## **Technical Test Inc. Challenge** Overall Architecture

## 

## Use cases list (Actors in bold)

USE CASE 1: Process an Online payment

1. **The online shopping user** getproducts list and status from the **ONLINE Website** which turns the products request to the **Product Manager**;
2. **The Online website** sent a “payment request” to the **Account Manager** (providing information as: payment Id, product Id, account Id, request timestamp);
3. **The account manager** checks for payment request and send through the **Account-Payment message adaptor** an ONLINE payment message to the Message Queue Manager, furthermore remains on hold by an appending request to the **Payment Processor (PPS)** that will provide a response with a Transaction Id and a related payment status;
4. The **PPS (Payments Processor System)** reads messages from the **Message Queue** **Manager**, then it generates a transaction ID (txId) and update an “in memory DB” storing the transaction status in “Processing”;
5. The **PPS** checks the bank account status and waits for an “OK” response from the **BANK integration module** (http://localhost:9000/payment), after the verification it puts the transaction status in “Validated” on the **Transaction datagrid** , then will persist the transaction result information (txId, ProductId, AccountId, payment Id, payment status) in the DB calling the PPS REST API http://localhost:9000/log  .
6. The **Account Manager**, that waits “on hold”, gets the response status, store the txId status in the DB and return it to the **ONLINE Website**

USE CASE 2: Process an shop payment

1. The **shop operator** uses The **CRM console** to send a payment request;
2. The CRM console send through the **Account-Payment message adaptor** an OFFLINE payment message to the **Message Queue Manager**;
3. The **PPS** checks the bank account status and waits for an “OK” response from the **BANK integration module** (http://localhost:9000/payment), after the verification it puts the transaction status in “Validated” on the **Transaction datagrid** , then will persist the transaction result information (txId, ProductId, AccountId, payment status) in the DB.
4. The **PPS** return payment information to the **CRM console**;

USE CASE 3: Get all transactions in failure status

1. The **Log System console** request for a list of all transaction with in “Failure” status to the **PPS** through the PPS REST API (http://localhost:9000/logs )
2. The **PPS** queries the Transaction database and return the result set to the **Log System**

This use case is introduced to better explain the role of the **Account-Payment message adaptor**

USE CASE 3 (OPTIONAL): Recovering a transaction failure

1. The **Log System Console** request for a txId in “Failure” status processed by the ONLINE channel by the **PPS REST API**
2. The **log System** fix theTransaction issue by sending a new message to the **Message Queue** Manager through the **Account-Payment message adaptor**;
3. The **PPS** checks the bank account status and waits for an “OK” response from the **BANK integration module**, after the verification it puts the transaction status in “Validated” on the **Transaction datagrid** , then will persist the transaction result information (txId, ProductId, AccountId, payment status) in the DB.