AVANTICA
an indecomm company

# Appetize GO Training Project

By:
Ariel Orozco
Katheryne Fallas
Fernando Ocampo
Allan Cedeno

Version 1.0

# Executive Summary

Avantica is having issues trying to hire GO developers, so in order to fulfill Appetize needs we are creating this Go training project. The goal is to have the developers get their hands on the language now that they have the basic knowledge. This will help them on future interviews with the client and will prepare them to work in the project itself.

# 1 Project Description

## 1.1 Project Objectives

The following are the general and specific objectives.

### 1.1.1 General Objective

- To strengthen the developer's skills in Go in order to be a good candidate for Appetize

### 1.1.2 Specific Objectives

- To prepare the candidate for interviews and tests related to Go
- To extend basic knowledge on Go
- To provide Avantica with the Go Developers they need

## 1.2 Project Scope

### 1.2.1 Project Assumptions

- The developer has a basic GO knowledge (book The Go Programming Language from Alan A.A Donovan and Brian W. Kernighan, chapters 1 to 7)
- The developer is going to work full time in this project (at least 8hrs per weekday)
- The developer already has a development environment with GO tools.

## 1.3 Project Timeline

This project was planned to take around 3 weeks of duration. Depending on the circumstances it could be extended one week more.

## 1.4 Communication Channels

The developer can use the following communication channels:

- Assigned coach that is going to help during the project
- Support group in Skype and Slack.

## 1.5 Coverage

1. Go Project Structure
2. Microservice Structure
3. Concurrency (because of workers and servers)
4. GRPC (Server)
5. REST API (POST/GET)
6. Streaming (Producer/Consumer)
7. Persistence (CRUD)

# 2 Project

The developer is going to build a **Checkout Process** that will handle incoming **Orders** requests. The complete process will consist of three small separate Microservices.

The **Order Microservice** will receive the order as a *JSON payload*, transform the data and save it to the ***Orders Database***, after that it will publish the pending order to the ***Pending Orders Stream***. This Microservice will also be subscribed to the ***Processed Orders Stream*** and after pulling a processed order from the stream it will update the status on the ***Orders Database***.

The **Checkout Microservice** will be subscribed to the ***Pending Orders Stream***, whenever it pulls an order from the stream it will send it to an external *Payment Gateway* service, receive a response status, register the payment transaction to the payments database and publishes the order was processed or not over to the ***Processed Orders Stream***.

The **Audit Microservice** will also be subscribed to the ***Processed Orders Stream*****,** and whenever it pulls a processed order from the stream it will transform it and save it to the ***Audit Database.*** This database will contain just basic information regarding the order status, payment method and date. This microservice will also expose a *gRPC endpoint* which will be used by other platforms to get data about processed orders audit data.

In order to query processed orders audit data you are going to build a command line interface client which will consume **audit microservice** through **gRPC** protocol. Audit data can be get with a given order id or using date range.

## 2.1 Project Prerequisites

- Go
- AWS account (it could be the free account)
- MongoDB

## 2.2 Development Considerations

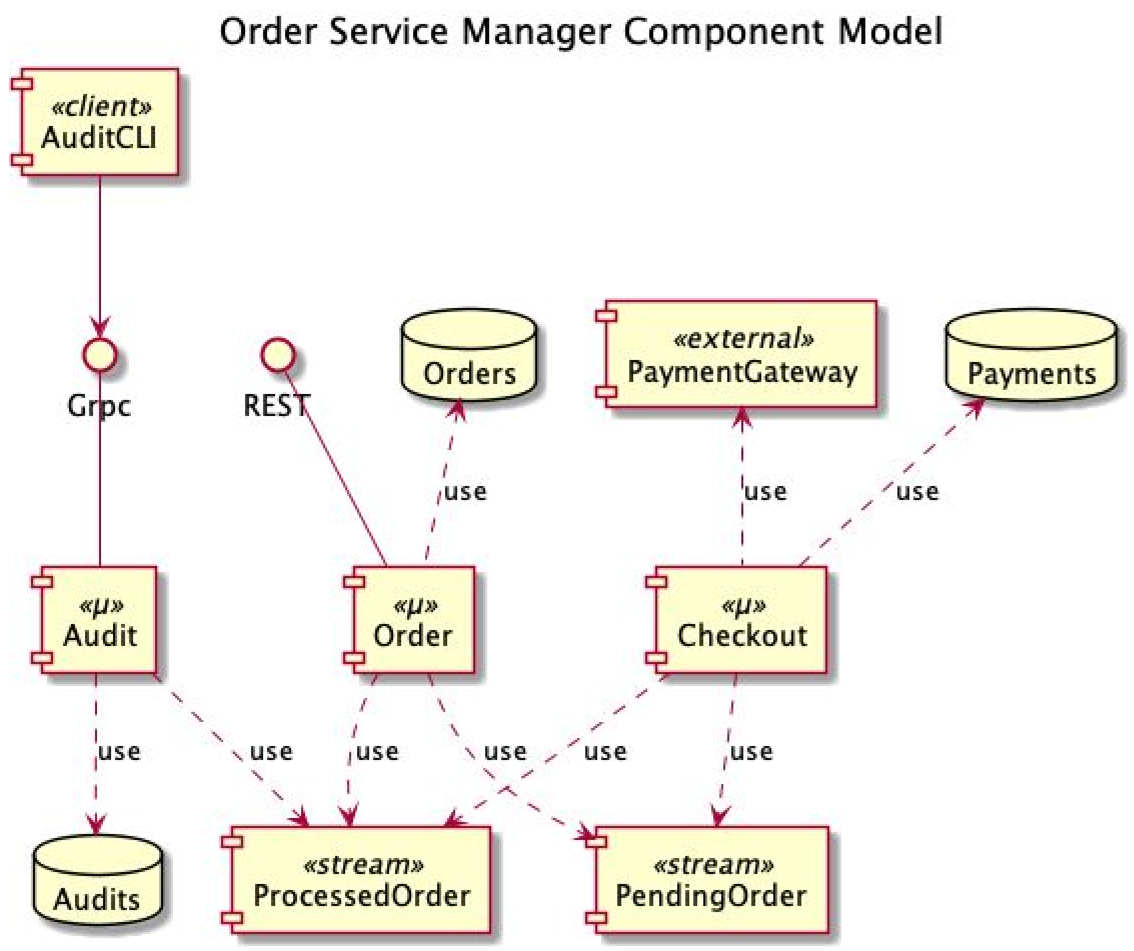
- The JSON payload will be provided to you (See **appendix A**).
- You must define the final structure on how the data will be transformed and stored on each database (Order and Audit).
- The *Streams* will contain the transformed data with the structure as it was defined on the Orders Database
- The *External Payment Gateway* endpoint will be a dummy REST HTTP Service that will return random statuses, this will be provided to you.

## 2.3 Implementation Requirements

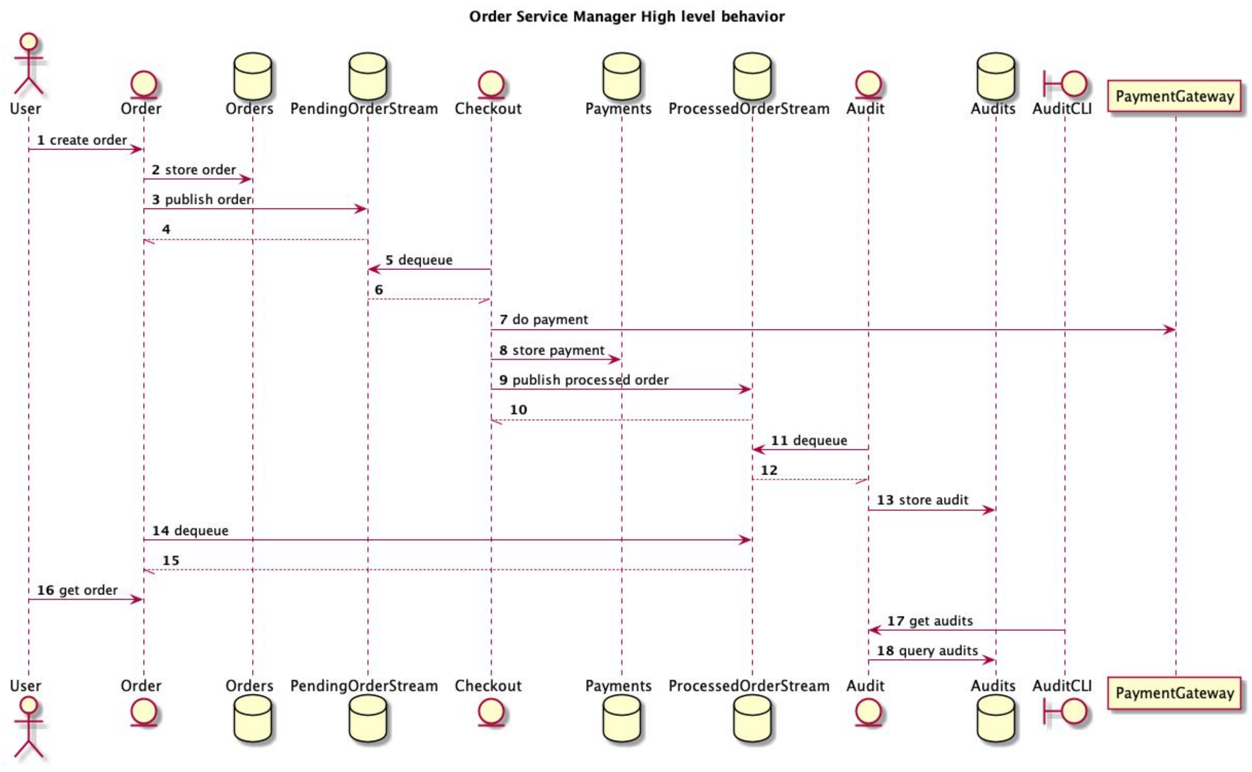
- All of the Microservices **must** be implemented using go-kit
- **Follow** the Onion Architecture within cmd/pkg go folder structure.
- **Use** gorilla/mux as the HTTP router
- **Use** MongoDB for the database
- **Use** AWS Kinesis for the streams
- Manage dependencies with go mod
- **Use** Testify as the unit test library.
- Unit tests **are required** (provide screenshots with successful tests)
- **SOLID** principles are required.
- **Code clean** is a must.

- **Golint and Vet** must executed before each commit. You will have to provide screenshots in order to prove this.
- Expose **Health check** endpoint.
- Application should log logic using Debug, Info and Error levels.

## 2.4 Service Diagrams



| ORDER SERVICE MANAGER COMPONENTS | | |
|---|---|---|
| COMPONENT | TYPE | RESPONSABILITY |
| Order | Microservice | Exposes REST API to create orders.<br>Exposes REST API to query orders.<br>Stores orders data.<br>Publishes orders to PendingOrder stream.<br>Consumes ProcessedOrder stream.<br>Updates orders state. |
| Checkout | Microservice | Consumes PendingOrder stream.<br>Consumes PaymentGateway to process payment.<br>Stores payments transaction data.<br>Publishes processed orders to ProcessedOrder stream. |
| Audit | Microservice | Exposes gRPC API to query orders audit data.<br>Consumes ProcessedOrder stream.<br>Stores processed orders audit data. |
| Orders | Repository | Orders database. |
| Payments | Repository | Payment transactions database. |
| Audits | Repository | Orders audit data database. |
| PendingOrder | Stream | Stores new order events pending for processing. |
| ProcessedOrder | Stream | Stores processed order events. |
| PaymentGateway | Service | Hypothetical payment processor.<br>Exposes REST API to process payment. |



## 2.5 Project Approach

1. Write user stories to guide developers step by step in development. Small work increase moral.
2. TDD implementation.
3. Coaching guided by Code Reviews.
4. Skype group for questions.

## 3 Project Evaluation

| Project Evaluation | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **Subject** | **Deficient** | **Basic Knowledge** | **Regular** | **Good** | **Outstanding** |
| Order microservice implemented with go-kit | | | | | |
| Checkout microservice implemented with go-kit | | | | | |
| Audit microservice implemented with go-kit | | | | | |
| Concurrency implemented correctly | | | | | |
| The Onion Architecture was followed correctly | | | | | |
| gorilla/mux was used as HTTP router | | | | | |
| Databases were created with MongoDB | | | | | |
| AWS Kinesis was used correctly | | | | | |
| Go mod was used correctly | | | | | |
| Created unit tests with Testify | | | | | |
| Code is clean | | | | | |
| Golint and Vet screenshots were provided | | | | | |
| SOLID principles were followed | | | | | |
| The health check endpoint was exposed | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Answers correctly any question related to the project | | | | | |

## APPENDIX A. ORDER PAYLOAD JSON FORMAT

```
{
   "id": 100000,
   "code": "asfs92342klsdfsf",
   "version": "1.2.3",
   "app_id": 12345,
   "channel": "Portal web",
   "customer": {
      "id": 1298723,
      "document": {
         "type": "DD",
         "number": "A23232421"
      },
      "first_name": "Lucio",
      "last_name": "Vero"
   },
   "items": [
      {
         "id": 1223,
         "code": "SHIPAHSSD1234",
         "description": "SHIPPING",
         "type": "SHIPPING",
         "sku": "123",
         "price": 4,
         "quantity": 1,
         "taxable": false
      },
      {
         "id": 12233445,
         "code": "SEDAHSSD1234",
         "description": "NIKE SHOES 123",
         "type": "FASHION",
         "sku": "400000301570",
         "price": 22.56,
         "quantity": 1,
         "taxable": true,
         "taxes": [
            {
               "id": 34,
               "name": "IVA",
                "rate": 7.75,
                "value": 1.75
            }
        ]
    }
],
"payments": [{
    "type": "cash",
    "tender_amount": 50,
    "change": 23.44,
    "amount": 26.56
}],
"total": 26.56,
"shipping_id": "A2345YZ234",
"currency": "USD",
"create_date": "2014-01-01T10:30:00+01:00"
}
```