   private final OrderService orderService;
   public OrderController(OrderService orderService) {
      this.orderService = orderService;
   }
   @PostMapping
   public ResponseEntity<String> createOrder(@RequestBody
CreateOrderCommand command) {
      String orderId = orderService.createOrder(command);
      return ResponseEntity.ok(orderId);
   }
}
```

Bước 3: Thiết kế API Gateway

API Gateway đóng vai trò là điểm vào duy nhất cho các client (web, mobile). Sử dụng Spring Cloud Gateway để định tuyến requests và xử lý authentication.

Appplication.yaml

```
spring:
  cloud:
    gateway:
      routes:
        - id: ordering_route
          uri: lb://ordering-service
          predicates:
            - Path=/api/orders/**
        - id: catalog_route
          uri: lb://catalog-service
          predicates:
            - Path=/api/products/**
      default-filters:
        - TokenRelay
  security:
    oauth2:
      client:
        provider:
          keycloak:
            issuer-uri: http://keycloak:8080/auth/realms/ecommerce
        registration:
          keycloak:
            client-id: ecommerce-gateway
            client-secret: your-client-secret
server:
  port: 8080
```

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: ordering_route
          uri: lb://ordering-service
          predicates:
            - Path=/api/orders/**
        - id: catalog_route
          uri: lb://catalog-service
          predicates:
            - Path=/api/products/**
      default-filters:
        - TokenRelay
  security:
    oauth2:
      client:
        provider:
          keycloak:
            issuer-uri: http://keycloak:8080/auth/realms/ecommerce
        registration:
          keycloak:
            client-id: ecommerce-gateway
            client-secret: your-client-secret
server:
  port: 8080
```

Bước 4: Giao tiếp giữa Microservices

Các microservices giao tiếp qua:

1. **Synchronous Communication**: Sử dụng REST hoặc gRPC khi cần phản hồi tức thì (ví dụ: Ordering Service gọi Catalog Service để lấy thông tin sản phẩm).
2. **Asynchronous Communication**: Sử dụng Kafka để publish/subscribe Domain Events.

4.1. Ví dụ: Gửi Domain Event qua Kafka

Thêm dependency Kafka vào pom.xml:

<project>

```
    <dependencies>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
            <version>3.2.0</version>
        </dependency>
    </dependencies>
</project>
```

Tạo Domain Event:

// OrderCreatedEvent.java

```
package com.example.ordering.domain;
public class OrderCreatedEvent {
    private String orderId;
    private String customerId;
    public OrderCreatedEvent(String orderId, String customerId) {
        this.orderId = orderId;
        this.customerId = customerId;
    }
    public String getOrderId() {
        return orderId;
    }
    public String getCustomerId() {
        return customerId;
    }
}
```

Publish event trong OrderService:

// OrderService.java

```
package com.example.ordering.application;
import com.example.ordering.domain.Address;
import com.example.ordering.domain.Order;
import com.example.ordering.domain.OrderCreatedEvent;
import com.example.ordering.domain.OrderItem;
import com.example.ordering.infrastructure.OrderRepository;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;
import java.util.UUID;
@Service
public class OrderService {
    private final OrderRepository orderRepository;
    private final KafkaTemplate<String, OrderCreatedEvent>
kafkaTemplate;
   public OrderService(OrderRepository orderRepository,
KafkaTemplate<String, OrderCreatedEvent> kafkaTemplate) {
        this.orderRepository = orderRepository;
        this.kafkaTemplate = kafkaTemplate;
    }
    public String createOrder(CreateOrderCommand command) {
        Order order = new Order(
            UUID.randomUUID().toString(),
```

```
            new Address(command.getStreet(), command.getCity(),
command.getPostalCode())
        );
        for (CreateOrderItemCommand item : command.getItems()) {
            order.addItem(new OrderItem(item.getProductId(),
item.getQuantity(), item.getPrice()));
        }
        orderRepository.save(order);
        kafkaTemplate.send("order-created", new
OrderCreatedEvent(order.getId(), "customer-123"));
        return order.getId();
    }
}
```

4.2. Inventory Service lắng nghe event

Inventory Service sẽ subscribe topic order-created để cập nhật kho.

Bước 5: Containerization với Docker

Tạo Dockerfile cho Ordering Service:

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY target/ordering-service-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Bước 6: Thiết lập GitLab CI/CD

Tạo pipeline để build, test, và deploy ứng dụng: *.gitlab-ci.yml*

```
stages:
  - build
```

```
  - test
  - deploy
build_job:
  stage: build
  image: maven:3.8-openjdk-17
  script:
    - mvn clean package
  artifacts:
    paths:
      - target/*.jar

test_job:
  stage: test
  image: maven:3.8-openjdk-17
  script:
    - mvn test
deploy_job:
  stage: deploy
  image: docker:20.10
  services:
    - docker:dind
  script:
    - docker build -t ordering-service:latest .
    - docker push registry.gitlab.com/your-group/ordering-service:latest
```

```yaml
stages:
  - build
  - test
  - deploy

build_job:
  stage: build
  image: maven:3.8-openjdk-17
  script:
    - mvn clean package
  artifacts:
    paths:
      - target/*.jar

test_job:
  stage: test
  image: maven:3.8-openjdk-17
  script:
    - mvn test
```

```yaml
deploy_job:
  stage: deploy
  image: docker:20.10
  services:
    - docker:dind
  script:
    - docker build -t ordering-service:latest .
    - docker push registry.gitlab.com/your-group/ordering-service:latest
```

Bước 7: Triển khai với Kubernetes

Tạo Kubernetes manifests để deploy Ordering Service và API Gateway.

File**: ordering-deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: ordering-service

spec:

replicas: 2

```
  selector:
    matchLabels:
      app: ordering-service
  template:
    metadata:
      labels:
        app: ordering-service
    spec:
      containers:
        - name: ordering-service
          image: registry.gitlab.com/your-group/ordering-service:latest
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: ordering-service
spec:
  selector:
    app: ordering-service
  ports:
    - port: 80
      targetPort: 8080
  type: ClusterIP
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ordering-service
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ordering-service
  template:
    metadata:
      labels:
        app: ordering-service
    spec:
      containers:
        - name: ordering-service
          image: registry.gitlab.com/your-group/ordering-service:lat
          ports:
            - containerPort: 8080
---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: ordering-service
spec:
  selector:
    app: ordering-service
  ports:
    - port: 80
      targetPort: 8080
  type: ClusterIP
```

Tương tự, tạo deployment cho API Gateway.

Để triển khai API Gateway sử dụng Kubernetes, chúng ta cần tạo một file manifest tương tự như file ordering-deployment.yaml đã được cung cấp trước đó. API Gateway (dựa trên Spring Cloud Gateway trong ví dụ) sẽ được container hóa và triển khai như một microservice độc lập, với cấu hình để định tuyến các request đến các microservices khác (như Ordering Service, Catalog Service, v.v.). File manifest sẽ bao gồm một Deployment để chạy ứng dụng và một Service để expose API Gateway trong cluster Kubernetes.

File: **gateway-deployment.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: api-gateway
  template:
    metadata:
      labels:
        app: api-gateway
    spec:
      containers:
        - name: api-gateway
          image: registry.gitlab.com/your-group/api-gateway:latest
          ports:
            - containerPort: 8080
          env:
            - name: SPRING_CLOUD_GATEWAY_ROUTES_ORDERING_URI
              value: "lb://ordering-service"
            - name: SPRING_CLOUD_GATEWAY_ROUTES_CATALOG_URI
              value: "lb://catalog-service"
```

```
        - name: SPRING_SECURITY_OAUTH2_CLIENT_PROVIDER_KEYCLOAK_ISSUER_URI
          value: "http://keycloak:8080/auth/realms/ecommerce"
        resources:
          limits:
            cpu: "500m"
            memory: "512Mi"
          requests:
            cpu: "200m"
            memory: "256Mi"
---
apiVersion: v1
kind: Service
metadata:
  name: api-gateway
  namespace: default
spec:
  selector:
    app: api-gateway
  ports:
    - port: 80
      targetPort: 8080
  type: LoadBalancer
```

Ảnh của file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: api-gateway
  template:
    metadata:
      labels:
        app: api-gateway
    spec:
      containers:
        - name: api-gateway
          image: registry.gitlab.com/your-group/api-gateway:latest
          ports:
            - containerPort: 8080
          env:
            - name: SPRING_CLOUD_GATEWAY_ROUTES_ORDERING_URI
              value: "lb://ordering-service"
            - name: SPRING_CLOUD_GATEWAY_ROUTES_CATALOG_URI
              value: "http://keycloak:8080/auth/realms/ecommerce"
          resources:
            limits:
              cpu: "500m"
              memory: "512Mi"
            requests:
              cpu: "200m"
              memory: "256Mi"
---
apiVersion: v1
kind: Service
metadata:
  name: api-gateway
  namespace: default
spec:
  selector:
    app: api-gateway
  ports:
    - port: 80
      targetPort: 8080
  type: LoadBalancer
```

Bước 8: Cài đặt trong Eclipse

1. **Cài đặt Eclipse**:
    - Sử dụng Eclipse IDE for Java EE Developers.
    - Cài plugin Spring Tools 4 từ Eclipse Marketplace.
2. **Tạo dự án**:
    - File > New > Maven Project, chọn archetype spring-boot-starter.
    - Thêm các dependency cần thiết (Spring Web, Spring Data JPA, Spring Kafka, Spring Cloud Gateway).
3. **Chạy ứng dụng**:
    - Right-click pom.xml > Run As > Maven Install.
    - Run OrderingServiceApplication.java để khởi động microservice.
4. **Debug**:
    - Sử dụng Debug mode trong Eclipse để kiểm tra REST endpoints hoặc Kafka events.

Bước 9: Testing và Monitoring

- **Unit Testing**: Sử dụng JUnit và Mockito để test domain logic.
- **Integration Testing**: Test REST API với Postman hoặc RestAssured.
- **Monitoring**: Tích hợp Prometheus và Grafana để theo dõi metrics.
- **Logging**: Sử dụng ELK Stack để thu thập logs từ các microservices.

Bước 10: Best Practices và Lưu ý

- **Saga Pattern**: Sử dụng Orchestration Saga để quản lý distributed transactions (ví dụ: Order -> Payment -> Inventory).
- **Circuit Breaker**: Áp dụng Resilience4j để xử lý lỗi khi gọi REST giữa các services.
- **Database per Service**: Ordering Service dùng PostgreSQL, Catalog dùng MongoDB, v.v.
- **Security**: Tích hợp Keycloak cho OAuth2, JWT cho authentication.

PHỤ LỤC

PHỤ LỤC -1:

Hướng dẫn này giả định bạn đã cài đặt Eclipse IDE và Java Development Kit (JDK) trên máy tính.

---

1. Cài đặt môi trường cần thiết

Trước khi bắt đầu, hãy đảm bảo bạn đã cài đặt các công cụ sau:

- **JDK**: Phiên bản 11 hoặc cao hơn (khuyến nghị JDK 17). Tải từ Oracle hoặc sử dụng OpenJDK.
- **Eclipse IDE**: Tải phiên bản Eclipse dành cho Java EE hoặc Eclipse IDE for Enterprise Java and Web Developers từ trang chính thức.
- **Maven**: Được tích hợp sẵn trong Eclipse, nhưng bạn có thể cài đặt riêng từ Maven nếu cần.
- **Spring Tool Suite (STS) Plugin** (tùy chọn): Để hỗ trợ tốt hơn cho Spring Boot, bạn có thể cài plugin STS trong Eclipse qua **Help > Eclipse Marketplace > Tìm "Spring Tools 4" > Install**.

---

2. Tạo dự án Spring Boot

Bước 1: Tạo dự án mới

1. Mở **Eclipse**.
2. Vào **File > New > Project**.
3. Trong cửa sổ "New Project", chọn **Spring Boot > Spring Starter Project** (nếu đã cài STS). Nếu không, bạn có thể chọn **Maven Project** và cấu hình thủ công sau.
4. Nhấn **Next** và điền thông tin dự án:
    - **Name**: HelloWorldSpringBoot
    - **Type**: Maven
    - **Java Version**: Chọn phiên bản JDK (ví dụ: 17)
    - **Packaging**: Jar (khuyến nghị cho Spring Boot)
    - **Group Id**: com.example

- **Artifact Id**: helloworld
- Nhấn **Next**.

5. Trong cửa sổ **Dependencies**, chọn các phụ thuộc:
   - **Spring Web**: Để tạo ứng dụng web cơ bản.
   - (Tùy chọn) **Spring Boot DevTools**: Hỗ trợ reload tự động khi code thay đổi.
6. Nhấn **Finish** để Eclipse tự động tạo dự án.

Bước 2: Kiểm tra cấu trúc dự án

Sau khi tạo, cấu trúc thư mục dự án sẽ như sau:

text

```
HelloWorldSpringBoot
├── src/main/java
│   └── com.example.helloworld
│       └── HelloworldApplication.java
├── src/main/resources
│   └── application.properties
├── pom.xml
```

- **HelloworldApplication.java**: Lớp chính để khởi chạy ứng dụng Spring Boot.
- **pom.xml**: File cấu hình Maven, chứa các phụ thuộc và plugin cần thiết.
- **application.properties**: File cấu hình ứng dụng (có thể để trống ban đầu).

Bước 3: Cấu hình file pom.xml

Nếu bạn không sử dụng Spring Starter Project, hãy đảm bảo pom.xml có nội dung sau:

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version> <!-- Phiên bản Spring Boot mới nhất tại thời điểm
10/2025 -->
        <relativePath/>
    </parent>
    <groupId>com.example</groupId>
    <artifactId>helloworld</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>HelloWorldSpringBoot</name>
    <description>Hello World Spring Boot Project</description>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
      </dependency>
    </dependencies>

    <build>
      <plugins>
        <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
      </plugins>
    </build>
</project>
```

Lưu ý: Kiểm tra phiên bản Spring Boot mới nhất tại Maven Central.

---

3. Tạo Controller Hello World

1. Trong thư mục src/main/java/com/example/helloworld, tạo một file mới tên là HelloWorldController.java.
2. Thêm mã sau:

java

```
package com.example.helloworld;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorldController {
```

```
    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, World!";
    }
}
```

- **@RestController**: Đánh dấu lớp này là một REST Controller để xử lý các yêu cầu HTTP.
- **@GetMapping("/hello")**: Xử lý yêu cầu GET tới endpoint /hello.

---

4. Cấu hình ứng dụng (nếu cần)

Mở file src/main/resources/application.properties và thêm cấu hình cơ bản (nếu cần):

properties

```
server.port=8080
```

- Mặc định, Spring Boot chạy trên cổng 8080. Bạn có thể thay đổi cổng nếu cần.

---

5. Dịch và chạy dự án

Bước 1: Dịch dự án

1. Nhấp chuột phải vào dự án trong **Project Explorer** > Chọn **Run As > Maven Build**.
2. Trong cửa sổ "Edit Configuration", nhập clean install vào mục **Goals**.
3. Nhấn **Run** để biên dịch dự án. Maven sẽ tải các phụ thuộc và tạo file JAR trong thư mục target.

Bước 2: Chạy ứng dụng

1. Nhấp chuột phải vào file HelloworldApplication.java > Chọn **Run As > Spring Boot App**.
2. Eclipse sẽ khởi động ứng dụng Spring Boot. Bạn sẽ thấy thông báo trong console, ví dụ:

   text

Started HelloworldApplication in 2.345 seconds (JVM running for 3.123)

Bước 3: Kiểm tra endpoint

1. Mở trình duyệt hoặc công cụ như Postman.
2. Truy cập: http://localhost:8080/hello.
3. Bạn sẽ thấy phản hồi: Hello, World!.

---

6. Debug (nếu cần)

1. Để debug, nhấp chuột phải vào HelloworldApplication.java > Chọn **Debug As > Spring Boot App**.
2. Đặt breakpoint trong HelloWorldController.java (nhấp đúp vào lề trái của dòng code).
3. Truy cập lại http://localhost:8080/hello để kích hoạt debug.

---

7. Một số lưu ý

- **Cập nhật phụ thuộc**: Nếu gặp lỗi liên quan đến phụ thuộc, nhấp chuột phải vào dự án > **Maven > Update Project** để tải lại các thư viện.
- **Spring Boot DevTools**: Nếu đã thêm DevTools, bạn có thể sửa code và ứng dụng sẽ tự động reload khi lưu.
- **Tùy chỉnh thêm**: Bạn có thể thêm các tính năng như cơ sở dữ liệu (Spring Data JPA), bảo mật (Spring Security), hoặc các endpoint REST khác.

PHỤ -LỤC-2:

Hướng dẫn chi tiết tạo, cấu hình, build và chạy dự án Web Service SOAP kiểu Hello World với Spring Boot trong Eclipse

Dưới đây là hướng dẫn đầy đủ, cụ thể để tạo một dự án Web Service SOAP đơn giản kiểu "Hello World" sử dụng Spring Boot trong Eclipse. Tôi sẽ sử dụng cách tiếp cận **contract-first** (bắt đầu từ XSD để định nghĩa contract, sau đó generate code), vì đây là cách được khuyến nghị cho Spring-WS. Dự án này sẽ tạo một service đơn giản nhận tên và trả về lời chào "Hello [name]!" dưới dạng SOAP response.

Lưu ý:

- "Dịch" trong câu hỏi của bạn có lẽ ám chỉ "build" hoặc "compile" (xây dựng dự án). Tôi sẽ giải thích phần build và run.
- Yêu cầu: Bạn cần Eclipse IDE (phiên bản mới nhất, ví dụ Eclipse 2024-09), JDK 17+, và plugin Spring Tools 4 (STS) để hỗ trợ Spring Boot tốt hơn.
- Dự án sử dụng Maven để quản lý dependencies.

Bước 1: Cài đặt Eclipse và plugin cần thiết

1. Tải và cài đặt Eclipse IDE for Enterprise Java and Web Developers từ eclipse.org.
2. Mở Eclipse, vào **Help > Eclipse Marketplace**, tìm kiếm "Spring Tools 4" (hoặc STS), cài đặt và restart Eclipse.
3. Cấu hình JDK: Vào **Window > Preferences > Java > Installed JREs**, thêm JDK 17+ nếu chưa có.

Bước 2: Tạo dự án Spring Boot mới trong Eclipse

(có thể dùng star.spring.io để tạo dự án)

1. Trong Eclipse, chọn **File > New > Spring Starter Project** (nếu có STS plugin; nếu không, sử dụng Spring Initializr online và import).
2. Điền thông tin dự án:
    - Name: soap-hello-world
    - Type: Maven
    - Packaging: JAR
    - Java: 17 (hoặc cao hơn)
    - Group: com.example
    - Artifact: soap-hello-world
    - Description: SOAP Hello World with Spring Boot
3. Chọn dependencies:
    - Spring Boot DevTools (cho hot reload)
    - Spring Web
    - Spring Web Services
4. Click **Next** > **Finish**. Dự án sẽ được tạo và import dependencies tự động.

Nếu không có Spring Starter Project, sử dụng start.spring.io:

- Chọn tương tự, generate ZIP, unzip và import vào Eclipse: **File > Import > Maven > Existing Maven Projects**.

Bước 3: Thêm dependencies bổ sung vào pom.xml

Mở file pom.xml trong dự án, thêm các dependencies sau vào phần <dependencies> (nếu chưa có):

xml

```
<dependency>
   <groupId>wsdl4j</groupId>
   <artifactId>wsdl4j</artifactId>
</dependency>
```

Thêm plugin JAXB để generate Java classes từ XSD:

xml

```
<build>
   <plugins>
      <plugin>
         <groupId>org.codehaus.mojo</groupId>
         <artifactId>jaxb2-maven-plugin</artifactId>
         <version>3.1.0</version>
         <executions>
            <execution>
               <id>xjc</id>
               <goals>
                  <goal>xjc</goal>
               </goals>
            </execution>
         </executions>
         <configuration>
            <sources>
               <source>${project.basedir}/src/main/resources/hello.xsd</source>
```

```
            </sources>
            <outputDirectory>${project.build.directory}/generated-
sources/jaxb</outputDirectory>
            <clearOutputDir>true</clearOutputDir>
          </configuration>
        </plugin>
    </plugins>
</build>
```

Right-click dự án > Maven > Update Project để cập nhật.

Bước 4: Định nghĩa XSD (XML Schema) cho contract

Tạo file hello.xsd trong thư mục src/main/resources (nếu chưa có, right-click > New > Folder > src/main/resources).

Nội dung hello.xsd (định nghĩa request và response cho "Hello World"):

xml

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="http://example.com/soap-hello"
           targetNamespace="http://example.com/soap-hello"
           elementFormDefault="qualified">

    <xs:element name="getHelloRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="getHelloResponse">
        <xs:complexType>
```

```
        <xs:sequence>
            <xs:element name="message" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XSD này định nghĩa:

- Request: getHelloRequest với param name.
- Response: getHelloResponse với message (ví dụ: "Hello [name]!").

Bước 5: Generate Java classes từ XSD

1. Right-click dự án > Run As > Maven build...
2. Goals: clean compile.
3. Chạy. Các class sẽ được generate trong target/generated-sources/jaxb/com/example/soaphello (ví dụ: GetHelloRequest, GetHelloResponse).

Thêm thư mục generated vào source path: Right-click thư mục target/generated-sources/jaxb > Build Path > Use as Source Folder.

Bước 6: Tạo Endpoint cho service

Tạo package com.example.soaphelloworld.endpoint (Right-click src/main/java > New > Package).

Tạo class HelloEndpoint.java:

java

```
package com.example.soaphelloworld.endpoint;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;
```

```java
import com.example.soaphello.GetHelloRequest;
import com.example.soaphello.GetHelloResponse;

@Endpoint
public class HelloEndpoint {
    private static final String NAMESPACE_URI = "http://example.com/soap-hello";

    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "getHelloRequest")
    @ResponsePayload
    public GetHelloResponse getHello(@RequestPayload GetHelloRequest request) {
        GetHelloResponse response = new GetHelloResponse();
        response.setMessage("Hello " + request.getName() + "!");
        return response;
    }
}
```

Lớp này xử lý request SOAP và trả về response.

Bước 7: Cấu hình Web Service

Tạo package com.example.soaphelloworld.config.

Tạo class WebServiceConfig.java:

java

```java
package com.example.soaphelloworld.config;

import org.springframework.boot.web.servlet.ServletRegistrationBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.ws.config.annotation.EnableWs;
import org.springframework.ws.config.annotation.WsConfigurerAdapter;
```

```
import org.springframework.ws.transport.http.MessageDispatcherServlet;
import org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;
import org.springframework.xml.xsd.SimpleXsdSchema;
import org.springframework.xml.xsd.XsdSchema;

@EnableWs
@Configuration
public class WebServiceConfig extends WsConfigurerAdapter {

    @Bean
    public ServletRegistrationBean<MessageDispatcherServlet>
messageDispatcherServlet(ApplicationContext applicationContext) {
        MessageDispatcherServlet servlet = new MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean<>(servlet, "/ws/*");
    }

    @Bean(name = "hello")
    public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema helloSchema) {
        DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("HelloPort");
        wsdl11Definition.setLocationUri("/ws");
        wsdl11Definition.setTargetNamespace("http://example.com/soap-hello");
        wsdl11Definition.setSchema(helloSchema);
        return wsdl11Definition;
    }

    @Bean
```

```
    public XsdSchema helloSchema() {
        return new SimpleXsdSchema(new ClassPathResource("hello.xsd"));
    }
}
```

Cấu hình này đăng ký servlet cho SOAP tại /ws/* và expose WSDL tại /ws/hello.wsdl.

Bước 8: Build (Compile) dự án

1. Right-click dự án > Maven > Update Project.
2. Right-click > Run As > Maven build... > Goals: clean install.
3. Kiểm tra lỗi trong console. Nếu thành công, file JAR sẽ ở target/soap-hello-world-0.0.1-SNAPSHOT.jar.

Bước 9: Chạy dự án trong Eclipse

1. Right-click class chính SoapHelloWorldApplication.java (tạo tự động) > Run As > Spring Boot App.
2. Ứng dụng chạy tại http://localhost:8080.
3. Kiểm tra WSDL: Mở browser, truy cập http://localhost:8080/ws/hello.wsdl. Nếu thấy XML WSDL, thành công.

Bước 10: Test service

Sử dụng công cụ như Postman hoặc SoapUI:

1. Tạo request SOAP (POST đến http://localhost:8080/ws).
2. Header: Content-Type: text/xml.
3. Body (XML request):

xml

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:hel="http://example.com/soap-hello">
  <soapenv:Header/>
  <soapenv:Body>
    <hel:getHelloRequest>
      <hel:name>World</hel:name>
    </hel:getHelloRequest>
```

```
  </soapenv:Body>
</soapenv:Envelope>
```

Response mong đợi:

xml

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:getHelloResponse xmlns:ns2="http://example.com/soap-hello">
      <ns2:message>Hello World!</ns2:message>
    </ns2:getHelloResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Nếu gặp lỗi:

- Kiểm tra console Eclipse.
- Đảm bảo port 8080 không bị chiếm.
- Update Maven nếu dependencies lỗi.

PHU LUC-3:

Hướng dẫn chi tiết tạo dự án Spring Boot RESTful Hello World sử dụng start.spring.io, cấu hình, build và chạy trong Eclipse

Dưới đây là hướng dẫn **chi tiết, cụ thể và đầy đủ** để tạo một dự án Webservice kiểu RESTful đơn giản "Hello World" với Spring Boot. Chúng ta sẽ sử dụng công cụ **start.spring.io** để khởi tạo dự án, sau đó import vào Eclipse, cấu hình, build (dịch) và chạy dự án.

Yêu cầu trước khi bắt đầu:

- Máy tính đã cài **JDK 17 hoặc cao hơn** (Spring Boot 3.x yêu cầu JDK 17+). Kiểm tra bằng lệnh java -version trong terminal.

- **Eclipse IDE** đã cài đặt (tốt nhất là Eclipse phiên bản mới nhất, như Eclipse 2024-09). Nếu chưa có plugin hỗ trợ Spring, hãy cài **Spring Tools 4 (STS)** từ Eclipse Marketplace (Help > Eclipse Marketplace > Tìm "Spring Tools 4" và cài đặt).
- Kết nối internet để tải dependency từ Maven.

Dự án này sẽ tạo một API RESTful đơn giản trả về "Hello World" khi truy cập endpoint /hello.

Bước 1: Tạo dự án Spring Boot trên start.spring.io

1. Mở trình duyệt web và truy cập địa chỉ: **https://start.spring.io**.
2. Cấu hình các thông số dự án như sau (chọn các tùy chọn mặc định nếu không chỉ định):
    - **Project**: Chọn "Maven Project" (hoặc Gradle nếu bạn quen dùng, nhưng hướng dẫn này dùng Maven).
    - **Language**: Java.
    - **Spring Boot**: Chọn phiên bản mới nhất (ví dụ: 3.3.4 hoặc cao hơn, tùy thời điểm).
    - **Project Metadata**:
        - Group: com.example (mặc định).
        - Artifact: hello-world-rest (tên dự án).
        - Name: HelloWorldRest (tên class chính).
        - Description: RESTful Hello World with Spring Boot (tùy chọn).
        - Package name: com.example.helloworldrest (tự động).
        - Packaging: Jar (mặc định).
        - Java: 17 hoặc 21 (tùy JDK của bạn).
    - **Dependencies**: Nhấn "Add Dependencies" và thêm:
        - **Spring Web** (để hỗ trợ RESTful API).
        - (Tùy chọn: Thêm **Spring Boot DevTools** để hỗ trợ hot reload khi phát triển).
3. Nhấn **Generate** (nút xanh ở dưới). Dự án sẽ được tải về dưới dạng file ZIP (ví dụ: hello-world-rest.zip).

4. Giải nén file ZIP vào một thư mục trên máy tính (ví dụ: C:\Projects\hello-world-rest).

Bước 2: Import dự án vào Eclipse

1. Mở Eclipse IDE.
2. Chọn **File > Import**.
3. Trong hộp thoại Import, tìm và chọn **Maven > Existing Maven Projects**.
4. Nhấn **Next**.
5. Trong **Root Directory**, browse đến thư mục đã giải nén (ví dụ: C:\Projects\hello-world-rest).
6. Eclipse sẽ tự động phát hiện file pom.xml. Đảm bảo dự án được chọn, sau đó nhấn **Finish**.
7. Chờ Eclipse tải dependency từ Maven (có thể mất vài phút lần đầu). Nếu có lỗi, right-click dự án > Maven > Update Project.

Lúc này, dự án đã được import. Cấu trúc dự án cơ bản:

- src/main/java/com/example/helloworldrest/HelloWorldRestApplication.java: Class chính để chạy ứng dụng.
- src/main/resources/application.properties: File cấu hình.
- pom.xml: File Maven quản lý dependency.

Bước 3: Cấu hình dự án (Configure)

1. **Thêm Controller cho RESTful API**:
    - Right-click thư mục src/main/java/com/example/helloworldrest > New > Class.
    - Tên class: HelloController.
    - Package: com.example.helloworldrest.controller (tạo package mới nếu cần).
    - Nội dung class (copy-paste mã sau):

      java

      ```
      package com.example.helloworldrest.controller;


      import org.springframework.web.bind.annotation.GetMapping;
      ```

```
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello World from Spring Boot RESTful API!";
    }
}
```

- Giải thích:
  - @RestController: Đánh dấu class này là controller RESTful.
  - @GetMapping("/hello"): Endpoint GET tại /hello sẽ trả về chuỗi "Hello World...".

2. **Cấu hình file application.properties** (nếu cần):
   - Mở src/main/resources/application.properties.
   - Thêm dòng: server.port=8080 (cổng chạy mặc định là 8080, bạn có thể thay đổi nếu cần).
   - Lưu file.
3. **Cấu hình Run Configuration (nếu cần tùy chỉnh)**:
   - Right-click class HelloWorldRestApplication.java > Run As > Spring Boot App (nếu có STS plugin).
   - Hoặc: Run > Run Configurations > Tạo mới "Spring Boot App" và chọn dự án.
4. **Kiểm tra dependency trong pom.xml**:
   - Mở pom.xml và đảm bảo có dependency sau (đã thêm từ start.spring.io):

```
xml
<dependency>
    <groupId>org.springframework.boot</groupId>
```

<artifactId>spring-boot-starter-web</artifactId>
</dependency>

- Nếu thiếu, thêm vào phần <dependencies> và update Maven.

Bước 4: Build (Dịch) dự án

1. Right-click dự án > Maven > Clean (để xóa build cũ).
2. Right-click dự án > Maven > Install (hoặc Build).
   - Điều này sẽ compile mã nguồn, tải dependency nếu thiếu, và tạo file JAR trong thư mục target (ví dụ: target/hello-world-rest-0.0.1-SNAPSHOT.jar).
3. Kiểm tra lỗi: Xem console Eclipse. Nếu có lỗi (ví dụ: JDK không khớp), sửa bằng cách right-click dự án > Properties > Java Build Path > Edit JDK.

Bước 5: Chạy (Run) dự án

1. **Chạy từ Eclipse**:
   - Right-click class HelloWorldRestApplication.java (có annotation @SpringBootApplication).
   - Chọn **Run As > Spring Boot App**.
   - Chờ console hiển thị thông báo: Tomcat started on port(s): 8080 (http) hoặc tương tự. Ứng dụng đang chạy!
2. **Kiểm tra API**:
   - Mở trình duyệt web hoặc công cụ như Postman.
   - Truy cập URL: http://localhost:8080/hello.
   - Kết quả mong đợi: Trang hiển thị chuỗi "Hello World from Spring Boot RESTful API!".
   - Nếu lỗi 404, kiểm tra endpoint trong controller.
3. **Chạy từ command line (tùy chọn, để kiểm tra)**:
   - Mở terminal, đi đến thư mục dự án.
   - Chạy lệnh: mvn clean install (build lại).
   - Sau đó: java -jar target/hello-world-rest-0.0.1-SNAPSHOT.jar.
   - Truy cập URL như trên.

Bước 6: Debug và Troubleshooting (Xử lý lỗi phổ biến)

- **Lỗi import dependency**: Right-click dự án > Maven > Update Project > Check "Force Update".
- **Lỗi port bị chiếm**: Thay đổi server.port trong application.properties thành 8081 hoặc kill process chiếm port (Windows: netstat -ano | findstr :8080 rồi taskkill).
- **Lỗi JDK**: Đảm bảo Project Facets là Java 17+ (Properties > Project Facets).
- **Không có Spring Boot App trong Run As**: Cài lại Spring Tools 4 plugin.
- **Hot reload không hoạt động**: Thêm dependency DevTools và restart.
- Nếu dự án lớn hơn, sử dụng logging: Thêm @Slf4j vào controller và dùng log.info().

Dự án này là cơ bản nhất cho RESTful Hello World. Nếu bạn muốn mở rộng (thêm database, security,...),