**1. Notification Service**

**Database Schema**

* **Notification Table**  
  This table will store details about each notification.

sql

Copy code

CREATE TABLE notifications (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

user\_id BIGINT NOT NULL,

type VARCHAR(50) NOT NULL, -- e.g., "Order Confirmation", "Promotion"

message TEXT NOT NULL,

status VARCHAR(20) DEFAULT 'Pending', -- "Pending" or "Sent"

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

sent\_at TIMESTAMP NULL

);

* Optionally, create an **index on status** for faster retrieval of pending notifications:

sql

Copy code

CREATE INDEX idx\_notifications\_status ON notifications(status);

**Cron Job Setup**

1. Use Spring’s @Scheduled annotation to process notifications periodically:
   * Enable scheduling in your Spring Boot application by adding @EnableScheduling to your main class:

java

Copy code

@SpringBootApplication

@EnableScheduling

public class NotificationServiceApplication {

public static void main(String[] args) {

SpringApplication.run(NotificationServiceApplication.class, args);

}

}

1. Define the cron job in a service class:

java

Copy code

@Service

public class NotificationProcessor {

@Autowired

private NotificationRepository notificationRepository;

@Scheduled(fixedRate = 60000) // Run every 1 minute

public void processPendingNotifications() {

List<Notification> pendingNotifications = notificationRepository.findByStatus("Pending");

for (Notification notification : pendingNotifications) {

boolean isSent = sendNotification(notification);

if (isSent) {

notification.setStatus("Sent");

notification.setSentAt(LocalDateTime.now());

notificationRepository.save(notification);

}

}

}

private boolean sendNotification(Notification notification) {

// Example logic: Email or SMS sending

System.out.println("Sending notification to user ID: " + notification.getUserId());

// Return true if the notification was successfully sent

return true;

}

}

1. Add a repository interface:

java

Copy code

public interface NotificationRepository extends JpaRepository<Notification, Long> {

List<Notification> findByStatus(String status);

}

**REST API Endpoints**

* **POST /notifications**: Create a new notification.
* **GET /notifications**: Fetch notifications for a user.

Example Controller:

java

Copy code

@RestController

@RequestMapping("/notifications")

public class NotificationController {

@Autowired

private NotificationRepository notificationRepository;

@PostMapping

public ResponseEntity<Notification> createNotification(@RequestBody Notification notification) {

notification.setCreatedAt(LocalDateTime.now());

Notification savedNotification = notificationRepository.save(notification);

return ResponseEntity.status(HttpStatus.CREATED).body(savedNotification);

}

@GetMapping("/{userId}")

public ResponseEntity<List<Notification>> getUserNotifications(@PathVariable Long userId) {

List<Notification> userNotifications = notificationRepository.findByUserId(userId);

return ResponseEntity.ok(userNotifications);

}

}

**2. Order Service**

**Database Schema**

* **Order Table**

sql

Copy code

CREATE TABLE orders (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

user\_id BIGINT NOT NULL,

status VARCHAR(20) DEFAULT 'Pending', -- "Pending", "Completed", etc.

total\_price DECIMAL(10, 2) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

* **Order Item Table**

sql

Copy code

CREATE TABLE order\_items (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

order\_id BIGINT NOT NULL,

product\_id BIGINT NOT NULL,

quantity INT NOT NULL,

price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (order\_id) REFERENCES orders(id)

);

**REST API Endpoints**

* **POST /orders**: Create a new order.
* **GET /orders/{id}**: Fetch order details by ID.
* **GET /users/{userId}/orders**: Fetch all orders for a user.

Example Controller:

java

Copy code

@RestController

@RequestMapping("/orders")

public class OrderController {

@Autowired

private OrderRepository orderRepository;

@Autowired

private OrderItemRepository orderItemRepository;

@PostMapping

public ResponseEntity<Order> createOrder(@RequestBody OrderRequestDto orderRequest) {

Order order = new Order();

order.setUserId(orderRequest.getUserId());

order.setTotalPrice(orderRequest.getTotalPrice());

order.setStatus("Pending");

order.setCreatedAt(LocalDateTime.now());

Order savedOrder = orderRepository.save(order);

for (OrderItemRequestDto item : orderRequest.getItems()) {

OrderItem orderItem = new OrderItem();

orderItem.setOrderId(savedOrder.getId());

orderItem.setProductId(item.getProductId());

orderItem.setQuantity(item.getQuantity());

orderItem.setPrice(item.getPrice());

orderItemRepository.save(orderItem);

}

return ResponseEntity.status(HttpStatus.CREATED).body(savedOrder);

}

@GetMapping("/{id}")

public ResponseEntity<Order> getOrderById(@PathVariable Long id) {

Optional<Order> order = orderRepository.findById(id);

return order.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@GetMapping("/users/{userId}")

public ResponseEntity<List<Order>> getOrdersByUser(@PathVariable Long userId) {

List<Order> userOrders = orderRepository.findByUserId(userId);

return ResponseEntity.ok(userOrders);

}

}

**Repositories**

java

Copy code

public interface OrderRepository extends JpaRepository<Order, Long> {

List<Order> findByUserId(Long userId);

}

public interface OrderItemRepository extends JpaRepository<OrderItem, Long> {

List<OrderItem> findByOrderId(Long orderId);

}

Would you like more details on integration with the resource server or specific improvements?

4o