

# Phase 1 : Synthèse des problèmes

Tchadel Icard

8 octobre 2024

# 1 Introduction

Ce document présente une analyse approfondie du logiciel IntOrder, utilisé pour la gestion des commandes e-commerce, et examine les principales problématiques identifiées au sein de son architecture. À travers des simulations, nous avons exploré divers scénarios mettant en lumière les dysfonctionnements potentiels, tels que des incohérences dans la gestion des paiements ou des statistiques de ventes, ainsi que les risques liés à la montée en charge et à la défaillance des systèmes. Cette synthèse propose une réflexion sur les solutions envisageables pour améliorer la fiabilité et la robustesse de l'application.

L'analyse des risques a été réalisée en collaboration avec Thibaud Couchet, avec qui nous avons travaillé pour identifier les principaux problèmes du logiciel. Toutefois, bien que cette collaboration ait permis de croiser nos observations, nous avons rédigé nos rapports respectifs de manière indépendante afin d'apporter nos propres réflexions et solutions.

## 2 Analyse des risques

Nom du risque	Proba- bilité (1-5)	Impact (1-5)	Mesure de contrôle (1-5)	Criticité (P x I x C)	Commentaire
Crash sous haute charge	4	5	1	20	Le système n'est pas adapté pour des charges élevées.
Format incohérent des statistiques	3	3	2	18	Problème de formata- ge après crash.
Paiement avant va- lidation de com- mande	2	4	3	24	Risque financier et in- satisfaction client.
Non-redémarrage de l'OMS après crash	5	5	1	25	Interruption prolon- gée du service.
Doubles paiements	1	3	2	6	Impact majeur sur la confiance client.

## 3 Description des risques identifiés

### 3.1 Paiements qui arrivent avant la commande

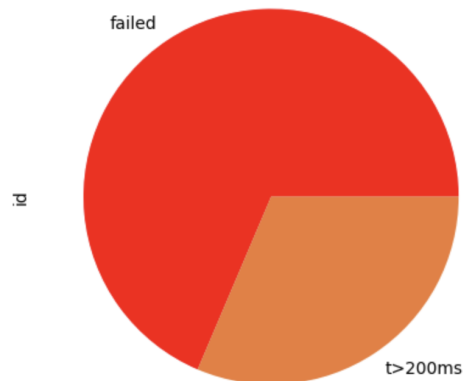
Nous avons remarqué en effectuant nos simulations que par moment les requêtes de *callback* de paiements arrivaient avant que la requête de création de commandes soit entièrement traitée par l'*OMS*. Ceci est dû au fait que l'*OMS* attend la réponse du *WMS* avant d'enregistrer la commande dans la base de données. Le *WMS* met entre 2 et 4 secondes pour traiter une commande faisant que la commande est enregistrée dans la base de données qu'après ce délai. Cependant, les requêtes de *callback* sont envoyées dans les 2 à 4 secondes après que la demande de création de commandes soit faite et de manière asynchrone ce qui fait que le paiement peut arriver avant que la commande existe. Nous avons réussi à bien mettre en évidence ce problème en augmentant significativement le temps de traitement d'une commande par le *WMS*.

```

data > stats.csv > data
1 900000001,172837590067,/order/1/payment-callback,12,ko,404
2 900000000,1728375900597,/order/0/payment-callback,2,ko,404
3 900000009,1728375901125,/order/9/payment-callback,4,ko,404
4 900000006,1728375901197,/order/6/payment-callback,2,ko,404
5 900000007,1728375901298,/order/7/payment-callback,4,ko,404
6 900000013,1728375901404,/order/13/payment-callback,2,ko,404
7 900000010,1728375901476,/order/10/payment-callback,2,ko,404
8 900000008,1728375901526,/order/8/payment-callback,2,ko,404
9 900000002,1728375901543,/order/2/payment-callback,2,ko,404
10 900000011,1728375901827,/order/11/payment-callback,2,ko,404
11 900000021,1728375901962,/order/21/payment-callback,2,ko,404
12 0,1728375897597,/order,4376,ok,200
13 900000025,1728375902243,/order/25/payment-callback,5,ko,404
14 4,1728375897994,/order,4255,ok,200
15 7,1728375898297,/order,4258,ok,200
16 900000016,1728375902582,/order/16/payment-callback,1,ko,404
17 900000026,1728375902592,/order/26/payment-callback,0,ko,404
18 8,1728375898398,/order,4264,ok,200
19 1,1728375897691,/order,5282,ok,200
20 9,1728375898499,/order,4588,ok,200

```

(a) Extrait du fichier `stats.csv` avec les paiements avant commandes



(b) Graphique montrant le temps de traitement des requêtes

FIGURE 1 – Figures de la simulation avec les paiements arrivant avant les commandes

Nous observons bien dans la figure 1a que le *callback* de la commande 0 arrive avant le traitement avec succès de la commande. La deuxième figure 1b permet de voir que le changement de temps de traitement d'une commande par le *WMS* fait qu'on a le double de requêtes en erreur par rapport à celles qui sont correctement traitées.

Une solution qu'on pourrait imaginer pour résoudre ce problème serait d'enregistrer le plus rapidement possible la commande dans la base de données sans vérifier la disponibilité des produits. La vérification des stocks pourrait être faite dans un second temps que la commande soit annulée et remboursée si les produits ne sont pas en stock. Je présume que c'est le comportement adopté par les sites d'*e-commerce* lorsqu'ils ont des flux de commande importants par exemple le *black friday* ou les commandes finissent par être remboursées s'il y a une rupture de stock.

### 3.2 Double de paiements

Un autre problème que nous avons identifié est qu'il est possible de créer des doublons de paiements en modifiant le code source de la fonction `post_random_order/2` en y mettant plusieurs appels à la fonction `post_payment/2`. Cependant, les paiements en double ne sont pas correctement stockés dans la base de données de l'*OMS*. La base de données de l'*OMS* ne stocke qu'un seul `transaction_id` dans la structure de données et non une liste de transactions pour correctement gérer les doublons. Voir l'extrait de code source ci-dessous. Ceci est un gros problème, car si un client nous indique qu'il a été prélevé plusieurs fois pour la même commande on n'est pas en mesure de retrouver cette information.

```

1 # payment arrived , get order and process package delivery !
2 post "/order/:orderid/payment-callback" do
3   {:ok, bin, conn} = read_body(conn)
4   %{ "transaction_id" => transaction_id } = Poison.decode!(bin)
5   case MicroDb.HashTable.get("orders", orderid) do
6     nil-> conn |> send_resp(404, "") |> halt()
7     order->
8       # If a new transaction for the same order arrives later, the transaction id is
8       # overwritten
9       order = Map.put(order, "transaction_id", transaction_id)
10      :httpc.request(:post, { 'http://localhost:9091/order/process_delivery', [], '
11      application/json', Poison.encode!(order) }, [], [])
12      MicroDb.HashTable.put("orders", orderid, order)
13      conn |> send_resp(200, "") |> halt()
14    end
15  end

```

La version modifiée de la fonction `post_random_order/2` afin de simuler 3 paiements à chaque commande est disponible ci-dessous.

```

1 def post_random_order(req_id, logfile) do
2   order =
3     Poison.encode!({

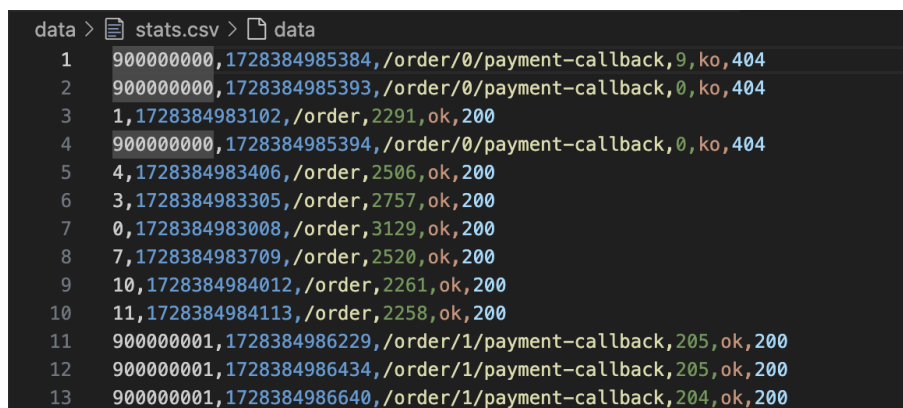
```

```

4      id: "#{req_id}",
5      products: [
6        %{id: :rand.uniform(1000), quantity: :rand.uniform(5)},
7        %{id: :rand.uniform(1000), quantity: :rand.uniform(5)}
8      ]
9    })
10
11    ts = :erlang.system_time(:milli_seconds)
12
13    Task.start(fn ->
14      {time, {ok?, other}} =
15        :timer.tc(fn ->
16          case :httpc.request(:post, {'#{@url}/order', [], 'application/json', order},
17            [], []) do
18            {:ok, {{_, code, _}, _, _}} when code < 400 -> {:ok, code}
19            {:ok, {{_, code, _}, _, _}} -> {:ko, code}
20            #{:error, reason} -> {:ko, "#{inspect reason}"}
21            {:error, _} -> {:ko, :failed}
22          end
23        end)
24
25    IO.write(logfile, "#{req_id},#{ts},/order,#{div(time,1000)},#{ok?},#{other}\n")
26
27    Task.start(fn ->
28      :timer.sleep(2_000 + (:rand.uniform(16) * 125)) # the paiement arrives after the
29      # Send 3 payment callbacks instead of 1
30      post_payment(req_id, logfile)
31      post_payment(req_id, logfile)
32      post_payment(req_id, logfile)
33    end)
34  end

```

Nous observons bien dans la figure 2 que les 3 paiements pour la commande 1 ont bien été pris en compte par l'OMS. Ce qui est le comportement souhaité. Cependant, au niveau de notre base de données, ceci provoquera un problème de cohérence dans les données.



```

data > stats.csv > data
1 900000000,1728384985384,/order/0/payment-callback,9,ko,404
2 900000000,1728384985393,/order/0/payment-callback,0,ko,404
3 1,1728384983102,/order,2291,ok,200
4 900000000,1728384985394,/order/0/payment-callback,0,ko,404
5 4,1728384983406,/order,2506,ok,200
6 3,1728384983305,/order,2757,ok,200
7 0,1728384983008,/order,3129,ok,200
8 7,1728384983709,/order,2520,ok,200
9 10,1728384984012,/order,2261,ok,200
10 11,1728384984113,/order,2258,ok,200
11 900000001,1728384986229,/order/1/payment-callback,205,ok,200
12 900000001,1728384986434,/order/1/payment-callback,205,ok,200
13 900000001,1728384986640,/order/1/payment-callback,204,ok,200

```

FIGURE 2 – Extrait du fichier `stats.csv` montrant la prise en compte des doublons de paiements

Une solution possible pour résoudre ce problème de cohérence de données serait de stocker toutes les transactions associées à une commande afin d'avoir l'historique des paiements en cas de paiement en double ou même de remboursement de paiement.

### 3.3 Crash de l'OMS à plus de 20 requêtes par seconde

Après de nombreux tests de charge, nous nous sommes rendu compte que l'OMS a du mal à tenir la charge. Cependant, le problème ne vient pas de l'agrégation des statistiques. Nous avons réussi à tester l'agrégation des statistiques avec une forte charge sur l'OMS. Le problème intervient surtout lorsque la charge est importante en requête de création de nouvelles commandes. Il est compliqué de passer la charge à plus de 20 requêtes par seconde sans que les processus de l'OMS se mettent à crash. En revanche, cela ne semble pas être un crash complet. On remarque que parfois des requêtes sont traitées et parfois elles

ne le sont pas. Ceci est potentiellement dû au fait que le processus est relancé automatiquement. Voir la figure 3 qui nous permet de visualiser un fort taux de requêtes non traité lorsque des crashes interviennent sur l'OMS.

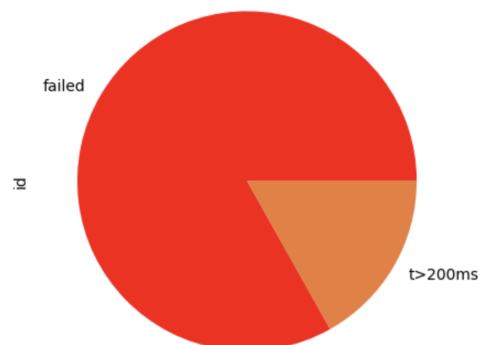


FIGURE 3 – Graphique montrant le temps de traitement des requêtes avec des crashes de l'OMS

Une solution potentielle pour corriger le problème serait d'avoir plusieurs instances de l'OMS qui tournent en parallèle. Cependant, ceci pourrait potentiellement créer des problèmes de cohérence des données dans la base de données de l'OMS. Je n'ai pas complètement réussi à saisir la gestion des processus avec Plug.Router et comment les requêtes sont traitées par l'OMS.

## 4 Format incohérent des statistiques

Lors de l'analyse des problèmes dans l'application, nous nous sommes rendu compte que le *notebook* n'arrivait plus à lire le fichier de statistiques. Voir la figure 4. On remarque la présence de caractères qui empêchaient le fichier à être correctement parsé. Voir la figure 5. Nous avons donc décidé d'effectuer des modifications dans le code du simulateur de l'application frontale pour que le fichier soit correctement généré même en cas d'erreur de l'OMS. L'ensemble des modifications apportées aux fonctions sont détaillées ci-dessous.

```
/var/folders/0h/crjpwv4n1kl1tc30x1vhjhf80000gn/T/ipykernel_43082/1237947535.py:1: FutureWarning: The argument 'date_parser' is deprecated and
will be removed in a future version. Please use 'date_format' instead, or read your data in as 'object' dtype and then call 'to_datetime'.
d = pd.read_csv('data/stats.csv', parse_dates=['ts'], date_parser=lambda x: pd.to_datetime(x, unit='ms').tz_localize('UTC').tz_convert('Europe/Paris')),
-----
ParserError                                Traceback (most recent call last)
Cell In[2], line 1
----> 1 d = pd.read_csv('data/stats.csv', parse_dates=['ts'], date_parser=lambda x: pd.to_datetime(x, unit='ms').tz_localize('UTC').tz_convert('Europe/Paris'),
      2                                     names=['id','ts','path','latency','res','res_desc'])
      3 d["grouppath"] = d.apply(lambda x: "/".join(x['path'].split("/")[:2]), axis=1)
      4 d.head()
```

FIGURE 4 – Erreur du parsing du fichier csv dans le notebook

```
387 157,1728386698106,/order,8963,ko,500
388 900000229,1728386707160,/order/229/payment-callback,1,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
389 165,1728386698779,/order,8390,ko,500
390 900000227,1728386707242,/order/227/payment-callback,1,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
391 266,1728386707267,/order,0,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
392 900000232,1728386707288,/order/232/payment-callback,1,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
393 267,1728386707351,/order,1,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
394 900000230,1728386707373,/order/230/payment-callback,2,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
395 900000233,1728386707374,/order/233/payment-callback,2,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
396 900000065,1728386693188,/order/65/payment-callback,14207,ko,500
397 269,1728386707519,/order,1,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
398 900000226,1728386707533,/order/226/payment-callback,1,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
399 174,1728386699535,/order,8001,ko,500
400 900000241,1728386707543,/order/241/payment-callback,2,ko,404
401 900000244,1728386707545,/order/244/payment-callback,2,ko,404
402 900000242,1728386707627,/order/242/payment-callback,1,ko,{failed_connect, [{to_address, {~c"localhost", 9090}}, {inet, [:inet], :emfile}}]
403 172,1728386699367,/order,8287,ko,500
```

FIGURE 5 – Erreur du format du fichier csv généré lorsque l'OMS crash

Dans la fonction `get_req_ko400/3`, nous avons du modifier le case afin de retirer l'`inspect` pour le remplacer par un `:failed`. Ceci permet de stocker `failed` dans le fichier à la place de l'erreur et donc de permettre au *notebook* de pouvoir correctement parser le fichier.

```

1 def get_req_ko400(id, path, logfile) do
2   ts = :erlang.system_time(:milli_seconds)
3
4   {time, {ok?, other}} = :timer.tc(fn ->
5     case :httpc.request("#{@url}#{path}") do
6       {:ok, {_, code, _}, _, _} when code < 400 -> {:ok, code}
7       {:ok, {_, code, _}, _, _} -> {:ko, code}
8       # Don't do an inspect because the output can't be parsed
9       #{:error, reason} -> {:ko, "#{inspect reason}"}
10      {:error, _} -> {:ko, :failed}
11    end
12  end)
13
14  IO.write(logfile, "#{id},#{ts},#{String.replace(path, " ", "-")},#{div(time,1000)},#{ok
15    ?},#{other}\n")
16 end

```

Nous devons aussi modifier le l'appel de `Task.start` dans la fonction `post_random_order/2` afin d'y appliquer les mêmes modifications pour les mêmes raisons.

```

1 Task.start(fn ->
2   {time, {ok?, other}} =
3     :timer.tc(fn ->
4       case :httpc.request(:post, {'#{@url}/order', [], 'application/json', order},
5         [], []) do
6         {:ok, {_, code, _}, _, _} when code < 400 -> {:ok, code}
7         {:ok, {_, code, _}, _, _} -> {:ko, code}
8         # Don't do an inspect because the output can't be parsed
9         #{:error, reason} -> {:ko, "#{inspect reason}"}
10        {:error, _} -> {:ko, :failed}
11      end
12    end)
13
14    IO.write(logfile, "#{req_id},#{ts},/order,#{div(time,1000)},#{ok?},#{other}\n")
15  end)

```

Les mêmes modifications sont également appliquées sur `post_payment/2`. Cependant, il y avait une autre modification à effectuer sur l'écriture, car il y avait un guillemet en trop.

```

1 def post_payment(order_id, logfile) do
2   transaction = Poison.encode!(%{transaction_id: :rand.uniform(10_000)})
3   ts = :erlang.system_time(:milli_seconds)
4
5   {time, {ok?, other}} =
6     :timer.tc(fn ->
7       case :httpc.request(:post, {'#{@url}/order/#{order_id}/payment-callback', [], '
8         application/json', transaction}, [], []) do
9         {:ok, {_, code, _}, _, _} when code < 400 -> {:ok, code}
10        {:ok, {_, code, _}, _, _} -> {:ko, code}
11        # Don't do an inspect because the output can't be parsed
12        #{:error, reason} -> {:ko, "#{inspect reason}"}
13        {:error, _} -> {:ko, :failed}
14      end
15    end)
16
17    # Remove the single quote from the line
18    #IO.write(logfile, "#{900000000 + order_id},#{ts},/order/#{order_id}/payment-
19      callback',#{div(time, 1000)},#{ok?},#{other}\n")
20    IO.write(logfile, "#{900000000+order_id},#{ts},/order/#{order_id}/payment-callback
21      ,#{div(time,1000)},#{ok?},#{other}\n")
22  end

```

Grâce à ces modifications, on obtient maintenant un fichier csv qui se parse correctement et nous réussissons bien à observer un taux de crash quasiment de 100%.