GOLDEN BIT

RAPPORT

Plateforme de Vente de Billets d'Événements

2025
2026

AABBOU IMRANE
MOHAMMED FEJJAR
MOHAMED MACHLOU
BRAHIM DARGUI

Résumé

Ce projet est une application en ligne de commande (CLI) développée en Python pour gérer des événements (concerts et conférences), vendre des billets (Standard et VIP) et produire des rapports d'analyse.

L'architecture est modulaire (inspirée de Laravel) pour favoriser la maintenabilité et la scalabilité. L'interface s'appuie sur la librairie questionary pour des saisies interactives et des validations en ligne.

Le système stocke les données en JSON (fichiers dans storage/) et produit un rapport visuel (image) présentant la répartition VIP vs Standard par événement.

Introduction

Dans un contexte où l'industrie de l'événementiel connaît une croissance rapide, la gestion efficace des inscriptions et des ventes de billets représente un enjeu crucial. De nombreuses petites structures s'appuient encore sur des outils rudimentaires, tels que des tableurs, qui deviennent rapidement inadaptés face à l'augmentation du volume de données et à la nécessité d'assurer une expérience fluide pour les participants comme pour les organisateurs.

C'est dans ce cadre que s'inscrit le présent projet : la conception et le développement d'une **plateforme de billetterie interne en ligne de commande (CLI)**, destinée à centraliser l'ensemble des opérations liées à l'organisation et à la gestion d'événements. Cette application vise à répondre à plusieurs problématiques :

Centraliser les informations relatives aux événements (concerts, conférences) et aux acheteurs.

Assurer la fiabilité des ventes de billets grâce à une gestion rigoureuse des capacités, évitant ainsi les situations de survente.

Automatiser et fiabiliser la sauvegarde et la persistance des données, en remplaçant les tableurs traditionnels par un système structuré basé sur des fichiers JSON.

Fournir des outils d'analyse pour évaluer la performance des événements et comprendre la répartition des ventes (Standard vs VIP).

Au-delà de la simple gestion des inscriptions, ce projet constitue également un exercice complet de mise en pratique des compétences acquises durant la formation. Il intègre les principes de la **programmation orientée objet (POO) avec héritage**, la **gestion des données persistantes**, la **construction d'une interface utilisateur CLI robuste et intuitive**, ainsi que l'exploitation de bibliothèques telles que **Pandas** et **Matplotlib** pour l'analyse statistique et la visualisation graphique.

Objectifs

- Concevoir une application CLI ergonomique et fiable pour gérer événements, acheteurs et ventes.
- Appliquer une architecture modulaire (inspirée par Laravel) : séparation des responsabilités, tests facilités, maintenance.
- Assurer la validation stricte des entrées (typing / règles de validation).
- Générer des rapports analytiques (tableaux + graphique) pour visualiser la répartition Standard vs VIP et le chiffre d'affaires par événement.
- Documenter clairement l'installation, l'usage et les choix techniques.

Architecture & Conception

Ce projet s'inspirent des bonnes pratiques observées dans les frameworks modernes comme **Laravel**, en privilégiant une séparation claire des responsabilités et une organisation structurée en dossiers. Cette approche modulaire vise à améliorer la lisibilité, la maintenabilit, l'évolutivité et la séparation des responsabilité du code, tout en facilitant la collaboration entre développeurs. Chaque composant est isolé dans un espace dédié (modèles, services, validations, utilitaires, etc.), ce qui permet de centraliser la logique métier et de limiter les dépendances croisées.

Afin d'assurer la portabilité et le suivi des dépendances, le projet a également été encapsulé dans un **environnement virtuel (venv)**, garantissant une exécution homogène sur différentes machines ainsi qu'une gestion automatisée des bibliothèques nécessaires. Cette structure offre donc une base solide, comparable aux standards utilisés dans des environnements professionnels, et prépare le terrain pour une application CLI robuste et extensible.

- **main.py** : point d'entrée de l'application.
- **models/** : classes représentant les entités métier : Evenement, Concert, Conference, Billet, Vente, Acheteur. Chaque modèle contient les propriétés, une validation basique et des helpers.
- **services/**:
  - **evenement_service.py** : CRUD des événements, calcul de disponibilité.
  - **vente_service.py** : enregistrement d'une vente, vérification de la capacité, mise à jour des places_vendues.
  - **rapport_service.py** : préparation des données pour les tableaux et graphiques.
- **validations/** : règles de validation plus fines (ex : dates, prix > 0, quantités entières positives).
- **utils/** : helpers (chargement JSON, logger, clear screen, formats de date).
- **seeders/** : scripts pour pré-remplir storage/ pour tests/démos.

- **storage/** : fichiers JSON persistants (evenements.json, ventes.json, acheteurs.json) et images générées (rapport_ventes.png).

Installation

Pour garantir une installation simple et reproductible, toutes les dépendances du projet sont centralisées dans le fichier requirements.txt.

Ainsi, après avoir créé et activé un environnement virtuel Python, il suffit d'exécuter la commande `pip install -r requirements.txt` afin d'installer automatiquement l'ensemble des librairies nécessaires.

Une fois cette étape terminée, l'application peut être lancée directement depuis le point d'entrée main.py avec `python main.py`.

Cette démarche assure à la fois la portabilité du projet et une expérience d'exécution cohérente entre tous les environnements.

Modèle de données

Avant de présenter directement le code, il est important d'expliquer la logique qui a guidé la conception du **modèle de données**.
Notre projet repose sur une représentation claire et bien structurée des différentes entités qui interviennent dans une plateforme de vente de billets d'événements.

Chaque entité correspond à un objet réel du domaine, avec ses propres propriétés et son rôle dans le fonctionnement global du système.
Par exemple, un **Événement** peut être un concert ou une conférence ; il possède des informations essentielles comme un titre, une date, un lieu et une capacité d'accueil.
Un **Billet** est lié à un type (Standard, VIP, etc.) et comporte un prix défini par rapport à la base tarifaire de l'événement.
Un **Acheteur** représente la personne qui achète des billets, tandis qu'une **Vente** est l'opération qui associe un acheteur, un événement et un ou plusieurs billets.

Ce découpage en entités nous permet d'assurer plusieurs avantages :

Une **clarté** dans l'organisation des données, chaque objet ayant une responsabilité bien délimitée.

Une **extensibilité**, car il est facile d'ajouter de nouveaux types d'événements ou de billets sans modifier toute la structure.

Une **fiabilité**, car le modèle impose des contraintes (par exemple, la capacité maximale d'un événement ou la quantité de billets disponibles).

Une **réutilisabilité**, puisque les classes de base (Événement, Billet, Vente, Acheteur) peuvent servir de fondations pour d'autres extensions futures.

Ainsi, avant de passer à la partie technique et au code, nous avons réfléchi à un schéma conceptuel clair qui se traduit par une hiérarchie de classes Python organisées dans le dossier `models/`.
Chaque modèle contient non seulement les attributs essentiels, mais aussi des méthodes spécifiques permettant de gérer la logique métier (comme la réservation, la vérification des places restantes ou le calcul du prix final).

voici comment les données résultantes issues des modèles sont enregistrées dans les fichiers de stockage au format JSON.

```json
[
  {
    "id_evenement": 1,
    "titre": "Conférence Internationale sur l'IA",
    "date": "2025-11-05T09:00:00",
    "lieu": "Palais des Congrès, Paris",
    "prix_base": 150.0,
    "capacite": 300,
    "places_vendues": 123,
    "orateur_principal": "Dr. Jeanne Dupont"
  },
  {
    "id_evenement": 2,
    "titre": "Concert Symphonique",
    "date": "2025-12-12T20:00:00",
    "lieu": "Opéra Garnier, Paris",
    "prix_base": 80.0,
    "capacite": 500,
    "places_vendues": 250,
    "artiste": "Orchestre National de France"
  }
]
```

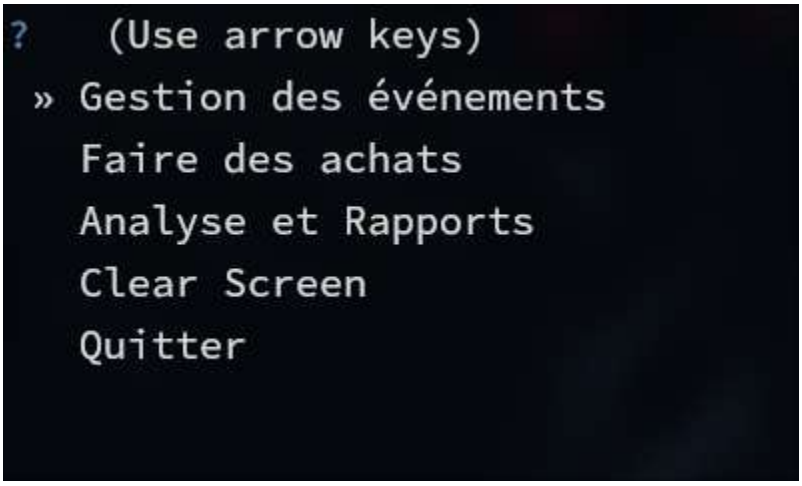
Image 1 - evenements.json

```
[
  {
    "id_vente": 1,
    "id_evenement": 1,
    "id_acheteur": 101,
    "type_billet": "Standard",
    "quantite": 3,
    "date": "2025-10-02T11:16:11",
    "prix_total": 450.0
  },
  {
    "id_vente": 2,
    "id_evenement": 1,
    "id_acheteur": 102,
    "type_billet": "VIP",
    "quantite": 2,
    "date": "2025-10-03T09:30:00",
    "prix_total": 450.0
  }
]
```

Image 2 - ventes.json

Méthodologie

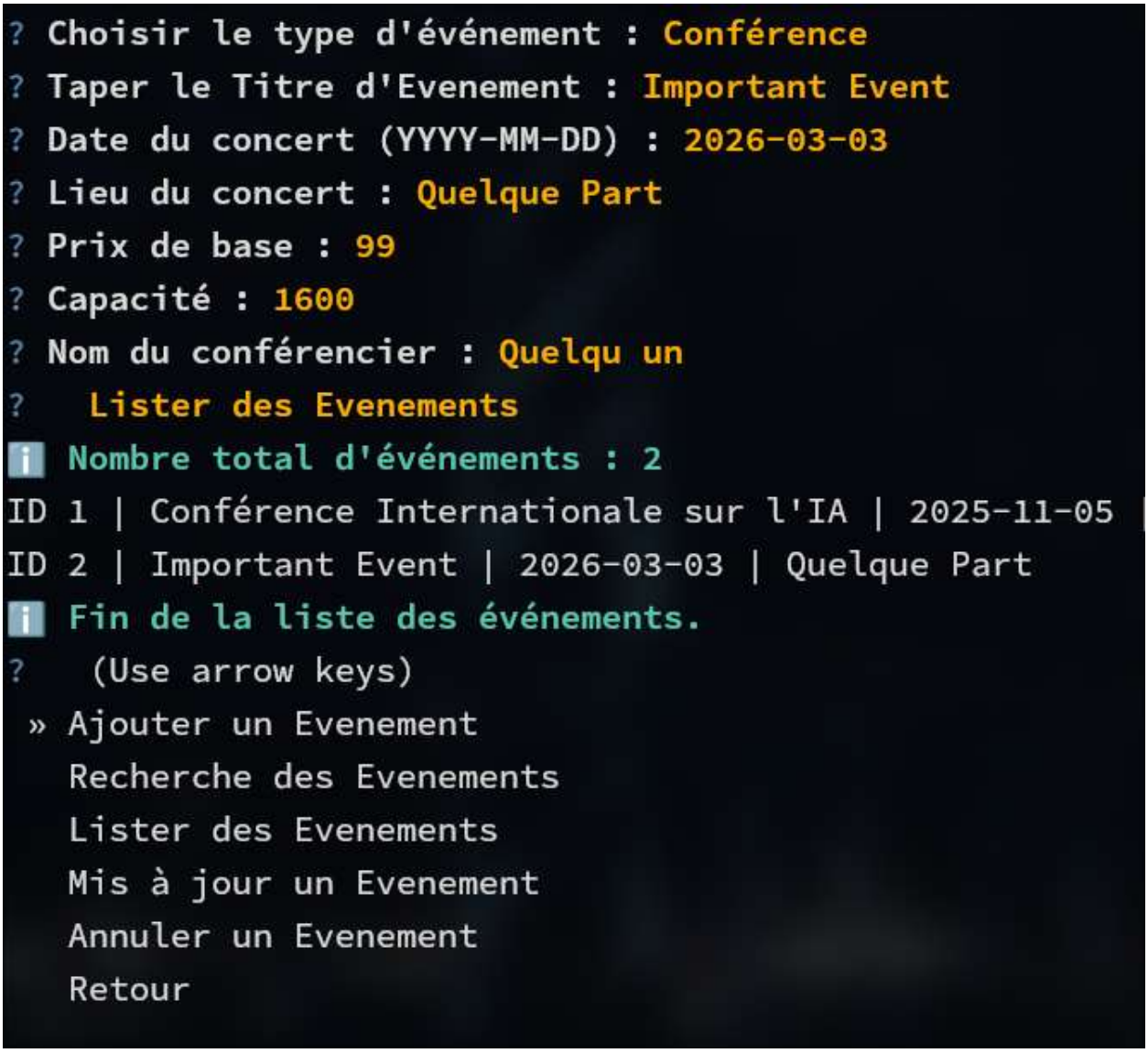
L'utilisateur exécute `python main.py`.



Le menu principal propose : `Gérer événements`, `Gérer acheteurs`, `Vendre billet`, `Rapports`, `Netoyage d'ecran`, `Quitter`.

Lors l'option de Gestion des événements:

Lister ou recherche des événements spécific ou bien l'ajout d'un nouvelle événement et aussi la gestion de leur donné ou bien l'annulé.



Lors l'option de Faire des ventes/achats:

Choix d'acheteur / Creation d'un nouvaux acheteur.

l'achat ou l'annulation ou bien la modfication des billets déja acheté.

l'orsque une nouvelle achat:

Sélection de l'événement .

Choix du type de billet (Standard / VIP).

Saisie de la quantité (validation : disponible >= quantité demandée).

Enregistrement de la vente dans `ventes.json` et mise à jour de . `places_vendues` dans les evenement sélectionné.

Lors l'option de nalyse et Rapports:

====== Analyse et Rapports ======

Revenu par type d'événement

	type	prix_total
0	Concert	850
1	Conference	900

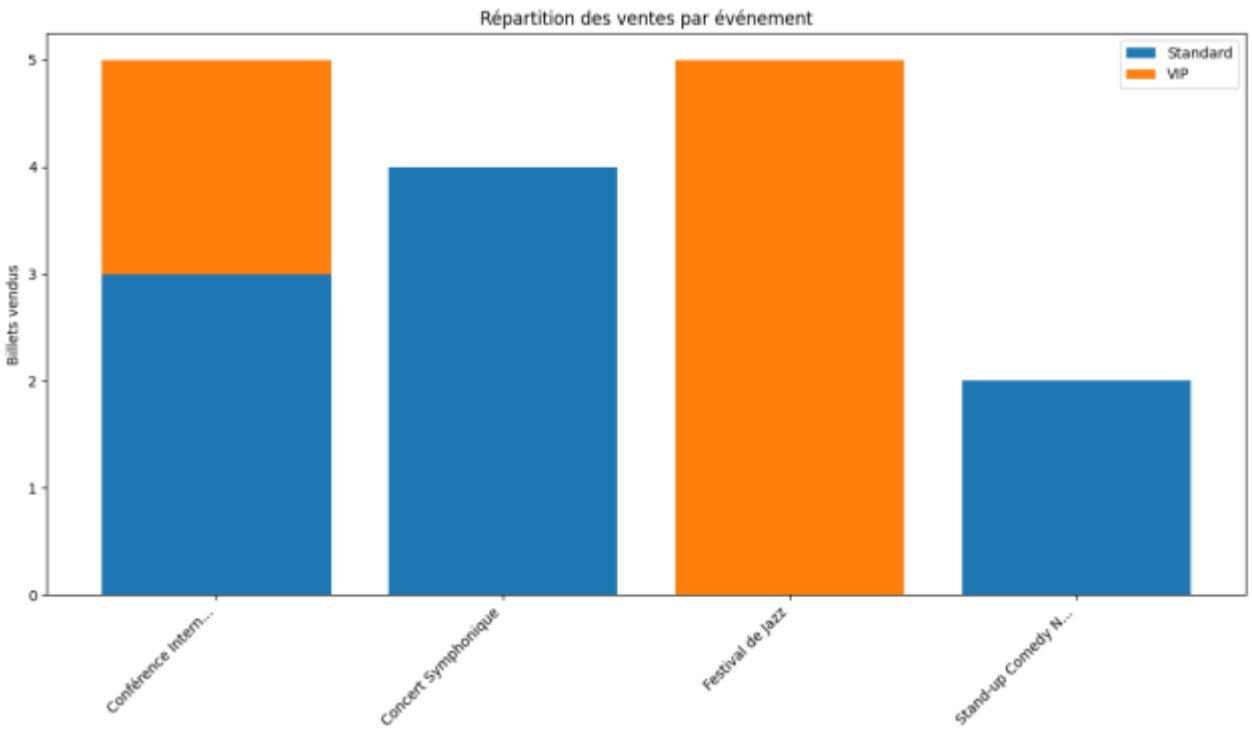
Statistiques globales

transactions	total_billets	moyenne_billets_par_vente	revenu_total
5	16	3.2	1750

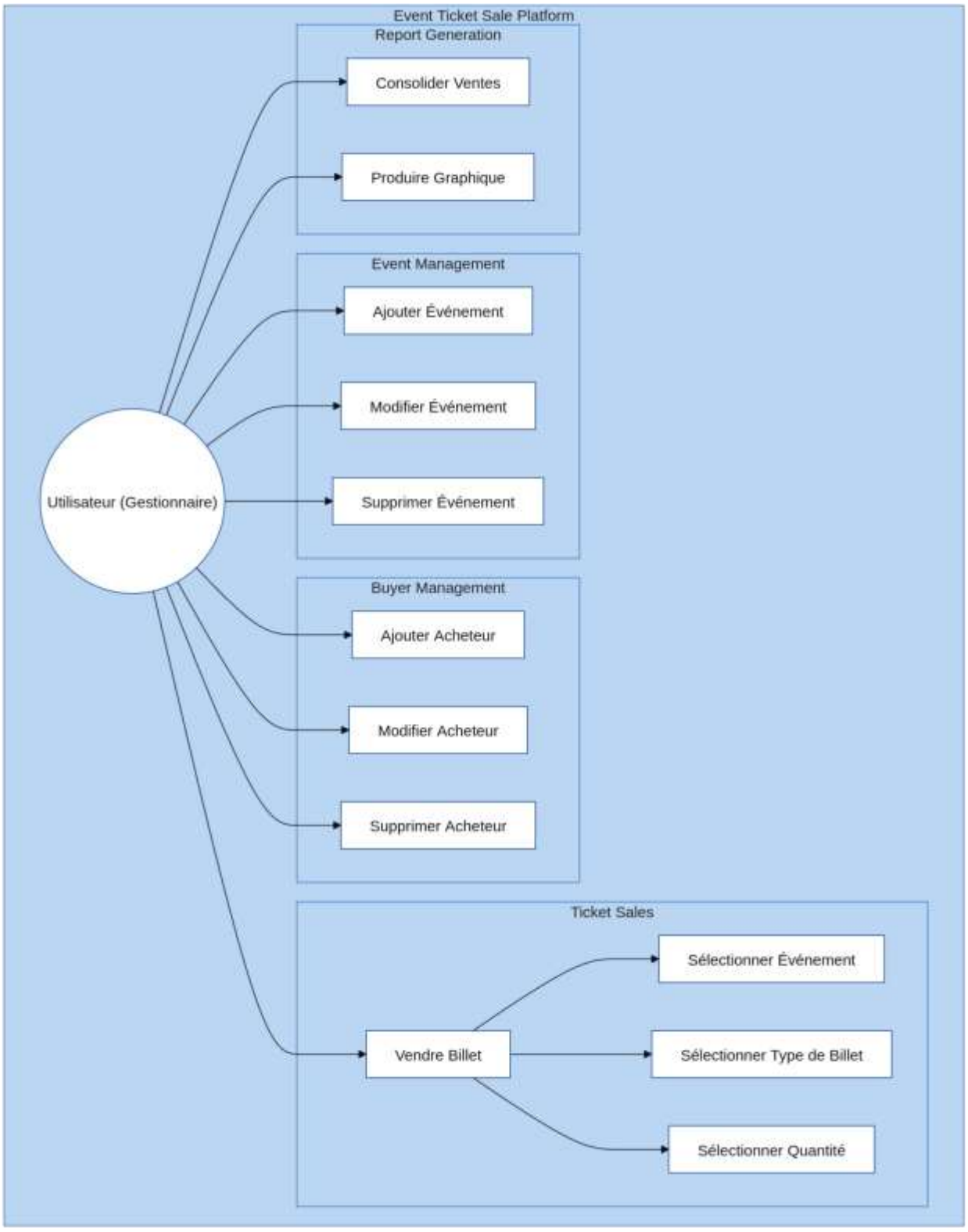
Répartition billets par événement

	id_evenement	titre	type	Standard	VIP
0	1	Conférence Internationale sur l'IA	Conference	3	2
1	2	Concert Symphonique	Concert	4	0
2	3	Festival de Jazz	Concert	0	5
3	4	Stand-up Comedy Night	Concert	2	0

• Graphique sauvegardé dans: storage/rapport_ventes.png



Tout ce que nous avons décrit précédemment peut être **simplifié et représenté visuellement** sous la forme du **diagramme de cas d'utilisation suivant:**


Event Ticket Sale Platform
Report Generation
Consolider Ventes
Produire Graphique
Event Management
Ajouter Événement
Modifier Événement
Supprimer Événement
Utilisateur (Gestionnaire)
Buyer Management
Ajouter Acheteur
Modifier Acheteur
Supprimer Acheteur
Ticket Sales
Sélectionner Événement
Vendre Billet
Sélectionner Type de Billet
Sélectionner Quantité

Conclusion

Cette Plateforme de **Vente de Billets d'Événements** est une solution pédagogique complète pour comprendre la conception d'un système de billetterie. Grâce à une architecture modulaire inspirée de Laravel et une interface CLI riche (questionary), l'application offre des fonctionnalités essentielles : gestion des événements, vente de billets, validation, stockage et génération de rapports analytiques. Les résultats montrent l'utilité des visualisations (répartition VIP/Standard) pour la prise de décision commerciale.