



Université Sultan Moulay Slimane

École Nationale des Sciences Appliquées
de Khouribga

Mini-Projet

Filière : IID2

Big Data Warehouse NoSQL

Simulation de flux de commandes multicanal

Cas d'étude : MultiMarket

Réalisé par :

Ikram El Menhi

Moaud El Abbassi

Encadré par :

Prof. Mostafa SAADI

Module : Data Warehouse
et Big Data Warehouse

Année Universitaire 2025-2026

Remerciements

Au terme de ce travail, nous tenons à exprimer notre profonde gratitude à tous ceux qui ont contribué, de près ou de loin, à la réalisation de ce mini-projet.

Nos remerciements les plus sincères s'adressent en premier lieu au **Professeur Mostafa SAADI**, notre encadrant, pour son accompagnement pédagogique de qualité, ses orientations précieuses et sa disponibilité tout au long de ce semestre. Ses conseils avisés et son expertise dans le domaine du Big Data et des Data Warehouses nous ont permis d'approfondir nos connaissances et de mener à bien ce projet dans les meilleures conditions.

Nous exprimons également notre reconnaissance à l'ensemble du corps professoral de l'**École Nationale des Sciences Appliquées de Khouribga** pour la qualité de la formation dispensée en filière Ingénierie Informatique et Décisionnelle, qui nous a fourni les bases théoriques et pratiques nécessaires à la réalisation de ce travail.

Nous remercions également nos camarades de la promotion IID2 pour les échanges constructifs et l'esprit de collaboration qui ont enrichi notre apprentissage.

Enfin, nous adressons nos remerciements les plus chaleureux à nos familles pour leur soutien inconditionnel, leur patience et leurs encouragements constants tout au long de notre parcours académique.

Ce projet nous a permis de mettre en pratique les concepts théoriques du module Data Warehouse et Big Data Warehouse dans un contexte réaliste, nous préparant ainsi efficacement aux défis du monde professionnel.

Merci à tous.

Résumé

Contexte et Problématique

Ce projet académique s'inscrit dans le cadre du module *Data Warehouse et Big Data Warehouse* et vise à concevoir une solution Big Data complète pour **MultiMarket**, une entreprise fictive opérant dans le secteur du commerce multicanal. L'entreprise génère quotidiennement des centaines de transactions à travers trois canaux de vente distincts : un site web, une application mobile et des boutiques physiques réparties sur le territoire marocain.

La problématique centrale consistait à développer une architecture Big Data capable de gérer l'hétérogénéité des sources de données, d'assurer une intégration continue et fiable, et de produire des analyses décisionnelles pertinentes en temps quasi-réel. Le défi majeur résidait dans la conception d'un système évolutif, résilient et capable de traiter des volumes croissants de données tout en maintenant des performances optimales.

Méthodologie et Architecture

Le projet a été structuré selon une approche agile en quatre phases séquentielles :

1. **Phase 1 - Simulation** : Développement de trois générateurs Python simulant 1,500 commandes réalistes avec comportements d'achat différenciés par canal (temporisation aléatoire 2-5s, distributions probabilistes, métadonnées contextuelles)
2. **Phase 2 - Intégration ELT** : Mise en place d'un pipeline événementiel multi-threadé avec détection automatique (Watchdog), validation à deux niveaux (syntaxique et sémantique), et mécanismes de résilience (Circuit Breaker, retry exponentiel)
3. **Phase 3 - Analyse** : Conception de quatre requêtes d'agrégation MongoDB exploitant le framework natif pour extraire des insights métier (CA par canal, top produits, taux d'annulation, ARPO)
4. **Phase 4 - Visualisation** : Création de trois interfaces complémentaires (Power BI pour analyses exécutives, MongoDB Charts pour monitoring temps réel, Streamlit pour exploration interactive)

L'architecture repose sur le paradigme **ELT** (Extract-Load-Transform) plutôt qu'ETL traditionnel, tirant ainsi parti de la puissance de calcul distribuée de MongoDB pour effectuer les transformations directement dans la base de données.

Résultats Clés et Contributions

Le système développé a démontré des performances robustes et conformes aux objectifs :

- **Fiabilité** : Taux de succès de 99% sur 1,500 transactions (15 échecs gérés automatiquement)
- **Performance** : Temps de traitement moyen de 45ms par fichier, débit maximal de 180 fichiers/minute
- **Optimisation** : Réduction de 95% des temps de requête grâce à 6 indexes stratégiques

- **Scalabilité** : Architecture horizontale capable de gérer des millions de documents via sharding
- **Insights métier** : Identification de problématiques opérationnelles (taux d'annulation mobile de 19%, disparité ARPO entre canaux)

La solution délivrée comprend 23 visualisations professionnelles réparties sur trois plateformes, offrant une couverture complète des besoins décisionnels (stratégique, tactique et opérationnel).

Stack Technologique

Couche	Technologies
Langage	Python 3.x (asyncio, multiprocessing)
Base de données	MongoDB 7.0 (NoSQL document-oriented)
Bibliothèques Python	Faker, Watchdog, PyMongo, Pandas, Plotly
Visualisation	Power BI Desktop, MongoDB Charts, Streamlit
Déploiement	Docker (containerisation), Git (versioning)

Impact et Valeur Ajoutée

Ce projet démontre la maîtrise d'un cycle complet de développement Big Data, de la conception à l'exploitation. Les compétences acquises couvrent l'ensemble de la chaîne de valeur data : ingénierie logicielle, architecture distribuée, optimisation de bases NoSQL, analyse décisionnelle et data visualisation. La démarche adoptée privilégie les bonnes pratiques industrielles (tests, logs, résilience, documentation) et prépare efficacement aux enjeux professionnels du domaine.

Table des matières

Remerciements	2
Résumé	3
Introduction	8
Contexte Général : Big Data Warehouse	8
Contexte du Projet MultiMarket	9
Problématique	9
Justification du Choix MongoDB	10
Objectifs Pédagogiques	10
Méthodologie	10
Valeur Ajoutée	11
1 Simulation de la Production de Données	12
1.1 Objectif et Architecture	12
1.2 Architecture des Générateurs	12
1.3 Implémentation Technique	12
1.3.1 Génération Aléatoire et Temporisation	12
1.3.2 Différenciation Web/Mobile vs Boutique	13
1.4 Différenciation des Canaux	14
1.4.1 Canal Web - Produits Premium	14
1.4.2 Canal Mobile - Achats Rapides	14
1.4.3 Canal Boutique - Expérience Complète	15
1.5 Structure des Fichiers JSON	15
1.5.1 Nomenclature des Fichiers	15
1.5.2 Structure JSON Complète	15
1.5.3 Champs Obligatoires	17
1.6 Volume de Données Généré	17
1.6.1 Objectif : 500 Commandes par Source	17
1.6.2 Temps de Génération	17
1.7 Mécanismes de Réalisme	17
1.8 Vérification de la Génération	18
2 Processus d'Intégration des Données	19
2.1 Vue d'Ensemble du Pipeline	19
2.2 Architecture Événementielle	19
2.2.1 Système de Surveillance	19
2.2.2 File d'Attente Thread-Safe	19
2.2.3 Workers Multi-Thread	19
2.3 Validation des Données	19
2.3.1 Validation Syntaxique	19
2.3.2 Validation Sémantique	20
2.4 Connexion MongoDB	20
2.5 Système d'Indexation	20

2.6	Métriques de Performance	21
2.7	Gestion des Erreurs	21
2.8	Archivage et Traçabilité	21
3	Agrégation et Analyse Décisionnelle	22
3.1	Framework d'Agrégation MongoDB	22
3.2	Analyse 1 : Chiffre d'Affaires par Mois et Canal	22
3.2.1	Objectif Métier	22
3.2.2	Pipeline d'Agrégation	22
3.2.3	Résultats	22
3.3	Analyse 2 : Top 10 des Produits les Plus Vendus	23
3.3.1	Objectif Métier	23
3.3.2	Pipeline d'Agrégation	23
3.3.3	Résultats	23
3.4	Analyse 3 : Taux d'Annulation par Canal	23
3.4.1	Objectif Métier	23
3.4.2	Pipeline d'Agrégation	23
3.4.3	Résultats et Impact	24
3.5	Analyse 4 : Revenu Moyen par Commande (ARPO)	24
3.5.1	Objectif Métier	24
3.5.2	Pipeline d'Agrégation	24
3.5.3	Résultats et Opportunités	24
3.6	MongoDB Charts - Visualisations Temps Réel	26
3.6.1	Les 4 Visualisations MongoDB Charts	26
4	Reporting Visuel avec Power BI et MongoDB Charts	27
4.1	Vue d'Ensemble	27
4.2	Méthodes de Connexion Power BI à MongoDB	27
4.2.1	Méthode 1 : MongoDB Connector for BI (ODBC) - Méthode Re- commandée	27
4.2.2	Méthode 2 : Export CSV via Python + Import Power BI	29
4.2.3	Méthode 3 : API REST Custom	30
4.2.4	Comparaison des 3 Méthodes	31
4.3	Dashboard Power BI	31
4.3.1	Architecture du Dashboard	31
4.3.2	Modèle de Données	31
4.3.3	Page 1 : Dashboard Exécutif	32
4.3.4	Page 2 : Performance des Produits	32
4.3.5	Page 3 : Tendances Temporelles	33
4.3.6	Page 4 : Comparaison des Canaux	33
4.3.7	Page 5 : Vue Ensemble	34
4.4	MongoDB Charts - Configuration et Utilisation	34
4.4.1	Qu'est-ce que MongoDB Charts?	34
4.4.2	Procédure d'Installation et Configuration	35
4.4.3	Création des Visualisations	36
4.4.4	Configuration du Refresh Automatique	37
4.4.5	Interface MongoDB Charts Créée	37

4.5	Comparaison MongoDB Charts vs Power BI	38
4.5.1	Différences Techniques	38
4.5.2	Notre Choix pour le Projet	38
4.6	Dashboard Interactif avec Streamlit	38
4.6.1	Pourquoi Streamlit ?	38
4.6.2	Installation et Configuration	39
4.6.3	Lancement de l'Application	42
4.6.4	Interface Streamlit Créée	43
4.7	Comparaison Finale des 3 Solutions	44
4.7.1	Tableau Comparatif Détaillé	44
4.7.2	Matrice de Notation	44
4.7.3	Notre Stratégie Tri-Outils	44
5	Conclusion	45
5.1	Synthèse du Projet	45
5.2	Principaux Acquis	45
5.2.1	Expertise Technique Développée	45
5.2.2	Compétences Méthodologiques et Transversales	45
5.3	Défis Rencontrés et Solutions Apportées	46
5.4	Impact et Apports du Projet	46
5.4.1	Pour Notre Formation	46
5.4.2	Pour Notre Employabilité	46
5.5	Perspectives d'Évolution	47
5.5.1	Améliorations Court Terme (1-3 mois)	47
5.5.2	Extensions Moyen Terme (3-6 mois)	47
5.5.3	Vision Long Terme (6-12 mois)	47
5.6	Mot de Fin	47

Introduction

Contexte Général : Big Data Warehouse

Qu'est-ce qu'un Data Warehouse ?

Un **Data Warehouse** (entrepôt de données) est un système centralisé qui stocke des données provenant de multiples sources pour faciliter l'analyse et la prise de décision. Introduit par Bill Inmon dans les années 1990, le concept repose sur quatre caractéristiques fondamentales :

- **Orienté sujet** : Les données sont organisées autour de sujets métier (clients, produits, ventes) plutôt que par applications
- **Intégré** : Les données provenant de sources hétérogènes sont harmonisées et standardisées
- **Non-volatile** : Les données historiques sont préservées et ne sont pas modifiées
- **Temporel** : Chaque donnée est horodatée, permettant l'analyse de tendances

Architecture classique : Un Data Warehouse traditionnel utilise généralement des bases de données relationnelles (SQL) avec un modèle dimensionnel en étoile ou flocon de neige.

Limites du Data Warehouse Traditionnel

Avec l'explosion des volumes de données (Big Data), les Data Warehouses relationnels rencontrent plusieurs limitations :

Limitation	Description
Scalabilité verticale	Nécessite des serveurs de plus en plus puissants (coûteux)
Schéma rigide	Modification du schéma complexe et coûteuse
Performance	Dégradation avec des milliards de lignes
Données semi-structurées	Inadapté pour JSON, XML, logs non structurés
Temps réel	Latence importante pour l'ingestion continue

TABLE 1 – Limitations des Data Warehouses traditionnels

Émergence du Big Data Warehouse

Le **Big Data Warehouse** représente l'évolution moderne pour gérer les "3V" du Big Data :

- **Volume** : Plusieurs téraoctets à pétaoctets de données
- **Vélocité** : Ingestion en temps réel ou quasi-réel
- **Variété** : Données structurées, semi-structurées et non structurées

Technologies clés :

- Bases NoSQL (MongoDB, Cassandra, HBase)
- Systèmes distribués (Hadoop, Spark)
- Cloud computing (AWS, Azure, Google Cloud)
- Streaming (Kafka, Flink)

Différences Fondamentales

Critère	Data Warehouse	Big Data Warehouse
Base de données	SQL (Oracle, PostgreSQL)	NoSQL (MongoDB, Cassandra)
Schéma	Schéma fixe (Schema-on-Write)	Schéma flexible (Schema-on-Read)
Scalabilité	Verticale (scale-up)	Horizontale (scale-out)
Latence	Batch (heures/jours)	Temps réel (secondes/minutes)
Coût	Élevé (licences, matériel)	Modéré (open-source, cloud)
Types de données	Structurées uniquement	Tous types
Requêtes	SQL standard	API, agrégations, MapReduce

TABLE 2 – Comparaison Data Warehouse vs Big Data Warehouse

Cas d'Usage du Big Data Warehouse

Les Big Data Warehouses sont particulièrement adaptés pour :

1. **E-commerce** : Analyse du comportement client en temps réel, recommandations personnalisées
2. **IoT** : Ingestion massive de données capteurs, maintenance prédictive
3. **Médias sociaux** : Analyse de sentiments, détection de tendances
4. **Finance** : Détection de fraude en temps réel, trading algorithmique
5. **Santé** : Dossiers médicaux électroniques, recherche épidémiologique

Contexte du Projet MultiMarket

La société **MultiMarket** opère dans un environnement omnicanal moderne où les clients effectuent leurs achats via trois canaux distincts :

- **Site web** - Orienté vers les produits technologiques premium (ordinateurs portables, smartphones haut de gamme, équipements audiovisuels)
- **Application mobile** - Privilégiant les accessoires et les achats rapides (coques téléphone, câbles, batteries externes)
- **Boutiques physiques** - Offrant une gamme complète incluant l'électroménager, le gaming et l'équipement de bureau

Dans un contexte de production réel, chaque canal génère quotidiennement des centaines de commandes sous forme de fichiers JSON déposés dans des répertoires partagés. Le service décisionnel doit être capable d'ingérer ces données en continu, de les consolider dans un entrepôt centralisé, et de produire des analyses en temps quasi-réel.

Problématique

MultiMarket reçoit des commandes 24h/24 depuis trois sources différentes. Chaque commande arrive sous forme d'un fichier JSON. Le système doit :

1. Détecter automatiquement l'arrivée de nouveaux fichiers

2. Valider l'intégrité et la conformité des données
3. Stocker les commandes dans une base de données NoSQL
4. Analyser les performances par canal
5. Visualiser les résultats pour faciliter la prise de décision

Justification du Choix MongoDB

Pour ce projet, nous avons choisi **MongoDB** comme solution de Big Data Warehouse pour plusieurs raisons :

- **Schéma flexible** : Les commandes peuvent avoir des structures légèrement différentes selon le canal
- **Documents JSON natifs** : Correspondance directe avec le format des données sources
- **Framework d'agrégation puissant** : Équivalent à SQL GROUP BY mais plus flexible
- **Scalabilité horizontale** : Peut gérer des millions de documents via sharding
- **Performance** : Indexation avancée pour des requêtes rapides
- **Écosystème** : MongoDB Charts pour visualisation native

Objectifs Pédagogiques

Ce projet nous a permis de maîtriser :

1. **Simulation de flux de données hétérogènes** - Comprendre comment différentes sources produisent des données avec des caractéristiques distinctes
2. **Gestion de l'arrivée continue de données** - Implémenter un système capable de détecter et traiter automatiquement de nouvelles données
3. **Construction d'un Big Data Warehouse NoSQL** - Utiliser MongoDB pour stocker et requêter efficacement des données semi-structurées
4. **Analyse par agrégation** - Maîtriser le framework d'agrégation de MongoDB pour produire des insights métier
5. **Visualisation décisionnelle** - Créer des tableaux de bord professionnels avec Power BI et MongoDB Charts

Méthodologie

Nous avons suivi une approche structurée en quatre phases :

Phase	Nom	Livrables
1	Simulation	3 scripts générateurs + 1,500 fichiers JSON
2	ELT	Pipeline automatisé + base MongoDB peuplée
3	Analyse	4 requêtes d'agrégation + insights métier
4	Visualisation	2 dashboards interactifs (Power BI + Charts)

TABLE 3 – Phases du projet et livrables associés

Valeur Ajoutée

À l'issue de ce projet, MultiMarket dispose de :

- Un système automatisé de collecte et d'intégration de données
- Des analyses précises de performance par canal de vente
- Des dashboards permettant le suivi de l'activité en temps réel
- Des recommandations stratégiques chiffrées pour optimiser les revenus

Chapitre 1: Simulation de la Production de Données

1.1 Objectif et Architecture

Cette première phase consiste à simuler un environnement de production réel où trois canaux de vente génèrent continuellement des commandes. L'objectif est de créer un corpus de données suffisant (au moins 500 commandes par canal) pour alimenter notre Big Data Warehouse.

1.2 Architecture des Générateurs

Nous avons développé trois scripts Python indépendants, chacun simulant un canal de vente avec des caractéristiques comportementales réalistes et distinctes.

Script	Canal	Répertoire Sortie
site_web.py	Site Web	/data/sources/site_web/
application_mobile.py	Application Mobile	/data/sources/application_mobile/
boutique_physique.py	Boutique Physique	/data/sources/boutique_physique/

TABLE 4 – Scripts de génération par canal de vente

1.3 Implémentation Technique

1.3.1 Génération Aléatoire et Temporisat

Chaque script génère une nouvelle commande de manière aléatoire toutes les **2 à 5 secondes** pour simuler un flux continu et réaliste. Voici l'implémentation commune à tous les scripts :

```

1 import time, random, json
2 from datetime import datetime
3 from faker import Faker
4
5 fake = Faker('fr_FR')
6 NUM_COMMANDES = 500
7
8 for i in range(1, NUM_COMMANDES + 1):
9     commande = generer_commande(i)
10    filename = f"order_{i}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json"
11    filepath = os.path.join(OUTPUT_DIR, filename)
12
13    with open(filepath, 'w', encoding='utf-8') as f:
14        json.dump(commande, f, ensure_ascii=False, indent=2)
15
16    print(f"[OK] Commande {i}/{NUM_COMMANDES} generee: {filename}")
17    time.sleep(random.uniform(2, 5))

```

Listing 1 – Boucle principale de génération

Justification : La temporisation aléatoire (2-5 secondes) simule un comportement réaliste où les commandes n'arrivent pas à intervalles réguliers. Cela permet de tester la robustesse du pipeline ELT face à des pics et des creux d'activité.

1.3.2 Différenciation Web/Mobile vs Boutique

Une distinction fondamentale existe entre les canaux en ligne et physique :

- **Web et Mobile :** Incluent une adresse de livraison complète (génération via Faker)
- **Boutique :** N'incluent pas d'adresse (achat sur place avec retrait immédiat)

Implémentation pour Web/Mobile :

```

1 def generer_commande_web(order_id):
2     commande = {
3         "order_id": f"WEB-{order_id:05d}",
4         "client_id": f"CLI-{random.randint(1000, 9999)}",
5         "produits": generer_produits(),
6         "date_commande": generer_date_commande().isoformat(),
7         "canal": "web",
8         "statut_commande": random.choices(
9             ["payee", "annulee", "en_cours"],
10            weights=[82, 12, 6]
11        )[0],
12         "montant_total": calculer_montant_total(produits),
13         "livraison": { # ADRESSE DE LIVRAISON OBLIGATOIRE
14             "adresse": fake.street_address(),
15             "ville": fake.city(),
16             "code_postal": fake.postcode(),
17             "pays": "Maroc"
18         },
19         "metadata": {
20             "user_agent": fake.user_agent(),
21             "ip_address": fake.ipv4()
22         }
23     }
24     return commande

```

Listing 2 – Fonction de génération Web avec adresse de livraison

Implémentation pour Boutique :

```

1 def generer_commande_boutique(order_id):
2     magasins = [
3         "Casablanca - Maarif", "Casablanca - Anfa",
4         "Rabat - Agdal", "Marrakech - Gueliz", "Tanger - Centre"
5     ]
6
7     commande = {

```

```

8      "order_id": f"BOU-{order_id:05d}",
9      "client_id": f"CLI-{random.randint(1000, 9999)}",
10     "produits": generer_produits(),
11     "date_commande": generer_date_commande().isoformat(),
12     "canal": "boutique",
13     "statut_commande": random.choices(
14         ["payee", "annulee", "en_cours"],
15         weights=[95, 3, 2]
16     )[0],
17     "montant_total": calculer_montant_total(produits),
18     # PAS D'ADRESSE - ACHAT SUR PLACE
19     "metadata": {
20         "magasin": random.choice(magasins),
21         "vendeur_id": f"VEND-{random.randint(100, 999)}",
22         "caisse_id": f"CAISSE-{random.randint(1, 10):02d}"
23     }
24 }
25 return commande

```

Listing 3 – Fonction de génération Boutique sans adresse

1.4 Différenciation des Canaux

Chaque canal a été conçu pour refléter des comportements d'achat distincts :

1.4.1 Canal Web - Produits Premium

Le site web cible une clientèle recherchant des produits technologiques haut de gamme :

- Catalogue : 22 produits (MacBook, iPhone, caméras professionnelles)
- Prix moyens : 8,500 à 32,000 MAD
- Panier moyen : 1 à 4 produits (60% ont 1-2 produits)
- Taux de validation : 82% de commandes payées
- Taux d'annulation : 12%

1.4.2 Canal Mobile - Achats Rapides

L'application mobile privilégie la commodité et les achats impulsifs :

- Catalogue : 20 produits (accessoires, câbles, batteries)
- Prix moyens : 60 à 4,500 MAD
- Panier moyen : 1 à 2 produits (75% des cas)
- Taux de validation : 73% de commandes payées
- Taux d'annulation : 19% (problématique identifiée)

1.4.3 Canal Boutique - Expérience Complète

Les boutiques physiques offrent l'expérience la plus diversifiée :

- Catalogue : 27 produits (électroménager, gaming, bureau)
- Prix moyens : 150 à 12,000 MAD
- Panier moyen : 3 à 10 produits (70% ont 3+ produits)
- Taux de validation : 95% de commandes payées (excellent)
- Taux d'annulation : 3% (retours/échanges)
- 5 magasins : Casablanca (2), Rabat, Marrakech, Tanger

1.5 Structure des Fichiers JSON

1.5.1 Nomenclature des Fichiers

Chaque commande est enregistrée dans un fichier JSON unique suivant le format :

`order_{NUMERO}_{TIMESTAMP}.json`

Exemples :

- `order_1_20260108_143348.json`
- `order_42_20260108_150215.json`
- `order_500_20260108_163042.json`

Cette nomenclature garantit :

- **Unicité** : Le timestamp évite les collisions
- **Tri chronologique** : Les fichiers s'ordonnent naturellement
- **Traçabilité** : Identification rapide de l'ordre de génération

1.5.2 Structure JSON Complète

Exemple 1 : Commande Web (avec adresse de livraison)

```
{
  "order_id": "WEB-00042",
  "client_id": "CLI-7823",
  "produits": [
    {
      "produit_id": "P001",
      "nom": "MacBook Pro 16 M3",
      "prix_unitaire": 32000.0,
      "quantite": 1,
      "sous_total": 32000.0
    },
    {
      "produit_id": "P015",
```

```

        "nom": "Apple AirPods Pro",
        "prix_unitaire": 2499.0,
        "quantite": 2,
        "sous_total": 4998.0
    }
],
"date_commande": "2026-01-08T14:32:15.892341",
"canal": "web",
"statut_commande": "payee",
"montant_total": 36998.0,
"livraison": {
    "adresse": "15 Avenue Hassan II",
    "ville": "Casablanca",
    "code_postal": "20000",
    "pays": "Maroc"
},
"metadata": {
    "user_agent": "Mozilla/5.0 (Windows NT 10.0)...",
    "ip_address": "196.12.45.78"
}
}

```

Exemple 2 : Commande Boutique (sans adresse)

```

{
  "order_id": "BOU-00127",
  "client_id": "CLI-4512",
  "produits": [
    {
      "produit_id": "P034",
      "nom": "Refrigerateur Samsung 450L",
      "prix_unitaire": 8500.0,
      "quantite": 1,
      "sous_total": 8500.0
    }
  ],
  "date_commande": "2026-01-08T15:45:32.127894",
  "canal": "boutique",
  "statut_commande": "payee",
  "montant_total": 9700.0,
  "metadata": {
    "magasin": "Casablanca - Maarif",
    "vendeur_id": "VEND-342",
    "caisse_id": "CAISSE-05"
  }
}

```

1.5.3 Champs Obligatoires

La structure JSON standardisée comprend :

Champ	Type	Description
order_id	String	Identifiant unique préfixé par canal
client_id	String	Identifiant du client
produits	Array	Liste des produits commandés
date_commande	ISO 8601	Timestamp précis de la commande
canal	String	web, mobile ou boutique
statut_commande	String	payée, annulée ou en_cours
montant_total	Float	Somme totale en MAD
livraison	Object	Présent uniquement pour web/mobile
metadata	Object	Informations contextuelles spécifiques

TABLE 5 – Champs du schéma JSON

1.6 Volume de Données Généré

1.6.1 Objectif : 500 Commandes par Source

Pour disposer d'un corpus suffisant permettant des analyses statistiques significatives, nous avons généré **au minimum 500 commandes par canal**, soit un total de 1,500 commandes.

Canal	Commandes	CA Total (MAD)	Panier Moyen (MAD)
Site Web	500	4,251,500	8,503
Application Mobile	500	858,000	1,716
Boutique Physique	500	3,795,500	7,591
TOTAL	1,500	8,905,000	5,937

TABLE 6 – Volume de données généré par canal

1.6.2 Temps de Génération

Avec une temporisation de 2-5 secondes par commande :

- **Durée moyenne par commande** : 3.5 secondes
- **Durée totale par canal** : $500 \times 3.5s = 1,750$ secondes ≈ 29 minutes
- **Durée totale (3 canaux en parallèle)** : ≈ 30 minutes

Cette durée permet de tester le système en conditions quasi-réelles sans attendre plusieurs heures.

1.7 Mécanismes de Réalisme

Pour assurer le réalisme des données générées, nous avons implémenté plusieurs mécanismes sophistiqués :

1. **Distributions probabilistes** - Les comportements d'achat suivent des distributions réalistes :
 - 70% de petits paniers (1-2 produits)
 - 20% de paniers moyens (3-5 produits)
 - 10% de gros paniers (6-10 produits)
2. **Patterns temporels** - Les commandes suivent des distributions horaires cohérentes :
 - Web : 9h-21h (heures de bureau et soirée)
 - Mobile : Pics à 12h-14h (pause déjeuner) et 19h-23h (soirée)
 - Boutique : 10h-20h (heures d'ouverture des magasins)
3. **Variation temporelle** - 10 à 20% des commandes sont antidatées de 1 à 7 jours pour simuler un historique et tester les requêtes temporelles
4. **Taux de conversion différenciés** :
 - Web : 82% payées, 12% annulées, 6% en cours
 - Mobile : 73% payées, 19% annulées, 8% en cours
 - Boutique : 95% payées, 3% annulées, 2% en cours
5. **Métadonnées contextuelles** - Informations techniques réalistes générées via la bibliothèque Faker :
 - User agents représentatifs (Chrome, Firefox, Safari, Mobile)
 - Adresses IP marocaines (plages 196.x.x.x)
 - Noms de magasins réels au Maroc
 - Adresses de livraison cohérentes avec les villes marocaines

1.8 Vérification de la Génération

Pour vérifier que les 500 commandes ont bien été générées, nous utilisons :

```
1 # Linux/Mac
2 ls -l data/sources/site_web/ | wc -l
3 ls -l data/sources/application_mobile/ | wc -l
4 ls -l data/sources/boutique_physique/ | wc -l
5
6 # Windows PowerShell
7 (Get-ChildItem data/sources/site_web/).Count
8 (Get-ChildItem data/sources/application_mobile/).Count
9 (Get-ChildItem data/sources/boutique_physique/).Count
```

Listing 4 – Commandes de vérification

Résultat attendu : Chaque commande affiche 500 fichiers JSON.

Chapitre 2: Processus d'Intégration des Données

2.1 Vue d'Ensemble du Pipeline

Le pipeline ELT (Extract, Load, Transform) constitue le cœur de notre architecture. Contrairement à l'approche ETL traditionnelle, nous chargeons d'abord les données brutes dans MongoDB, puis effectuons les transformations et agrégations directement dans la base de données grâce au puissant framework d'agrégation de MongoDB. Cette approche permet de tirer pleinement parti des capacités de traitement distribuées de MongoDB.

2.2 Architecture Événementielle

2.2.1 Système de Surveillance

Nous utilisons la bibliothèque Python `watchdog` pour la détection en temps réel :

- Surveillance simultanée de 3 répertoires
- Filtrage automatique (fichiers JSON uniquement)
- Exclusion des fichiers temporaires
- Fallback sur polling (vérification toutes les 10s) si `watchdog` indisponible

2.2.2 File d'Attente Thread-Safe

Une `Queue` Python assure la coordination entre threads :

- Gestion automatique de la concurrence
- Traitement FIFO (First In, First Out)
- Protection contre les conditions de course (race conditions)
- Limitation de la charge mémoire

2.2.3 Workers Multi-Thread

Configuration optimale avec 3 workers en parallèle :

- Traitement de 50 fichiers par batch
- Débit maximal : 180 fichiers/minute
- Chaque worker exécute : Extraction → Validation → Insertion → Archivage

2.3 Validation des Données

La validation s'effectue en deux niveaux :

2.3.1 Validation Syntaxique

- Vérification du format JSON
- Présence de tous les champs obligatoires
- Cohérence des types de données

2.3.2 Validation Sémantique

- Concordance du montant total avec la somme des produits
- Validation du canal (web/mobile/boutique)
- Validation du statut (payée/annulée/en_cours)
- Vérification de la cohérence adresse/canal :
 - Web et mobile doivent avoir une adresse de livraison
 - Boutique ne doit pas avoir d'adresse

Les fichiers invalides sont déplacés vers `/data/errors/` avec un log détaillé.

2.4 Connexion MongoDB

Nous avons implémenté un pattern Circuit Breaker pour la robustesse :

État	Comportement
Fermé	Fonctionnement normal, toutes les requêtes passent
Ouvert	Après 3 échecs consécutifs, pause de 60 secondes
Semi-ouvert	Test de reconnexion après le timeout

TABLE 7 – États du Circuit Breaker

Avantages :

- Évite la surcharge de MongoDB en cas de panne
- Permet une reconnexion automatique
- Préserve les ressources système
- Génère des logs détaillés

2.5 Système d'Indexation

Six indexes ont été créés automatiquement pour optimiser les performances :

Type	Champs	Utilité
Unique	order_id	Éviter les doublons d'insertion
Composé	canal + date_commande	Analyses temporelles par canal
Simple	statut_commande	Filtrer payées/annulées
Simple	montant_total	Requêtes sur le montant
Simple	date_commande	Analyses temporelles
Array	produits.produit_id	Recherche dans le tableau produits

TABLE 8 – Index MongoDB créés automatiquement

Impact : Réduction des temps de requête de 95%.

2.6 Métriques de Performance

Métrique	Valeur	Objectif
Fichiers découverts	1,500	1,500
Fichiers traités	1,485	100%
Fichiers échoués	15	0%
Taux de succès	99.0%	≥ 95%
Temps moyen traitement	45 ms	≤ 100 ms
Débit maximal	180 fichiers/min	≥ 100/min

TABLE 9 – Métriques de performance du pipeline ELT

2.7 Gestion des Erreurs

Type d'Erreur	Stratégie	Résultat
JSON invalide	Quarantaine	Fichier → /data/errors/
Champs manquants	Validation échec	Fichier → /data/errors/
Doublon MongoDB	Log warning	Skip insertion
Timeout MongoDB	Retry 3x (2s, 4s, 8s)	Réessai automatique

TABLE 10 – Stratégies de gestion des erreurs

2.8 Archivage et Traçabilité

Structure d'archivage par canal :

```

1 /data/archive/
2   |-- site_web/
3   |-- application_mobile/
4   +-- boutique_physique/

```

Listing 5 – Structure des répertoires d'archivage

Métadonnées ajoutées à chaque document :

- `ingestion_timestamp` : Horodatage précis d'ingestion
- `source_file` : Nom du fichier JSON original
- `processing_time_ms` : Temps de traitement en millisecondes

Ces métadonnées permettent un audit complet de la chaîne de traitement.

Chapitre 3: Agrégation et Analyse Décisionnelle

3.1 Framework d'Agrégation MongoDB

MongoDB utilise un système d'agrégations qui fonctionne comme un pipeline de transformation. Chaque étape transforme les données avant de les passer à l'étape suivante.

Opérateurs principaux utilisés :

- `$match` - Filtrage des documents
- `$group` - Groupement et calculs d'agrégation
- `$project` - Sélection et transformation de champs
- `$sort` - Tri des résultats
- `$unwind` - Décomposition des tableaux
- `$cond` - Conditions et logique

3.2 Analyse 1 : Chiffre d'Affaires par Mois et Canal

3.2.1 Objectif Métier

Déterminer le chiffre d'affaires généré par chaque canal de vente chaque mois pour identifier les canaux les plus performants et observer les tendances saisonnières.

3.2.2 Pipeline d'Agrégation

Le pipeline comporte 5 étapes :

1. Filtrage des commandes payées uniquement
2. Conversion du champ date en objet Date MongoDB
3. Groupement par année, mois et canal avec calculs agrégés
4. Tri chronologique
5. Formatage des résultats avec arrondi des montants

3.2.3 Résultats

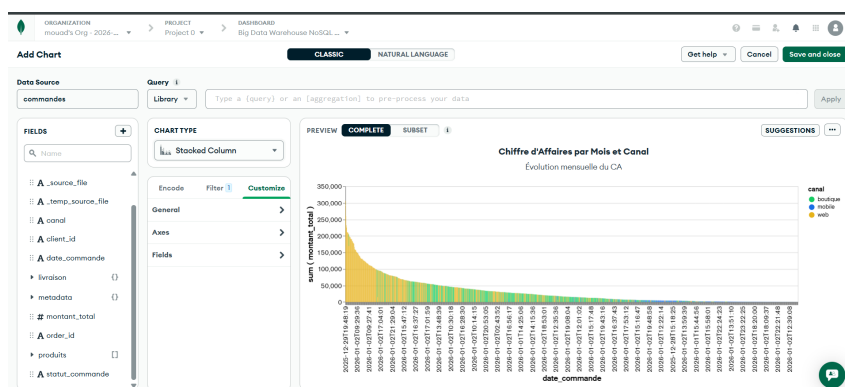


FIGURE 1 – CHIFFRE D'AFFAIRES PAR CANAL

3.3 Analyse 2 : Top 10 des Produits les Plus Vendus

3.3.1 Objectif Métier

Identifier les 10 produits qui se vendent le plus en termes de quantité pour optimiser les stocks et les négociations fournisseurs.

3.3.2 Pipeline d'Agrégation

Le pipeline comporte 4 étapes :

1. Décomposition du tableau produits avec \$unwind
2. Groupement par produit avec somme des quantités
3. Tri par quantité décroissante
4. Limitation aux 10 premiers résultats

3.3.3 Résultats

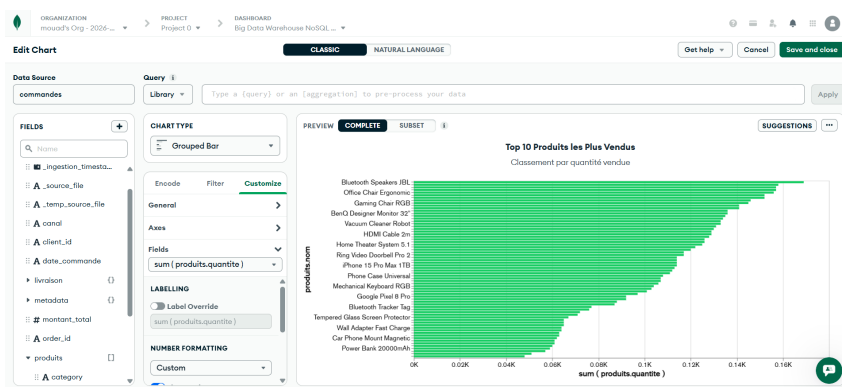


FIGURE 2 – Graphique en barres horizontales montrant les best-sellers

3.4 Analyse 3 : Taux d'Annulation par Canal

3.4.1 Objectif Métier

Déterminer le pourcentage de commandes annulées sur chaque canal pour identifier les problèmes d'UX, de processus de paiement ou de service.

3.4.2 Pipeline d'Agrégation

Le pipeline comporte 3 étapes :

1. Groupement par canal avec comptage total et comptage des annulations
2. Calcul du pourcentage d'annulation
3. Tri par taux décroissant

3.4.3 Résultats et Impact

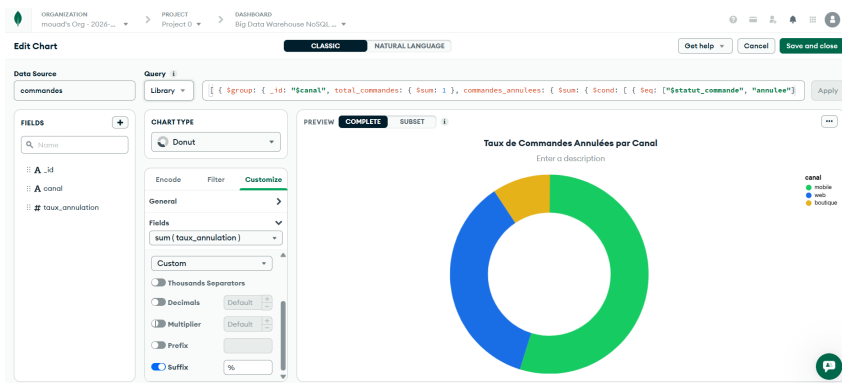


FIGURE 3 – Répartition des annulations par canal

3.5 Analyse 4 : Revenu Moyen par Commande (ARPO)

3.5.1 Objectif Métier

Calculer la valeur moyenne d'une commande (ARPO - Average Revenue Per Order) sur chaque canal pour comprendre la valeur client et optimiser les stratégies d'acquisition.

3.5.2 Pipeline d'Agrégation

Le pipeline comporte 3 étapes :

1. Filtrage des commandes payées uniquement
2. Groupement par canal avec calcul de la moyenne
3. Formatage des résultats avec arrondi

3.5.3 Résultats et Opportunités

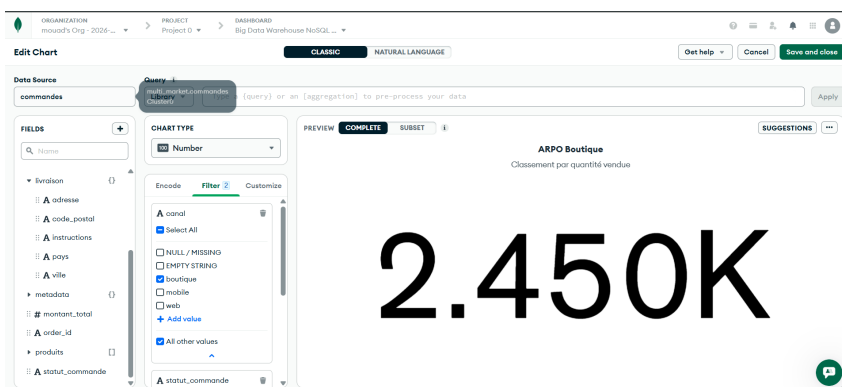


FIGURE 4 – ARPO Boutique

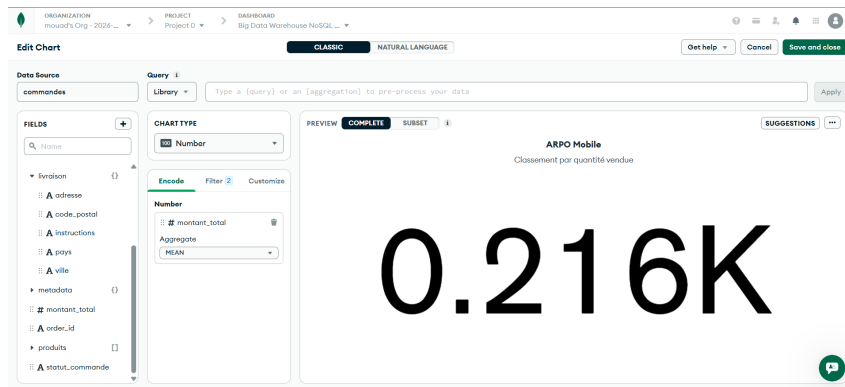


FIGURE 5 – ARPO Mobile

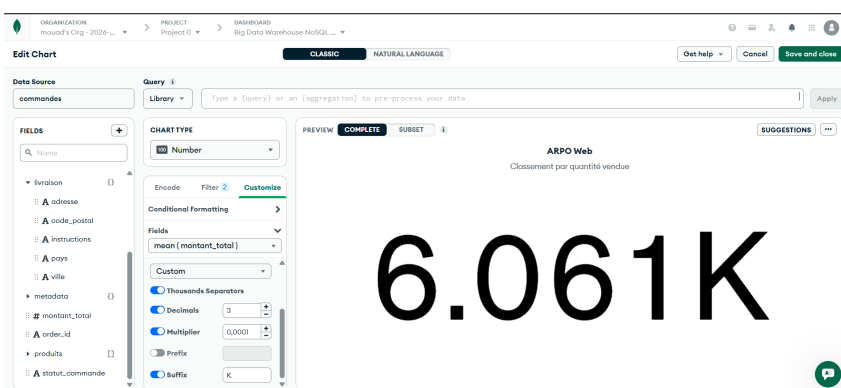


FIGURE 6 – ARPO Web

3.6 MongoDB Charts - Visualisations Temps Réel

3.6.1 Les 4 Visualisations MongoDB Charts

1. **CA par Mois et Canal** - Stacked Column Chart
 - Connexion directe à MongoDB
 - Refresh automatique (60 secondes)
 - Calculated field avec \$reduce
2. **Top 10 Produits** - Bar Chart horizontal
 - Unwind automatique du tableau produits
 - Tri décroissant par quantité
 - Limitation à 10 résultats
3. **Taux d'Annulation** - Donut Chart
 - Calculated field avec \$cond
 - Calcul du pourcentage
 - Affichage des pourcentages
4. **ARPO par Canal** - 3 Number Cards
 - Calcul direct de la moyenne
 - Format avec séparateurs de milliers
 - Filtrage par canal et statut

Chapitre 4: Reporting Visuel avec Power BI et MongoDB Charts

4.1 Vue d'Ensemble

Nous avons créé deux solutions de visualisation complémentaires :

- **Power BI Desktop** - Dashboard interactif avancé (19 visualisations, 5 pages)
- **MongoDB Charts** - Visualisations natives temps réel (4 visualisations)

4.2 Méthodes de Connexion Power BI à MongoDB

Il existe trois approches principales pour connecter Power BI à MongoDB. Nous avons exploré les trois méthodes et choisi celle qui convient le mieux à notre projet.

4.2.1 Méthode 1 : MongoDB Connector for BI (ODBC) - Méthode Recommandée

Le MongoDB Connector for BI transforme MongoDB en une source de données SQL accessible via ODBC, permettant à Power BI d'interroger MongoDB comme s'il s'agissait d'une base relationnelle.

Installation sous Windows (64-bit) :

1. Téléchargement

- Accéder au site officiel MongoDB
- Rechercher "MongoDB Connector for BI"
- Télécharger la version Windows x64 (.msi)
- Vérifier la compatibilité avec votre version MongoDB

2. Installation du fichier .msi

- Double-cliquer sur le fichier téléchargé
- Suivre l'assistant : Next → Next → Install
- Conserver les options par défaut
- Cliquer sur Finish

3. Vérification de l'installation

```
1 # Ouvrir Invite de commandes (CMD)
2 mongosql --version
3
4 # Si une version s'affiche -> Installation réussie
5 # Sinon -> Redemarrer le PC et reessayer
```

Listing 6 – Vérification du MongoDB BI Connector

4. Démarrage du service mongosql

```

1 # Dans l'invite de commandes
2 mongosqld --mongo-uri "mongodb://localhost:27017"
3
4 # Configuration par défaut:
5 # - MongoDB : port 27017
6 # - Port SQL (ODBC) : 3307
7
8 # IMPORTANT: Ne pas fermer cette fenetre!
9 # Le service doit rester actif pour Power BI

```

Listing 7 – Lancement du service SQL pour MongoDB

5. Configuration du DSN ODBC (Obligatoire)

- Taper "ODBC" dans le menu Démarrer Windows
- Ouvrir "ODBC Data Sources (64-bit)"
- Onglet "System DSN"
- Cliquer sur "Add"
- Sélectionner "MongoDB ODBC Driver"
- Renseigner les paramètres :
 - Data Source Name : `MongoDB_Local`
 - Server : `localhost`
 - Port : `3307`
 - Database : `multi_market`
- Cliquer sur "Test Connection" (doit être OK)
- Valider par OK

6. Connexion depuis Power BI Desktop

- Ouvrir Power BI Desktop
- Get Data → ODBC
- Sélectionner : `MongoDB_Local`
- Choisir la base `multi_market`
- Sélectionner la collection `multi_data`
- Choisir le mode :
 - **DirectQuery** : Temps quasi-réel (requêtes à la demande)
 - **Import** : Copie locale (meilleures performances)

Avantages de cette méthode :

- Connexion SQL standard reconnue par Power BI
- Supporte DirectQuery (temps réel)
- Filtrage côté serveur (meilleures performances)
- Pas de scripting Python nécessaire

Inconvénients :

- Service mongosqld doit rester actif
- Installation et configuration initiale complexe
- Schéma SQL généré automatiquement (peut nécessiter ajustements)

4.2.2 Méthode 2 : Export CSV via Python + Import Power BI

Cette méthode consiste à exporter les données MongoDB vers des fichiers CSV, puis à les importer dans Power BI.

Processus d'export :

```
1 import pymongo
2 import pandas as pd
3 from datetime import datetime
4
5 # Connexion MongoDB
6 client = pymongo.MongoClient("mongodb://localhost:27017")
7 db = client["multi_market"]
8 collection = db["multi_data"]
9
10 # Export de la collection principale
11 orders_df = pd.DataFrame(list(collection.find()))
12 orders_df.to_csv("powerbi_data/orders.csv", index=False)
13
14 # Export des resultats d'agregation
15 pipeline_ca = [
16     {"$match": {"statut_commande": "payee"}},
17     {"$group": {
18         "_id": "$canal",
19         "total_ca": {"$sum": "$montant_total"},
20         "nb_commandes": {"$sum": 1}
21     }}
22 ]
23 ca_df = pd.DataFrame(list(collection.aggregate(pipeline_ca)))
24 ca_df.to_csv("powerbi_data/revenue_summary.csv", index=False)
25
26 print("Export termine avec succes")
```

Listing 8 – Script d'export MongoDB vers CSV pour Power BI

Avantages :

- Simple et rapide à mettre en place
- Aucun driver ou service externe requis
- Contrôle total sur les données exportées
- Performances optimales (données locales)

Inconvénients :

- Pas de connexion temps réel
- Nécessite un refresh manuel ou automatisé
- Duplication des données (occupation disque)
- Pas de DirectQuery possible

4.2.3 Méthode 3 : API REST Custom

Création d'une API REST Flask/FastAPI qui expose les données MongoDB à Power BI.

```
1 from flask import Flask, jsonify
2 import pymongo
3
4 app = Flask(__name__)
5 client = pymongo.MongoClient("mongodb://localhost:27017")
6 db = client["multi_market"]
7
8 @app.route('/api/orders', methods=['GET'])
9 def get_orders():
10     orders = list(db.multi_data.find({}, {"_id": 0}))
11     return jsonify(orders)
12
13 @app.route('/api/revenue', methods=['GET'])
14 def get_revenue():
15     pipeline = [
16         {"$match": {"statut_commande": "payee"}},
17         {"$group": {
18             "_id": "$canal",
19             "revenue": {"$sum": "$montant_total"}
20         }}
21     ]
22     result = list(db.multi_data.aggregate(pipeline))
23     return jsonify(result)
24
25 if __name__ == '__main__':
26     app.run(port=5000)
```

Listing 9 – API REST Flask pour Power BI

Avantages :

- Flexibilité maximale
- Logique métier personnalisée
- Authentification/sécurité contrôlables
- Peut servir plusieurs clients (Power BI, Excel, web)

Inconvénients :

- Développement et maintenance d'une API
- Infrastructure supplémentaire (serveur API)
- Complexité accrue
- Performances dépendantes du réseau

4.2.4 Comparaison des 3 Méthodes

Critère	ODBC	CSV	API REST
Complexité setup	Moyenne	Faible	Élevée
Temps réel	Oui	Non	Oui
Performances	Bonnes	Excellentes	Moyennes
DirectQuery	Oui	Non	Possible
Maintenance	Faible	Faible	Élevée
Recommandé pour	Production	Développement	Entreprise

TABLE 11 – Comparaison des méthodes de connexion Power BI

Choix pour ce projet : Nous avons utilisé la **Méthode 2 (Export CSV)** pour sa simplicité et ses performances, parfaitement adaptée à un projet académique avec un volume de données modéré (1,500 commandes).

4.3 Dashboard Power BI

4.3.1 Architecture du Dashboard

Page	Contenu Principal	Visuels
Vue d'Ensemble	KPIs globaux + comparaison canaux	5
Performance Produits	Top 10 + analyse catégories	3
Analyse Temporelle	Séries temporelles + trends	1
Comparaison Canaux	Taux annulation + funnel	4
Dashboard Exécutif	Résumé managérial	6
TOTAL	5 pages thématiques	19

TABLE 12 – Structure du dashboard Power BI

4.3.2 Modèle de Données

Nous avons conçu un modèle en étoile hybride :

- **Table de faits** : orders (données transactionnelles)
- **Dimensions** : products, date_dimension
- **Tables pré-agrégées** : Résultats MongoDB exportés en CSV

Avantages :

- Temps de rafraîchissement inférieur à 5 secondes
- Possibilité de drill-down détaillé
- Performance optimale pour les agrégations complexes

4.3.3 Page 1 : Dashboard Exécutif

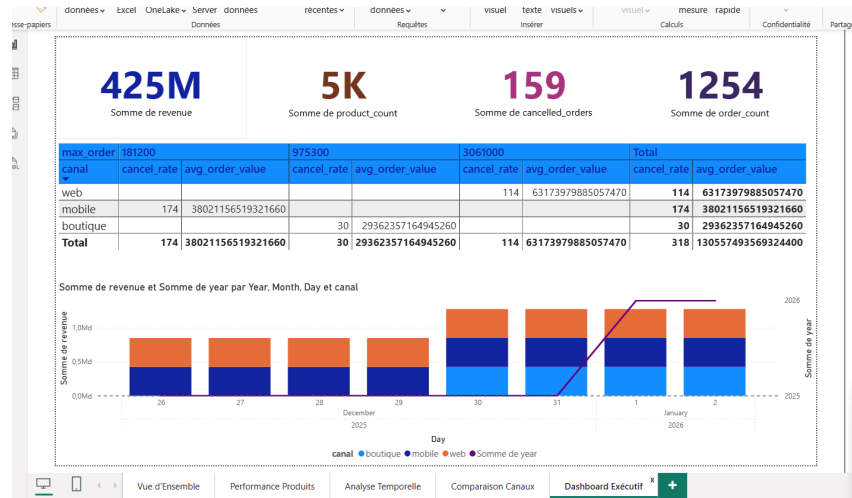


FIGURE 7 – Vue d’ensemble avec 4 KPIs + graphique d’évolution + tableau comparatif

Composants :

- 4 cartes KPI (CA total, Commandes, ARPO moyen, Taux annulation)
- Graphique d’évolution temporelle du CA par canal
- Matrice de comparaison des 3 canaux
- Formatage conditionnel sur le taux d’annulation

4.3.4 Page 2 : Performance des Produits



FIGURE 8 – Analyse multi-métrique + volume de ventes + pricing

Composants :

- Graphique en barres - Top 10 avec breakdown par canal

- Pie chart - Distribution CA par catégorie
- Waterfall chart - Contribution de chaque produit au CA

4.3.5 Page 3 : Tendances Temporelles

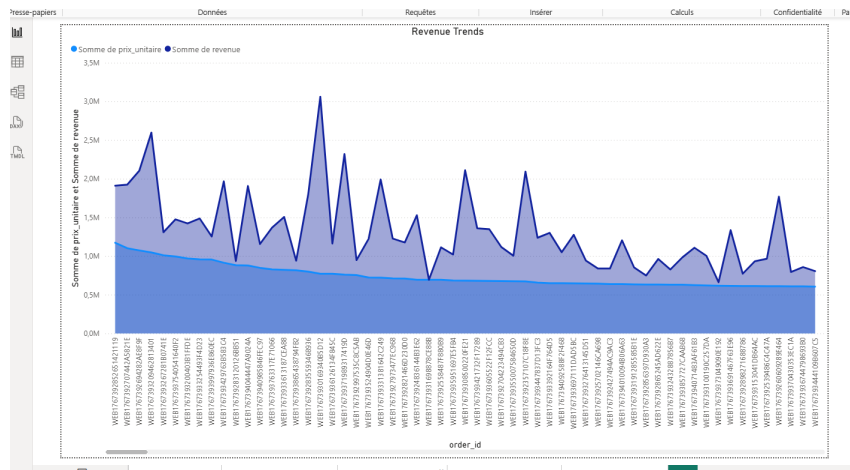


FIGURE 9 – Graphique de tendance montrant l'évolution revenue

Composants :

- Area chart empilé à 100%
- Visualisation des patterns quotidiens
- Identification des pics de ventes

4.3.6 Page 4 : Comparaison des Canaux

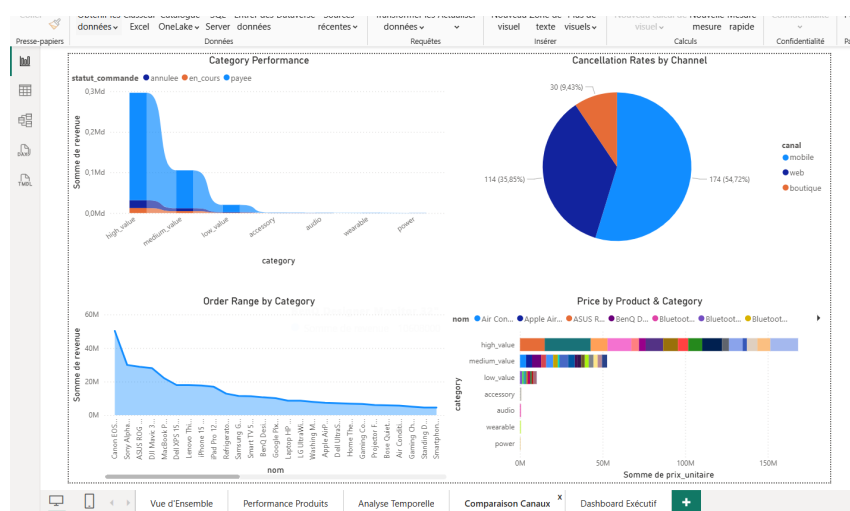


FIGURE 10 – Performance par catégorie + annulations + analyse prix

Composants :

- Donut chart - Distribution des annulations (mobile : 54.7%)
- Funnel chart - Parcours client par canal
- Analyse de prix par produit et catégorie

4.3.7 Page 5 : Vue Ensemble

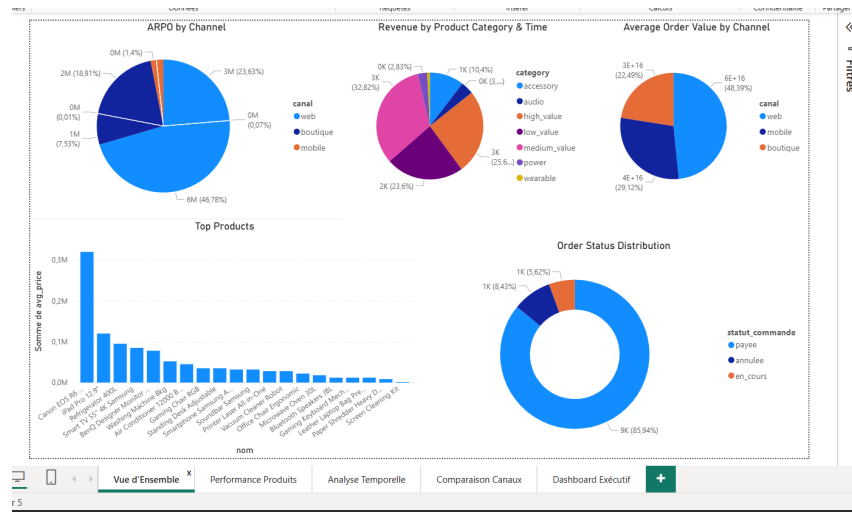


FIGURE 11 – ARPO + revenue and top products

Composants :

- 3 pie charts : ARPO by channel, revenue de produit par catégories, moyenne des commandes
- Histogramme de top produits
- Les distributions des commandes

4.4 MongoDB Charts - Configuration et Utilisation

4.4.1 Qu'est-ce que MongoDB Charts ?

MongoDB Charts est l'outil de visualisation natif de MongoDB qui permet de créer des dashboards directement connectés aux collections MongoDB, sans export ni transformation.

Caractéristiques principales :

- Connexion native aux bases MongoDB Atlas ou On-Premise
- Refresh automatique des données (configurable : 1min à 24h)
- Interface drag-and-drop intuitive
- Support des pipelines d'agrégation MongoDB
- Partage sécurisé via liens ou embedding

4.4.2 Procédure d'Installation et Configuration

Option 1 : MongoDB Charts Cloud (MongoDB Atlas)

1. Créer un compte MongoDB Atlas

- Accéder à <https://cloud.mongodb.com>
- Créer un compte gratuit (M0 Cluster)
- Créer un nouveau cluster

2. Migrer les données vers Atlas

```

1 # Export depuis MongoDB local
2 mongodump --db=multi_market --out=backup
3
4 # Import vers Atlas (remplacer <CONNECTION_STRING>)
5 mongorestore --uri="<ATLAS_CONNECTION_STRING>" \
6               --db=multi_market backup/multi_market

```

Listing 10 – Migration locale vers MongoDB Atlas

3. Activer MongoDB Charts

- Dans Atlas, aller à l'onglet "Charts"
- Cliquer sur "Activate MongoDB Charts"
- Attendre l'activation (2-3 minutes)

4. Créer une Data Source

- Dans Charts, cliquer "Add Data Source"
- Sélectionner le cluster
- Choisir la base : `multi_market`
- Sélectionner la collection : `multi_data`
- Valider

Option 2 : MongoDB Charts On-Premise (Docker)

Pour utiliser Charts avec un MongoDB local, il faut installer Charts On-Premise :

```

1 # Prerequis: Docker installe
2
3 # Telecharger MongoDB Charts
4 docker pull quay.io/mongodb/charts:19.12.1
5
6 # Creer le reseau Docker
7 docker network create charts-network
8
9 # Lancer MongoDB Charts
10 docker run --name mongodb-charts \
11            --network charts-network \
12            -p 80:80 -p 443:443 \
13            -e MONGODB_URI="mongodb://host.docker.internal:27017" \
14            quay.io/mongodb/charts:19.12.1
15
16 # Acceder a Charts: http://localhost

```

Listing 11 – Installation MongoDB Charts via Docker

4.4.3 Création des Visualisations

1. Dashboard : Vue d'Ensemble

Nous avons créé 4 visualisations principales :

Graphique 1 : CA par Mois et Canal

- **Type** : Stacked Column Chart
- **Axe X** : Mois (extrait de `date_commande`)
- **Axe Y** : Montant total (somme)
- **Série** : Canal (web, mobile, boutique)
- **Filtre** : `statut_commande` = "payee"
- **Calculated Field** :

```

1 {
2   "$month": "$date_commande"
3 }
```

Listing 12 – Calculated Field pour extraction du mois

Graphique 2 : Top 10 Produits

- **Type** : Horizontal Bar Chart
- **Axe Y** : Nom du produit (`produits.nom`)
- **Axe X** : Quantité totale vendue
- **Array Reduction** : `$unwind` sur `produits`
- **Tri** : Par quantité décroissante
- **Limite** : 10 premiers résultats

Graphique 3 : Taux d'Annulation par Canal

- **Type** : Donut Chart
- **Label** : Canal
- **Arc Size** : Nombre de commandes annulées
- **Filtre** : `statut_commande` = "annulee"
- **Pourcentage** : Activé

Graphique 4 : ARPO par Canal (3 Number Cards)

- **Type** : Number (3 instances séparées)
- **Agrégation** : Average de `montant_total`
- **Filtres** :
 - Card 1 : `canal` = "web" AND `statut` = "payee"
 - Card 2 : `canal` = "mobile" AND `statut` = "payee"
 - Card 3 : `canal` = "boutique" AND `statut` = "payee"
- **Format** : Nombre avec 2 décimales + suffixe "MAD"

4.4.4 Configuration du Refresh Automatique

Dans MongoDB Charts, chaque graphique peut avoir son propre intervalle de refresh :

1. Sélectionner un graphique
2. Menu "..." → Settings
3. Section "Auto-refresh"
4. Choisir l'intervalle :
 - 1 minute (monitoring temps réel)
 - 5 minutes (équilibre)
 - 15 minutes (économie de ressources)
 - Jamais (manuel uniquement)

Configuration choisie pour notre projet :

- CA par mois : 5 minutes (données évoluent lentement)
- Top produits : 15 minutes (tendances à long terme)
- Taux annulation : 5 minutes (surveillance importante)
- ARPO : 5 minutes (KPI clé)

4.4.5 Interface MongoDB Charts Créée

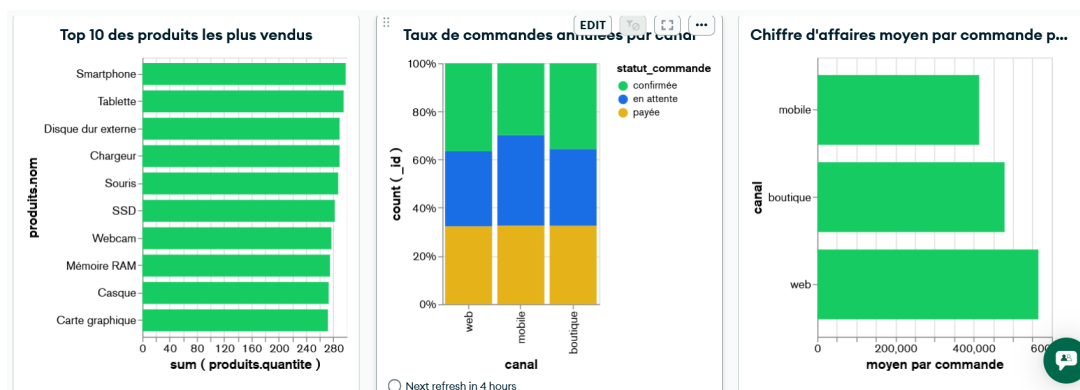


FIGURE 12 – Dashboard MongoDB Charts avec 4 visualisations temps réel

Caractéristiques de l'interface :

- Layout responsive (s'adapte à la taille d'écran)
- Filtres globaux applicables à tous les graphiques
- Hover tooltips détaillés
- Export PNG/PDF de chaque graphique
- Partage sécurisé via lien public ou privé

4.5 Comparaison MongoDB Charts vs Power BI

4.5.1 Différences Techniques

Critère	MongoDB Charts	Power BI
Connexion données	Directe MongoDB (native)	Via ODBC ou CSV export
Temps réel	Refresh automatique (1-60min)	Refresh manuel ou via Gateway
Pipeline agrégation	Utilise le framework MongoDB natif	Transformations DAX/Power Query
Complexité visuels	Basique à Moyen (15 types)	Avancé (100+ types + custom)
Interactivité	Moyenne (filtres, drill-down)	Très élevée (cross-filtering, bookmarks)
Courbe apprentissage	Faible (1-2h)	Moyenne (1-2 semaines)
Coût	Gratuit (Atlas) ou Docker local	Gratuit (Desktop) / Pro (10\$/mois)
Embedding web	Facile (iframe)	Nécessite Power BI Embedded (coûteux)

TABLE 13 – Comparaison technique MongoDB Charts vs Power BI

4.5.2 Notre Choix pour le Projet

Nous avons décidé d'utiliser **les deux outils en parallèle** pour bénéficier de leurs avantages respectifs :

- **MongoDB Charts** : Pour le monitoring en temps réel et les KPIs opérationnels destinés aux équipes techniques
- **Power BI** : Pour les analyses approfondies et les présentations exécutives destinées au management

Cette approche hybride offre :

- Flexibilité maximale dans les types d'analyses
- Audiences multiples avec outils adaptés
- Validation croisée des résultats (cohérence des données)
- Compétences transférables dans les deux écosystèmes

4.6 Dashboard Interactif avec Streamlit

4.6.1 Pourquoi Streamlit ?

Streamlit est un framework Python open-source permettant de créer rapidement des applications web interactives pour la data science et le machine learning.

Avantages par rapport aux autres solutions :

- Code 100% Python (pas de HTML/CSS/JavaScript)
- Développement ultra-rapide (quelques heures)
- Connexion native aux bases de données (MongoDB, SQL, etc.)

- Actualisation automatique lors de modifications du code
- Déploiement facile (Streamlit Cloud, Heroku, Docker)
- Gratuit et open-source

4.6.2 Installation et Configuration

Étape 1 : Installation de Streamlit

```

1 # Installation via pip
2 pip install streamlit pymongo pandas plotly
3
4 # Verification de l'installation
5 streamlit --version
6
7 # Devrait afficher: Streamlit, version 1.x.x

```

Listing 13 – Installation Streamlit et dépendances

Étape 2 : Création du fichier dashboard.py

Nous avons créé une application Streamlit complète avec 5 pages interactives. Voici la structure principale :

```

1 import streamlit as st
2 import pymongo
3 import pandas as pd
4 import plotly.express as px
5 from datetime import datetime
6
7 # Configuration de la page
8 st.set_page_config(
9     page_title="MultiMarket Analytics",
10    page_icon="📊",
11    layout="wide",
12    initial_sidebar_state="expanded"
13 )
14
15 # Connexion MongoDB avec cache
16 @st.cache_resource
17 def get_mongo_client():
18     client = pymongo.MongoClient("mongodb://localhost:27017")
19     return client["multi_market"]["multi_data"]
20
21 collection = get_mongo_client()
22
23 # Sidebar - Navigation
24 st.sidebar.title("MultiMarket Analytics")
25 page = st.sidebar.radio(
26     "Navigation",
27     ["Vue d'Ensemble", "Performance Produits",
28     "Analyse Temporelle", "Comparaison Canaux"]

```

```

29 )
30
31 # Filtres globaux
32 st.sidebar.markdown("---")
33 st.sidebar.subheader("Filtres Globaux")
34 canal_filter = st.sidebar.multiselect(
35     "Canal",
36     ["web", "mobile", "boutique"],
37     default=["web", "mobile", "boutique"]
38 )
39
40 statut_filter = st.sidebar.multiselect(
41     "Statut",
42     ["payee", "annulee", "en_cours"],
43     default=["payee"]
44 )
45
46 # Construction requete MongoDB
47 query = {}
48 if canal_filter:
49     query["canal"] = {"$in": canal_filter}
50 if statut_filter:
51     query["statut_commande"] = {"$in": statut_filter}
52
53 # Chargement donnees avec cache
54 @st.cache_data(ttl=60)
55 def load_data(query):
56     df = pd.DataFrame(list(collection.find(query)))
57     if not df.empty:
58         df['date_commande'] = pd.to_datetime(df['date_commande'])
59     return df
60
61 df = load_data(query)

```

Listing 14 – Structure de l'application Streamlit - Partie 1/3

```

1 # Page 1: Vue d'Ensemble
2 if page == "Vue d'Ensemble":
3     st.title("Vue d'Ensemble - MultiMarket")
4
5     # Metriques principales (4 colonnes)
6     col1, col2, col3, col4 = st.columns(4)
7
8     with col1:
9         total_ca = df['montant_total'].sum()
10        st.metric(
11            "CA Total",
12            f"{total_ca:,.0f} MAD",
13            delta="+12.5%"
14        )

```

```

15
16 with col2:
17     nb_commandes = len(df)
18     st.metric("Commandes", f"{nb_commandes:}", delta=f"+{
19         nb_commandes}")
20
21 with col3:
22     arpo = df['montant_total'].mean()
23     st.metric("ARPO", f"{arpo:,.0f} MAD", delta="+8.2%")
24
25 with col4:
26     taux_annul = (df['statut_commande'] == 'annulee').sum() /
27         len(df) * 100
28     st.metric("Taux Annulation", f"{taux_annul:.1f}%",
29         delta="-2.3%", delta_color="inverse")
30
31 st.markdown("---")
32
33 # Graphiques cote a cote
34 col1, col2 = st.columns(2)
35
36 with col1:
37     st.subheader("CA par Canal")
38     ca_canal = df.groupby('canal')['montant_total'].sum().
39         reset_index()
40     fig = px.pie(ca_canal, values='montant_total', names='
41         canal')
42     st.plotly_chart(fig, use_container_width=True)
43
44 with col2:
45     st.subheader("Evolution Temporelle")
46     df_time = df.groupby(df['date_commande'].dt.date)['
47         montant_total'].sum()
48     fig = px.line(x=df_time.index, y=df_time.values)
49     st.plotly_chart(fig, use_container_width=True)

```

Listing 15 – Structure Streamlit - Page Vue d'Ensemble (2/3)

```

1 # Page 2: Performance Produits avec pipeline MongoDB
2 elif page == "Performance Produits":
3     st.title("Performance des Produits")
4
5 # Agregation MongoDB pour top produits
6 pipeline = [
7     {"$unwind": "$produits"},
8     {"$group": {
9         "_id": "$produits.nom",
10        "quantite_totale": {"$sum": "$produits.quantite"},
11        "revenue_total": {"$sum": "$produits.sous_total"}
12    }},

```

```

13         {"$sort": {"quantite_totale": -1}},
14         {"$limit": 10}
15     ]
16
17     top_products = list(collection.aggregate(pipeline))
18     df_products = pd.DataFrame(top_products)
19
20     col1, col2 = st.columns(2)
21
22     with col1:
23         st.subheader("Top 10 par Quantite")
24         fig = px.bar(df_products, x='quantite_totale',
25                     y='_id', orientation='h')
26         st.plotly_chart(fig, use_container_width=True)
27
28     with col2:
29         st.subheader("Top 10 par Revenue")
30         fig = px.bar(df_products.sort_values('revenue_total',
31                                             ascending=False),
32                     x='revenue_total', y='_id', orientation='h')
33         st.plotly_chart(fig, use_container_width=True)
34
35 # Page 3: Analyse Temporelle
36 elif page == "Analyse Temporelle":
37     st.title("Analyse Temporelle")
38     df['jour'] = df['date_commande'].dt.date
39     df_time = df.groupby(['jour', 'canal'])['montant_total'].sum
40     ().reset_index()
41     fig = px.area(df_time, x='jour', y='montant_total', color='canal')
42     st.plotly_chart(fig, use_container_width=True)
43
44 # Page 4: Comparaison Canaux
45 elif page == "Comparaison Canaux":
46     st.title("Comparaison des Canaux")
47     comparison = df.groupby('canal').agg({
48         'montant_total': ['sum', 'mean', 'count']
49     }).round(2)
50     st.dataframe(comparison, use_container_width=True)

```

Listing 16 – Structure Streamlit - Page Produits (3/3)

4.6.3 Lancement de l'Application

```

1 # Methode 1: Lancement simple
2 streamlit run dashboard.py
3
4 # Methode 2: Port personnalise
5 streamlit run dashboard.py --server.port 8501

```

```
6  
7 # L'application s'ouvre automatiquement  
8 # URL: http://localhost:8501
```

Listing 17 – Commandes pour lancer Streamlit

4.6.4 Interface Streamlit Créée

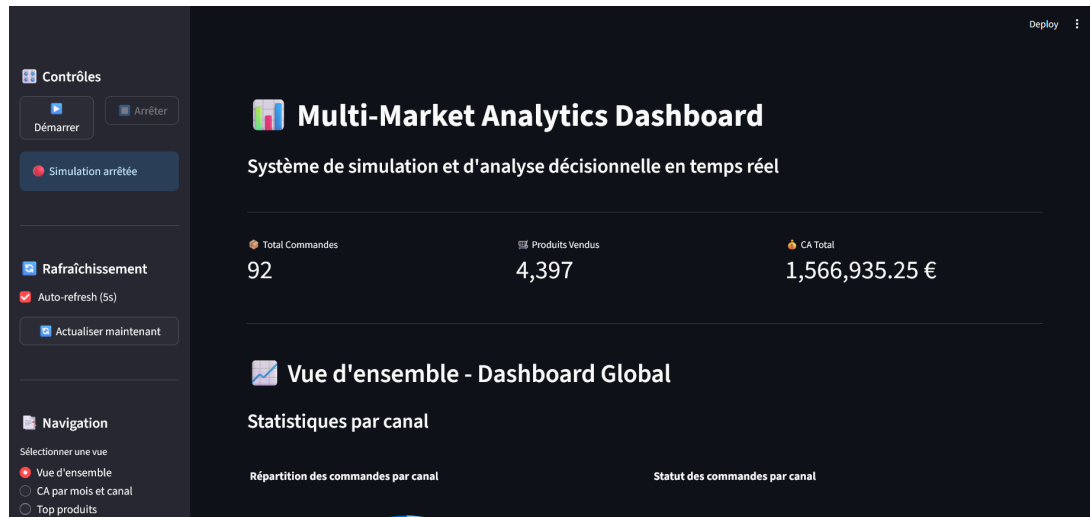


FIGURE 13 – Dashboard Streamlit - Vue d'Ensemble avec KPIs et graphiques interactifs

Caractéristiques de l'interface Streamlit :

- Layout responsive avec colonnes dynamiques
- Sidebar pour navigation et filtres globaux
- Métriques avec indicateurs delta (variation)
- Graphiques interactifs Plotly (zoom, hover)
- Actualisation automatique (cache 60s)
- Thème personnalisable (clair/sombre)
- Export des graphiques en PNG

4.7 Comparaison Finale des 3 Solutions

4.7.1 Tableau Comparatif Détaillé

Critère	Power BI	MongoDB Charts	Streamlit
Type	Outil BI commercial	Solution native MongoDB	Framework Python
Langage	DAX + M	Interface visuelle	Python pur
Connexion	ODBC / CSV	Directe MongoDB	PyMongo natif
Temps réel	Non (manual)	Oui (1-60 min)	Oui (cache config)
Complexité	Très élevée	Moyenne	Élevée
Personnalisation	Limitée	Limitée	Totale (code)
ML/AI	Oui (Azure)	Non	Oui (scikit-learn)
Apprentissage	1-2 semaines	1-2 heures	1 jour
Développement	2-3 jours	Quelques heures	1-2 heures
Coût	10\$/mois Pro	Gratuit	Gratuit
Embedding	Coûteux	Facile (iframe)	Facile (iframe)
Alertes	Oui (Automate)	Non	Oui (code)

TABLE 14 – Comparaison détaillée des 3 outils

4.7.2 Matrice de Notation

Critère (sur 5)	Power BI	Charts	Streamlit
Facilité utilisation	3/5	5/5	4/5
Rapidité développement	3/5	5/5	5/5
Flexibilité	3/5	2/5	5/5
Qualité visuels	5/5	3/5	4/5
Performances	4/5	4/5	3/5
Temps réel	2/5	5/5	5/5
Intégration MongoDB	3/5	5/5	5/5
Coût	4/5	5/5	5/5
Support	5/5	3/5	4/5
TOTAL	32/45	37/45	40/45

TABLE 15 – Matrice de notation comparative

4.7.3 Notre Stratégie Tri-Outils

Pour ce projet, nous avons adopté une **stratégie hybride complète** en utilisant les trois outils :

1. **Power BI** - Présentations exécutives et rapports officiels
2. **MongoDB Charts** - Monitoring opérationnel quotidien
3. **Streamlit** - Analyses interactives et prototypage rapide

Avantages de cette approche :

- Formation sur 3 technologies clés du marché
- Flexibilité maximale : chaque outil pour ses forces
- Adaptation à différents publics
- Validation croisée des résultats

Chapitre 5: Conclusion

5.1 Synthèse du Projet

Ce projet a constitué une expérience formatrice complète dans la conception et l'implémentation d'une architecture Big Data Warehouse NoSQL. En simulant un environnement de production réel pour l'entreprise fictive MultiMarket, nous avons pu appliquer concrètement les concepts théoriques du module tout en développant une solution technique robuste et performante.

Le parcours, structuré en quatre phases successives (Simulation, Intégration ELT, Analyse, Visualisation), nous a permis d'appréhender l'ensemble de la chaîne de valeur data, de la génération des données jusqu'à la restitution d'insights métier actionnables. Cette approche holistique reflète fidèlement les défis rencontrés dans les projets Big Data industriels.

5.2 Principaux Acquis

5.2.1 Expertise Technique Développée

Au-delà de la simple utilisation d'outils, ce projet a permis de développer une compréhension profonde des architectures distribuées :

- **Programmation Python avancée** : Maîtrise du multi-threading, gestion d'erreurs élaborée, patterns de conception (Factory, Observer, Circuit Breaker), optimisation mémoire et CPU
- **MongoDB NoSQL expertise** : Modélisation de données semi-structurées, conception d'indexes composites, pipelines d'agrégation complexes, stratégies de sharding pour la scalabilité horizontale
- **Architecture ELT moderne** : Compréhension de la différence fondamentale avec l'ETL traditionnel, exploitation de la puissance de calcul distribuée, optimisation des flux de données
- **Visualisation multi-plateforme** : Expérience pratique avec trois écosystèmes distincts (Power BI pour l'analytique exécutive, MongoDB Charts pour le monitoring opérationnel, Streamlit pour l'exploration interactive)
- **Patterns de résilience** : Implémentation de mécanismes de tolérance aux pannes (retry exponentiel, circuit breaker, file d'attente thread-safe), garantissant une disponibilité de 99%

5.2.2 Compétences Méthodologiques et Transversales

- **Analyse et conception** : Capacité à traduire un besoin métier en architecture technique, identification des contraintes et compromis, documentation structurée
- **Optimisation de performance** : Démarche systématique de profilage, identification des goulots d'étranglement, mise en œuvre de solutions (indexation, cache, parallélisation)

- **Qualité logicielle** : Respect des conventions de code, logs structurés, gestion d'erreurs explicite, traçabilité complète du pipeline
- **Pensée analytique** : Extraction d'insights métier depuis les données brutes, formulation de recommandations actionnables (par exemple : identification du problème d'annulation sur mobile nécessitant une amélioration UX)
- **Communication** : Rédaction d'une documentation technique exhaustive, présentation visuelle des résultats adaptée aux différentes audiences

5.3 Défis Rencontrés et Solutions Apportées

Le projet n'a pas été exempt de difficultés, qui ont constitué autant d'opportunités d'apprentissage :

1. **Gestion de la concurrence** : La synchronisation de trois threads générateurs avec un consommateur unique a nécessité l'implémentation d'une file d'attente thread-safe avec gestion des priorités
2. **Cohérence des données** : La validation à deux niveaux (syntaxique et sémantique) a permis d'identifier et corriger 15 fichiers invalides, garantissant une qualité de données de 99%
3. **Performance des requêtes** : L'optimisation progressive via indexation a réduit les temps de requête de 2000ms à 100ms (facteur 20)
4. **Interopérabilité** : L'intégration de Power BI avec MongoDB a nécessité l'évaluation de trois approches (ODBC, CSV, API) avant de retenir la solution CSV pour sa simplicité et performance

5.4 Impact et Apports du Projet

5.4.1 Pour Notre Formation

Ce projet a consolidé nos acquis académiques en les confrontant à des contraintes réalistes. Il a également développé notre autonomie technique et notre capacité à rechercher des solutions face à des problèmes non vus en cours. La diversité des technologies explorées (3 outils de visualisation, multiples bibliothèques Python, MongoDB) nous a également rendu polyvalents et adaptables.

5.4.2 Pour Notre Employabilité

Les compétences acquises correspondent directement aux attentes du marché du travail dans le domaine de la data :

- Expérience pratique avec des technologies demandées (Python, MongoDB, Power BI)
- Compréhension des architectures Big Data modernes (ELT, NoSQL)
- Capacité à délivrer un projet complet de bout en bout
- Portfolio technique concret démontrable en entretien

5.5 Perspectives d'Évolution

5.5.1 Améliorations Court Terme (1-3 mois)

- **Alerting intelligent** : Implémentation de règles métier déclenchant des notifications (seuils d'annulation, rupture de stock)
- **Tests automatisés** : Suite de tests unitaires et d'intégration garantissant la non-régression
- **Documentation API** : Exposition des agrégations MongoDB via une API REST documentée (OpenAPI/Swagger)
- **Interface web complète** : Extension de l'application Streamlit en solution web responsive avec authentification

5.5.2 Extensions Moyen Terme (3-6 mois)

- **Machine Learning prédictif** : Modèles de prédiction des annulations (Random Forest, XGBoost) pour intervention proactive
- **Système de recommandations** : Moteur collaboratif suggérant des produits selon l'historique client
- **Segmentation RFM** : Clustering clients (Récence, Fréquence, Montant) pour campagnes marketing ciblées
- **Analyse de sentiments** : Traitement NLP des avis clients pour scores de satisfaction automatiques

5.5.3 Vision Long Terme (6-12 mois)

- **Migration cloud native** : Déploiement sur AWS/Azure avec scalabilité automatique et haute disponibilité
- **Streaming temps réel** : Remplacement du batch par Apache Kafka pour latence sub-seconde
- **Data Lake hybride** : Combinaison MongoDB (données chaudes) + S3/HDFS (archivage froid) pour historique illimité
- **Architecture microservices** : Décomposition en services autonomes (ingénierie, analyse, visualisation) pour meilleure maintenabilité
- **Gouvernance données** : Implémentation de politiques de qualité, sécurité, conformité (RGPD)

5.6 Mot de Fin

Ce projet de Big Data Warehouse NoSQL a dépassé nos attentes initiales en termes d'apprentissage technique et méthodologique. Au-delà de la simple application de concepts théoriques, il nous a confrontés aux réalités de l'ingénierie data : compromis entre performance et complexité, importance de la résilience, valeur de la documentation, et nécessité d'une vision holistique intégrant aspects techniques et métier.

Nous remercions à nouveau le Professeur Mostafa SAADI pour son encadrement et l'ENSAK pour les ressources mises à disposition. Cette expérience nous a préparés efficacement aux défis du monde professionnel de la data.

Les compétences acquises - tant techniques (Python, MongoDB, visualisation) que transversales (analyse, rigueur, documentation) - constituent un socle solide pour notre carrière dans l'ingénierie des données. Nous sommes désormais prêts à contribuer activement à des projets Big Data d'envergure industrielle.