



Projet de Gestion de processus métiers - M2 DataScale

**Construction d'un Système de Traitement
de Commandes Asynchrone avec FastAPI**

Présenté par :

Thivagini SUGUMAR

Mohamed OUMEZZAOUCHE

Encadré par :

Yehia TAHER

10 mars 2024

Table des matières

1. Introduction.....	3
2. Modélisation BPMN.....	3
3. Implémentation.....	5
4. Exécution.....	6
5. Tests.....	7
5.1. Cas où tout se déroule normalement.....	8
5.2. Cas où la validation de commande est non validée.....	11
5.3. Cas où le fournisseur refuse le devis.....	12
5.4. Cas où le client refuse le devis.....	14
5.5. Cas où le client n'est pas d'accord avec la réalisation du service.....	16

1. Introduction

Dans le cadre de l'UE "Gestion de processus métiers", nous avons réalisé un projet dans lequel nous avons développé un système de traitement de commandes asynchrone composé d'un processus client pour passer des commandes et d'un processus fournisseur pour gérer les demandes de commandes, générer des devis et confirmer des commandes. Pour mettre en œuvre ce projet, nous avons utilisé FastAPI, un framework web Python pour créer des API REST.

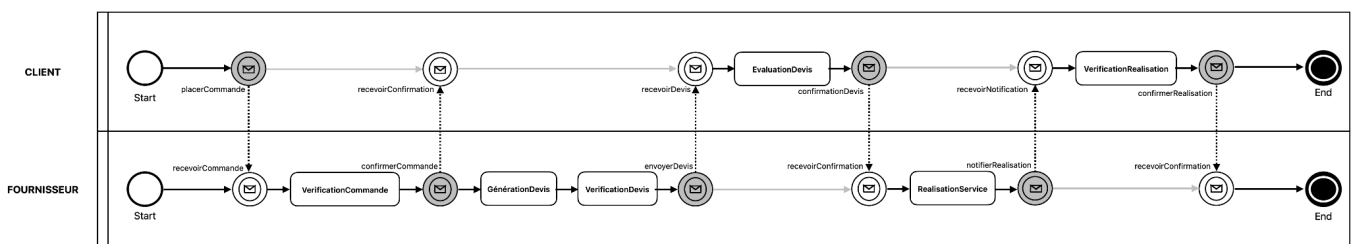
Dans un premier temps, on a modélisé ce système en modèle BPMN puis dans un deuxième temps, nous avons concrétisé ce modèle à l'aide de FastAPI. A la fin de ce rapport, vous retrouverez l'exécution ainsi que les différents tests de notre système que nous avons implémenté.

2. Modélisation BPMN

Dans la première phase de notre projet, nous avons utilisé la modélisation BPMN pour concevoir de manière précise et structurée le système de traitement de commandes asynchrones entre le client et le fournisseur.

La modélisation BPMN (Business Process Model and Notation) est une méthode graphique utilisée pour représenter les processus métier d'une organisation. Ses symboles graphiques standardisés permettent de fournir une représentation visuelle des processus métiers.

Pour notre cas, nous avons effectué le modèle suivant :






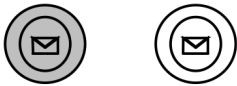

Ce modèle modélise les interactions entre les deux processus : le client et le fournisseur.

Voici les étapes de notre système qu'on a modélisé :

- (1) Le client place sa commande qui sera réceptionné par le fournisseur.
- (2) Le fournisseur vérifie la commande du client.
- (3) Le fournisseur confirme la commande et le client reçoit la confirmation de commande.
- (4) Le fournisseur génère et vérifie le devis.
- (5) Le fournisseur envoie le devis au client qui sera réceptionné par le client.
- (6) Le client évalue le devis.
- (7) Le client confirme le devis qui sera reçu par le fournisseur.
- (8) Le fournisseur réalise le service.
- (9) Le fournisseur notifie la réalisation du service à la fin qui sera réceptionné par le client.
- (10) Le client vérifie si le fournisseur a bien réalisé le service.
- (11) Le client confirme le service réalisé et sera reçu par le fournisseur.

Voici la légende de notre modèle :

LEGENDE	
	Début événement
	Fin événement
	Tâches

	Événement intermédiaire (capture, émission)
	Flux de données

3. Implémentation

Dans la continuité de notre processus de développement, nous abordons maintenant la phase d'implémentation du modèle BPMN que nous avons rigoureusement élaborée dans la partie précédente.

Pour le processus client/fournisseur asynchrone, nous avons utilisé FastAPI pour définir des endpoints pour chaque étape du processus client et fournisseur, tels que :

```
[...]
@app.post("/place_order")
[...]
@app.get("/check_order/{order_id}")
[...]
@app.get("/check_quote/{order_id}")
[...]
@app.get("/check_realization/{order_id}")
[...]
```

Pour rendre ces actions asynchrones, nous utilisons des fonctions de type asynchrone “async def” et des opérations asynchrones avec “await asyncio.sleep()”. Nous utilisons parfois les tâches en arrière-plan “BackgroundTasks” afin d'exécuter ces tâches asynchrones en arrière-plan pendant le traitement d'une requête HTTP, ce qui permet la parallélisation.

Voici les prototypes de nos fonctions :

```
async def place_order(order: OrderCreate, background_tasks: BackgroundTasks)
async def validate_order(order_id: int, background_tasks: BackgroundTasks)
async def check_order(order_id: int)
def check_validation(order_id: int)
def generate_quote(order_id: int, name: str, email: str)
def process_and_validate_quote(order_id: int, name: str, email: str,
background_tasks: BackgroundTasks)
async def evaluate_quote(order_id: int, background_tasks: BackgroundTasks)
async def check_quote(order_id: int)
def realization_service(order_id: int, background_tasks: BackgroundTasks)
async def check_realization(order_id: int)
async def validation_realization_service(order_id: int, background_tasks:
BackgroundTasks):
```

Pour stocker les données des processus, nous utilisons SQLite pour sauvegarder ces données qui sont les suivantes :

```
id INTEGER NOT NULL,
name VARCHAR,
email VARCHAR,
validated BOOLEAN,
validated_quote_supplier BOOLEAN,
validated_quote_client BOOLEAN,
service_realization BOOLEAN,
quote_file_name VARCHAR,
PRIMARY KEY (id)
```

4. Exécution

Après avoir vu l'implémentation, on va vous montrer comment exécuter notre programme pour utiliser notre système.

Voici les commandes à utiliser pour démarrer le serveur:

- Pour démarrer le serveur la première fois : `uvicorn main:app --port 8803`
- Pour redémarrer le serveur après fermeture : `Get-Process | Where-Object { $_.ProcessName -eq "uvicorn" } | Stop-Process`

Une fois le serveur démarré, voici les requêtes HTTP que le client doit saisir pour :

- placer une commande :

```
http POST http://127.0.0.1:8803/place_order name="Jon Doe"
email="john@example.com"
```

- vérifier le placement de commande:

```
http GET http://127.0.0.1:8803/check_order/1
```

- vérifier le devis :

```
http GET http://127.0.0.1:8803/check_quote/1
```

- vérifier la réalisation du service :

```
http GET http://127.0.0.1:8803/check_realization/1
```

5. Tests

Des cas de tests sont proposés dans la suite afin de découvrir notre système implémenté et de voir les fonctionnalités qu'il propose, voici les tests que vous allez voir dans la suite :

- (1) Cas où tout se déroule normalement
- (2) Cas où la validation de commande est non validée
- (3) Cas où le fournisseur refuse le devis
- (4) Cas où le client refuse le devis
- (5) Cas où le client n'est pas d'accord avec la réalisation du service, le fournisseur refait le service, le client accepte ensuite le service réalisé

5.1. Cas où tout se déroule normalement

[CLIENT] : Place sa commande et regarde l'état de sa commande qui est reçue par le fournisseur mais en attente de validation.

```
PS C:\Users\OUMEZZAUCHE Mohamed> http POST http://127.0.0.1:8803/place_order name="Jon Doe" email="john@example.com"
HTTP/1.1 200 OK
content-length: 68
content-type: application/json
date: Fri, 08 Mar 2024 16:17:11 GMT
server: uvicorn

{
  "message": "Commande reçue, en attente de validation",
  "order_id": 1
}
```

[FOURNISSEUR] : Reçoit la requête du client, vérifie la commande et valide la commande en saisissant ok.

```
PS C:\Users\OUMEZZAUCHE Mohamed> http GET http://127.0.0.1:8803/check_order/1
HTTP/1.1 200 OK
content-length: 47
content-type: application/json
date: Fri, 08 Mar 2024 16:22:01 GMT
server: uvicorn

{
  "message": "Votre commande a été vérifiée"
}
```

[CLIENT] : Regarde l'état de sa commande qui indique que la commande a été vérifiée et donc validée.

```
Vérifier la commande 1. Tapez 'ok' pour valider, 'non' pour invalider: ok
Commande 1 validée. Notification envoyée au client.
```

[FOURNISSEUR] : Génère le devis de la commande : création de quote1.txt, informations saisies par le client et ajout du coût de sa commande.

```
Devis généré : quote_1.txt

1 id: 1
2 name: Jon Doe
3 email: john@example.com
4 cost: 38
```


main.ipynb	11 days ago	12.1 KB
main.py	2 hours ago	9.7 KB
quote_1.txt	1 hour ago	55 B
test.db	1 hour ago	16 KB

[CLIENT] : Peut voir l'état de sa commande qui attend la validation du devis par le fournisseur pendant ce temps là.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_quote/1
HTTP/1.1 200 OK
content-length: 54
content-type: application/json
date: Fri, 08 Mar 2024 16:26:31 GMT
server: uvicorn

{
  "message": "Votre devis est en attente de validation"
}
```

[FOURNISSEUR] : Vérifie et valide le devis en saisissant ok.

Le devis associé à la commande 1 a été généré et validé, veuillez le consulter et tapez 'ok' pour le valider, 'non' pour invalider:
Réponse: ok

[CLIENT] : Peut voir l'état de sa commande qui indique que le devis est validé par le fournisseur.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_quote/1
HTTP/1.1 200 OK
content-length: 44
content-type: application/json
date: Fri, 08 Mar 2024 16:27:41 GMT
server: uvicorn

{
  "message": "Votre devis a été vérifiée"
}
```

[CLIENT] : Reçoit la notification que le devis est arrivé, évalue le devis et valide en saisissant ok.

Vérifier le devis 1. Tapez 'ok' pour valider, 'non' pour invalider:
ok

[FOURNISSEUR] : Reçoit la notification que le devis a été accepté par le client.

Devis 1 accepté.

[FOURNISSEUR] : Réalise le service demandé dans la commande et indique le message suivant lorsque le service est réalisé.

Service associé à la commande 1 réalisé.

[CLIENT] : Consulte l'état de sa commande qui indique que le service a été réalisé par le fournisseur.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_realization/1
HTTP/1.1 200 OK
content-length: 42
content-type: application/json
date: Fri, 08 Mar 2024 16:40:35 GMT
server: uvicorn

{
  "message": "Le service a été réalisé"
}
```

[CLIENT] : Reçoit un message du fournisseur qui demande si le service a bien été réalisé selon son souhait. Il vérifie et valide par ok pour dire que tout s'est bien passé.

```
Le service associé à la commande 1 a-t-il été réalisé comme convenu ? Tapez 'ok' pour valider, 'non' pour invalider: ok
Service associé à la commande 1 validé.
```

5.2. Cas où la validation de commande est non validée

[CLIENT] : Place sa commande et regarde l'état de sa commande qui est reçue par le fournisseur mais en attente de validation.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http POST http://127.0.0.1:8803/place_order name="Marc Doe" email="marc@example.com"
HTTP/1.1 200 OK
content-length: 68
content-type: application/json
date: Fri, 08 Mar 2024 18:23:58 GMT
server: uvicorn

{
  "message": "Commande reçue, en attente de validation",
  "order_id": 2
}
```

[FOURNISSEUR] : Reçoit la requête du client, vérifie la commande et ne valide pas la commande en saisissant non.

```
Vérifier la commande 2. Tapez 'ok' pour valider, 'non' pour invalider: non
Commande 2 invalidée. Notification envoyée au client.
```

[CLIENT] : Regarde l'état de sa commande qui indique que la commande a été invalidée et doit donc remplacer sa commande.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_order/2
HTTP/1.1 200 OK
content-length: 81
content-type: application/json
date: Fri, 08 Mar 2024 18:24:04 GMT
server: uvicorn

{
  "message": "Votre commande a été invalidée, veuillez remplacer votre commande"
}
```

5.3. Cas où le fournisseur refuse le devis

[CLIENT] : Place sa commande et regarde l'état de sa commande qui est reçue par le fournisseur mais en attente de validation.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http POST http://127.0.0.1:8803/place_order name="Martin Doe" email="martin@example.com"
HTTP/1.1 200 OK
content-length: 68
content-type: application/json
date: Fri, 08 Mar 2024 18:28:06 GMT
server: uvicorn

{
  "message": "Commande reçue, en attente de validation",
  "order_id": 2
}
```

[FOURNISSEUR] : Reçoit la requête du client, vérifie la commande et valide la commande en saisissant ok.

```
Vérifier la commande 2. Tapez 'ok' pour valider, 'non' pour invalider: ok
Commande 2 validée. Notification envoyée au client.
```

[CLIENT] : Regarde l'état de sa commande qui indique que la commande a été vérifiée et donc validée.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_order/2
HTTP/1.1 200 OK
content-length: 47
content-type: application/json
date: Fri, 08 Mar 2024 18:29:06 GMT
server: uvicorn

{
  "message": "Votre commande a été vérifiée"
}
```

[FOURNISSEUR] : Génère le devis de la commande : création de quote2.txt, informations saisies par le client et ajout du coût de sa commande.

```
Devis généré : quote_2.txt
1 id: 2
2 name: Martin Doe
3 email: martin@example.com
4 cost: 188
```

Name	Last Modified	File Size
main.ipynb	11 days ago	12.1 KB
main.py	2 hours ago	9.7 KB
quote_1.txt	2 hours ago	55 B
quote_2.txt	2 minutes ago	61 B
test.db	2 minutes ago	16 KB

[FOURNISSEUR] : Vérifie et ne valide pas le devis en saisissant non.

```
Vérifier le devis 2. Tapez 'ok' pour valider, 'non' pour invalider:
non
Devis 2 non validé.
```

[CLIENT] : Peut voir l'état de sa commande qui indique que le devis n'est pas validé par le fournisseur et qu'il faut remplacer la commande.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_quote/2
HTTP/1.1 200 OK
content-length: 78
content-type: application/json
date: Fri, 08 Mar 2024 18:34:54 GMT
server: uvicorn

{
  "message": "Votre devis a été invalidée, veuillez remplacer votre commande"
}
```

5.4. Cas où le client refuse le devis

[CLIENT] : Place sa commande et regarde l'état de sa commande qui est reçue par le fournisseur mais en attente de validation.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http POST http://127.0.0.1:8803/place_order name="Martin Doe" email="martin@example.com"
HTTP/1.1 200 OK
content-length: 68
content-type: application/json
date: Fri, 08 Mar 2024 18:28:06 GMT
server: uvicorn

{
  "message": "Commande reçue, en attente de validation",
  "order_id": 2
}
```

[FOURNISSEUR] : Reçoit la requête du client, vérifie la commande et valide la commande en saisissant ok.

```
Vérifier la commande 2. Tapez 'ok' pour valider, 'non' pour invalider: ok
Commande 2 validée. Notification envoyée au client.
```

[CLIENT] : Regarde l'état de sa commande qui indique que la commande a été vérifiée et donc validée.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_order/2
HTTP/1.1 200 OK
content-length: 47
content-type: application/json
date: Fri, 08 Mar 2024 18:29:06 GMT
server: uvicorn

{
  "message": "Votre commande a été vérifiée"
}
```

[FOURNISSEUR] : Génère le devis de la commande : création de quote2.txt, informations saisies par le client et ajout du coût de sa commande.

```
Devis généré : quote_2.txt
1 id: 2
2 name: Martin Doe
3 email: martin@example.com
4 cost: 188
```

Name	Last Modified	File Size
main.ipynb	11 days ago	12.1 KB
main.py	2 hours ago	9.7 KB
quote_1.txt	2 hours ago	55 B
quote_2.txt	2 minutes ago	61 B
test.db	2 minutes ago	16 KB

[FOURNISSEUR] : Vérifie et valide le devis en saisissant ok.

```
Vérifier le devis 2. Tapez 'ok' pour valider, 'non' pour invalider: ok
Devis 2 validé.
```

[CLIENT] : Reçoit la notification que le devis est arrivé, évalue le devis et ne valide pas le devis en saisissant non. La commande est annulée.

```
Le devis associé à la commande 2 a été généré et validé, veuillez le consulter et tapez 'ok' pour le valider, 'non' pour invalider:
Réponse: non
Devis 2 refusé.
```

5.5. Cas où le client n'est pas d'accord avec la réalisation du service

[CLIENT] : Place sa commande et regarde l'état de sa commande qui est reçue par le fournisseur mais en attente de validation.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http POST http://127.0.0.1:8803/place_order name="Martin Doe" email="martin@example.com"
HTTP/1.1 200 OK
content-length: 68
content-type: application/json
date: Fri, 08 Mar 2024 18:28:06 GMT
server: uvicorn

{
  "message": "Commande reçue, en attente de validation",
  "order_id": 2
}
```

[FOURNISSEUR] : Reçoit la requête du client, vérifie la commande et valide la commande en saisissant ok.

```
Vérifier la commande 2. Tapez 'ok' pour valider, 'non' pour invalider: ok
Commande 2 validée. Notification envoyée au client.
```

[CLIENT] : Regarde l'état de sa commande qui indique que la commande a été vérifiée et donc validée.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_order/2
HTTP/1.1 200 OK
content-length: 47
content-type: application/json
date: Fri, 08 Mar 2024 18:29:06 GMT
server: uvicorn

{
  "message": "Votre commande a été vérifiée"
}
```

[FOURNISSEUR] : Génère le devis de la commande : création de quote2.txt, informations saisies par le client et ajout du coût de sa commande.

```
Devis généré : quote_2.txt

1 id: 2
2 name: Martin Doe
3 email: martin@example.com
4 cost: 188
```


Name	Last Modified	File Size
main.ipynb	11 days ago	12.1 KB
main.py	2 hours ago	9.7 KB
quote_1.txt	2 hours ago	55 B
quote_2.txt	2 minutes ago	61 B
test.db	2 minutes ago	16 KB

[FOURNISSEUR] : Vérifie et valide le devis en saisissant ok.

```
Vérifier le devis 2. Tapez 'ok' pour valider, 'non' pour invalider: ok
Devis 2 validé.
```

[CLIENT] : Reçoit la notification que le devis est arrivé, évalue le devis et valide en saisissant ok.

```
Le devis associé à la commande 2 a été généré et validé, veuillez le consulter et tapez 'ok' pour le valider, 'non' pour invalider:
Réponse: ok
```

[FOURNISSEUR] : Reçoit la notification que le devis a été accepté par le client.

```
Devis 2 accepté.
```

[CLIENT] : Consulte l'état de sa commande qui indique que le service n'a pas encore été réalisé par le fournisseur.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_realization/2
HTTP/1.1 200 OK
content-length: 109
content-type: application/json
date: Fri, 08 Mar 2024 18:41:20 GMT
server: uvicorn

{
  "message": "Le service n'a pas encore été réalisé, des étapes préliminaires doivent être effectuées"
}
```

[FOURNISSEUR] : Réalise le service demandé dans la commande et indique le message suivant lorsque le service est réalisé.

```
Service associé à la commande 2 réalisé.
```

[CLIENT] : Reçoit un message du fournisseur qui demande si le service a bien été réalisé selon son souhait. Il vérifie, il voit que le service n'a pas été fait et répond au fournisseur par non afin qu'il reconsidère le service.

```
Le service associé à la commande 2 a t-il été réalisé comme convenu ? Tapez 'ok' pour valider, 'non' pour invalider: non
Service associé à la commande 2 non validé. Le service doit être reconsidéré.
```

[CLIENT] : Consulte l'état de sa commande qui indique que le service n'a pas encore été réalisé par le fournisseur.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_realization/2
HTTP/1.1 200 OK
content-length: 109
content-type: application/json
date: Fri, 08 Mar 2024 18:41:20 GMT
server: uvicorn

{
  "message": "Le service n'a pas encore été réalisé, des étapes préliminaires doivent être effectuées"
}
```

[FOURNISSEUR] : Refait le service demandé dans la commande et indique le message suivant lorsque le service est réalisé. (Tant que le client n'aura pas répondu que le service réalisé, il le refera)

Service associé à la commande 2 réalisé.

[CLIENT] : Reçoit un message du fournisseur qui demande si le service a bien été réalisé selon son souhait. Il vérifie et valide par ok pour dire que tout s'est bien passé.

```
Le service associé à la commande 2 a t-il été réalisé comme convenu ? Tapez 'ok' pour valider, 'non' pour invalider: ok
Service associé à la commande 2 validé.
```

[CLIENT] : Consulte l'état de sa commande qui indique que le service a été réalisé par le fournisseur.

```
PS C:\Users\OUMEZZAOUCHE Mohamed> http GET http://127.0.0.1:8803/check_realization/2
HTTP/1.1 200 OK
content-length: 42
content-type: application/json
date: Fri, 08 Mar 2024 18:43:43 GMT
server: uvicorn

{
  "message": "Le service a été réalisé"
}
```