Projet de Gestion de Portefeuille d'Actions Virtuelles

Jean Delepine | Kylian Parguer | Mathéo Guedes Berthézène | Nikola Marchal

IF3A – Groupe Y

Table des matières

I. Introduction

Dans le cadre de nos études en IF3A, nous avons conçu une application web de simulation boursière virtuelle permettant à un joueur d'investir un capital de 10 000 € dans un portefeuille d'actions numériques. L'objectif principal est d'offrir une expérience de gestion de portefeuille en pseudo-temps réel, intégrant l'évolution mensuelle des cours, la distribution annuelle de dividendes et une mécanique de perte automatique lorsque le capital total tombe en dessous de 1000 €.

Ce projet, développé en PHP et MySQL, couvre les aspects suivants : gestion des comptes (inscription, connexion, modification de mot de passe), fonctionnalités de suivi social (recherche et suivi d'autres joueurs) et mécanique de jeu (achats/ventes, mise à jour mensuelle des prix, distribution des dividendes, classement).

La suite du document présente, pour chaque fonctionnalité, l'objectif visé, le travail réalisé et une brève conclusion.

II. Fonctionnalités générales

2.1. Connexion utilisateur (login_form.php & login.php)

Fonctionnalités attendues :

- Affichage d'un formulaire de connexion (identifiant + mot de passe).
- Vérification des identifiants via base de données.
- Gestion des erreurs (identifiant inconnu, mot de passe incorrect).
- Connexion sécurisée avec sessions PHP.
- Redirection vers la page d'accueil en cas de succès.

Fonctionnalités réalisées :

a. Formulaire de connexion (`login_form.php`)

- **Objectif** : Afficher un formulaire simple et fonctionnel.
- **Travail réalisé** : Champs requis, envoi en POST, liens vers inscription et modification du mot de passe.
- **Conclusion** : Fonction complète et opérationnelle.

b. Traitement des identifiants (`login.php`)

- **Objectif** : Vérifier l'identifiant et le mot de passe.
- **Travail réalisé** : Connexion à la BDD, requête préparée, vérification via `password_verify()`.
- **Conclusion** : Traitement sécurisé et conforme aux bonnes pratiques.

c. Gestion des erreurs & session

- **Objectif** : Informer l'utilisateur et créer une session.
- **Travail réalisé** : Messages via `$_SESSION`, redirection selon le résultat.
- **Conclusion** : Fonctionnalité correctement gérée.

2.2. Création de compte (register_form.php & register.php)

Fonctionnalités attendues :

- Formulaire d'inscription avec identifiant et mot de passe.
- Vérification que l'utilisateur n'existe pas déjà.
- Hachage du mot de passe et enregistrement dans la base.
- Message d'erreur ou de confirmation affiché à l'utilisateur.

Fonctionnalités réalisées :

a. Formulaire (`register_form.php`)

- **Objectif** : Saisie des données d'inscription.
- **Travail réalisé** : Champs requis, lien retour, affichage des messages.
- **Conclusion** : Formulaire clair et fonctionnel.

b. Traitement (`register.php`)

- **Objectif** : Enregistrement sécurisé de l'utilisateur.
- **Travail réalisé** : Requête préparée, mot de passe hashé, solde initial défini.
- **Conclusion** : Inscription sécurisée, messages bien gérés.

2.3. Changement de mot de passe (password_form.php & password_change.php)

Fonctionnalités attendues :

- Interface de saisie de l'identifiant, ancien mot de passe, nouveau mot de passe + confirmation.
- Vérification des identifiants.
- Mise à jour sécurisée du mot de passe dans la base.
- Messages de confirmation ou d'erreur.

Fonctionnalités réalisées :

a. Formulaire (`password_form.php`)

- **Objectif** : Permettre la saisie des informations pour changer de mot de passe.
- **Travail réalisé** : Champs obligatoires, lien retour, affichage de messages.
- **Conclusion** : Interface claire et accessible.

b. Traitement (`password_change.php`)

- **Objectif** : Vérifier l'ancien mot de passe et enregistrer le nouveau.
- **Travail réalisé** :
  - Validation des données (confirmation, utilisateur existant, mot de passe correct).
  - Mise à jour sécurisée avec hachage (`password_hash`).
- **Conclusion** : Fonctionnement conforme, sécurité assurée.

2.4. Déconnexion (logout.php)

Fonctionnalités attendues :

- Déconnexion sécurisée de l'utilisateur.
- Nettoyage de la session.
- Redirection automatique vers la page de connexion.

Fonctionnalités réalisées :

- **Objectif** : Mettre fin à la session de l'utilisateur.
- **Travail réalisé** :

    - Suppression des données de session (`session_unset()` / `session_destroy()`).
    - Message de confirmation affiché.
    - Notification via `BroadcastChannel` pour informer d'une fermeture côté client.
    - Redirection automatique vers le formulaire de connexion.
- **Conclusion** : Déconnexion propre, expérience utilisateur fluide.

2.5. Page d'accueil principale (index.php)

Fonctionnalités attendues :

- Affichage dynamique de l'interface principale selon l'utilisateur connecté.
- Chargement conditionnel des pages internes via un système de routage (GET).
- Ouverture automatique d'une fenêtre de mise à jour si nécessaire.
- Sécurisation de l'accès par vérification de session.

Fonctionnalités réalisées :

- **Objectif** : Centraliser la navigation de l'application après connexion.
- **Travail réalisé** :
    - Vérification de session : redirection vers `login_form.php` si utilisateur non connecté.
    - Ouverture d'une pop-up de mise à jour via `window.open()` à la première connexion.
    - Routage interne : chargement de la page correspondant au paramètre `?p=` (`dashboard`, `portfolio`, `buy`, etc.).
    - Chargement de l'en-tête et du pied de page communs.
- **Conclusion** : Interface modulaire et sécurisée fonctionnelle, avec navigation dynamique et protection d'accès.

III. Fichiers communs

3.1. En-tête du site (header.php)

Fonctionnalités attendues :

- Affichage du menu de navigation avec des liens vers le tableau de bord, le portefeuille, et le classement.
- Gestion dynamique des liens de connexion ou de déconnexion en fonction de l'état de l'utilisateur.

Fonctionnalités réalisées :

- **Navigation dynamique** : Affiche des liens pour accéder à :
  - **Tableau de bord**
  - **Portefeuille**
  - **Classement**
- **Gestion de l'état de connexion** :
  - Si l'utilisateur est connecté, un lien de déconnexion est affiché.
  - Si l'utilisateur n'est pas connecté, un lien vers la page de connexion est proposé.

3.2. Pied de page (footer.php)

Fonctionnalités attendues :

- Affichage d'une mention légale ou d'un crédit.

Fonctionnalités réalisées :

- **Crédits** : Affiche le texte "Code produit par Jean, Kylian, Mathéo, Nikola".

3.3. Page d'erreur 404 (notfound.php)

Fonctionnalités attendues :

- Affichage d'un message d'erreur lorsque la page demandée n'existe pas (erreur 404).
- Proposition d'un lien pour revenir à la page d'accueil.

Fonctionnalités réalisées :

- **Message d'erreur** : Affiche un titre "404" suivi du message "Page non trouvée".
- **Lien de retour** : Un bouton permettant de revenir à l'accueil, avec un lien vers `index.php?p=dashboard`.

IV. Mise à jour du jeu

4.1. Mise à jour du jeu (update.php)

Fonctionnalités attendues :

- Gestion de la mise à jour mensuelle des éléments du jeu (prix des actions, dividendes, portefeuille).
- Suppression des utilisateurs dont la valeur du portefeuille est inférieure à 1000€.
- Notification et communication des changements via un canal Broadcast.
- Mise à jour de la base de données en temps réel.

Fonctionnalités réalisées :

- **Objectif** : Mettre à jour la date du jeu et ajuster les données des utilisateurs et actions.
- **Travail réalisé** :
  - **Mise à jour de la date** : Incrémentation de la date du jeu de 1 mois.
  - **Dividendes** : Calcul et versement des dividendes en fonction des actions détenues par les utilisateurs.
  - **Prix des actions** : Calcul d'une variation aléatoire des prix des actions et mise à jour de la base de données.
  - **Valeur du portefeuille** : Calcul de la valeur du portefeuille des utilisateurs et stockage dans l'historique.
  - **Vérification des pertes** : Suppression des utilisateurs dont la valeur totale du portefeuille est inférieure à 1000€, avec nettoyage des données associées.

    - **Notification de mise à jour** : Ajout d'un flag dans `update_flag.txt` et utilisation de `BroadcastChannel` pour notifier l'interface de la mise à jour.
- **Conclusion** : Mise à jour complète du jeu réalisée, incluant calculs de dividendes, ajustements des prix et gestion des pertes.

4.2. Fichier HTML associé (update.html)

- **Objectif** : Afficher un message de mise à jour en cours.
- **Travail réalisé** :
    - Affichage d'un message de mise à jour.
    - Surveillance de l'état de la mise à jour via `BroadcastChannel`.
    - Vérification périodique des fichiers `player_lost_flag.txt` et `update_flag.txt` pour déclencher des actions de fermeture ou notification.
- **Conclusion** : Interface de mise à jour fonctionnelle, avec surveillance continue de l'état.

v. Pages utilisateur

5.1. Tableau de bord (dashboard.php)

Fonctionnalités attendues :

- Affichage de l'état du jeu pour un utilisateur connecté.
- Récupération et affichage des informations personnelles (argent, actions, portefeuille).
- Vérification et gestion des utilisateurs dont la valeur du portefeuille est inférieure à 1000€.
- Surveillance des mises à jour du jeu via un fichier `update_flag.txt`.

Fonctionnalités réalisées :

- **Vérification de la connexion** : Assure que l'utilisateur est connecté avant d'afficher le tableau de bord.

- **Informations personnelles** : Récupère l'ID utilisateur et l'argent disponible de la base de données.
- **Vérification du portefeuille** : Calcule la valeur totale du portefeuille de l'utilisateur (actions + argent) et supprime l'utilisateur si cette valeur est inférieure à 1000€.
- **Affichage des actions possédées** : Récupère et affiche les actions détenues par l'utilisateur, y compris la quantité et la valeur actuelle.
- **Évolution du portefeuille** : Affiche l'historique de la valeur du portefeuille sur les 12 derniers mois.
- **Vérification périodique des mises à jour** : Vérifie toutes les 5 secondes si une mise à jour du jeu a eu lieu via le fichier `update_flag.txt`. Si le fichier a été mis à jour, recharge la page pour refléter les changements.
- **Conclusion** : Tableau de bord interactif avec mise à jour automatique et gestion des utilisateurs en fonction de leur valeur totale.

5.2. Gestionnaire des actions (actions.php)

Fonctionnalités attendues :

- Affichage de la liste des actions disponibles avec leurs prix.
- Filtrage des actions selon plusieurs critères : nom, prix minimum, prix maximum, et progression sur une période donnée (1 mois ou 1 an).
- Possibilité d'acheter et de vendre des actions.
- Affichage de l'évolution du prix d'une action sur 12 mois via un graphique.

Fonctionnalités implémentées :

1. **Affichage des actions disponibles :**
    a. **Objectif** : Afficher la liste des actions avec leur nom et prix.
    b. **Travail réalisé** : La liste des actions est récupérée depuis la base de données et affichée dans un tableau HTML. Chaque action dispose d'options pour l'achat, la vente et la visualisation de son évolution.
    c. **Conclusion** : Fonctionnalité bien implémentée.
2. **Filtrage des actions :**
    a. **Objectif** : Permettre de filtrer les actions par nom, prix et progression sur 1 mois ou 1 an.
    b. **Travail réalisé** : Des champs de saisie et un menu déroulant permettent à l'utilisateur de filtrer les actions. Les données sont

récupérées dynamiquement via une requête SQL préparée avec des conditions basées sur les entrées de l'utilisateur.

c. **Conclusion** : Fonctionnalité bien implémentée avec des requêtes SQL sécurisées et filtres dynamiques.

3. **Affichage de l'évolution des actions :**
   a. **Objectif** : Permettre à l'utilisateur de visualiser l'évolution du prix d'une action sur les 12 derniers mois.
   b. **Travail réalisé** : Un graphique est généré en utilisant `Chart.js` pour afficher l'évolution des prix sur 12 mois lorsque l'utilisateur choisit de voir l'évolution d'une action.
   c. **Conclusion** : Fonctionnalité bien implémentée, graphique dynamique fonctionnel.
4. **Interaction avec la base de données :**
   a. **Objectif** : Récupérer et afficher les actions à partir d'une base de données.
   b. **Travail réalisé** : Une connexion à la base de données est établie avec une requête SQL pour récupérer les données des actions. Les filtres sont appliqués dynamiquement en fonction des entrées de l'utilisateur.
   c. **Conclusion** : Connexion et récupération de données efficaces et sécurisées via des requêtes préparées.

5.3. Fichier d'achat des actions (buy.php)

Fonctionnalités attendues :

- Vérifier si l'utilisateur est connecté.
- Vérifier que l'utilisateur a suffisamment d'argent pour acheter les actions demandées.
- Mettre à jour le portefeuille de l'utilisateur et les transactions d'achat.

Fonctionnalités implémentées :

1. **Vérification de la connexion de l'utilisateur :**
   a. **Objectif** : S'assurer que l'utilisateur est connecté avant de pouvoir acheter des actions.
   b. **Travail réalisé** : Si l'utilisateur n'est pas connecté, un message d'erreur est affiché et une redirection est effectuée.
   c. **Conclusion** : Fonctionnalité correctement implémentée.

2. **Vérification des fonds suffisants :**
    a. **Objectif** : S'assurer que l'utilisateur dispose des fonds nécessaires pour l'achat.
    b. **Travail réalisé** : Le montant total de l'achat est calculé et comparé aux fonds disponibles de l'utilisateur. Si l'utilisateur a suffisamment d'argent, l'achat est effectué, sinon un message d'erreur est affiché.
    c. **Conclusion** : Fonctionnalité correcte, validation des fonds bien réalisée.
3. **Mise à jour du portefeuille et des transactions :**
    a. **Objectif** : Mettre à jour le portefeuille de l'utilisateur et enregistrer la transaction d'achat.
    b. **Travail réalisé** : Les informations de portefeuille sont mises à jour avec la quantité achetée, et une transaction est enregistrée dans la base de données.
    c. **Conclusion** : Fonctionnalité bien implémentée, mise à jour correcte du portefeuille.
4. **Gestion des erreurs et redirections :**
    a. **Objectif** : Fournir un retour clair en cas d'erreur (par exemple, si l'utilisateur n'a pas assez d'argent).
    b. **Travail réalisé** : Des messages d'erreur sont affichés en cas de problème (quantité incorrecte, fonds insuffisants, utilisateur non trouvé, etc.).
    c. **Conclusion** : Fonctionnalité de gestion des erreurs bien implémentée avec redirections appropriées.

5.4. Fichier de vente des actions (sell.php)

Fonctionnalités attendues :

- Vérifier si l'utilisateur est connecté.
- Vérifier que l'utilisateur possède suffisamment d'actions à vendre.
- Mettre à jour le portefeuille de l'utilisateur et enregistrer la transaction de vente.

Fonctionnalités implémentées :

1. **Vérification de la connexion de l'utilisateur :**

a. **Objectif** : S'assurer que l'utilisateur est connecté avant de pouvoir vendre des actions.
b. **Travail réalisé** : Si l'utilisateur n'est pas connecté, un message d'erreur est affiché et une redirection est effectuée.
c. **Conclusion** : Fonctionnalité correctement implémentée.

2. **Vérification des actions disponibles à la vente :**
   a. **Objectif** : Vérifier que l'utilisateur possède suffisamment d'actions pour vendre la quantité demandée.
   b. **Travail réalisé** : Le portefeuille de l'utilisateur est consulté pour s'assurer qu'il possède les actions à vendre en quantité suffisante.
   c. **Conclusion** : Fonctionnalité correcte, vérification des actions bien réalisée.
3. **Mise à jour du portefeuille et des transactions :**
   a. **Objectif** : Mettre à jour le portefeuille de l'utilisateur après la vente d'actions et enregistrer la transaction.
   b. **Travail réalisé** : Le portefeuille de l'utilisateur est mis à jour en fonction de la quantité vendue et une transaction est enregistrée dans la base de données.
   c. **Conclusion** : Fonctionnalité bien implémentée, mise à jour correcte du portefeuille.
4. **Gestion des erreurs et redirections :**
   a. **Objectif** : Fournir un retour clair en cas d'erreur (par exemple, si l'utilisateur ne possède pas suffisamment d'actions).
   b. **Travail réalisé** : Des messages d'erreur sont affichés en cas de problème (quantité incorrecte, action non trouvée, etc.).
   c. **Conclusion** : Fonctionnalité de gestion des erreurs bien implémentée avec redirections appropriées.

5.5. Affichage du portfolio (portfolio.php)

1. Vérification de la connexion de l'utilisateur

Objectif :

Avant d'afficher le portfolio, l'utilisateur doit être connecté. Si ce n'est pas le cas, un message d'erreur est affiché et l'accès au portfolio est empêché.

Travail réalisé :

Un contrôle est effectué au début du fichier pour vérifier si une session utilisateur est active. Si la session n'existe pas, un message d'erreur est affiché, et l'exécution du script est interrompue avec `exit()`. Si l'utilisateur est connecté, son nom d'utilisateur est récupéré pour l'affichage du portfolio.

Conclusion :

Cette fonctionnalité assure que seuls les utilisateurs connectés peuvent accéder à leur portfolio, respectant ainsi les critères de sécurité et de confidentialité.

2. Affichage du portfolio de l'utilisateur

Objectif :

Le fichier doit afficher un tableau des actions détenues par l'utilisateur, en récupérant les informations pertinentes depuis la base de données, incluant le nom des actions, la quantité détenue, la valeur par action et la valeur totale.

Travail réalisé :

1. Une connexion à la base de données est établie avec les paramètres définis.
2. L'ID de l'utilisateur est récupéré depuis la base de données en fonction du nom d'utilisateur de la session.
3. Une requête est ensuite exécutée pour récupérer les informations des actions détenues par l'utilisateur, via une jointure entre les tables `wallet` et `actions`.
4. Si des résultats sont trouvés, les données sont affichées sous forme de tableau HTML.

Conclusion :

Le portfolio est correctement affiché avec les informations des actions, permettant à l'utilisateur de visualiser ses investissements. La gestion des erreurs permet d'afficher un message adéquat si aucun portefeuille n'est trouvé.

3. Gestion des erreurs de connexion à la base de données

Objectif :

En cas d'erreur de connexion à la base de données, un message d'erreur doit être affiché, et le script doit arrêter son exécution.

Travail réalisé :

Une gestion des erreurs est mise en place pour la connexion à la base de données. Si la connexion échoue, un message d'erreur est affiché, et le script s'arrête immédiatement.

Conclusion :

Cette fonctionnalité garantit que l'utilisateur ne verra pas de pages vides ou de comportements inattendus si une erreur survient lors de la connexion à la base de données.

4. Vérification de l'existence d'un portefeuille pour l'utilisateur

Objectif :

L'objectif est de s'assurer qu'un portefeuille existe pour l'utilisateur. Si ce n'est pas le cas, un message d'erreur est affiché, et l'utilisateur est invité à effectuer une transaction pour créer un portefeuille.

Travail réalisé :

Une requête est exécutée pour vérifier si un portefeuille existe dans la table `wallet` pour l'utilisateur en question. Si aucun portefeuille n'est trouvé, un message d'erreur est affiché.

Conclusion :

Cela assure que les utilisateurs ne verront pas de portfolio vide s'ils n'ont pas encore effectué de transactions et n'ont donc pas encore de portefeuille.

5. Mise à jour en temps réel via une vérification du fichier `update_flag.txt`

Objectif :

Mettre à jour automatiquement le contenu du portfolio toutes les 5 secondes si des changements sont détectés dans le fichier `update_flag.txt`, signalant une modification.

Travail réalisé :

Un script JavaScript vérifie toutes les 5 secondes le fichier `update_flag.txt` pour détecter une modification. Si le fichier a changé, la page est rechargée afin de refléter les dernières données.

Conclusion :

Cette fonctionnalité permet de maintenir le portfolio de l'utilisateur à jour sans avoir besoin d'un rafraîchissement manuel de la page.

6. Bouton pour retourner à la page des actions disponibles

Objectif :

Un bouton permettant à l'utilisateur de revenir à la page des actions disponibles pour effectuer des transactions ou consulter d'autres informations.

Travail réalisé :

Un formulaire avec un bouton de soumission est ajouté au début du fichier. En cliquant sur ce bouton, l'utilisateur est redirigé vers la page des actions disponibles.

Conclusion :

Le bouton est fonctionnel et permet à l'utilisateur de naviguer facilement vers d'autres pages liées à la gestion des actions.

VI. Suivi et classement

6.1. Classement des joueurs (ranking.php)

1. Affichage du classement général des joueurs

Objectif :

L'objectif est d'afficher un classement général des utilisateurs en fonction de la somme d'argent qu'ils possèdent, trié par ordre décroissant.

Travail réalisé :

Le code PHP établit une connexion à la base de données et exécute une requête pour récupérer les utilisateurs et leur solde d'argent. Les résultats sont affichés sous forme de tableau HTML. Si aucun joueur n'est trouvé, un message d'erreur est affiché. Ce classement est affiché lorsque l'utilisateur n'a pas effectué de recherche ou lorsqu'il clique sur le bouton "Retour au classement général".

Conclusion :

Cette fonctionnalité est bien implémentée et permet de visualiser facilement les utilisateurs et leur solde d'argent dans un classement trié par valeur. L'affichage se fait de manière claire et sans erreurs.

2. Recherche d'un utilisateur spécifique

Objectif :

L'objectif est de permettre à l'utilisateur de rechercher un joueur spécifique par son identifiant. Si un utilisateur est trouvé, ses informations (nom et solde d'argent) ainsi que ses dernières transactions doivent être affichées.

Travail réalisé :

Un formulaire est mis en place pour saisir l'identifiant du joueur. Lorsqu'une recherche est effectuée, une requête SQL est envoyée pour chercher un utilisateur dont l'identifiant correspond au texte saisi. Si un utilisateur est trouvé, ses informations sont affichées dans un tableau,

suivies des transactions récentes de cet utilisateur. Ces transactions sont limitées à 10 et sont affichées sous forme de tableau, incluant des informations comme le type de transaction, le nom de l'action, la quantité, la valeur totale, et la date de la transaction.

Conclusion :

La fonctionnalité de recherche fonctionne correctement. L'utilisateur peut chercher un joueur par identifiant, et le système affiche les informations pertinentes et les transactions récentes de l'utilisateur recherché. Si aucune correspondance n'est trouvée, un message d'erreur est correctement affiché.

3. Affichage des transactions récentes d'un utilisateur

Objectif :

L'objectif est d'afficher les 10 dernières transactions d'un utilisateur, incluant le type de transaction, le nom de l'action, la quantité, la valeur totale, et la date.

Travail réalisé :

Après avoir récupéré les informations sur un utilisateur, une requête supplémentaire est exécutée pour obtenir les transactions récentes (limitées à 10). Si des transactions sont trouvées, elles sont affichées sous forme de tableau à l'intérieur du tableau principal des informations utilisateur. Les transactions sont triées par date et affichées avec les détails mentionnés.

Conclusion :

La fonctionnalité fonctionne comme prévu, et les transactions récentes sont bien affichées. Si un utilisateur n'a pas de transactions récentes, un message d'erreur est affiché pour indiquer qu'il n'y a aucune transaction disponible.

4. Retour au classement général

Objectif :

Permettre à l'utilisateur de revenir au classement général des joueurs après avoir effectué une recherche.

Travail réalisé :

Un bouton "Retour au classement général" a été ajouté sous le formulaire de recherche. Lorsqu'il est cliqué, il réinitialise la recherche en vérifiant si le bouton a été pressé et recharge le classement général en affichant les utilisateurs triés par leur solde d'argent.

Conclusion :

Cette fonctionnalité est implémentée correctement. Le bouton permet de revenir facilement au classement général sans difficulté.

5. Gestion des erreurs et des messages

Objectif :

L'objectif est de gérer les erreurs de manière appropriée, notamment si la recherche ne donne aucun résultat ou si aucune donnée n'est entrée pour la recherche.

Travail réalisé :

Des messages d'erreur sont affichés dans les cas suivants :

- Si aucun utilisateur n'est trouvé lors de la recherche.
- Si un utilisateur n'a aucune transaction.
- Si le champ de recherche est vide lorsque l'utilisateur essaie de rechercher un joueur.

Ces messages d'erreur sont affichés sous forme de texte dans la page HTML, ce qui aide l'utilisateur à comprendre la situation.

Conclusion :

La gestion des erreurs est bien réalisée, avec des messages d'erreur clairs et appropriés qui aident l'utilisateur à naviguer et à comprendre les éventuelles situations d'échec.

6. Sécurisation des entrées utilisateur

Objectif :

Assurer que les entrées utilisateur (notamment l'identifiant du joueur) sont sécurisées pour éviter les injections SQL.

Travail réalisé :

Les données saisies par l'utilisateur (nom d'utilisateur) sont protégées en utilisant la fonction `bind_param` pour préparer la requête SQL, ce qui empêche l'exécution de code malicieux. De plus, les valeurs récupérées de la base de données sont échappées avec `htmlspecialchars` pour éviter les attaques XSS.

Conclusion :

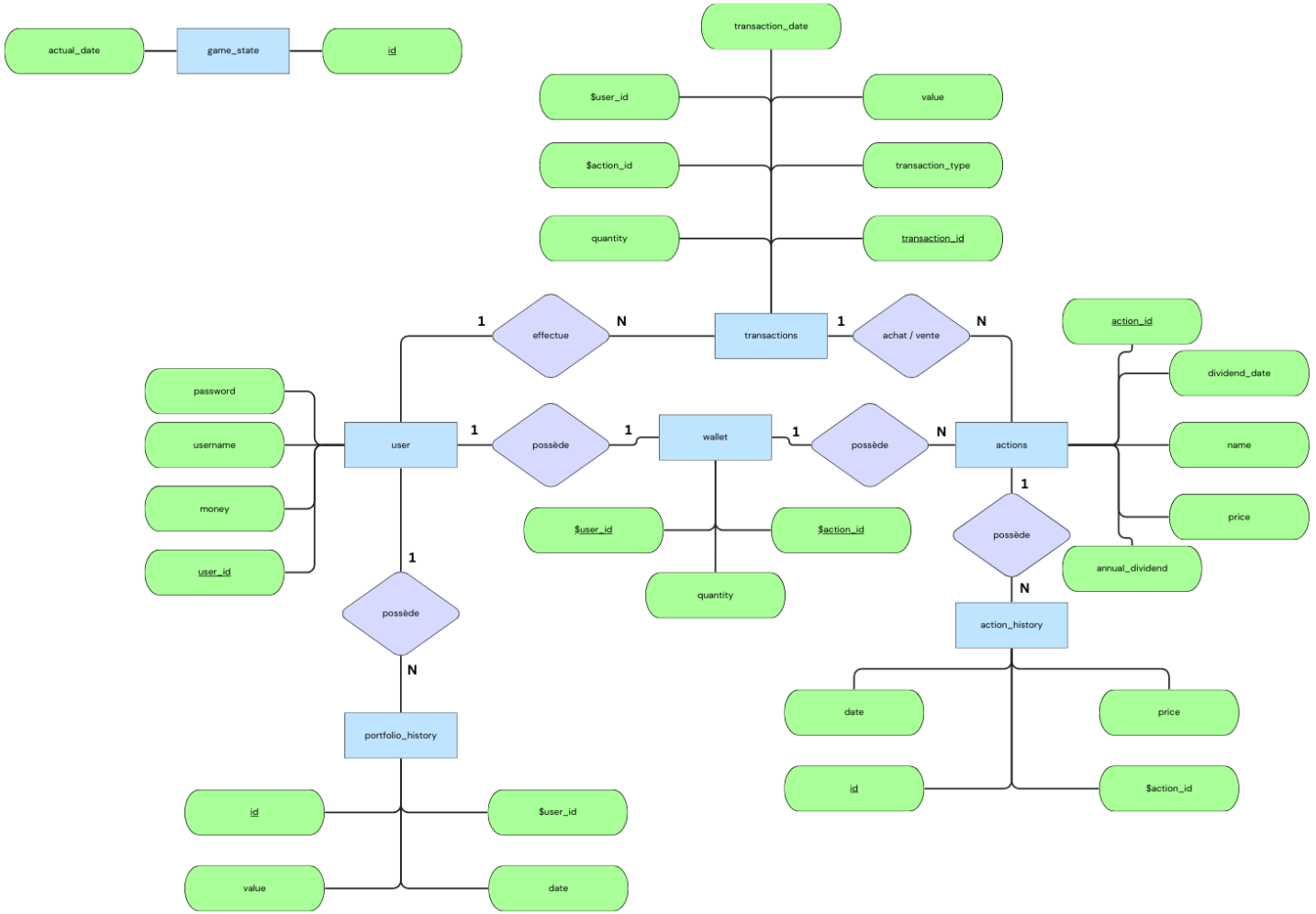
Les entrées utilisateur sont correctement sécurisées, protégeant ainsi l'application contre les attaques par injection SQL et les attaques de type Cross-Site Scripting (XSS).

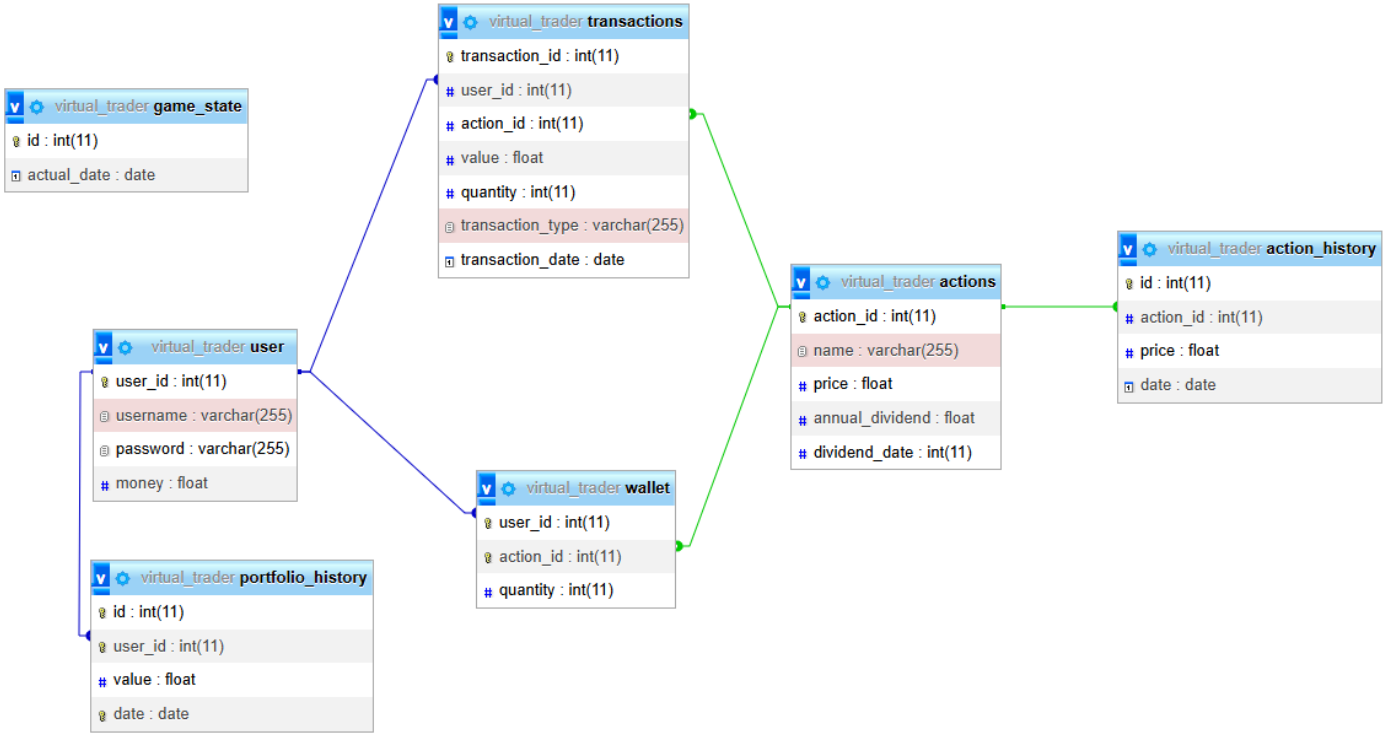
VII. Conclusion

Le projet de simulation de portefeuille virtuel mené en PHP/MySQL répond aux objectifs initiaux : création et gestion sécurisée des comptes, suivi social, transactions en temps réel, actualisation automatique des cours et distribution des dividendes, ainsi que visibilité historique et compétitive. Les modules sont modulaires et extensibles, offrant une base solide pour des améliorations futures (emprunts, options, websocket pour temps réel).

Au-delà des fonctionnalités de base, nous avons implémenté des pratiques de développement sécurisées (hashage, gestion des sessions) et des graphiques via Chart.js.

VIII. Annexes


actual_date
game_state
id
transaction_date
$user_id
value
$action_id
transaction_type
quantity
transaction_id
1
effectue
N
transactions
1
achat / vente
N
action_id
dividend_date
password
username
money
user_id
user
1
possède
1
wallet
1
possède
N
actions
name
price
annual_dividend
$user_id
$action_id
quantity
1
possède
N
action_history
1
possède
N
portfolio_history
date
price
id
$action_id
id
$user_id
value
date


virtual_trader game_state
id : int(11)
actual_date : date
virtual_trader transactions
transaction_id : int(11)
user_id : int(11)
action_id : int(11)
value : float
quantity : int(11)
transaction_type : varchar(255)
transaction_date : date
virtual_trader user
user_id : int(11)
username : varchar(255)
password : varchar(255)
money : float
virtual_trader wallet
user_id : int(11)
action_id : int(11)
quantity : int(11)
virtual_trader portfolio_history
id : int(11)
user_id : int(11)
value : float
date : date
virtual_trader actions
action_id : int(11)
name : varchar(255)
price : float
annual_dividend : float
dividend_date : int(11)
virtual_trader action_history
id : int(11)
action_id : int(11)
price : float
date : date