
# Stock Merch UGS

## Projet Master
## Livrable 2 - rapport d'avancement

*Thomas Siest*

M2 CYBER 2024-2025

# Sommaire

## Introduction

Dans le cadre de l'UE "Projet Master", je développe l'application Stock Merch UGS, destinée à faciliter la gestion de stocks de produits dérivés (*goodies*) pour mon association et pour les collectionneurs.

Ce projet suit une démarche agile, avec une approche mobile-first initiale sous forme de web app, évoluant ensuite vers une application mobile via Capacitor.

Ce rapport dresse l'état d'avancement actuel du projet, avec un focus particulier sur la partie Conception, finalisée comme exigé dans les modalités.

Vous allez remarquer tout au long de ce document, et même du projet, que le nom des termes techniques utilisés dans le code est en anglais ! Cela permet et permettra une meilleure compréhension de quiconque se joindra au projet par la suite. Ça permet l'internationalisation du projet.

## Avancement général

Les besoins ont été identifiés et discutés avec la partie prenante. Cette dernière n'a pas communiqué de refus. De plus, ils portent beaucoup d'enthousiasme dans le projet. Par conséquent, les objectifs ont été proposés et validés. Pour rappel, les voici :

- Faciliter la gestion des stocks de goodies lors des événements associatifs ;
- Permettre aux collectionneurs de gérer leur collection personnelle de manière simple et intuitive.

Une étude comparative menée montre des solutions déjà existantes. Par conséquent, notre projet se démarque par son modèle d'utilisation lié aux associations et aux contributeurs individuels.

La méthodologie de travail agile a été adoptée. Ainsi que le planning prévisionnel retranscrit dans l'outil de ticketing JIRA.

La conception de la solution (architecture, modèles de données, scénarios d'utilisation) est terminée. Par conséquent, la mise en place du squelette de l'application est également terminée.

Le projet étant disponible sur GitHub via ce lien : https://github.com/plugveg/stock-merch-ugs

On peut voir que j'ai mis en place énormément d'outils liés à la CI/CD (voir ce lien pour plus d'informations : https://github.com/plugveg/stock-merch-ugs/actions). Ces derniers me permettant de vérifier les failles dans le code, de revoir mon code, de mettre à jour les dépendances automatiquement.

# Conception de la solution (finalisée)

## Architecture technique

StockMerchUGS repose sur une architecture modulaire assurant la maintenabilité, la scalabilité et la portabilité. L'application est construite selon une approche mobile-first.

- ⇨ Frontend : ReactJS (mobile-first)
  - Ajout de Capacitor en phase finale du projet pour la création d'une application mobile (iOS/Android/PWA).
- ⇨ Backend : Convex (Backend serverless, temps réel).
- ⇨ Gestion de données : TanStack Query pour le fetching et la synchronisation.
- ⇨ Stockage média : Cloudinary ou Supabase pour les images.
- ⇨ Hébergement : Vercel.

## Fonctionnalités visées

**Pour les associations** :

- ➢ Gestion complète du stock (ajout, modification, suppression)
- ➢ Visualisation en temps réel des quantités
- ➢ Suivi des ventes et calcul des bénéfices
- ➢ Accès multi-utilisateur avec permissions

**Pour les collectionneurs individuels** :

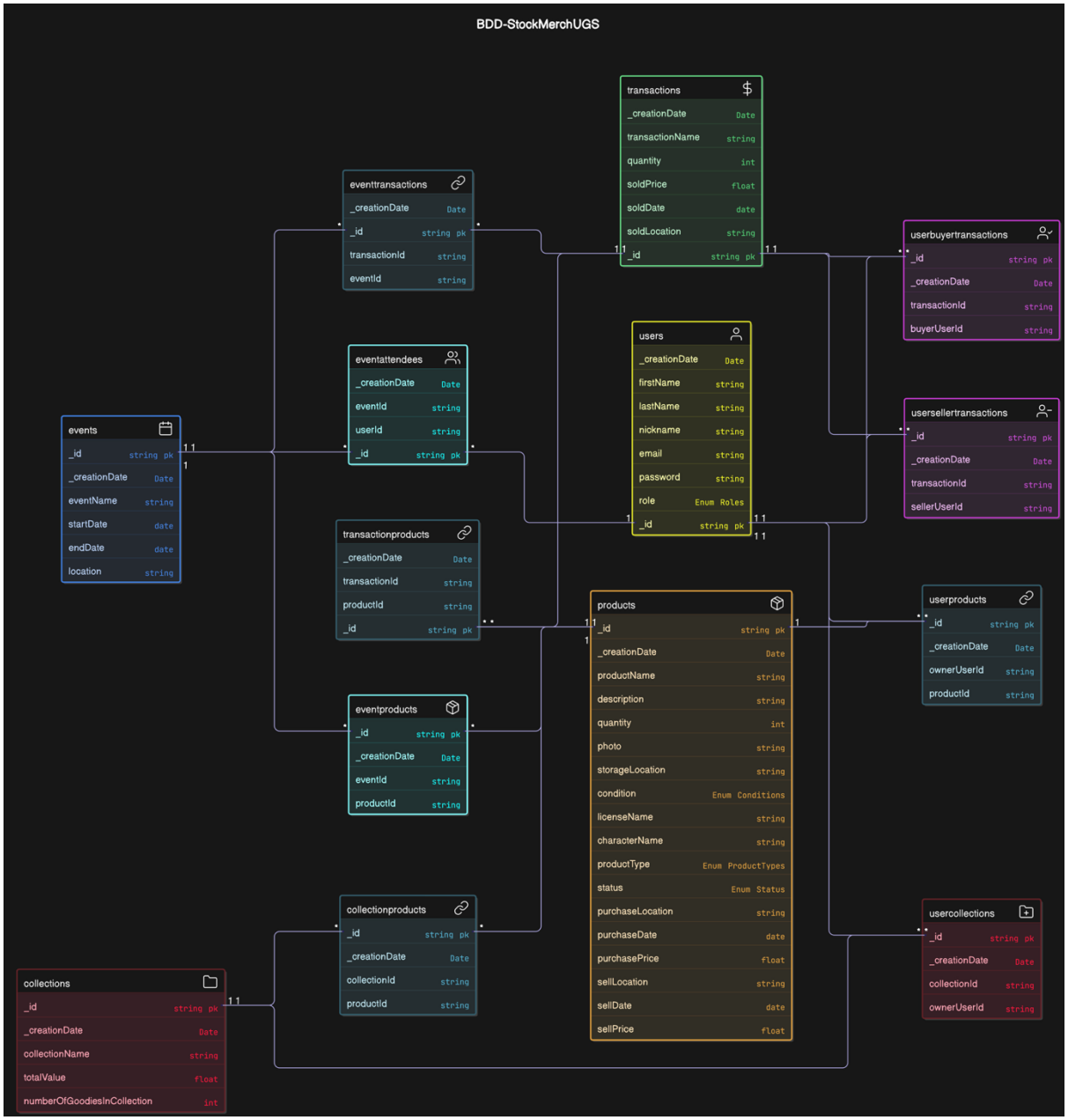
- ➢ Ajout d'articles personnels (nom, description, prix, photo, lieu de stockage)
- ➢ Statistiques personnalisées (valeur estimée, nombre d'objets)

**Fonctionnalités secondaires (dans une phase ultérieure)** :

- ➢ Historique des achats
- ➢ Comparaison de collections
- ➢ Mode enchère
- ➢ Export/partage de collection

## Modèles de données principaux

Ci-dessous, le schéma de la base de données qui est appliquée à l'application.




Schématisation de la base de données de StockMerchUGS

Si vous souhaitez plus de détail, ci-joint dans ce même rendu et en annexe de ce document, un fichier expliquant tous les choix fait pour les attributs, les cardinalités et les entités !

### Modèles d'utilisation

En convention, l'application servira de consultation pour voir les stocks disponibles sur le stand de l'association. Elle permettra également de mettre à jour rapidement le stock lors d'une vente. Les autres gestionnaires du stand pourront voir en tant direct la mise à jour du stock grâce à la technologie choisie Convex ! De plus, elle pourra également servir d'information pour les utilisateurs individuels. Ces derniers pourront voir en temps réels s'il y a encore des produits en vente qui les intéressent.

Dans le cadre d'un usage individuel, l'utilisateur pourra enregistrer tous ces produits personnels. Ainsi, il pourra créer des collections personnalisées avec un suivi des statistiques. Ces produits pourront également être proposé à la vente lors d'un événement, s'il l'a choisi.

## Méthodologie et organisation

Pour rappel, j'utilise une approche de travail de manière Agile. Pour m'aider dans cette approche, j'utilise un outil de ticketing se nommant JIRA. Ce dernier me permet de gérer, via des priorités, mes tâches à réaliser.

Pour la partie code, j'utilise GitHub pour stocker mon code source. Ainsi, il faut faire extrêmement attention sur la sécurité. C'est pour cela que j'ai mis en place une CI/CD en béton. Cette dernière vérifie toutes les versions, toutes les parties du code qui pourraient potentiellement être une faille ! À chaque nouveau changement dans le code source, il y a un outil de versioning qui permet de release un nouveau package. Ce package est utilisable directement !

Enfin, à chaque changement, l'outil se nommant Vercel me permet de release automatiquement en production le site internet. Vous pouvez aller le voir via cette URL : https://stock-merch-ugs.vercel.app/ (N'oubliez pas que le développement est encore en cours)

## Conclusion

Le projet StockMerchUGS n'avance pas assez rapidement. Cela est dû à la longue phase de conception et des choix des technologies. Ainsi que la mise en place des différents outils et surtout de la CI/CD !

La phase de Conception étant maintenant finalisée, le projet et la réalisation technique peut se lancer sereinement. Sans oublier, le maintien du respect de la méthodologie agile qui permet l'ajustement du développement en fonction des besoins et feedbacks des parties prenantes.

Annexe


# Stock Merch UGS

## Informations sur la BDD

*Thomas Siest*

## Sommaire

## À savoir

- Vous allez remarquer tout au long de ce document, et même du projet, que le nom des attributs, le nom des entités et autres sont en anglais ! Cela permet et permettra une meilleure compréhension de quiconque se joindra au projet par la suite. Ça permet l'internationalisation du projet.
- "Primary Key" ou **PK** veut dire cela est l'identifiant unique de chaque donnée dans une entité. Cela nous permettra de les sélectionner individuellement par la suite.
- "Foreign Key" ou **FK** veut dire **clé étrangère**. Je ferai référence à ces trois termes pour permettre l'établissement de cardinalités entre les entités, où une clé étrangère dans une entité fait référence à la clé primaire d'une autre entité, assurant ainsi l'intégrité référentielle et permettant de lier des données liées entre elles de manière cohérente.
- J'ai omis les attributs "_creationTime" & "_id" car ils sont créés automatiquement lors de la création d'une entrée. Ce "_id" est la primary key (PK).
- Dans mon schéma, j'utiliserai des Enums pour certains attributs pour permettre de sélectionner une option précise. Les Enums permettront par la suite de trier une donnée réaliste et non fausse ou créée l'utilisateur.
- Pour ce qui est des cardinalités entre les entités, on pourra avoir
    - 0..1 => une entité de A peut être reliée à aucune ou à une seule entité de B
    - 1..1 => une entité de A peut être reliée à une seule entité de B
    - 0..* =>une entité de A peut être reliée à aucune ou à plusieurs entités de B
    - 1..* => une entité de A peut être reliée à une ou plusieurs entités de B
- Une transaction sera possible même si l'acheteur ou le vendeur n'est pas connu dans la base de données. Il suffira de laisser l'option correspondante vide. Ou bien de demander à la personne concernée de se créer un compte.

## Enums

- Roles :
    - Administrator (Administrateur) : Ayant accéder à toutes les possibilités.
    - Board of directors (Conseil d'administration) : Personnes appartenant au conseil de l'association (ayant accès à la plupart des fonctionnalités) (à confirmer chaque année après l'assemblée générale extraordinaire).
    - Active members (Membres actifs) : Personnes ayant payés leurs adhésions pour avoir les avantages d'un "membre premium".
    - Founding members (Membres fondateurs) : Personnes ayant fondés l'association.
    - Member representative (Représentant des membres) : Personne (une seul par mois) participant aux réunions et aux décisions prises par le conseil. Il est le porte-parole de tous les membres.
    - Member (Membres) : Personne n'ayant pas payés l'adhésion à l'association mais participe à la vie de l'association.
    - Unknown user (Utilisateurs inconnus) : Personne non-connu de l'association ou personne temporaire
- Conditions :
    - **New** (Neuf) : Le produit est neuf, jamais utilisé ou ouvert.
    - **Used** (Occasion) : Le produit a été utilisé, mais reste en bon état.
    - **Damaged** (Abîmé) : Le produit présente des signes de dommage, mais reste fonctionnel.
    - **Refurbished** (Reconditionné) : Le produit a été remis à neuf ou réparé pour le remettre en vente.
    - **Mint** (Comme neuf) : Le produit est dans un état presque neuf, sans signes d'usure notables.
    - **Vintage** (Vintage) : Le produit est ancien, mais reste dans un bon état.
    - **Limited Edition** (Édition limitée) : Produit qui a été édité en quantité limitée, mais pas nécessairement neuf.
    - **Damaged Box** (Boîte abîmée) : Le produit est en bon état, mais l'emballage est endommagé.
- Status :
    - **In Stock** (En stock) : Le produit est actuellement dans l'inventaire et disponible.
    - **Sold** (Vendu) : Le produit a été vendu.
    - **Reserved** (Réservé) : Le produit est réservé par un client, mais pas encore payé.

- **Out of Stock** (Rupture de stock) : Le produit est actuellement épuisé et ne peut pas être vendu.
- **On Sale** (En vente) : Le produit est activement proposé à la vente (par exemple, pendant un événement).
- **In Collection** (Dans la collection) : Le produit appartient à un utilisateur et est enregistré dans sa collection personnelle.
- **Archived** (Archivé) : Le produit n'est plus actif ou disponible à la vente (par exemple, une ancienne édition qui n'est plus en stock).
- **Pre-Order** (Précommande) : Le produit est disponible en précommande, mais pas encore en stock ou livré.
- **In Auction** (En enchères) : Le produit est actuellement en vente aux enchères.

- ProductTypes :
  - **Prepainted** (Peint à l'avance) : Figurines ou objets qui sont déjà peints et prêts à l'exposition.
  - **Action/Dolls** (Figurines/Mannequins) : Figurines d'action ou poupées basées sur des personnages.
  - **Trading** (Cartes à échanger) : Cartes ou objets à collectionner et à échanger.
  - **Garage Kits** (Kits de garage) : Modèles en kit, généralement à assembler et peindre soi-même.
  - **Model Kits** (Maquettes) : Kits à assembler pour créer des modèles réduits (véhicules, bâtiments, etc.).
  - **Accessories** (Accessoires) : Objets accessoires liés à une licence ou un personnage (sacs, bijoux, porte-clés, etc.).
  - **Plushes** (Peluches) : Peluches représentant des personnages ou des animaux.
  - **Linens** (Linge) : Textiles tels que des draps, des serviettes, des couvertures, etc.
  - **Dishes** (Vaisselle) : Articles de vaisselle comme des tasses, assiettes, bols, etc.
  - **Hanged up** (Accroché) : Objets destinés à être accrochés comme des posters, des bannières, des figurines suspendues, etc.
  - **Apparel** (Vêtements) : Vêtements comme des t-shirts, des pulls, des chaussettes, des casquettes, etc.
  - **On Walls** (Sur les murs) : Décorations murales comme des posters, des panneaux, des tapisseries, etc.
  - **Stationeries** (Papeterie) : Objets de papeterie tels que des stylos, des carnets, des agendas, etc.

- **Misc** (Divers) : Autres articles qui ne correspondent pas aux catégories précédentes.
- **Books** (Livres) : Livres et mangas, en lien avec des licences spécifiques.
- **Music** (Musique) : CDs, vinyles, bandes sonores liées à des licences ou des personnages.
- **Video** (Vidéo) : DVDs, Blu-rays ou tout autre format vidéo en lien avec des franchises.
- **Games** (Jeux) : Jeux vidéo ou de société en lien avec des personnages ou des licences.
- **Software** (Logiciels) : Logiciels, applications ou jeux numériques à télécharger.

## Entités principales

1. ***Users*** : Représente les utilisateurs de l'application (membres de l'association ou collectionneur individuel). Chaque utilisateur peut avoir un ou plusieurs rôles (par exemple, administrateur, collaborateur). Ces Rôles seront régis par l'Enum ci-dessous.

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| firstName | String | Le prénom de l'utilisateur |
| lastName | String | Le nom de l'utilisateur |
| nickname | String | Le pseudo de l'utilisateur |
| email | String | Le mail de l'utilisateur |
| password | Passwd | Le mot de passe de l'utilisateur |
| role | Roles (*Enum*) | Le/les role(s) de l'utilisateur |

2. ***Products*** : Représente les produits ou goodies dans le stock.

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| productName | String | Nom du produit |
| description | LongString | Description libre du produit |
| quantity | Int | Quantité possédée |
| photo | String (*chemin vers le fichier*) | Une photo pour montrer le produit[1] |
| storageLocation | String | Lieu de stockage |
| condition | Conditions (*Enum*) | L'état du produit |
| licenseName | String | License auquel est reliée le produit[2] |
| characterName | String/List | Si le produit représente un ou plusieurs personnages ou un groupe[3] |
| productType | ProductTypes (*Enum*) | Le type du produit (une proposition à l'utilisateur sera faite) |
| status | Status (*Enum*) | Le statut du produit (acquis, en vente, en enchère…) |
| purchaseLocation | String | L'endroit, le site où le produit a été acheté |
| purchaseDate | Date | La date de l'achat |

| | | |
|---|---|---|
| purchasePrice | Float | Le prix d'achat du produit |
| sellLocation | String | L'endroit, le site où a été vendu le produit ou la clé étrangère lié à la transaction[5] |
| sellDate | Date | La date de la vente |
| sellPrice | Float | Le prix de vente du produit |

[1] : Pour le moment, le choix est de stocker au maximum une seule photo pour chaque produit. Par la suite, avec une meilleure base de données, nous pourrons stocker une multitude de photos pour chaque produit.

[2] : Pour le moment le champ sera une chaine de caractère libre. Par la suite, en ayant plus de données, nous pourrons créer une Enum à partir des données déjà récoltées.

[3] : De même ici, pour le moment le champ sera une chaine de caractère libre. Par la suite, en ayant plus de données, nous pourrons créer une Enum à partir des données déjà récoltées.

[4] : Cette clé étrangère ne sera pas "flexible". Ici, flexible veut dire que même si le produit est vendu à un autre utilisateur, la valeur de l'attribut *ownerUserId* ne changera pas pour le nouvel id de l'acheteur. Cela permettra d'avoir, dans la base de données de l'utilisateur, une trace de la vente. L'utilisateur saura qu'il a vendu son ancien produit. Dans ce cas, la valeur de l'attribut *status* sera considéré comme "vendu" ou autre selon le cas précisé.

[5] : Nous proposerons les deux types pour ces attributs car il est possible que l'utilisateur ne puisse pas utiliser l'application/le site web au moment de la transaction. Dans ce cas, la transaction ne sera pas possible. On laissera donc le renseignement des informations libre.

3. ***UserProducts*** : Représente l'entité qui regroupe l'utilisateur avec ses produits

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| ownerUserId | FK vers *users._id* | La clé étrangère pour relier l'utilisateur et son produit |
| productId | FK vers *products._id* | La clé étrangère qui relie l'utilisateur à son produit |

4. ***TransactionProducts*** : Représente l'entité qui regroupe les transactions avec leurs produits

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| transactionId | FK vers *transactions._id* | La clé étrangère pour relier la transaction à son produit vendu |
| productId | FK vers *products._id* | La clé étrangère qui relie la transaction à son produit |

5. ***Transactions*** : Enregistre une vente ou un mouvement de produit dans l'inventaire.

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| transactionName | String | Le nom de la transaction[1] |
| quantity | Int | La quantité du même produit vendu |
| soldPrice | Float | Le prix de vente individuel du produit |
| soldDate | Date | La date de vente |
| soldLocation | String | L'endroit, le site sur lequel a été vendu le produit |

[1] : On pourra, par la suite, retrouver les transactions par leurs noms.

[2] : On pourra via cette clé étrangère retrouver tous les détails du produit.

6. ***UserBuyerTransactions*** : Représente l'entité qui relie les acheteurs à leurs transactions

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| transactionId | FK vers *transactions._id* | La clé étrangère pour relier la transaction à son acheteur |
| buyerUserId | FK vers *users._id* | La clé étrangère qui relie l'acheteur à ses transactions |

7. ***UserSellerTransactions*** : Représente l'entité qui relie les vendeurs à leurs transactions

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| transactionId | FK vers *transactions._id* | La clé étrangère pour relier la transaction à son vendeur |

| | | |
|---|---|---|
| sellerUserId | FK vers *users._id* | La clé étrangère qui relie le vendeur à ses transactions |

8. ***EventTransactions*** : Représente l'entité qui relie les évènements avec les transactions qui ont eu lieu lors de ce dernier

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| transactionId | FK vers *transactions._id* | La clé étrangère pour relier la transaction à son évènement |
| eventId | FK vers *events._id* | La clé étrangère qui relie l'événement à ses transactions |

9. ***Collections*** : Représente la collection d'un utilisateur. Chaque utilisateur peut avoir plusieurs produits dans sa collection.

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| collectionName | String | Le nom de la collection |
| totalValue | Float | Calcul de la valeur totale selon les prix renseignés |
| numberOfGoodiesInCollection | Int | Nombre total de produits dans la collection |

10. ***CollectionProducts*** : Représente l'entité qui regroupe tous les produits appartenant à une collection

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| collectionId | FK vers *collections._id* | La clé étrangère pour relier la collection à ses produits |
| productId | FK vers *products._id* | La clé étrangère qui relie les produits à leurs collections |

11. ***UserCollections*** : Représente l'entité qui regroupe chaque utilisateur avec ses collections

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| collectionId | FK vers *collections._id* | La clé étrangère pour relier la collection à son utilisateur |
| ownerUserId | FK vers *users._id* | La clé étrangère qui relie l'utilisateur à ses collections |

12. ***Events*** : Représente un événement associatif où les produits sont vendus (exemple une convention).

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| eventName | String | Nom de l'événement |
| startDate | Date | Date de début de l'événement |
| endDate | Date | Date de la fin de l'événement |
| location | String | Lieu ou se déroule l'événement (peut-être en ligne) |

13. ***EventAttendees*** : Représente l'entité qui regroupe tous les utilisateurs allant à l'événement.

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| eventId | FK vers *events._id* | La clé étrangère qui représente l'événement |
| userId | FK vers *users._id* | La clé étrangère qui donne les informations des utilisateurs présents |

14. ***EventProducts*** : représente l'entité qui regroupe tous les produits disponibles lors d'un événement.

| Nom de l'attribut | Type de l'attribut | Description de l'attribut |
|---|---|---|
| eventId | FK vers *events._id* | La clé étrangère qui représente l'événement |
| productId | FK vers *products._id* | La clé étrangère qui donne des produits disponibles durant l'événement |

## Cardinalités

- Un **Utilisateur** peut effectuer plusieurs **Transactions**.
- Un **Utilisateur** peut avoir plusieurs **Collections**.
- Un **Utilisateur** peut avoir plusieurs **Produits**.
- Un **Utilisateur** peut participer à plusieurs **Événements**.
- Un **Produit** peut être associé à plusieurs **Transactions**.
- Un **Produit** peut appartenir à plusieurs **Collections** (pour la gestion des objets personnels des utilisateurs).
- Un **Produit** peut appartenir à plusieurs **Utilisateurs**.
- Un **Produit** peut appartenir à plusieurs **Événements**.
    - Attention, s'il n'y a qu'un produit de ce type, il ne pourra pas appartenir à plusieurs conventions qui se déroulent sur une même plage de date ou avec des dates qui se superposent.
- Une **Transaction** appartient à deux **Utilisateurs** (buyerUserId & sellerUserId).
- Une **Transaction** doit obligatoirement contenir un **Produit**.
- Une **Transaction** se rapporte à un **Évènement** (lorsqu'un produit est vendu lors d'un événement ou non lors d'une transaction personnelle (entre deux utilisateurs)).
- Une **Collection** est liée à un **Utilisateur** (chaque collection appartient à un utilisateur spécifique).
- Une **Collection** contient plusieurs **Produits**.
- Un **Événement** regroupe plusieurs **Utilisateurs**.
- Un **Événement** propose ou non des **Produits**.
- Un **Événement** peut avoir des **Transactions** liées.