Livrable 1 - Gestion d’un système de dépôt de vente

ROUSSEL Alexandre - SAUVAGE Leo

Sommaire

Sommaire **2**
Introduction **3**
Brève présentation du projet 3
Objectifs du livrable 3
Résumé des choix techniques et de conception 3
Plan d'Architecture Initial **4**
Diagramme d'Architecture Globale 4
Schéma de Base de Données Préliminaire 6
Explication des principaux choix de modélisation 8
Diagramme Entité-Relation 9
Clés primaires et étrangères 9
Justification des Choix Technologiques 11
Wireframes et Maquettes des Interfaces Utilisateur **12**
Définition Préliminaire des Endpoints API **26**
Sécurité et Authentification 29
Sécurisation des Endpoints 29
Authentification 29
Autorisations 30
Endpoints protégés 30
Conclusion et Prochaines Étapes **31**

Introduction

Brève présentation du projet

Le projet consiste à développer un système de gestion de dépôt-vente pour des jeux de société, utilisé lors d'événements comme des festivals de jeux. Ce système doit permettre aux gestionnaires de déposer les jeux des vendeurs, d'enregistrer les achats et de gérer efficacement les transactions, les stocks, ainsi que les bilans financiers.

Objectifs du livrable

Ce premier livrable vise à valider l'architecture technique et les choix de conception avant de commencer le développement de l'application. Ce document inclut un diagramme d'architecture, un schéma de base de données, des wireframes, ainsi qu'une définition des endpoints API.

Résumé des choix techniques et de conception

- **Architecture :**
  - Séparation front-end et back-end (gestion indépendante des interfaces utilisateur et des logiques métier).
  - front-end : AngularJS
  - back-end ExpressJS

- **Base de données :**
  - La base de données sera relationnelle.
  - PostgreSQL

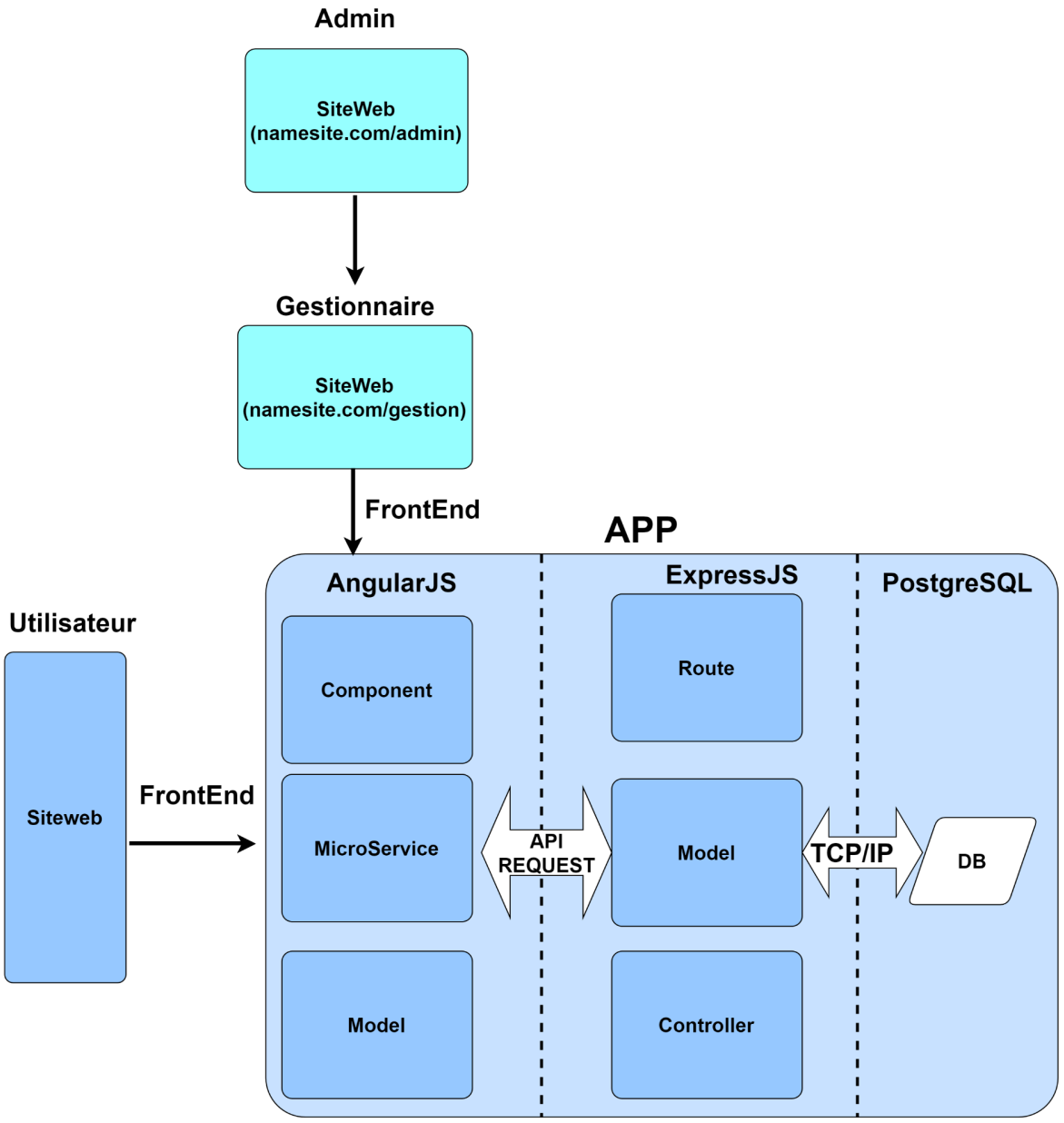
- **Endpoints API :**
  - Les endpoints RESTful seront définis pour permettre la gestion des dépôts, des ventes, et des stocks, ainsi que pour fournir des bilans financiers.

- **Front-end :**
  - L'interface utilisateur sera ergonomique et facile à utiliser, avec des wireframes conçus pour minimiser les opérations des gestionnaires afin de supporter le grand flux de clients.

- **Sécurité :**
  - Le système sera sécurisé pour empêcher l'utilisation de fonctionnalités critiques par d'autres utilisateurs que les gestionnaires et administrateurs. Des tests unitaires et d'intégration seront implémentés pour valider la robustesse du système.

Plan d'Architecture Initial

Diagramme d'Architecture Globale



Les utilisateurs pourront utiliser l'application avec le site web. Ils verront directement le front-end.

Un Admin possède les droits d'un gestionnaire. Ces 2 rôles accèderont à leurs fonctionnalités via deux routes distinctes et inaccessibles pour un utilisateur lambda.
Dans le front-end:

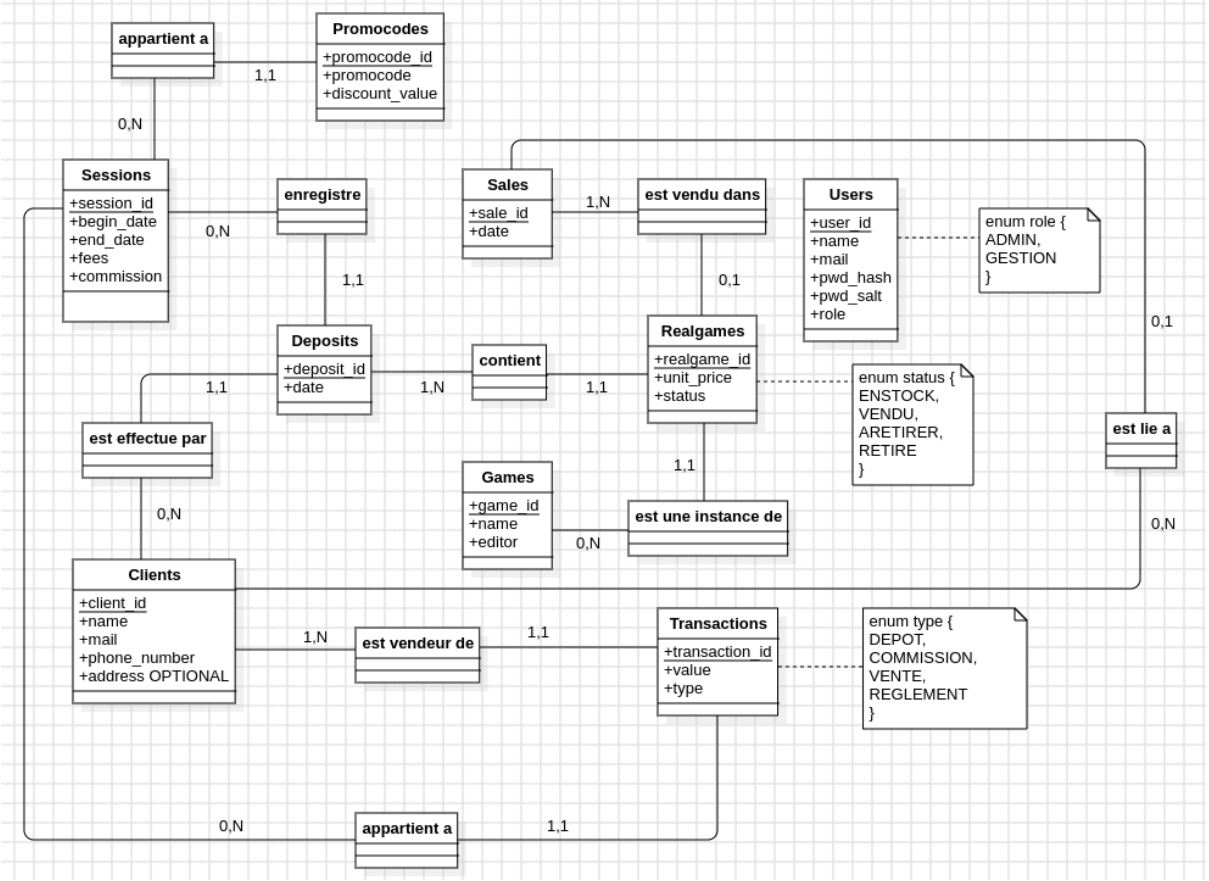
- Le component permet de gérer l'interaction avec les utilisateurs.
- Les microservices permettront de communiquer avec l'API et le back (on le décompose en éléments simples et indépendants).
- Le model permet de représenter la structure de données et la logique métier.
-

La relation entre le front-end et le back-end se fera via des requêtes API.

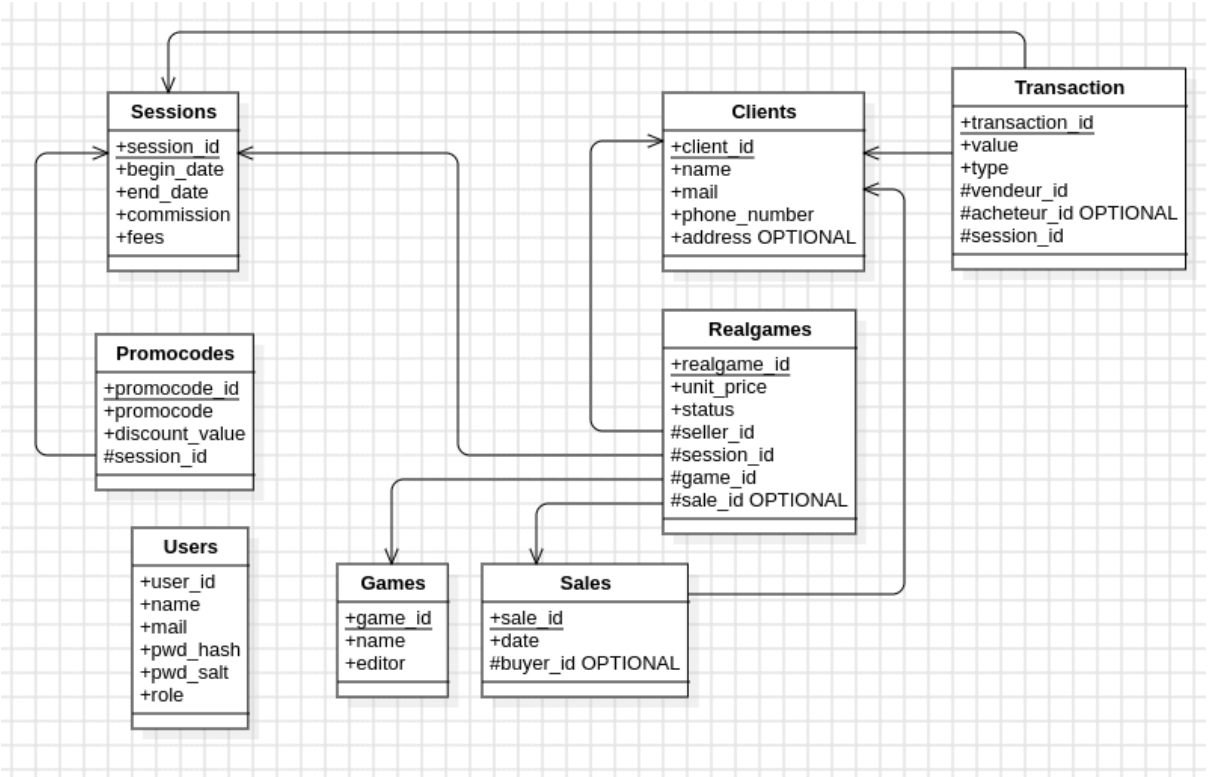
Dans le back-end :

- Les routes vont nous permettre d'accéder à une adresse API spécifique associé à une fonction que l'on retrouvera dans les controllers.
- Les models vont permettre d'intéragir avec la base de données.
- Les controllers vont traiter les demandes des clients et renvoyer une réponse.

Schéma de Base de Données Préliminaire



Modèle conceptuel de donnée



Modèle logique de données

La table *Deposits* a disparu car aucune fonctionnalité du projet (wireframe et endpoints) ne nécessite de connaître la date et le dépôt spécifique d'une instance réelle d'un jeu (Realgames). La table Deposits est laissée dans le MCD pour le contexte.

Explication des principaux choix de modélisation

- **Normalisation :**
    - Le modèle est normalisé afin de réduire la redondance des données et d'assurer la cohérence. Chaque table est liée par des clés étrangères, garantissant qu'il n'y a pas de duplication inutile des informations.

- **Performances :**
    - Les relations sont conçues pour être directes et efficaces. L'usage de clés étrangères facilite les jointures rapides. Le modèle prend également en compte la gestion des statuts des jeux via l'entité Realgames, qui permet un suivi en temps réel du stock.

- **Évolutivité :**
    - L'utilisation d'énumérations pour les rôles des utilisateurs (ADMIN, GESTION) et les statuts des jeux (ENSTOCK, VENDU, etc.) permet de facilement ajouter de nouvelles catégories. De plus, la gestion des transactions financières est flexible grâce à la table Transactions, permettant d'ajouter de nouveaux types de transactions si nécessaire.

Diagramme Entité-Relation

- **Sessions :**
  - Représente une période d'enregistrement de ventes avec une date de début et de fin, des frais et une commission.
- **Deposits :**
  - Décrit les dépôts réalisés par les clients.
- **Clients :**
  - Représente les utilisateurs (vendeurs ou acheteurs) du système.
- **Games :**
  - Les jeux disponibles dans le dépôt-vente.
- **Realgames :**
  - Correspond à une instance spécifique d'un jeu, avec un prix unitaire et un statut.
- **Transactions :**
  - Représente les différentes transactions financières (dépôts, ventes, commissions).
- **Users :**
  - Les administrateurs et gestionnaires de la plateforme.

Clés primaires et étrangères

- **Clés primaires :**
  - session_id pour Sessions.
  - client_id pour Clients.
  - game_id pour Games.
  - realgame_id pour Realgames.
  - sale_id pour Sales.
  - transaction_id pour Transactions.
  - user_id pour Users.
  - promocode_id pour Promocodes.

- **Clés étrangères :**
  - session_id dans Realgames (lien avec Sessions).
  - game_id dans Realgames (lien avec Games).
  - seller_id dans Realgames (lien avec Clients).
  - sale_id dans Realgames (lien avec Sales).
  - seller_id dans Transactions (lien avec Clients).
  - session_id dans Transactions (lien avec Sessions).
  - buyer_id dans Sales (lien avec Clients).
  - session_id dans Promocodes (lien avec Sessions).

Relations

- **Sessions et Deposits** : Une session peut enregistrer plusieurs dépôts.
- **Sessions et Promocodes** : Une session peut enregistrer plusieurs codes-promo.
- **Sessions et Transactions** : Une session peut enregistrer plusieurs transactions.
- **Clients et Transactions** : Un client peut être lié à plusieurs transactions.
- **Clients et Deposits** : Un client peut effectuer plusieurs dépôts.
- **Games et Realgames** : Un jeu peut avoir plusieurs instances réelles (Realgames).
- **Sales et Realgames** : Une vente peut contenir plusieurs instances de jeux vendus.

Justification des Choix Technologiques

- **Langages de programmation :**
    - JavaScript : Utilisé pour le développement du front-end et du back-end. Utilisation du même langage pour les deux parties de l'application, simplifiant ainsi l'intégration et la communication entre les différentes couches.

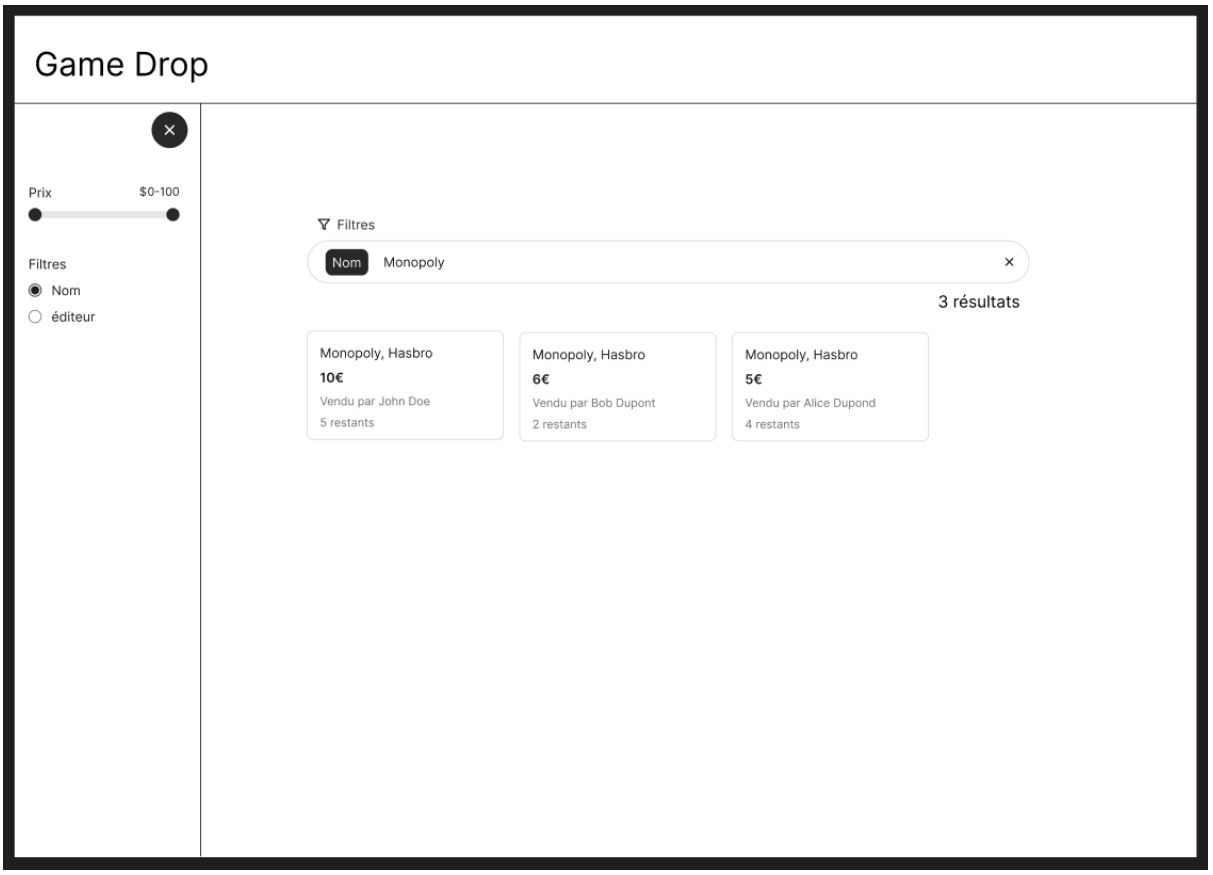
- **Frameworks :**
    - AngularJS (Front-end) : AngularJS est choisi pour ses fonctionnalités robustes et sa modularité.
    - ExpressJS (Back-end) : framework léger pour Node.js, choisi pour son efficacité à gérer des requêtes HTTP et à servir des API RESTful. Il est facile à configurer et offre une bonne performance pour un système de gestion de ventes en temps réel.

- **Base de données :**
    - PostgreSQL : base de données relationnelle permettant une gestion fiable et structurée des données. Idéale pour ce projet, qui nécessite une gestion rigoureuse des stocks et des transactions.

- **Hébergement :**
    - Dokku : plateforme basée sur Docker, idéale pour déployer des applications web et qui facilite le déploiement de conteneurs Docker.

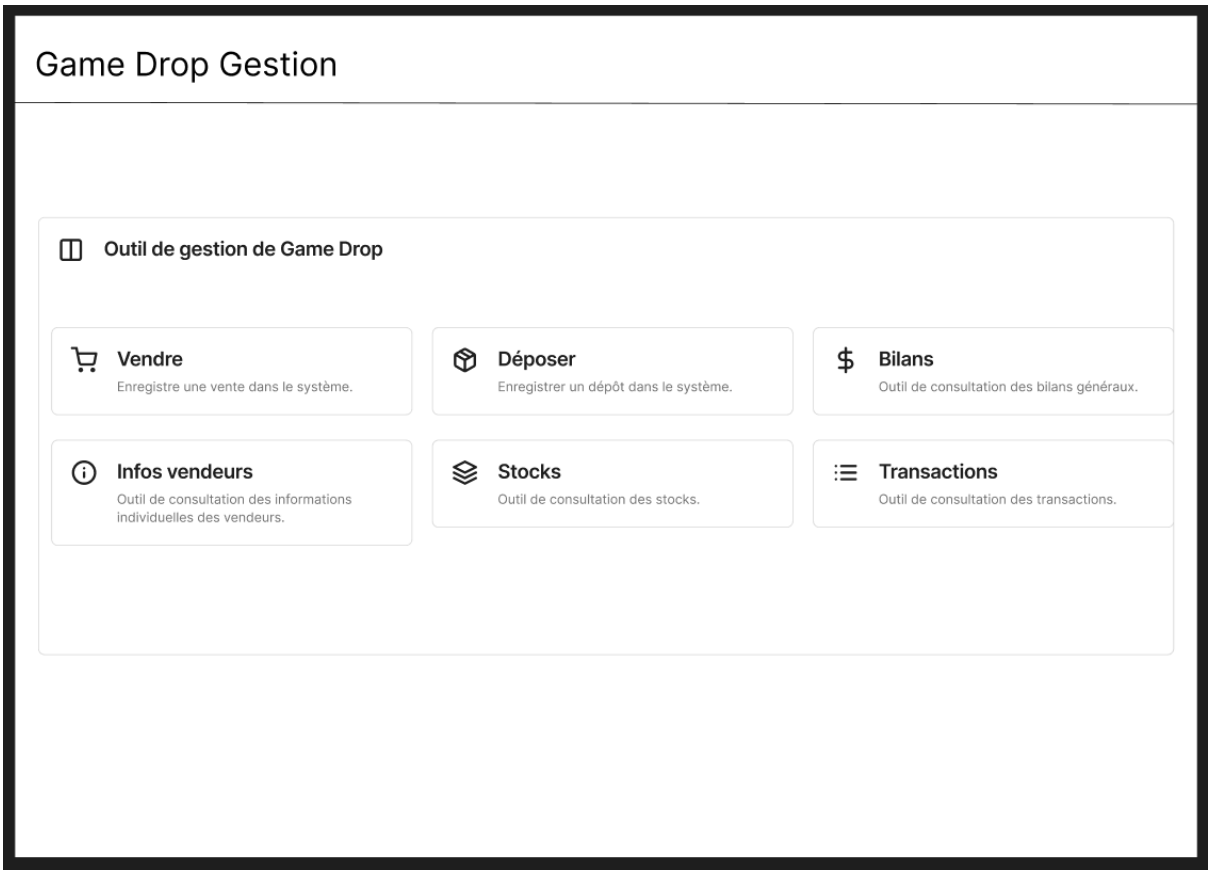
Wireframes et Maquettes des Interfaces Utilisateur

Lorsque un client se connecte à la racine de *Game Drop*, il est accueilli par le statut actuel des sessions, si une est en cours, l'interface le fait savoir à l'utilisateur et lui propose de consulter le catalogue. Si aucune session n'est en cours, l'utilisateur est notifié qu'il peut revenir plus tard pour consulter le catalogue.



Quand l’utilisateur clique sur le bouton “Consulter le catalogue”, il est alors redirigé sur la page ci-dessus. Cette page affiche l'entièreté du catalogue et donne la possibilité d’effectuer une recherche au moyen de filtres pour le nom, l'éditeur et le prix. Sa recherche affiche alors le nombre de résultats et montre les différentes offres en fonction du vendeur.

Ici, nous pouvons retrouver l'interface de gestion de *Game Drop*, seuls les gestionnaires et administrateurs y ont accès (théoriquement grâce à un pare-feu). Il s'agit ici d'une interface de connexion standard.



Une fois connectés, les gestionnaires ont alors rapidement accès aux différentes actions qu'ils doivent réaliser pour traiter un grand nombre de clients rapidement. Ils peuvent donc à partir de cette interface, enregistrer une vente, un dépôt, consulter les bilans, infos vendeurs, stocks et les transactions de la session.

Ici, nous pouvons retrouver l’interface de vente, celle-ci est simple et intuitive à utiliser, il suffit de rechercher l'étiquette unique du jeu que l’on désire enregistrer comme vendu. Celui-ci est suggéré pour vérification des informations. Lorsqu'on le sélectionne, celui-ci s'ajoute au panier. Si le client le désire, le gestionnaire peut cocher la case “imprimer une facture”, il sera ensuite possible d’entrer les informations du client afin d’imprimer la facture de son achat. Les acheteurs peuvent à tout moment venir demander leur facture s' ils ont fourni leurs informations à l’achat.

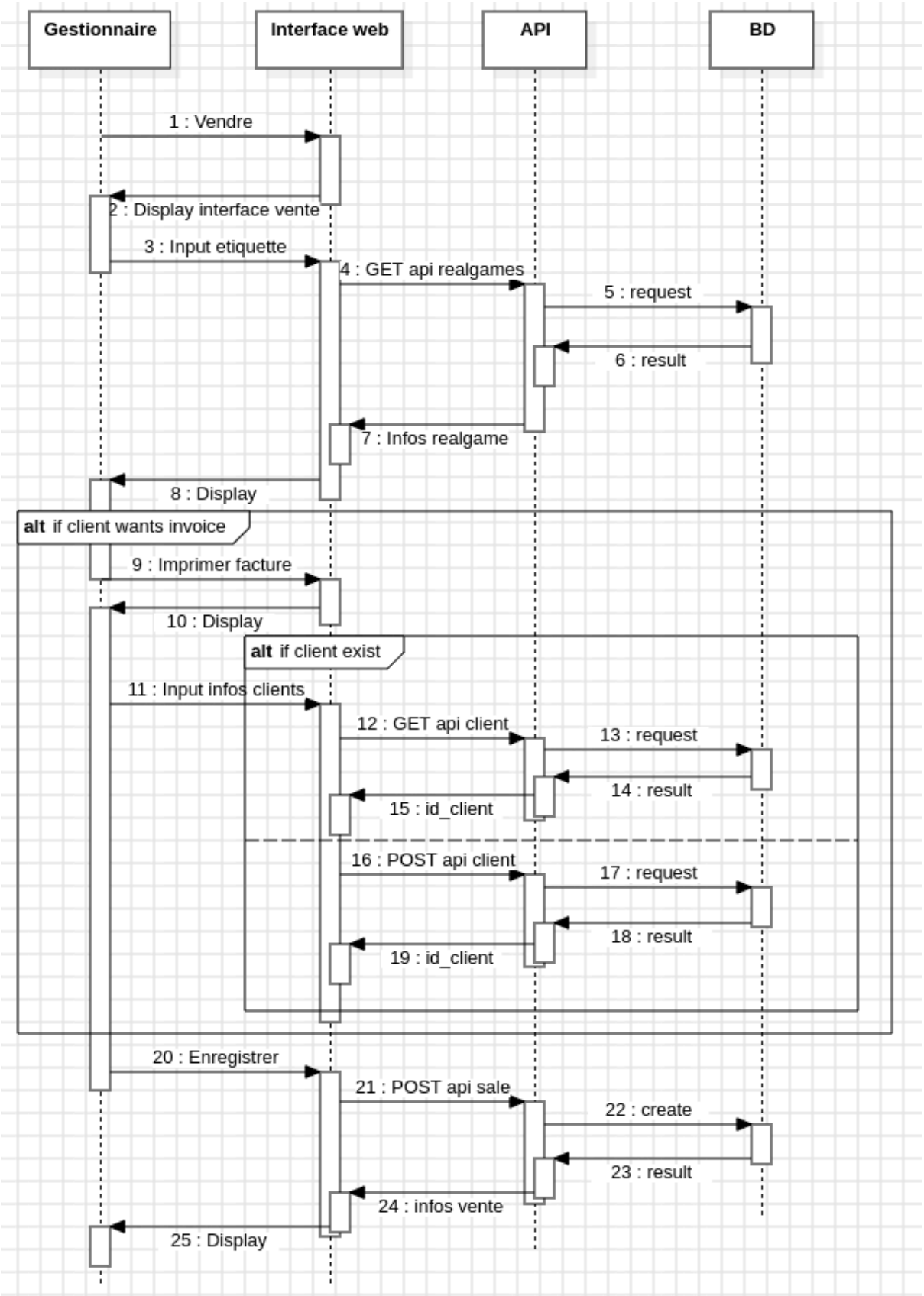
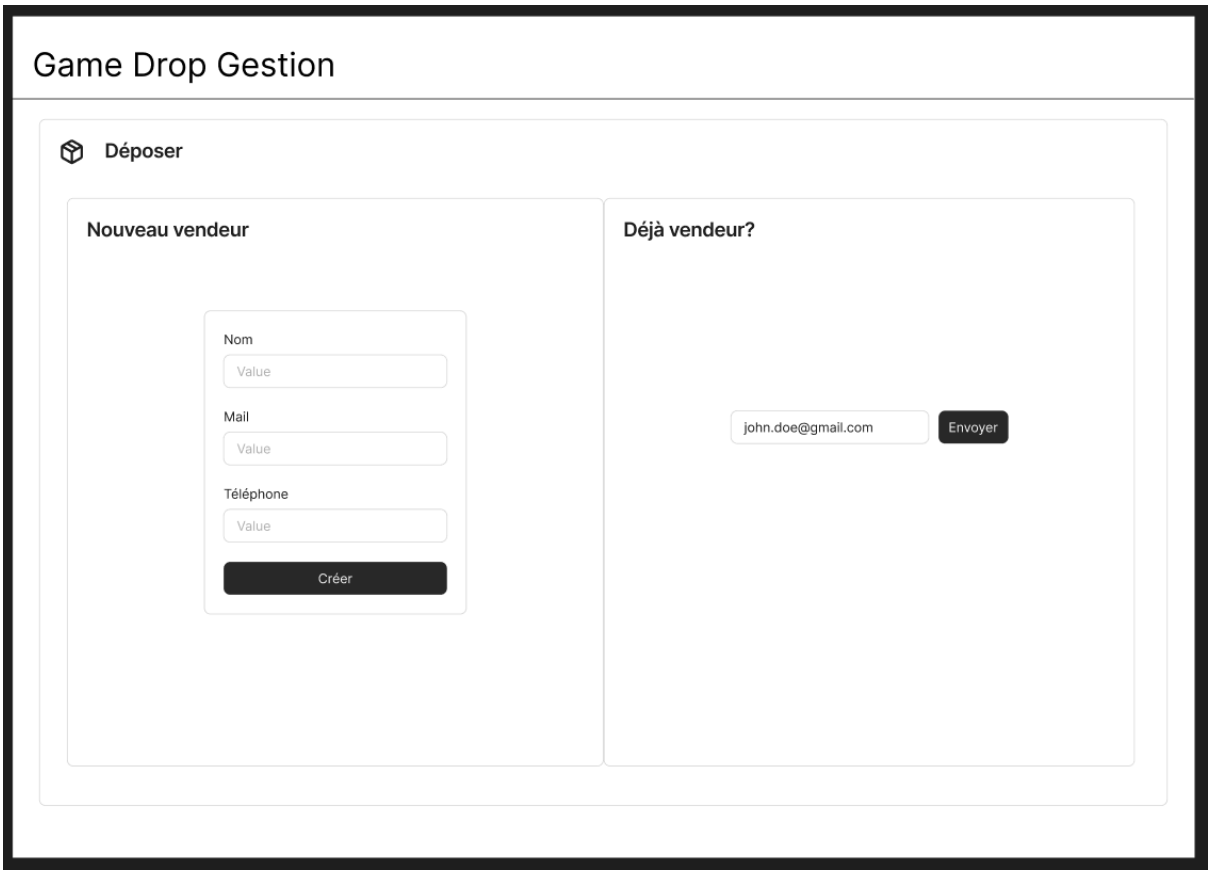
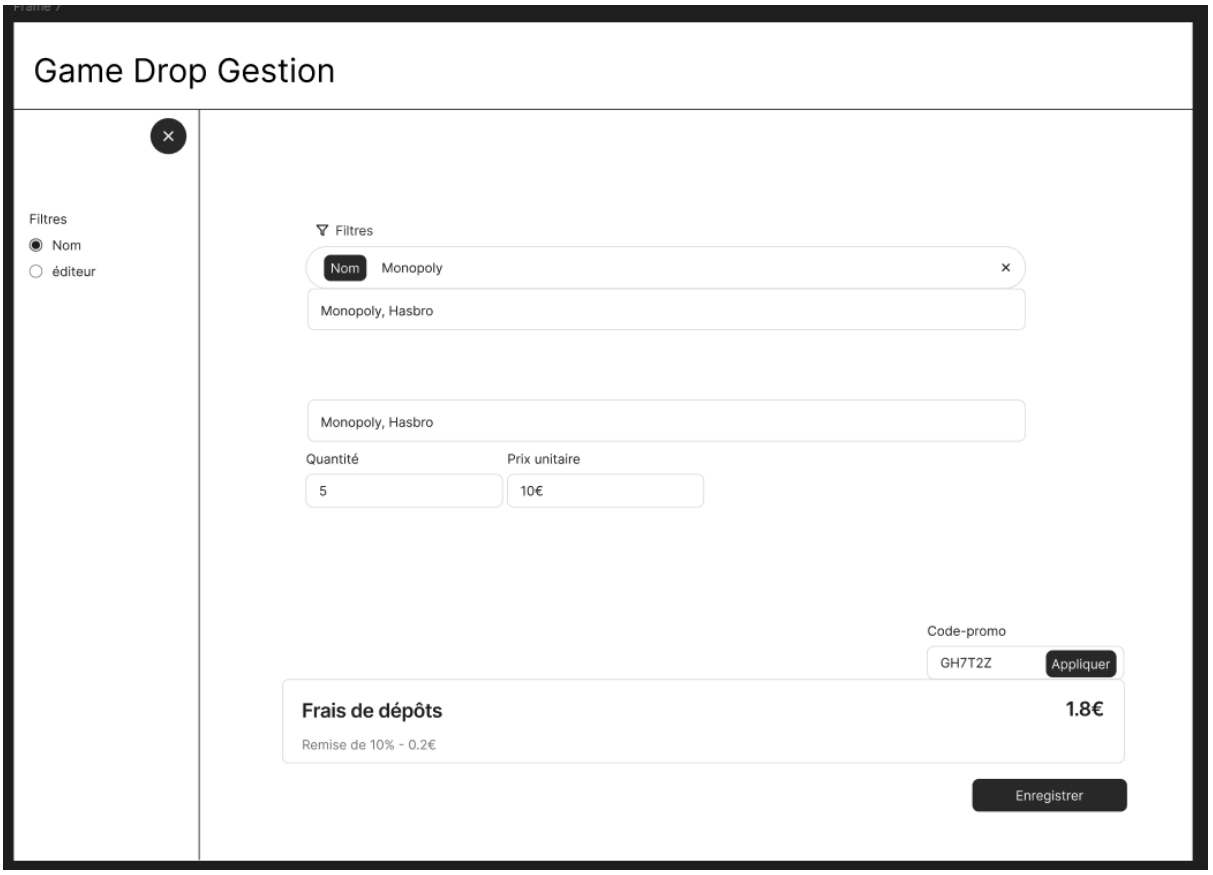


Diagramme de séquence d'une vente



Pour ce qui est de l’interface de dépôt, le gestionnaire doit d’abord demander s' il s’agit du premier dépôt du client. Si ce n’est pas le cas, le gestionnaire remplit les informations du client sinon il recherche le mail du client après consultation de sa carte d'identité ou vérification par mail.



Une fois le client authentifié, le gestionnaire recherche le nom ou l'éditeur du jeu à déposer puis donne la quantité et le prix unitaire désiré par le client. Le gestionnaire à aussi la possibilité d’entrer un code-promo afin d’appliquer une réduction aux frais de dépôts.

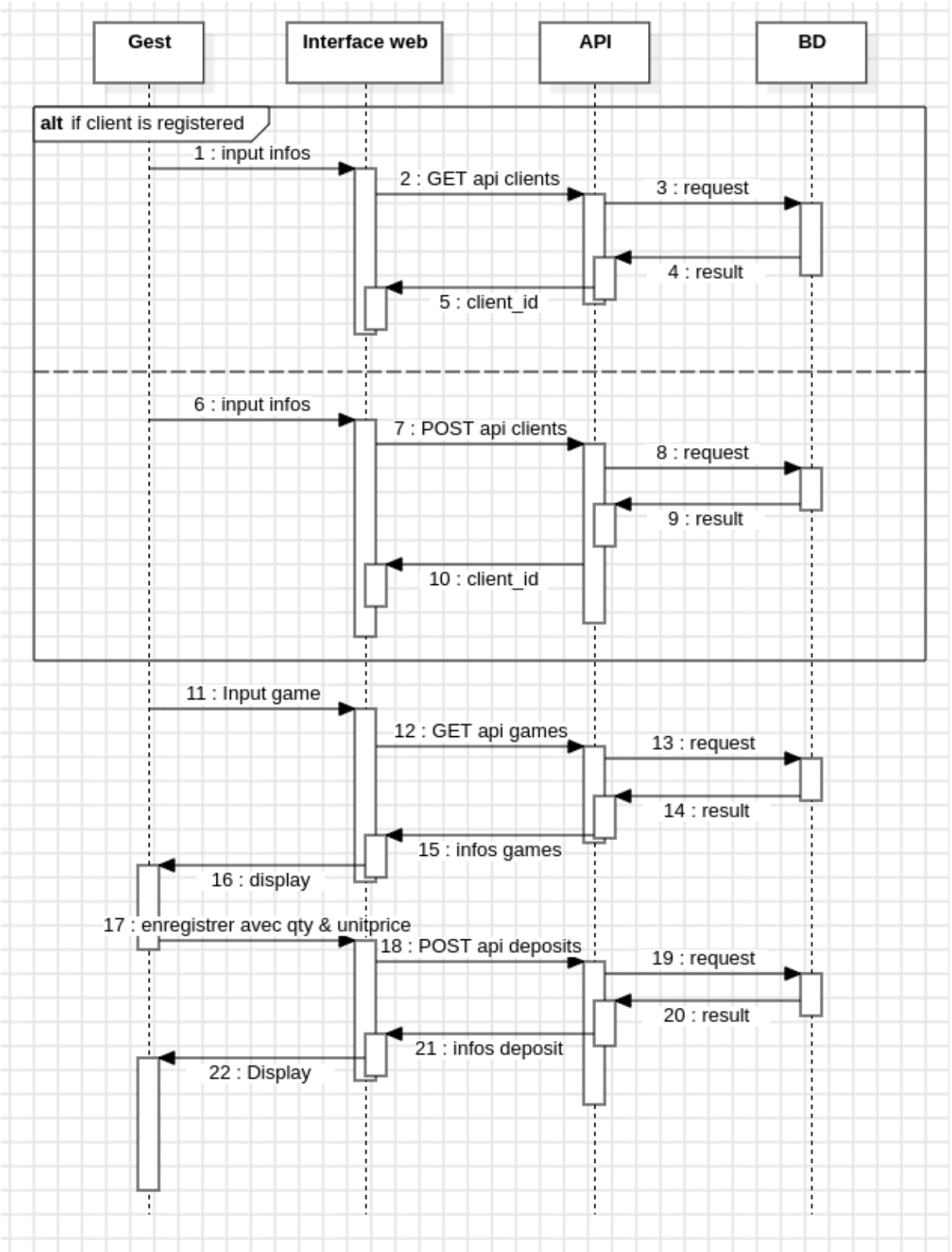
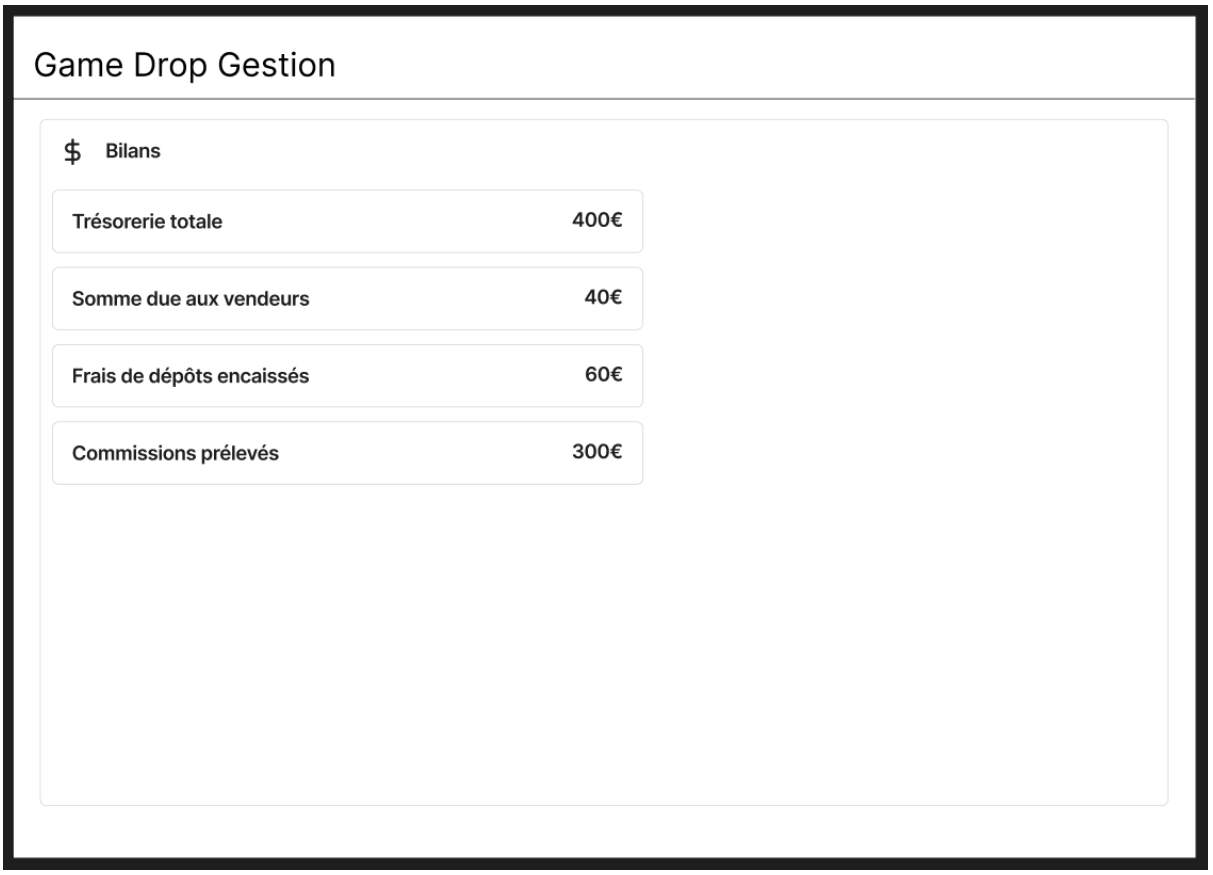
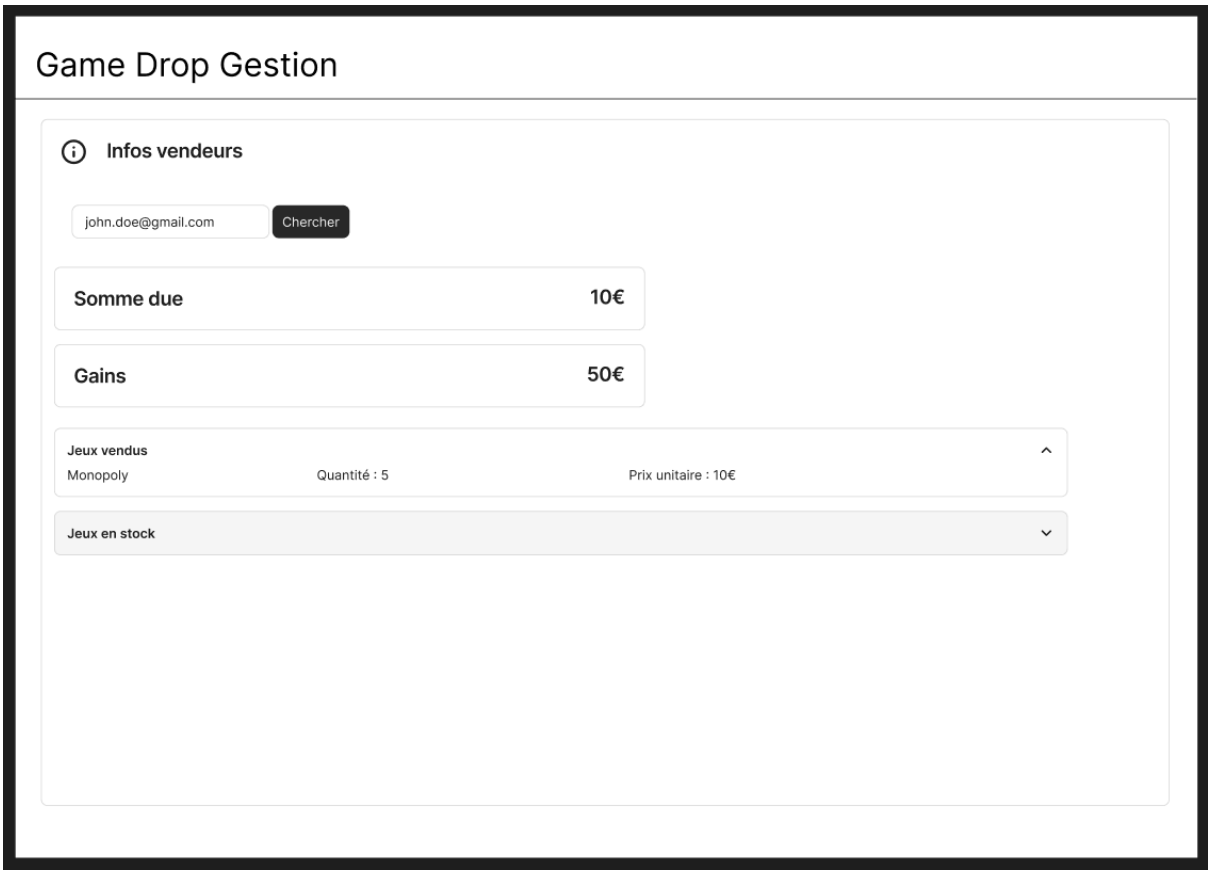


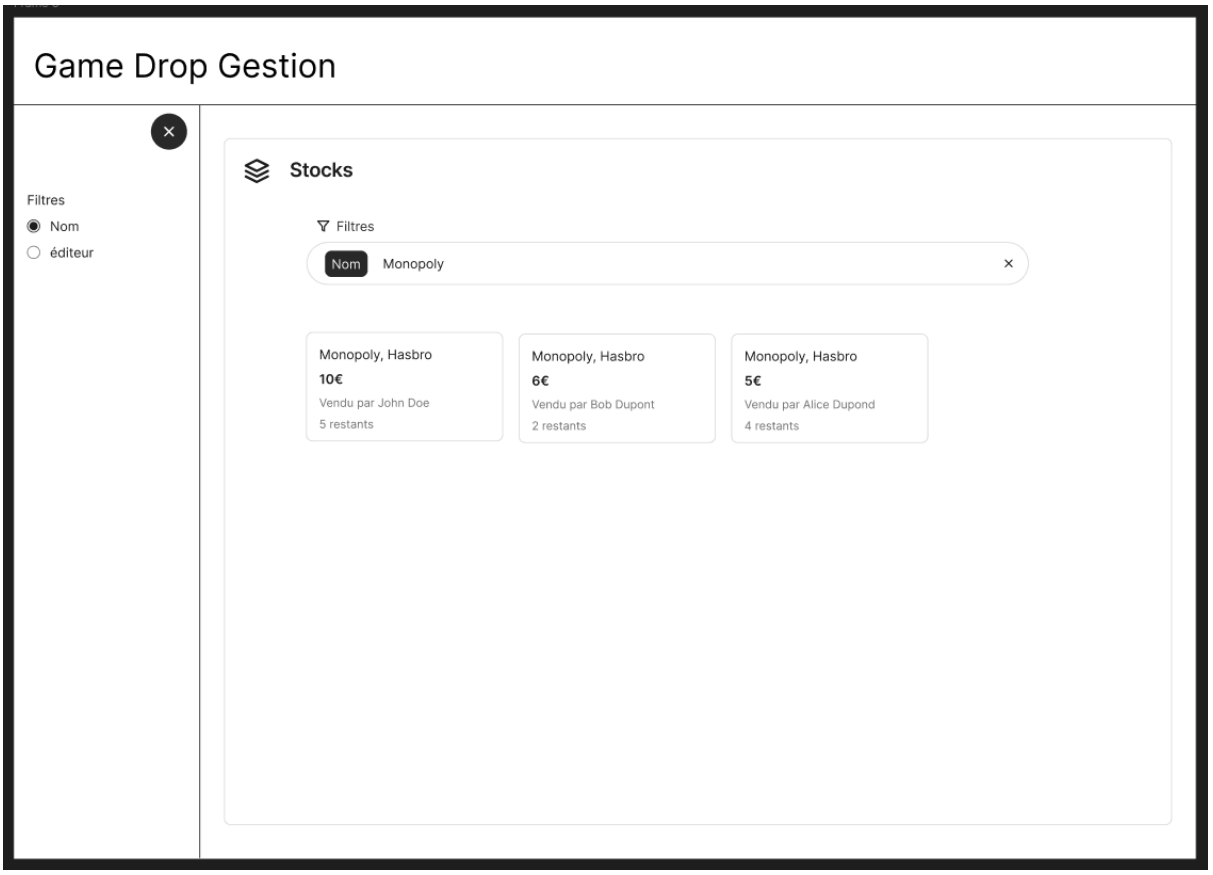
Diagramme de séquence d’un dépôt



Ici, nous retrouvons l'interface de consultation des bilans de *Game Drop*, les gestionnaires peuvent donc y consulter la trésorerie totale, la somme due aux vendeurs, les frais et commissions encaissées.

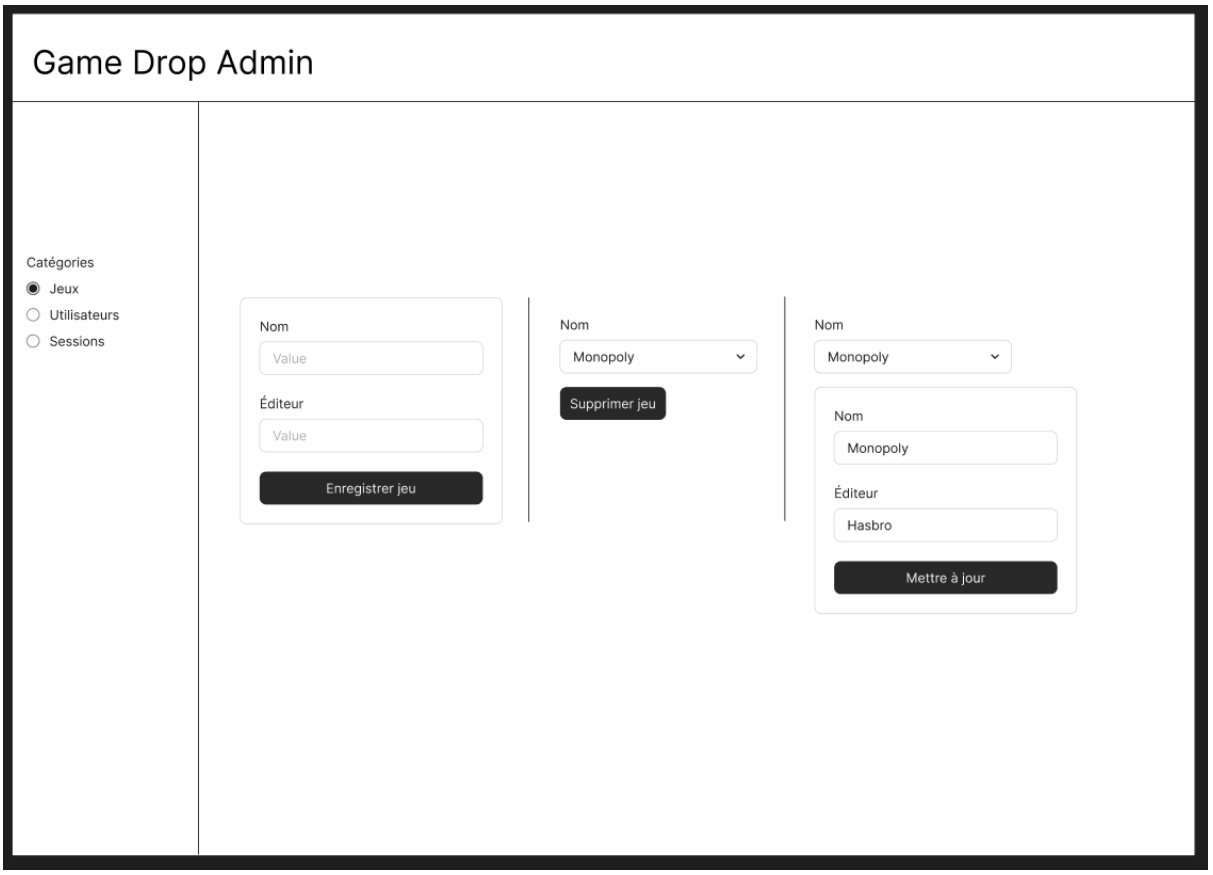


Ici, nous pouvons, à la demande d'un client, consulter ses ventes, la somme due et ses gains. Le gestionnaire pourra aussi créditer le client de la somme due et retirer ses jeux en stocks pour les lui rendre.



Ici, nous pouvons consulter les stocks généraux de *Game Drop*, la page affiche par défaut l'entièreté des stocks mais il est possible de filtrer les résultats par nom et éditeur afin de renseigner un client sur la disponibilité d’un jeu demandé.

Enfin, nous pouvons consulter l'interface des transactions. Cette interface affiche une table ou nous pouvons filtrer par date et montant croissant ou décroissant. Pour chaque transaction, son type, sa date, son montant et le client lié sont spécifiés.



Finalement, nous pouvons retrouver l’interface administrateur ou il sera possible pour ce dernier, de créer, modifier et supprimer des jeux, des utilisateurs et des sessions. Il pourra également modifier les frais de dépôt, les commissions et les codes-promos.

Définition Préliminaire des Endpoints API

Endpoint	Méthode	URI	Paramètres	Réponse
Get current session	GET	/api/session	-	id_current_session | null
Create new session	POST	/api/admin/session	begin date, end date, fees, commission, promo-codes	-
Update session	PUT	/api/admin/session	begin date, end date, fees, commission, promo-codes	-
Delete session	DELETE	/api/admin/session	session	-
Get catalog for query	GET	/api/catalog	query, filter, numpage	available games grouped by sellers
Gestionnaire, admin login	POST	/api/gestion/login	email, pwd	token JWT for API Auth
Get realgames for query	GET	/api/gestion/realgames	query	available realgames where id match query
Register product sale	POST	/api/gestion/sale	client_id | null, realgame_id list	invoice | null
Get client id from email	GET	/api/gestion/client	query	client_id | null
Create client from infos	POST	/api/gestion/client	email, name, phone number, address (OPTIONAL)	client_id
Update client from infos	PUT	/api/admin/client	client_id, email, name, phone number, address (OPTIONAL)	-
Delete client	DELETE	/api/admin/client	email	-

Get realgames of client	GET	/api/gestion/client/realgames	client_id	realgames of a seller
Get amount due to client	GET	/api/gestion/client/due	client_id	amount due to seller for current session
Get withdrawn money	GET	/api/gestion/client/withdrawn	client_id	withdrawn money by seller for current session
Get games for query	GET	/api/gestion/games	query, filter	games where name|editor match query
Create new game	POST	/api/admin/games	nom, editeur	-
Update game	PUT	/api/admin/games	nom, editeur	-
Delete game	DELETE	/api/admin/games	game_id	-
Get discount for query	GET	/api/gestion/promocode	query	discount | null
Register product deposit	POST	/api/gestion/deposit	client_id, game_id list with quantity & unit price, promocode (OPTIONAL)	-
Get deposit fees	GET	/api/gestion/fees	-	deposit fees for current session
Get total treasury	GET	/api/gestion/balance/treasury	-	total treasury for current session
Get amount due to sellers	GET	/api/gestion/balance/due	-	amount due to sellers for current session
Get deposit fees collected	GET	/api/gestion/balance/deposit-fees	-	collected deposit fees for current session
Get commissions	GET	/api/gestion/balance/c	-	collected

collected		ommissions		commissions for current session
Get transactions	GET	/api/gestion/transactions	-	transactions of current session

Sécurité et Authentification

Sécurisation des Endpoints

Authentification

Pour ce projet, une méthode d'authentification basée sur JWT (JSON Web Tokens) sera utilisée. Elle permet d'authentifier les utilisateurs et de s'assurer que seuls les administrateurs et gestionnaires puissent accéder aux endpoints de gestions.

- **Processus d'authentification :**

Lorsqu'un utilisateur (gestionnaire ou administrateur) se connecte, il envoie ses identifiants (email et mot de passe) via un endpoint dédié à la connexion.

Si les informations sont correctes, le serveur génère un JWT qui contient les informations de l'utilisateur (ID, rôle).

Ce token est ensuite renvoyé au client, qui le stocke dans un cookie sécurisé.

- **Validation du token :**

À chaque requête à un endpoint sécurisé, l'utilisateur inclut le token JWT dans le header HTTP (souvent dans le champ Authorization).

Le serveur vérifie la validité du token (vérification de la signature, expiration, etc.) avant d'autoriser l'accès aux ressources.

Autorisations

L’autorisation est gérée en fonction du rôle de l’utilisateur, défini lors de la génération du JWT. Les rôles possibles sont :

- **ADMIN**
- **GESTION**

Chaque rôle a un accès différencié aux endpoints selon ses responsabilités :

- **ADMIN** : a un accès complet à tous les endpoints (gestion des utilisateurs, sessions, ventes, dépôts, etc.).
- **GESTION** : a un accès aux endpoints nécessaires à la gestion des dépôts et ventes, mais n'a pas accès aux fonctions d'administration (création/suppression d’utilisateurs, etc.).
- **Endpoints publics** : Seul le catalogue des jeux est accessible sans authentification. Les autres endpoints nécessitent une vérification des rôles via JWT.

Endpoints protégés

Voici comment les endpoints sont sécurisés :

- Catalogue des jeux (endpoint public) :
    - Accessible à tous sans authentification (JWT non requis).
    - Exemple : GET /games

- Endpoints réservés aux gestionnaires et administrateurs :
    - Dépôts, ventes, sessions, transactions : Ces endpoints ne sont accessibles qu’aux utilisateurs authentifiés ayant un rôle GESTION ou ADMIN.
    - Les tokens JWT sont vérifiés à chaque requête et l’accès est accordé uniquement si le rôle de l’utilisateur le permet.

Conclusion et Prochaines Étapes

Le projet de site de gestion de dépôt et de vente de jeux a bien été modélisé en respectant les principes de normalisation et d'efficacité. L'utilisation des technologies AngularJS pour le front-end, ExpressJS pour le back-end, PostgreSQL pour la base de données, et le déploiement via Dokku assure une architecture robuste, scalable et facilement maintenable.

La sécurité des endpoints a été abordée avec la mise en place d’un système d’authentification via JWT pour protéger les parties sensibles de l’application, tout en laissant le catalogue des jeux accessible au public sans authentification. Le MCD et MLD ont permis de valider la structure des données, les clés primaires et étrangères ainsi que les relations entre les différentes entités.