UserZoom

.NET Core Senior Backend

Technical test

Builder Engineering Team

UserZoom

Technical Test

Contents

UserZoom

Introduction

Introduction

Thank you for taking the time to do our technical test.

Please take this considerations before you start:

- Spend no more than **1-2 hours**
  a. We acknowledge that the time suggested is not enough
  b. Add comments on how would you continue with the unfinished parts
- It consists of two parts:
  a. A coding test
  b. Some technical questions

Please, in order execute the technical test you should:

1. Share with us your Github username.
2. Clone our test repository into your account.
3. Perform the test and perform a Pull Request into your repository.
4. Share the new repository with us. Make sure that your repository is public.

Abstract and goals

- The described scenario and the resulting implementation are intended to **encourage discussion** around **formal correctness** and its related **implementation patterns and idioms**.
- Due to the limited time investment, it is not expected for the outcome implementation to be functionally complete, defect-free or production quality as long as it tries to fulfill the presented requirements and establishes a base for discussion.
- According to that, there is no need in this case to focus on what would otherwise be considered relevant production quality implementation topics such as security, maintainability, scalability, abstraction, observability, performance or functional completeness among others.

UserZoom

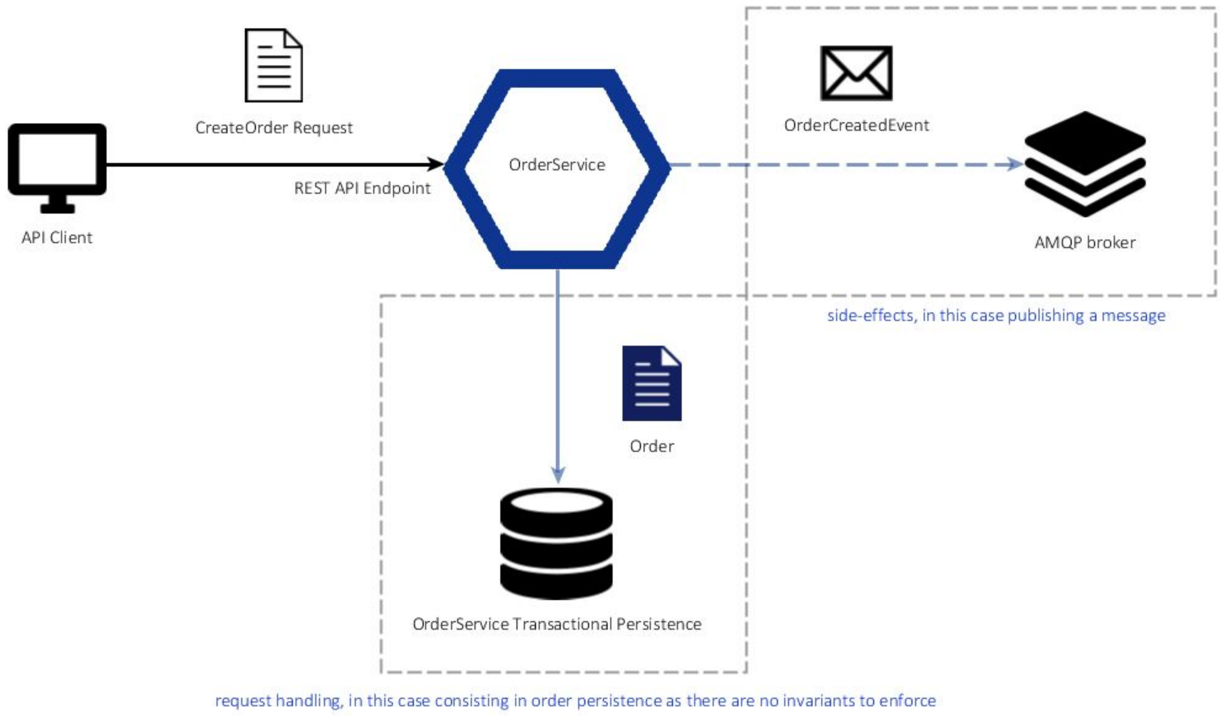
Code Testing

UserZoom

Scope and Context

Our goal is to implement an initial version of an orders handling service. This first version will only handle order creation requests which are performed by third-party clients invoking a REST endpoint.

The handling process consists of transforming the request into an appropriate model and persisting it on a postgres database. Once successfully persisted, the service publishes a message to a RabbitMQ broker to notify interested parties of the recently created order.


CreateOrder Request
API Client
REST API Endpoint
OrderService
OrderCreatedEvent
AMQP broker
side-effects, in this case publishing a message
Order
OrderService Transactional Persistence
request handling, in this case consisting in order persistence as there are no invariants to enforce

Scope and Context

Our orders service implementation must fulfill the following requirements

Order creation requests will conform to the following payload schema:

```
CreateOrderApiRequest
  - id: Guid                                   # order id is client generated
  - items: IDictionary<string, decimal>     # represents (productSku, quantity) tuples
```

Order creation request sample:

```
{
  "id": "34969054-5c26-4e88-afce-1b324528d53a",
  "items": {
    "product-sku-01": 100,
    "product-sku-02": 250
  }
}
```

Requirements

At the moment there are no invariants placed on orders, so we consider any present order items -represented as (productSku, quantity) tuples- as valid, independently of their productSku and quantity values.

The established API contract mandates that on successful request processing the service must return a 201 status code -created- response. The api client will consider that response as a successful outcome and take it as a guarantee that no state loss can occur whatsoever and that system consistency is guaranteed.

According to that, the implementation **must ensure** that, **once the order is successfully persisted**, a corresponding side-effect consisting in **publishing a notification message always occurs**. At the same time, the implementation must avoid scenarios consisting in persisted orders without their corresponding published messages and published messages without their corresponding persisted orders, thus guaranteeing system consistency.

Once our service returns a successful response, **system consistency must be guaranteed under all circumstances** -process crashes, node crashes, connectivity loss, resource unavailability, unresponsive downstream services- with the exception of persistent state -postgres, messaging queues- loss.

Implementation Hints

- Usage of additional tools -docker, docker-compose- and libraries -MediatR, AutoMapper, Brighter, EF, Marten, Npgsql, MassTransit- is welcome and encouraged as long as it provides value and its usage trade-offs are acceptable.
- When designing the persistence schema, consider if applying denormalization and leveraging postgres capabilities -such as json support- provides value and represent an acceptable trade-off.
- According to the scenario scope and its requirements, at-least-once delivery semantics are considered.
- The given repository has already a BoilerPlate that you can use in order to complete the exercise. Feel free to use as much or as little as needed, even ignoring the given code if you prefer to implement your own solution.

UserZoom

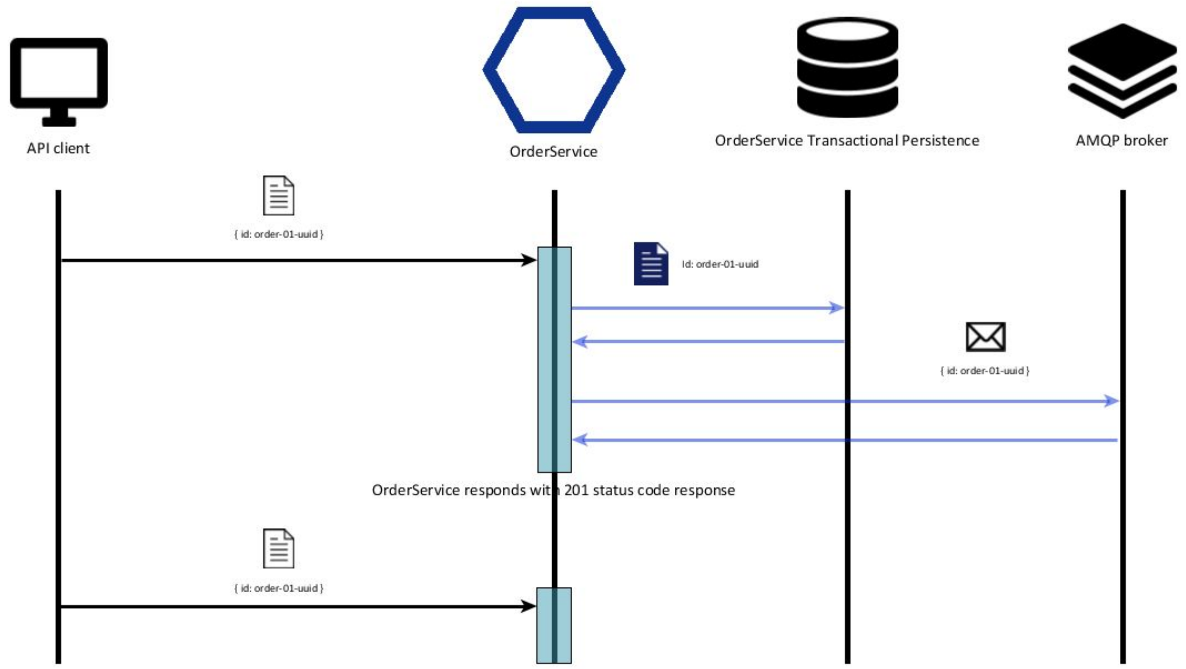
Questions

Please answer the following questions in a markdown file called:
"*Answers to technical questions.md*"

UserZoom

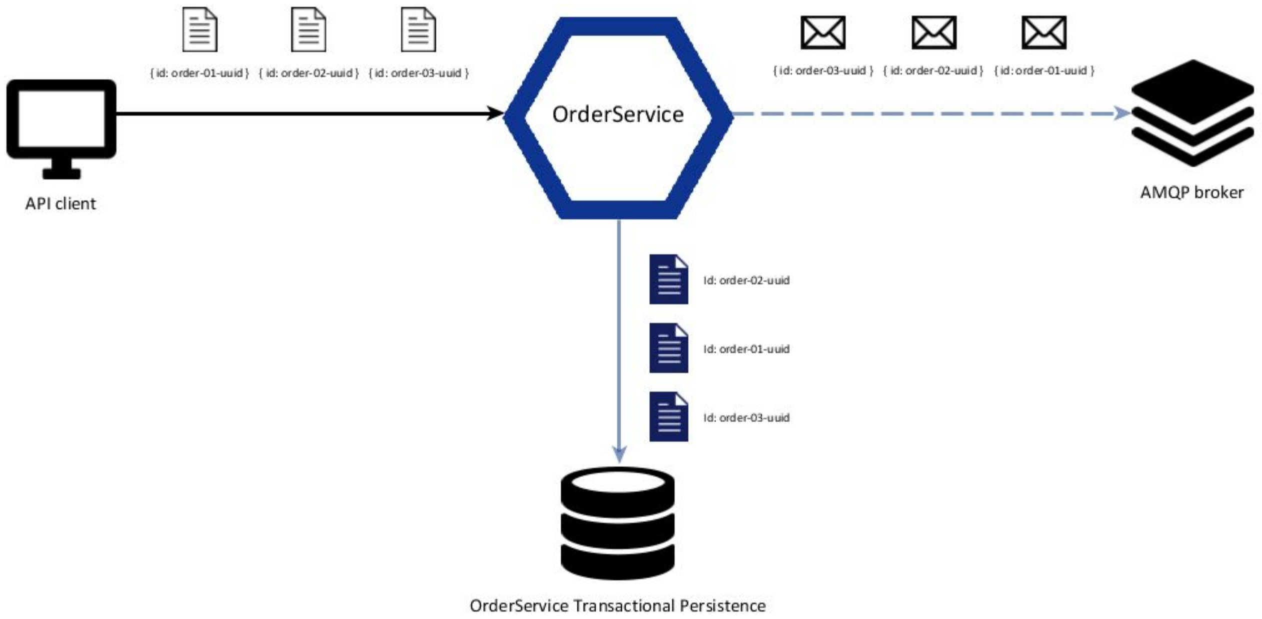
Resilience side-effects

- Is it possible that even when order service responds to a request with a 201 status code response the api client sends the same request again -for example as part of a retry policy on the api client side-?



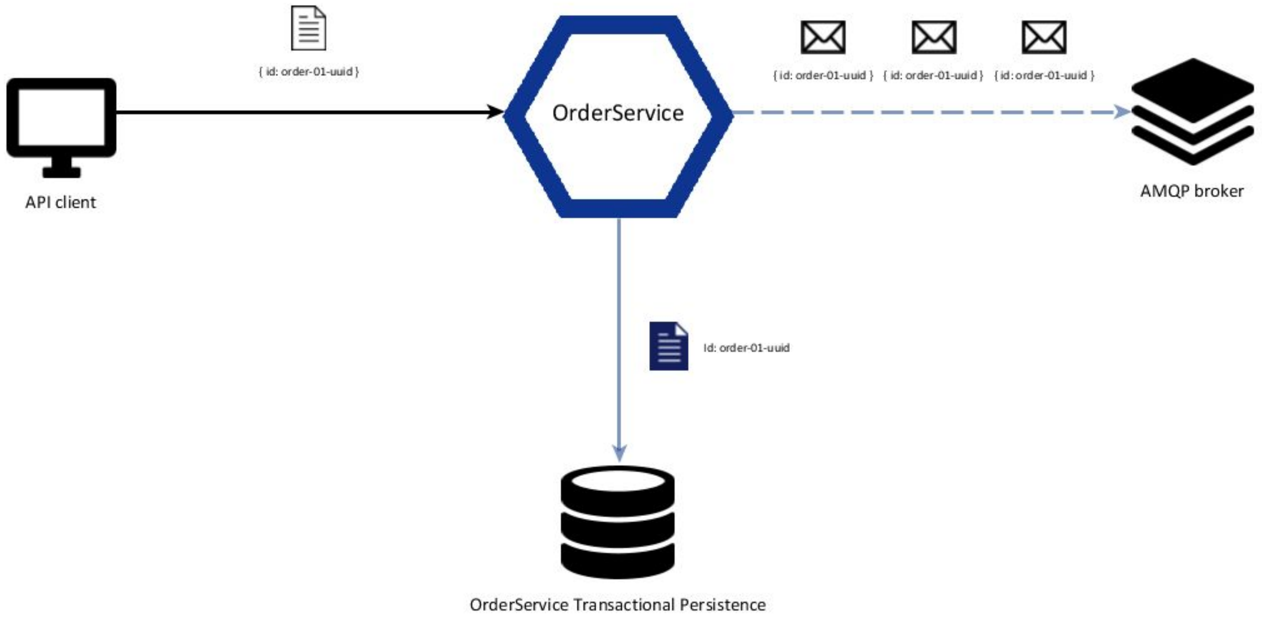
From the message producer standpoint

- Is it possible to publish messages in a different order than the one in which their respective orders were processed and persisted? If so, how can we avoid it? What trade-offs are considered?



From the message producer standpoint

- Is it possible to publish message duplicates?



From the message consumer standpoint

- Is it possible to consume messages out of order? If so, how can we avoid it?
- Is it possible to consume message duplicates? If so, how can we avoid it?

Message semantics

At the moment one of OrderCreatedEvent messages consumers is shipping service, which reacts to those messages creating order shipments accordingly. In case we decide to produce **ShipOrderCommand** messages instead of **OrderConfirmedEvent** ones,

- Which are the semantic differences?
- Does service interaction change in a meaningful way?
- Is there a need to introduce additional components or change system topology?

userzoom.com

Thank you!