Axon with Saga orchestration:

Saga class:

**Order microservice:**

**Start flow > Place Order:**

OrderCommandRestController:

**@RestController**

**@RequestMapping("/orders")**

public class OrdersCommandController {

**private final CommandGateway commandGateway;**

**private final QueryGateway queryGateway;**

public OrdersCommandController(CommandGateway commandGateway,

QueryGateway queryGateway) {

this.commandGateway = commandGateway;

this.queryGateway=queryGateway;

}

@PostMapping

public OrderSummary createOrder(@Valid **@RequestBody OrderCreateRes**t order) {

String userId = "27b95829-4f3f-4ddf-8983-151ba010e35b";

String orderId = UUID.randomUUID().toString();

CreateOrderCommand createOrderCommand = CreateOrderCommand.builder().addressId(order.getAddressId()) .productId(order.getProductId()).userId(userId).quantity(order.getQuantity()).orderId(orderId) .orderStatus(OrderStatus.CREATED).build();

**// FindOrderQuery is my created class. It is standard in CRQS pattern that we should used separate pojo to define criteria while searching data.**

FindOrderQuery **findOrderQuery**=new FindOrderQuery(orderId);

// **SubscriptionQueryResult we use this API to return the immediate result of createOrder Event to client.**

**SubscriptionQueryResult**<OrderSummary, OrderSummary> result=**queryGateway.subscriptionQuery**(**findOrderQuery**,

ResponseTypes.instanceOf(OrderSummary.class)

,ResponseTypes.instanceOf(OrderSummary.class));

try{

**commandGateway.sendAndWait(createOrderCommand);**

return result.updates().blockFirst();

}finally { result.close(); }

}

}

**Saga class:**

@Saga  
public class **OrderSaga** {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrderSaga.class);  
  
 @Autowired  
 private transient CommandGateway commandGateway;  
  
 @Autowired  
 private transient DeadlineManager deadlineManager;  
  
 private final String PAYMENT\_PROCESSING\_TIMEOUT\_DEADLINE="payment-processing-deadline";  
 @Autowired  
 private transient QueryGateway queryGateway;  
  
 @Autowired  
 private transient QueryUpdateEmitter queryUpdateEmitter;  
  
 private String scheduleId;  
  
 @StartSaga  
 @SagaEventHandler(associationProperty = "orderId")  
 public void handle(OrderCreatedEvent orderCreatedEvent){  
 **//on OrderCreatedEvent, as flow we need to reserve the product.  
 //so for that we need to raise a ReserveProductCommand to product microservice.**ReserveProductCommand reserveProductCommand = ReserveProductCommand.*builder*()  
 .orderId(orderCreatedEvent.getOrderId())  
 .productId(orderCreatedEvent.getProductId())  
 .quantity(orderCreatedEvent.getQuantity())  
 .userId(orderCreatedEvent.getUserId())  
 .build();  
  
 *LOGGER*.info("OrderCreatedEvent handled for orderId: " + reserveProductCommand.getOrderId() +  
 " and productId: " + reserveProductCommand.getProductId() );  
  
 commandGateway.send(reserveProductCommand,(commandMessage,commandResultMessage)->{  
 if(commandResultMessage.isExceptional()){  
 **//start compensating transaction.**RejectOrderCommand rejectOrderCommand=  
 new RejectOrderCommand(orderCreatedEvent.getOrderId(),  
 commandResultMessage.exceptionResult().getLocalizedMessage());  
 commandGateway.send(rejectOrderCommand);  
 }  
 });  
 }  
  
 @SagaEventHandler(associationProperty = "orderId")  
 public void handle(ProductReservedEvent productReservedEvent){  
 **//process user payment***.  
 LOGGER*.info("ProductReservedEvent is called for productId: "+ productReservedEvent.getProductId() +  
 " and orderId: " + productReservedEvent.getOrderId());  
  
 FetchUserPaymentDetailsQuery fetchUserPaymentDetailsQuery=  
 new FetchUserPaymentDetailsQuery(productReservedEvent.getUserId());  
  
 User user=null;  
 try{  
 user=queryGateway.query(fetchUserPaymentDetailsQuery, ResponseTypes.*instanceOf*(User.class)).join();  
 }catch (Exception e){  
 *LOGGER*.error(e.getLocalizedMessage());  
 **//start compensating transaction**cancelProductReservation(productReservedEvent,e.getLocalizedMessage());  
 return;  
 }  
  
 if(user==null){  
 ***//start compensating transaction***cancelProductReservation(productReservedEvent,"Could not able to fetch user payment detail");  
 return;  
 }  
  
 *LOGGER*.info("Successfully fetch user detail..:"+user.getPaymentDetails());  
  
 scheduleId = deadlineManager.schedule(Duration.*of*(120, ChronoUnit.*SECONDS*),  
 PAYMENT\_PROCESSING\_TIMEOUT\_DEADLINE, productReservedEvent);  
  
 *//if(true) return;* ProcessPaymentCommand proccessPaymentCommand = ProcessPaymentCommand.*builder*()  
 .orderId(productReservedEvent.getOrderId())  
 .paymentDetails(user.getPaymentDetails())  
 .paymentId(UUID.*randomUUID*().toString())  
 .build();  
  
 String result="";  
 try {  
 result=commandGateway  
 .sendAndWait(proccessPaymentCommand,10, TimeUnit.*SECONDS*);  
 }catch (Exception e){  
 *LOGGER*.error(e.getLocalizedMessage());  
 ***//start compensating transaction***cancelProductReservation(productReservedEvent,e.getLocalizedMessage());  
 }  
  
 if(result==null)  
 {  
 *LOGGER*.info("Result is null start compensating transaction");  
 ***//start compensating transaction***cancelProductReservation(productReservedEvent  
 ,"Could not process payment with provided payment detail");  
 }  
 }  
  
 @SagaEventHandler(associationProperty = "orderId")  
 public void handle(PaymentProcessedEvent paymentProcessedEvent){  
  
 cancelDeadline();  
  
 ***//now till here my payment is processed, so as next part  
 //I need to change order status from Created to Approve.  
 //then after I have ship the product on given address.*** *LOGGER*.info("PaymentProcessedEvent is handled.......");  
  
 ApproveOrderCommand approveOrderCommand=new ApproveOrderCommand(paymentProcessedEvent.getOrderId());  
 commandGateway.send(approveOrderCommand);  
 }  
  
 @EndSaga  
 @SagaEventHandler(associationProperty = "orderId")  
 public void handle(OrderApprovedEvent orderApprovedEvent){  
 *LOGGER*.info("happy path end here: " +  
 "Order is Approved, order saga is completed for order id" +  
 ":"+orderApprovedEvent.getOrderId());  
  
 ***//you can use SagaLifecycle.end() instead of @EndSaga,  
 //When to use: SagaLifecycle.end()  
 //If in certain condition is true then end the life cycle. then use this.  
  
 //BUt note that once SagaLifecycle.end() Or @EndSaga is used and it is called  
 // then saga life cycle is ended and this instance of order saga will not  
 //able to handle new events anymore.  
  
 //query emitter for updated result set.***queryUpdateEmitter.emit(FindOrderQuery.class  
 ,query->true  
 ,new OrderSummary(orderApprovedEvent.getOrderId()  
 ,orderApprovedEvent.getOrderStatus()  
 ,""));  
 }  
  
 @SagaEventHandler(associationProperty = "orderId")  
 public void handle(ProductReservationCancelledEvent productReservationCancelledEvent){  
 *LOGGER*.info("TODO: create and send RejectOrderCommand");  
 RejectOrderCommand rejectOrderCommand=  
 new RejectOrderCommand(productReservationCancelledEvent.getOrderId(),  
 productReservationCancelledEvent.getReason());  
 commandGateway.send(rejectOrderCommand);  
 }  
  
 @EndSaga  
 @SagaEventHandler(associationProperty = "orderId")  
 public void handle(OrderRejectedEvent orderRejectedEvent){  
 *LOGGER*.info("TODO: Received order rejected event, now what next.");  
  
 ***//query emitter for updated result set.***queryUpdateEmitter.emit(FindOrderQuery.class  
 ,query->true  
 ,new OrderSummary(orderRejectedEvent.getOrderId()  
 ,orderRejectedEvent.getOrderStatus(),  
 orderRejectedEvent.getReason()));  
 }  
  
 private void cancelProductReservation(ProductReservedEvent productReservedEvent, String reason) {  
  
 cancelDeadline();  
  
 CancelProductReservationCommand publishProductReservationCommand =  
 CancelProductReservationCommand.*builder*()  
 .orderId(productReservedEvent.getOrderId())  
 .productId(productReservedEvent.getProductId())  
 .quantity(productReservedEvent.getQuantity())  
 .userId(productReservedEvent.getUserId())  
 .reason(reason)  
 .build();  
  
 commandGateway.send(publishProductReservationCommand);  
  
 }  
  
 @DeadlineHandler(deadlineName=PAYMENT\_PROCESSING\_TIMEOUT\_DEADLINE)  
 public void handlePaymentDeadline(ProductReservedEvent productReservedEvent) {  
 *LOGGER*.info("Payment processing deadline took place. Sending a compensating " +  
 "command to cancel the product reservation");  
 cancelProductReservation(productReservedEvent, "Payment timeout");  
 }  
  
 private void cancelDeadline() {  
 if (scheduleId != null) {  
 deadlineManager.cancelSchedule(PAYMENT\_PROCESSING\_TIMEOUT\_DEADLINE, scheduleId);  
 scheduleId = null;  
 }  
 }  
}

**Order Aggregator:**

**@Aggregate**

public class **OrderAggregate** {

**@AggregateIdentifier**

private String orderId; private String productId; private String userId;

private int quantity; private String addressId; private OrderStatus orderStatus;

//constructor

**@CommandHandler**

public OrderAggregate(**CreateOrderCommand** createOrderCommand){

OrderCreatedEvent orderCreatedEvent=new OrderCreatedEvent();

BeanUtils.copyProperties(createOrderCommand,orderCreatedEvent);

**AggregateLifecycle.apply(orderCreatedEvent);**

}

**@EventSourcingHandler**

public void on(OrderCreatedEvent orderCreatedEvent) throws Exception {

this.orderId = orderCreatedEvent.getOrderId();

this.productId = orderCreatedEvent.getProductId();

this.userId = orderCreatedEvent.getUserId();

this.addressId = orderCreatedEvent.getAddressId();

this.quantity = orderCreatedEvent.getQuantity();

this.orderStatus = orderCreatedEvent.getOrderStatus();

}

**@CommandHandler**

public void handle(ApproveOrderCommand approveOrderCommand){

OrderApprovedEvent orderApprovedEvent=new OrderApprovedEvent(approveOrderCommand.getOrderId());

**AggregateLifecycle.apply(orderApprovedEvent);**

}

**@EventSourcingHandler**

public void on(OrderApprovedEvent orderApprovedEvent){

this.orderStatus=orderApprovedEvent.getOrderStatus();

}

**@CommandHandler**

public void handle(RejectOrderCommand rejectOrderCommand){

OrderRejectedEvent orderRejectedEvent=

new OrderRejectedEvent(rejectOrderCommand.getOrderId(),

rejectOrderCommand.getReason());

**AggregateLifecycle.apply(orderRejectedEvent);**

}

**@EventSourcingHandler**

public void on(OrderRejectedEvent orderRejectedEvent){

this.orderStatus=orderRejectedEvent.getOrderStatus();

}

}