

# TPI - Echange de cartes à collectionner

---

David Assayah – FIN2 - 2023

# Table des matières

---

1	Analyse préliminaire .....	4
1.1	Introduction.....	4
1.2	Objectifs .....	4
	Matériel à disposition.....	5
1.3	Prérequis.....	5
2	Analyse / Conception .....	8
2.1	Méthodologie de travail.....	8
2.2	Environnement de travail .....	9
2.3	Maquettes.....	9
2.4	Base de données .....	18
2.4.1	MCD.....	18
2.4.2	Entités.....	18
2.4.3	Cardinalités .....	21
2.4.4	MLD .....	23
2.5	Stratégie de test .....	24
2.5.1	Logiciels et outils supplémentaires.....	26
2.6	Risques techniques.....	26
3	Réalisation.....	27
	INTRO + VERSIONS OUTILS .....	27
3.1	Choix de la structure du projet .....	27
3.2	Base de données .....	28
3.2.1	Importation du fichier .SQL.....	28
3.2.2	MPD .....	30
3.2.3	Tables et type des attributs .....	30
3.3	Connexion à la base de données.....	34
3.4	Gestion de l'authentification .....	37
3.5	Gestion des rôles .....	40
3.5.1	Affichage de la page d'accueil selon le rôle de l'utilisateur .....	40
3.5.2	Navigation du header selon le rôle de l'utilisateur .....	41
3.5.3	Affichage du formulaire de login selon le rôle de l'utilisateur .....	42
3.5.4	Déconnexion .....	43
3.6	Création de compte utilisateur.....	45
3.7	Affichage du tableau des cartes en vente.....	57
3.8	Filtres .....	62
3.9	Mise en vente d'une carte.....	68
3.10	Affichage des détails d'une carte .....	77
3.11	Modification d'une carte .....	79
3.12	Suppression d'une carte.....	85
3.13	Système de transaction des points de crédits .....	87

3.13.1	Ajout d'une carte au panier .....	87
3.13.2	Confirmation de la commande .....	89
3.13.3	Création de la commande .....	92
3.14	Confirmation de réception d'une commande .....	98
4	Tests.....	104
4.1.1	.....	105
4.1.2	.....	106
4.2	Erreurs restantes .....	106
4.3	Liste des documents fournis .....	106
5	Conclusions.....	107
5.1	Bilan des fonctionnalités .....	107
5.2	Comparaison de la planification.....	107
5.3	Critiques / Finalité du projet .....	107
5.4	Difficultés particulières .....	107
5.5	Conclusion personnelle .....	107
6	Lexique .....	107
7	Table d'illustrations .....	107
8	Annexes.....	108
8.1	Résumé du rapport du TPI / version succincte de la documentation.	108
8.1.1	Situation de départ .....	108
8.1.2	Mise en œuvre .....	108
8.1.3	Résultats.....	108
8.2	Sources – Bibliographie .....	108
9	Bibliographie .....	108
9.1	Manuel d'Utilisation .....	108
9.2	Archives du projet .....	108

## 1 Analyse préliminaire

### 1.1 Introduction

Ce TPI est réalisé sous la supervision de M.Charmier – Chef de projet – et de M.Venries ainsi que M.Bertino – Experts – dans le cadre de la formation FPA de l'ETML.

Le projet a pour but de produire une application WEB permettant à des collectionneurs de cartes d'échanger celles qu'ils ont à double. Un système de transaction de points de crédits – valeur accordée à chaque carte – doit être mis en place afin de fonctionner comme monnaie d'échange entre acheteurs et vendeurs. Lors de l'achat d'une carte, les crédits dépensés par l'acheteur doivent être mis en attente de la confirmation de la bonne réception des articles. Dès que cela se produit, les crédits sont versés sur le compte du vendeur.

Les utilisateurs doivent pouvoir ajouter/modifier/supprimer leurs propres cartes et consulter/acheter celles des autres. Le tout en ayant la possibilité de filtrer ou trier les cartes mises en vente sur le site.

### 1.2 Objectifs

1. Les formulaires de l'application respectent les bonnes pratiques, à savoir :
  1. Affichage des erreurs contextuelles.
  2. Re-remplissage des champs lors d'erreur.
  3. Validation des champs.
2. La recherche multicritère possède des fonctionnalités de tris et de filtres pertinentes
3. Le candidat explique dans le rapport au moins trois mesures de sécurité qu'il a implémentées dans son projet.
4. La gestion de l'authentification et des rôles des utilisateurs garantit un accès sécurisé aux données.
5. Le système de mise en attente des points de crédits lors d'une opération d'achat/de vente de carte assure une gestion cohérente des points de crédits des utilisateurs.
6. Le manuel d'installation est présent sous forme de README.md dans le dépôt git. Evalué selon la reproductibilité et la qualité des instructions pour l'environnement de travail du candidat.
7. La modélisation de la base de données respecte les conventions de nommage de l'ETML et le MCD / MLD / MPD sont présents et détaillés (justification des différents choix).

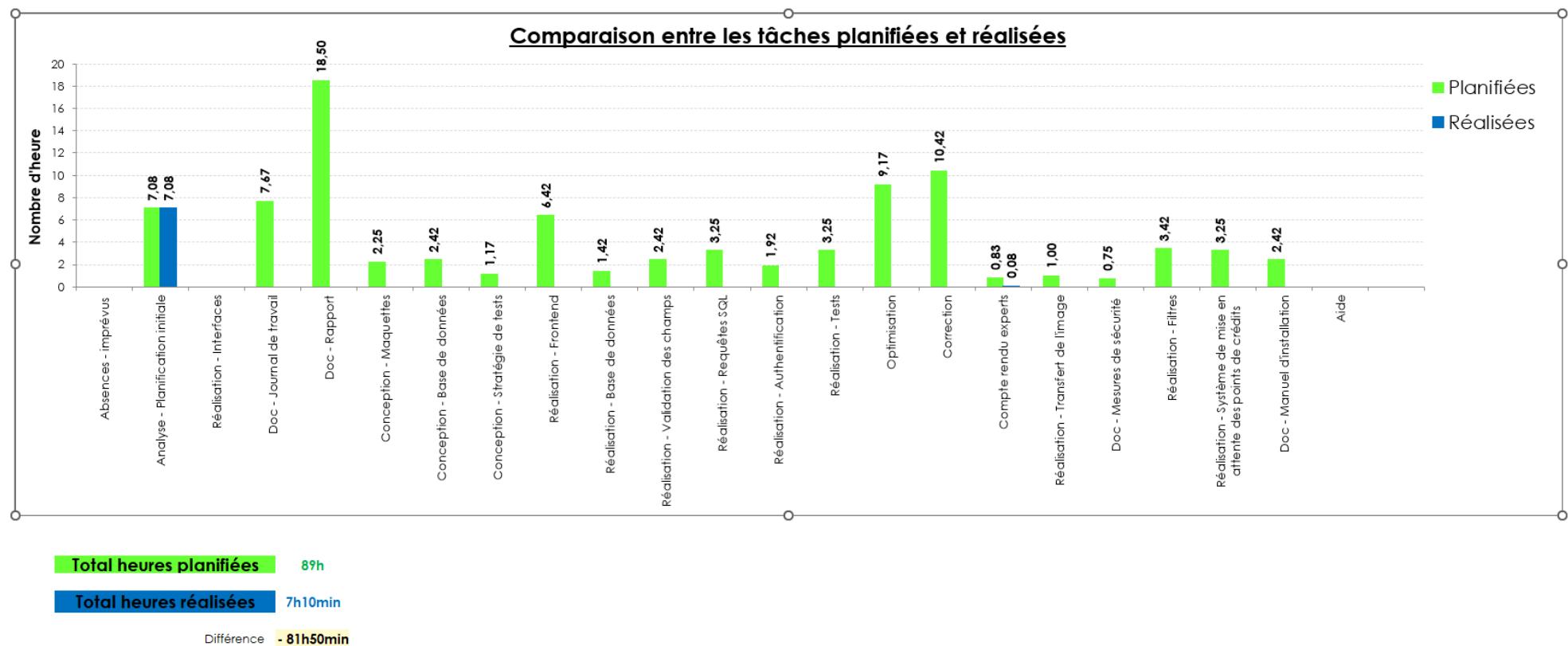
## **Matériel à disposition**

- Un PC standard de l'ETML (Windows 10).
- Visual Studio code avec environnement PHP installé.
- Serveur web local (uWamp ou autre).
- Suite Microsoft Office pour la documentation.
- Un dépôt GIT (GitHub, BitBucket ou autre).

### **1.3 Prérequis**

- Connaissances en programmation PHP et en POO (Modules ICT 403, 404, 226, 120, 326, 411, 133 et 151).
- Connaissances en modélisation et implémentation de base de données relationnelles (Modules ICT 104, 105, 153).

## 1.4 Planification initiale





## 2 Analyse / Conception

### 2.1 Méthodologie de travail

Afin de mener à bien ce projet, la méthodologie Waterfall, également connue sous le nom de méthode en cascade, a été choisie.

Cette approche de gestion de projet se caractérise par sa nature linéaire, fixe et structurée, suivant un processus séquentiel composé d'étapes clairement définies.

Cette méthodologie convient particulièrement pour ce projet dont le cahier des charges est fixe, car elle permet de planifier chaque phase en conséquence sans avoir besoin d'anticiper de changements.

Cette approche permet d'optimiser la conception et la réalisation du projet en s'assurant que chaque étape est bien définie et que les tâches sont effectuées dans l'ordre prévu.

De plus, les 6 étapes illustrées dans la figure ci-dessous offrent une catégorisation claire des différentes tâches à accomplir et permettent une planification judicieuse du projet.

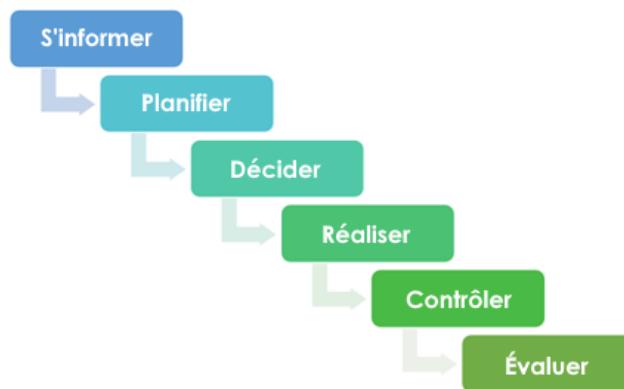


Figure 1 : Schéma de la méthode Waterfall des 6 pas

## 2.2 Environnement de travail

Afin de se rendre compte des difficultés et problématiques qui peuvent survenir lors de la réalisation de ce projet, il est important de redéfinir et de comprendre les fonctionnalités et les limites de chaque outils et concepts qui seront utilisés.

### 2.3 Maquettes

Différentes pages vont composer notre site. Nous verrons ici leur modèle de conception et leurs utilités pour notre projet.

#### Le header

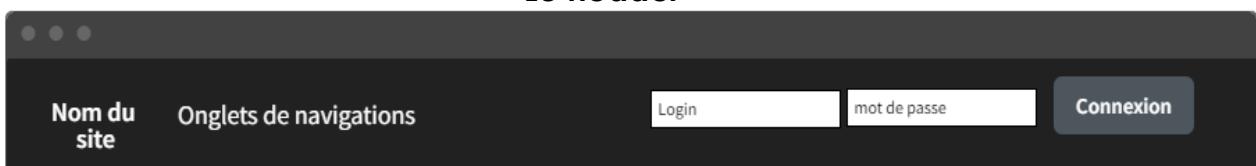


Figure 2 : modèle de conception du header du site

L'affichage du header est prévu pour être évolutif selon le profil de l'utilisateur connecté. Pour chaque cas, différentes informations seront affichées et accessibles en conséquence.

#### Cas no1 : L'utilisateur n'a pas de compte et ne peut pas se connecter

Dans ce cas de figure, l'utilisateur n'aura que la possibilité de créer un compte via les onglets de navigation ou de contacter un admin via les onglets de contact présents dans le footer. Tant qu'il n'aura pas créé de compte et ne sera pas connecté, il ne pourra effectuer aucune action en lien avec le contenu de l'application.

#### Cas no2 : L'utilisateur dispose d'un compte utilisateur et peut se connecter

Dans ce second cas de figure, dès l'instant où l'utilisateur se connecte à son compte en renseignant son login et son mot de passe, le formulaire de connexion disparaît pour laisser place à un message de bienvenue ainsi qu'un compteur de crédits pour rappeler à l'utilisateur l'état de son compte de crédits.

Les onglets de navigation lui permettent d'accéder à différentes pages :

- Mon profil
- Ajouter une carte
- Panier
- Accueil

### Cas no3 : L'utilisateur dispose d'un compte admin

Dans le cas où l'utilisateur est un administrateur, il a les mêmes possibilités qu'un utilisateur à la différence qu'il dispose de tous les droits.

#### Le footer



Figure 3 : modèle de conception du footer du site

L'affichage du footer n'est pas évolutif. Il contient simplement différentes possibilités d'entrer en contact avec l'administrateur de l'application via mail ou les réseaux sociaux.

#### La page d'accueil

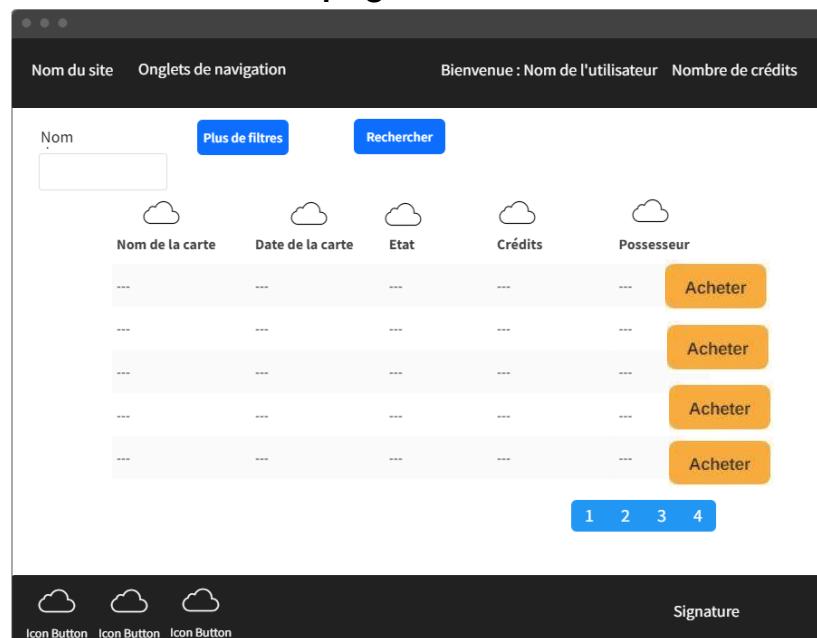


Figure 4: page d'accueil du site

La page d'accueil évolue en fonction du rôle de l'utilisateur connecté. Il est à tout moment possible de revenir à cette page en cliquant sur l'onglet accueil présent dans le header.

### Cas no1 : L'utilisateur n'a pas de compte et ne peut pas se connecter

La page d'accueil affiche uniquement un message invitant l'utilisateur à créer un compte. Aucune autre interaction n'est possible et aucune information relative aux cartes de collection n'est accessible.

### Cas no2 : L'utilisateur dispose d'un compte utilisateur ou administrateur et est connecté

La page d'accueil affiche un tableau contenant toutes les cartes mises en vente sur le site ainsi que certaines de leurs informations. Il est possible de les trier grâce à un bouton au-dessus des colonnes ainsi que de les filtrer. Un filtre sur le nom est présent de base et il est possible de cliquer sur un bouton plus de filters pour en afficher davantage et affiner la recherche. Ce bouton fait à nouveau disparaître les filtres lorsqu'un nouveau clic se produit. La recherche par filtre s'effectue lorsque le bouton rechercher est cliqué.

L'utilisateur a également la possibilité d'afficher la page contenant toutes les informations d'une carte en cliquant sur le bouton *détails* dans les options du tableau. Il peut aussi ajouter une carte à son panier en cliquant sur le bouton *acheter* dans les options du tableau.

### Page de création d'utilisateur

The screenshot shows a web page titled "Page de création d'utilisateur". At the top, there is a navigation bar with tabs for "Nom du site" and "Onglets de navigation", and buttons for "Connexion" and "Connexion". Below the navigation bar, there is a form for creating a new user account. The form includes fields for "Login", "Prénom", "Nom", "Adresse", "E-mail", and "Mot de passe". A green "Créer un compte" button is located at the bottom of the form. At the bottom of the page, there is a footer with three "Icon Button"s and a "Signature" field.

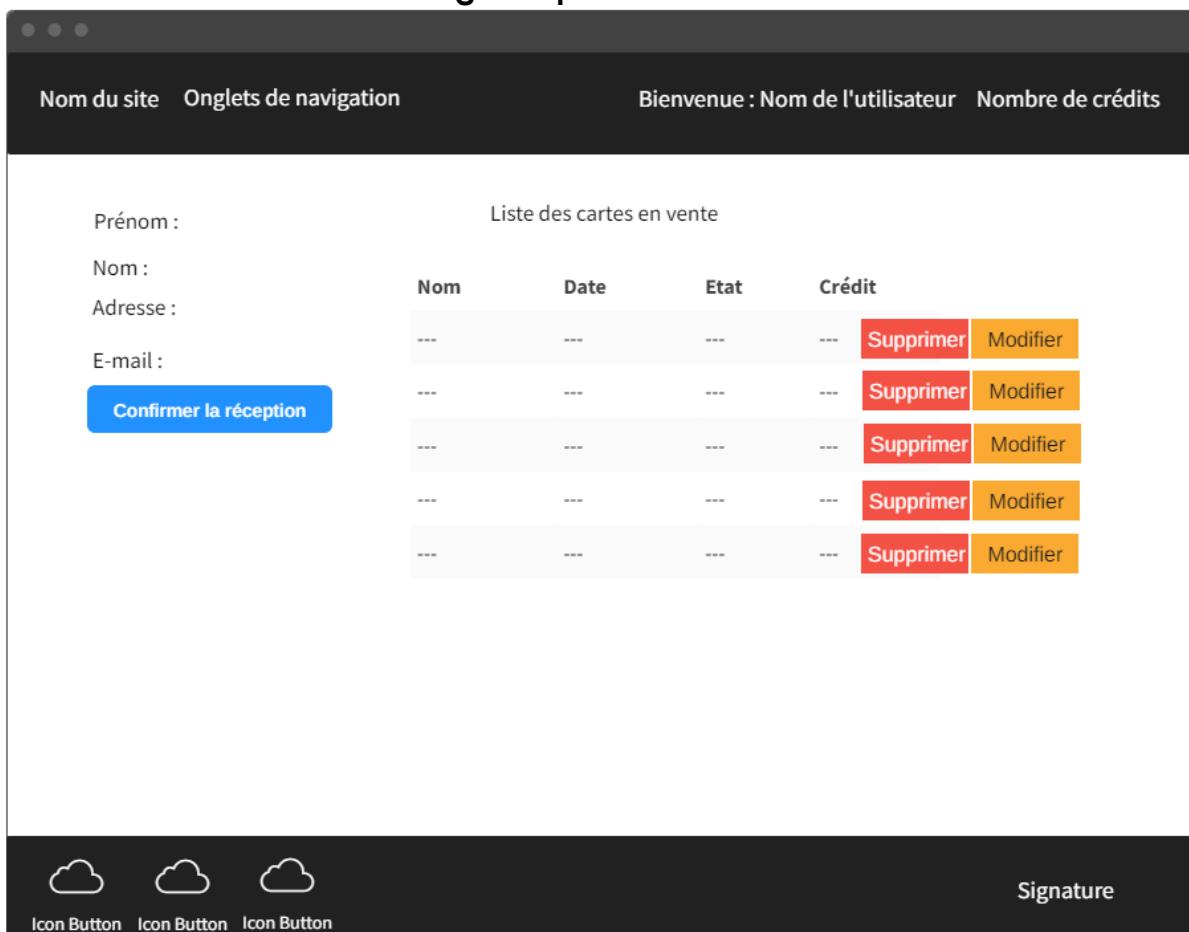
Figure 5 : page de création d'utilisateur

Lorsqu'un utilisateur n'a pas encore de compte, il a la possibilité d'en créer un en cliquant sur l'onglet *créer un compte*. Il doit ensuite renseigner les informations suivantes :

- Login
- Prénom
- Nom
- Adresse
- E-mail
- Mot de passe

Pour que l'inscription soit acceptée, une validation des champs est effectuée dès lors qu'il clique sur le bouton *Créer un compte*. S'il y a des erreurs, elles lui sont indiquées de façon contextuelle, sinon le compte est créé et ses informations sont enregistrées.

### Page de profil utilisateur



The screenshot shows a web application interface for managing user profiles and card sales. At the top, there's a navigation bar with 'Nom du site' and 'Onglets de navigation' on the left, and 'Bienvenue : Nom de l'utilisateur' and 'Nombre de crédits' on the right. Below this, on the left, is a form with fields for 'Prénom', 'Nom', 'Adresse', and 'E-mail', each with a placeholder '---'. A blue button labeled 'Confirmer la réception' is positioned below the email field. On the right, under the heading 'Liste des cartes en vente', is a table with columns 'Nom', 'Date', 'Etat', and 'Crédit'. Each row contains a set of three empty cells followed by two buttons: 'Supprimer' (red) and 'Modifier' (orange). At the bottom, there are three 'Icon Button' icons (clouds) and a 'Signature' input field.

Figure 6 : page de profil de l'utilisateur

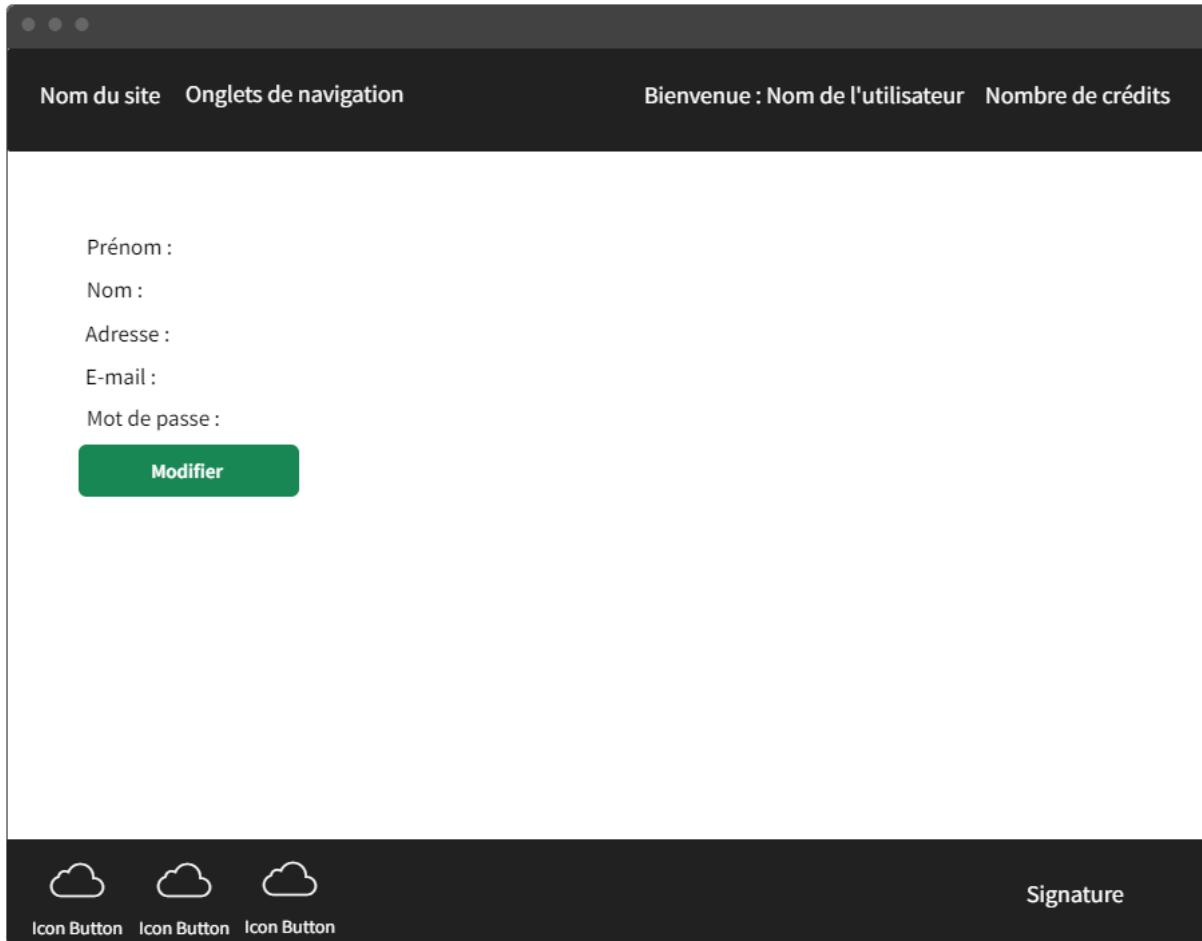
La page de profil d'un utilisateur contient toutes ses informations personnelles. Elles ne sont accessibles qu'à l'utilisateur en question en cliquant sur l'onglet *mon profil* à l'exception de l'administrateur s'il y a nécessité d'intervenir. Toutes

les cartes que l'utilisateur propose à la vente sont également affichées sur un tableau et il a la possibilité de supprimer ou modifier une carte en particulier s'il clique sur le bouton correspondant. Il peut aussi consulter une carte en cliquant sur le bouton *détails* dans les options du tableau. S'il le souhaite, il peut modifier les informations de son profil depuis cette page en cliquant sur le bouton *modifier mon profil*.

Si l'utilisateur a passé une commande sur le site, un bouton *confirmer la réception* à propos de la commande en question apparaîtra sur son profil. Dès lors qu'il clique sur ce bouton et confirme la réception, la transaction est considérée comme terminée.

Dans le cas où l'utilisateur n'a pas passé de commande, un simple message *aucune commande en attente* sera affiché.

### Page de modification d'un utilisateur



The screenshot shows a user profile modification form. At the top, there's a header bar with three dots on the left, followed by 'Nom du site' and 'Onglets de navigation' on the left, and 'Bienvenue : Nom de l'utilisateur' and 'Nombre de crédits' on the right. Below the header is a large input field containing five lines of placeholder text: 'Prénom :', 'Nom :', 'Adresse :', 'E-mail :', and 'Mot de passe :'. To the right of this input field is a green rectangular button with the word 'Modifier' in white. At the bottom of the page, there's a dark footer bar featuring three small cloud icons labeled 'Icon Button' each, and a 'Signature' placeholder area.

Figure 7 : Page de modification d'un utilisateur

Lorsqu'un utilisateur souhaite modifier les informations de son profil, il doit à nouveau renseigner les informations qu'il souhaite modifier. Les valeurs de son profil précédemment enregistrées lui sont retournées de base dans les champs concernés. Dès lors qu'il clique sur le bouton *modifier* une validation des

champs a lieu. S'il y a des erreurs, elles lui sont indiquées de façon contextuelle, sinon les modifications sont correctement enregistrées et son profil est mis à jour.

### Page d'ajout d'une carte

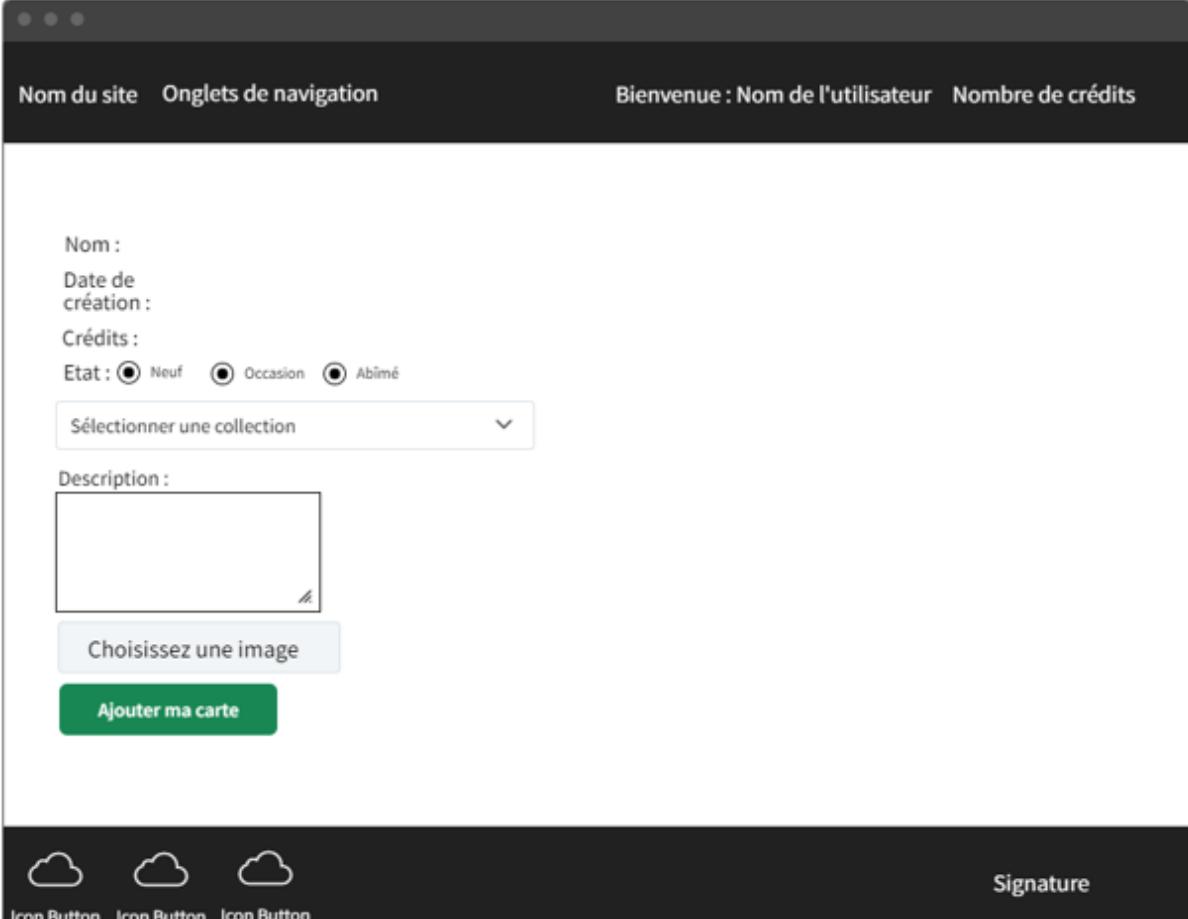


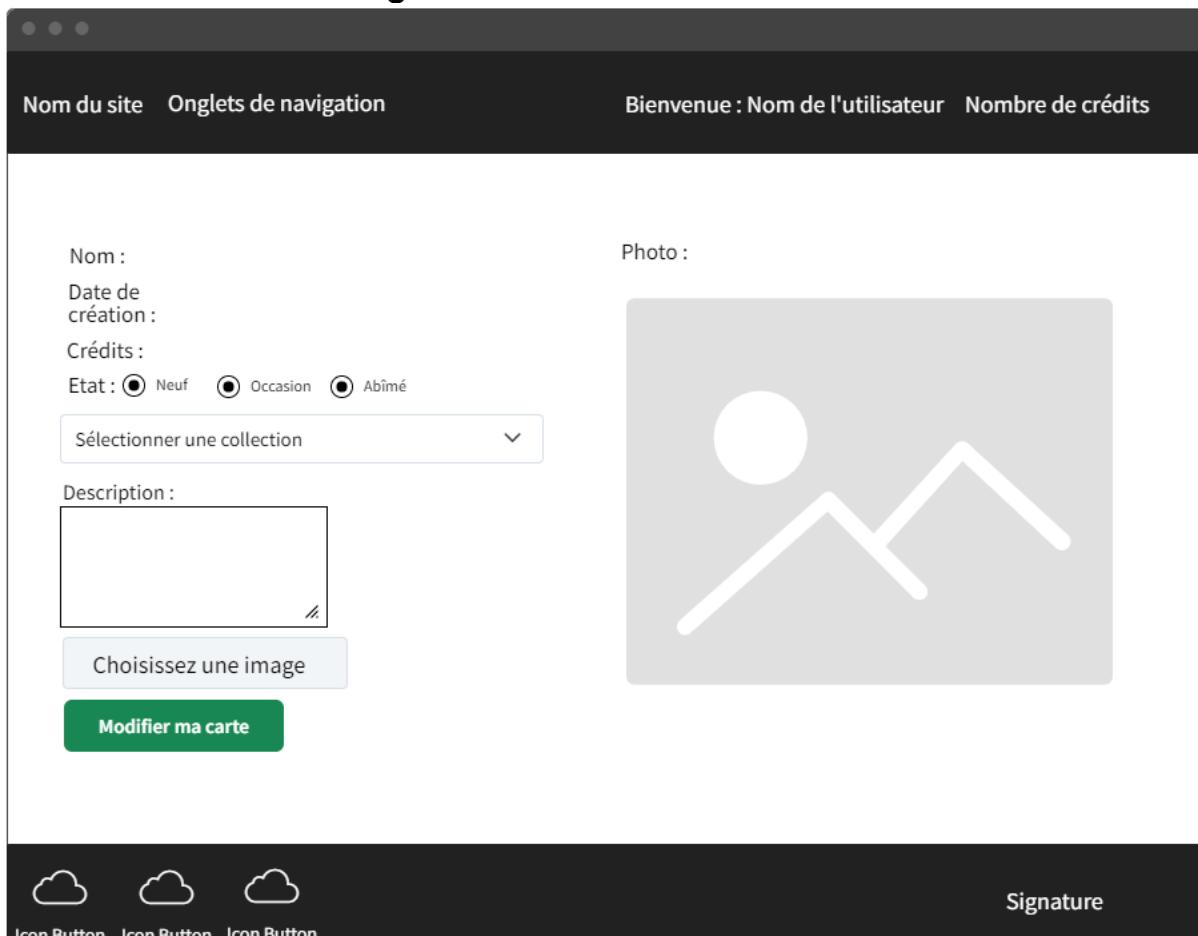
Figure 8 : Page d'ajout d'une carte

Un utilisateur connecté peut en tout temps ajouter une carte à vendre sur le site via l'onglet *Ajouter une carte*. Pour cela, il doit renseigner les informations suivantes :

- Nom
- Date de création
- Valeur en crédits
- Etat
- La collection
- Une description
- Une photo de la carte possédée

Dès lors qu'un clic est effectué sur le bouton *Ajouter une image* une validation des champs contrôle les informations renseignées. S'il y a des erreurs, elles sont affichées de façon contextuelle, sinon les informations sont bien enregistrées et la carte est mise en vente sur le site.

## Page de modification d'une carte



Nom du site   Onglets de navigation

Bienvenue : Nom de l'utilisateur   Nombre de crédits

Nom :  
Date de création :  
Crédits :  
Etat :  Neuf  Occasion  Abîmé

Sélectionner une collection

Description :

**Modifier ma carte**

Photo :

Icon Button   Icon Button   Icon Button

Signature

Figure 9 : Page de modification d'une carte

Un utilisateur connecté peut en tout temps modifier une carte qu'il a déjà mise en vente depuis son profil en cliquant sur le bouton modifier.

Lorsqu'un utilisateur souhaite modifier les informations de l'une de ses cartes, il doit à nouveau renseigner les informations qu'il souhaite modifier. Les valeurs de précédemment enregistrées de sa carte lui sont retournées de base dans les champs concernés. La photo enregistrée précédemment s'affiche également sur cette page. Dès lors qu'il clique sur le bouton *modifier ma carte* une validation des champs a lieu. S'il y a des erreurs, elles lui sont indiquées de façon contextuelle, sinon les modifications sont correctement enregistrées et sa carte est mise à jour.

|

## Page d'affichage d'une carte en particulier

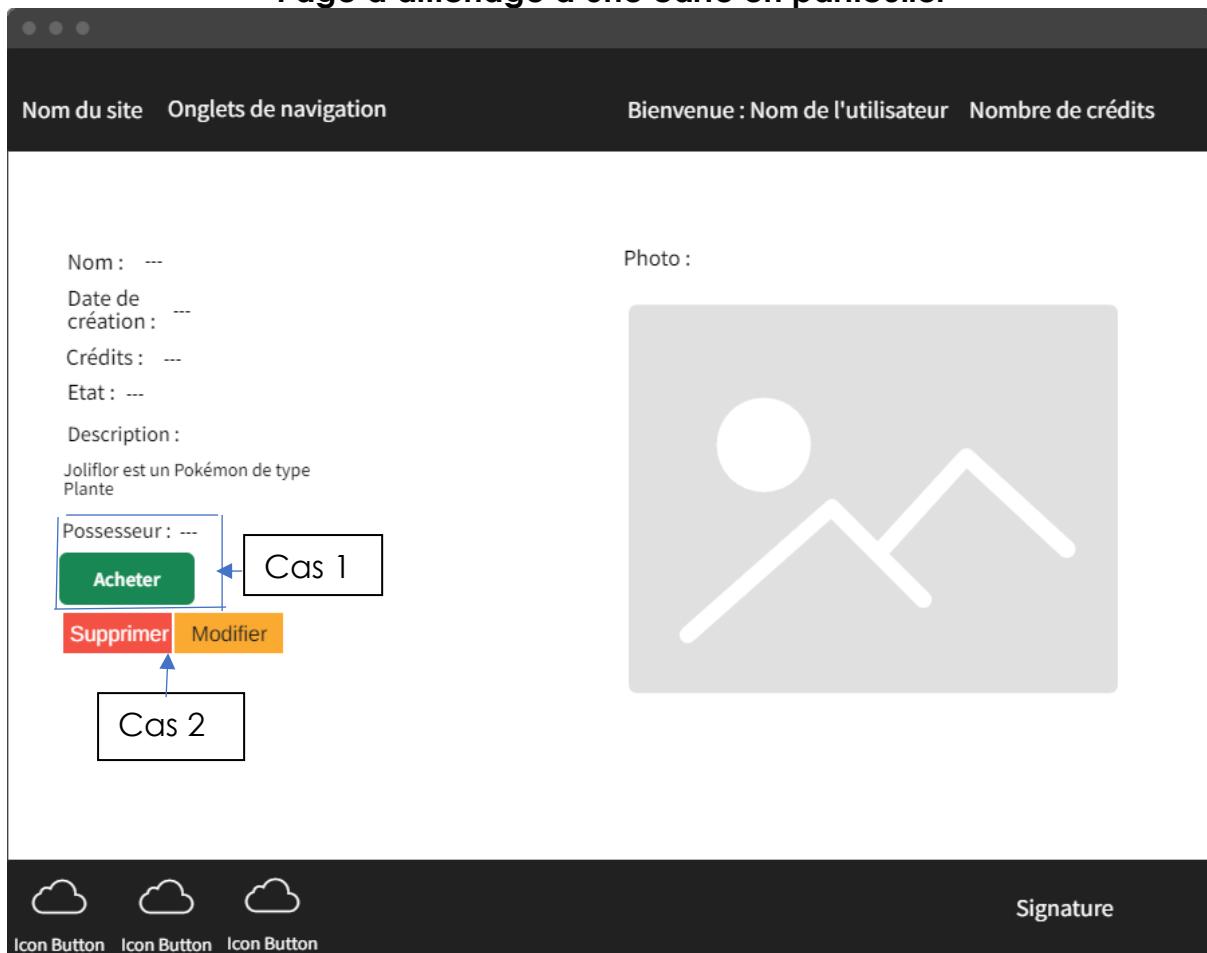


Figure 10 : Page d'affichage d'une carte en particulier

Un utilisateur connecté peut accéder en tout temps aux détails d'une carte en particulier en cliquant sur le nom de celle-ci. Il peut de cette façon consulter toutes les informations la concernant et y compris observer une photo de l'article.

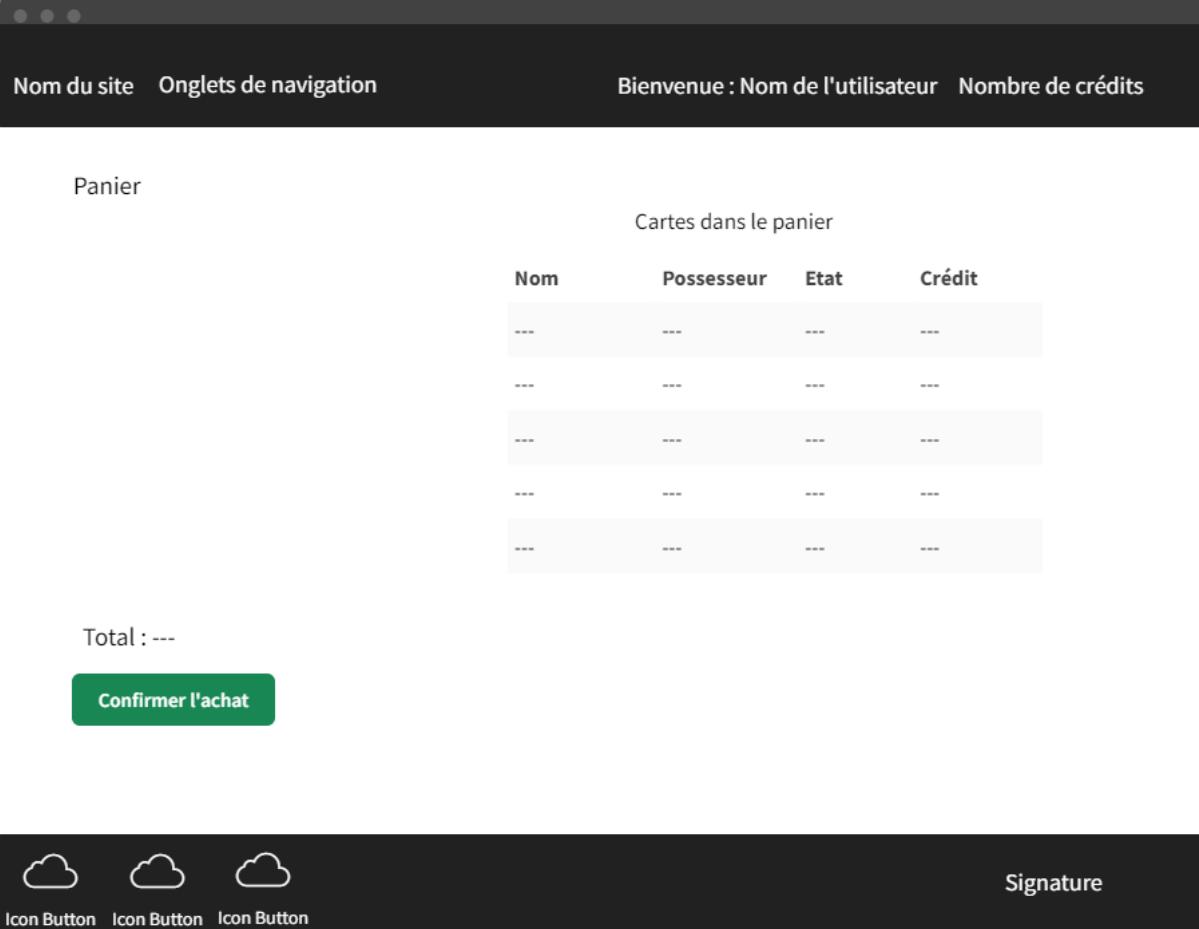
### Cas n°1 : La carte n'appartient pas à l'utilisateur connecté

Dans le cas où l'utilisateur consulte les informations d'une carte qu'il n'a pas lui-même mise en vente, le nom du possesseur de la carte ainsi qu'un bouton Acheter sont visibles sur la page. En cas d'achat, la carte est ajoutée au panier de l'utilisateur.

### Cas n°2 : La carte appartient à l'utilisateur connecté

Dans le cas où l'utilisateur consulte les informations d'une carte qu'il a lui-même mise en vente, le champ Possesseur : ainsi que le bouton Acheter ne sont pas visibles sur la page. En lieu et place un bouton Supprimer permettant de supprimer la carte ainsi qu'un bouton Modifier redirigeant sur la page de modification d'une carte sont affichés sur la page.

## Page de panier de l'utilisateur



The screenshot shows a user interface for managing a shopping cart. At the top, there's a header bar with the text "Nom du site" and "Onglets de navigation" on the left, and "Bienvenue : Nom de l'utilisateur" and "Nombre de crédits" on the right. Below the header, the word "Panier" is displayed. To the right, a section titled "Cartes dans le panier" contains a table with four columns: "Nom", "Possesseur", "Etat", and "Crédit". Each row in the table has three dashed entries ("---"). Below the table, the text "Total : ---" is shown. A green button labeled "Confirmer l'achat" is positioned below the total. At the bottom of the page, there are three "Icon Button" components, each represented by a cloud icon, followed by the text "Signature".

Figure 11 : Page de panier de l'utilisateur

Un utilisateur connecté peut accéder en tout temps à son panier via l'onglet *Mon panier*. Sur cette page, il a la possibilité de consulter tous les articles qu'il a ajouté à son panier et de confirmer sa commande. Dès lors qu'il clique sur le bouton *Confirmer l'achat* la transaction débute.

Les crédits nécessaires à l'opération sont déduits du compte de l'acheteur et sont temporairement mis en attente. Une fois que le vendeur a envoyé les articles, l'acheteur confirme la bonne réception de ceux-ci depuis son profil. Les crédits mis en attente sont alors ajoutés au compte du vendeur.

## 2.4 Base de données

### 2.4.1 MCD

Pour fonctionner, notre application a besoin que la base de données puisse stocker différentes informations.

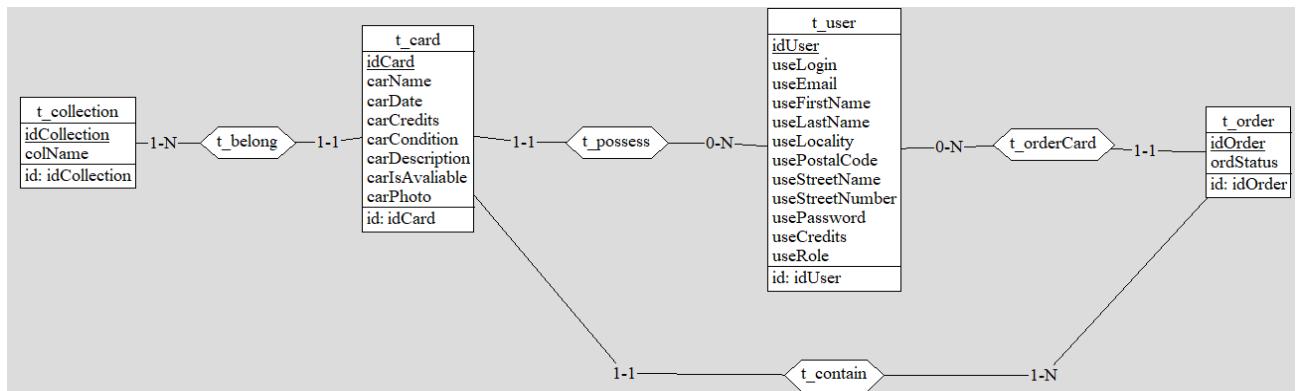


Figure 12 : MCD de la base de données

### 2.4.2 Entités

<b>t_user</b>	
<u><b>idUser</b></u>	
<b>useLogin</b>	
<b>useEmail</b>	
<b>useFirstName</b>	
<b>useLastName</b>	
<b>useLocality</b>	
<b>usePostalCode</b>	
<b>useStreetName</b>	
<b>useStreetNumber</b>	
<b>usePassword</b>	
<b>useCredits</b>	
<b>useRole</b>	
<b>id: idUser</b>	

Figure 13 : entité **t\_user** du MCD

L'entité **t\_user** contient toutes les informations d'un utilisateur donné. Elle permet par exemple de définir si l'utilisateur est connecté ou non et si oui de quel type d'utilisateur il s'agit. Elle est composée des propriétés suivantes :

- ***idUser*** : Identifiant unique de l'utilisateur.
- ***useLogin*** : Login de l'utilisateur.
- ***useEmail*** : Email de l'utilisateur.
- ***useFirstName*** : Prénom de l'utilisateur.
- ***useLastName*** : Nom de famille de l'utilisateur.
- ***useLocality*** : Localité dans laquelle vit l'utilisateur.
- ***usePostalCode*** : Code postal de l'endroit où vit l'utilisateur.
- ***useStreetName*** : Nom de la rue où vit l'utilisateur.
- ***useStreetNumber*** : Numéro de la rue où vit l'utilisateur.
- ***usePassword*** : Mot de passe de l'utilisateur pour se connecter sur le site.
- ***useCredits*** : Valeur en crédits sur le compte de l'utilisateur
- ***useRole*** : Rôle de l'utilisateur sur le site (profil utilisateur/administrateur).

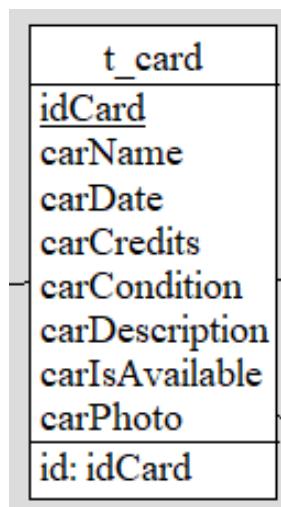


Figure 14 : entité **t\_card** du MCD

L'entité **t\_card** contient toutes les informations d'une carte donnée. Elle permet par exemple de définir la valeur en crédits, l'état ou le nom d'une carte. Elle est composée des propriétés suivantes :

- ***idCard*** : Identifiant unique de la carte.
- ***carName*** : Nom de la carte.
- ***carDate*** : Année de création de la carte.
- ***carCredits*** : Valeur en crédits de la carte.
- ***carCondition*** : Etat de la carte.
- ***carDescription*** : Description de la carte.
- ***carIsAvailable*** : Indicateur permettant de savoir si la carte est disponible à la vente ou non.
- ***carPhoto*** : Photo de la carte.

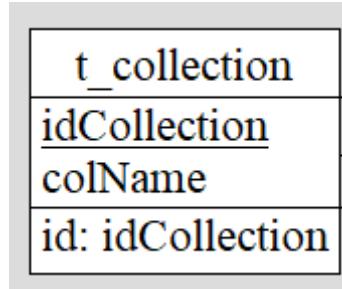


Figure 15 : entité **t\_collection** du MCD

L'entité **t\_collection** contient les informations relatives à la collection d'une carte. Elle est composée des propriétés suivantes :

- ***idCollection*** : Identifiant unique de la collection.
- ***colName*** : Nom de la collection.

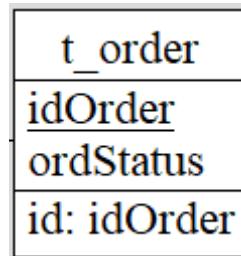


Figure 16 : entité **t\_order**

L'entité **t\_order** contient les informations relatives aux commandes réalisées par les utilisateurs et permet de gérer les crédits en attente lorsqu'une commande est passée. Elle est composée des propriétés suivantes :

- ***idOrder*** : Identifiant unique de la commande.
- ***ordStatus*** : Statut de la commande (en cours/terminée).

### 2.4.3 Cardinalités

Les cardinalités permettent d'établir le type de relation entre deux entités via une association.

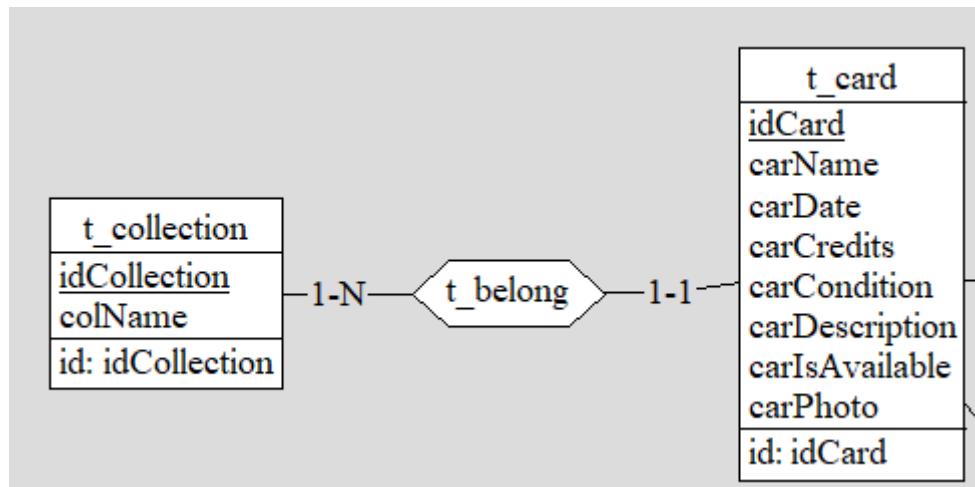


Figure 17 : Association **t\_belong** entre les entités **t\_collection** et **t\_card**

L'association **t\_belong** lie les entités **t\_card** et **t\_collection**. La cardinalité **1-1** indique qu'une carte ne peut appartenir qu'à une et une seule collection. En revanche, la cardinalité **1-N** indique qu'une ou plusieurs cartes peuvent appartenir à la même collection.

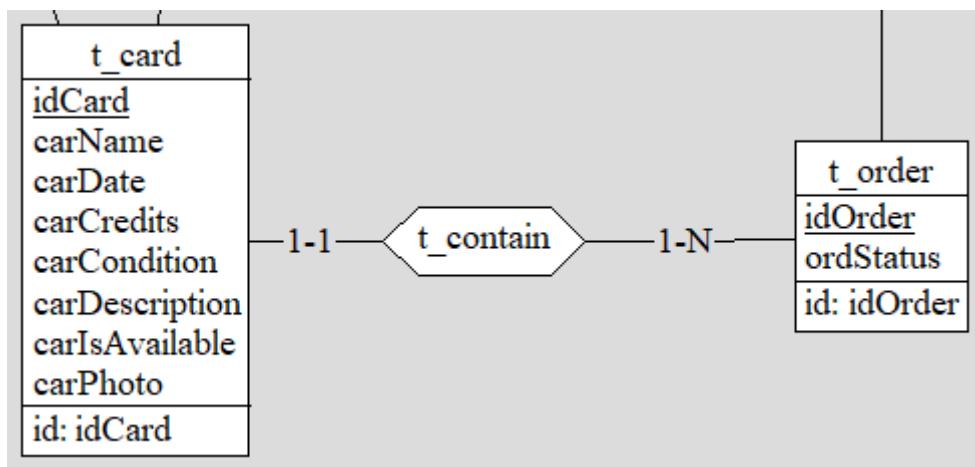
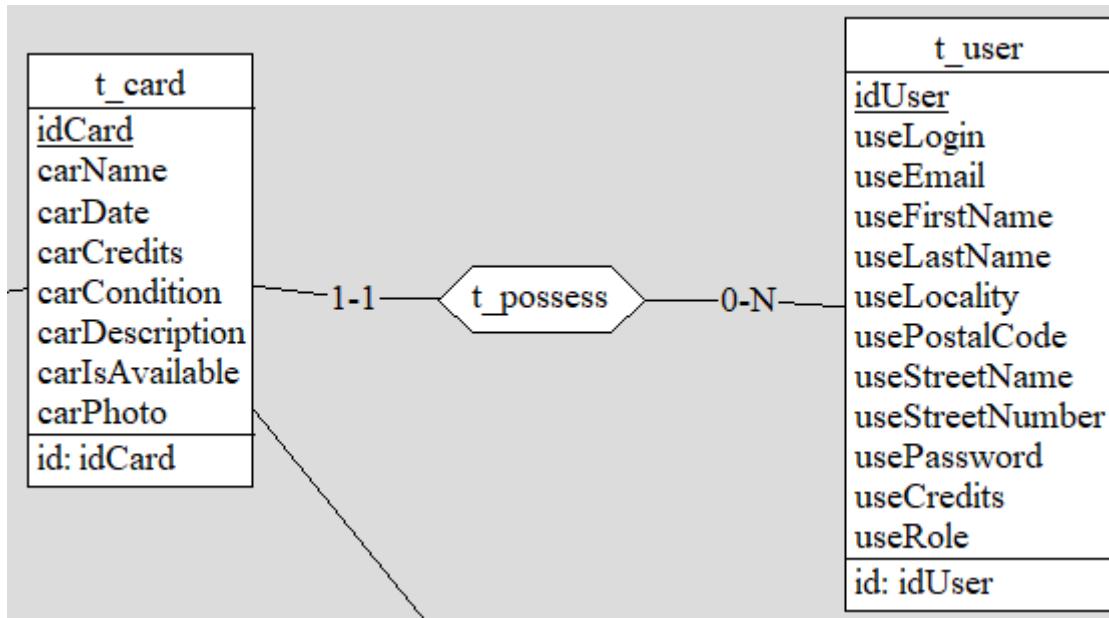
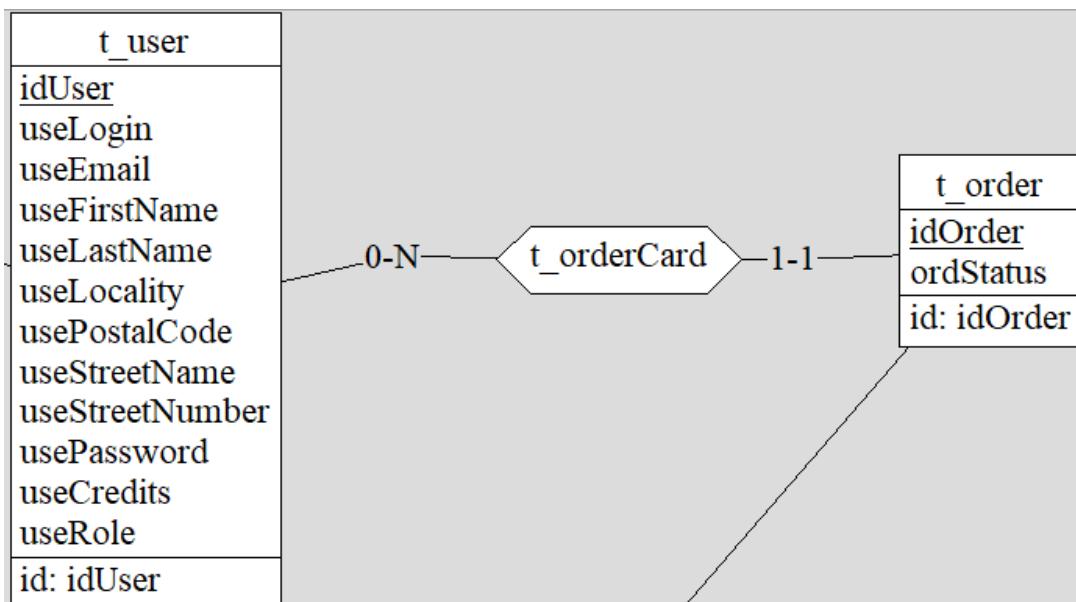


Figure 18 : Association **t\_contain** entre les entités **t\_card** et **t\_order**

L'association **t\_contain** lie les entités **t\_card** et **t\_order**. La cardinalité **1-1** indique que chaque carte, étant unique, ne peut être contenue que dans une et une seule commande à la fois. La cardinalité **1-N** indique quant à elle qu'une commande doit contenir au moins une carte mais peut également contenir plusieurs cartes différentes.

Figure 19 : association *t\_possess* entre les entités *t\_card* et *t\_user*

L'association **t\_possess** lie les entités **t\_card** et **t\_user**. La cardinalité **1-1** indique qu'une carte, étant un objet unique, ne peut être possédée qu'une et une seule fois par un utilisateur. La cardinalité **0-N** quant à elle indique qu'un utilisateur n'a pas l'obligation d'échanger une carte mais peut en échanger autant qu'il le souhaite.

Figure 20 : association *t\_orderCard* entre les entités *t\_user* et *t\_order*

L'association **t\_orderCard** lie les entités **t\_user** et **t\_order**. La cardinalité **0-N** indique qu'un utilisateur n'a pas l'obligation de passer une commande mais qu'il a la possibilité d'en passer autant qu'il le souhaite. La cardinalité **1-1** indique quant à elle qu'une commande, étant unique, ne peut être passée que par un et un seul utilisateur.

## 2.4.4 MLD

Selon notre **MCD**, voici le résultat du **MLD**. Des contraintes référentielles ont été créées et l'intégrité de nos données entre les différentes tables est maintenant assurée.

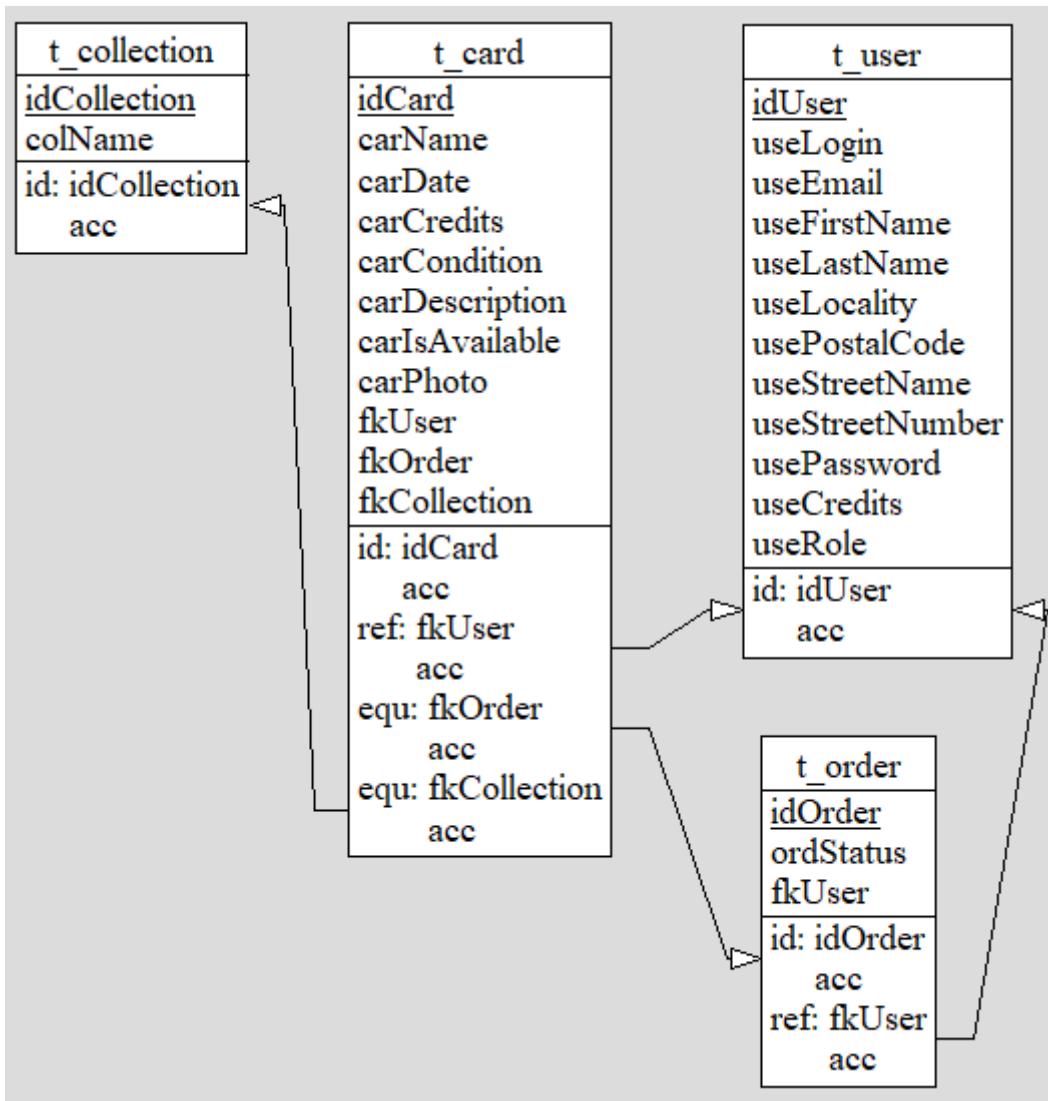


Figure 21 : MLD de la base de données

Nous pouvons constater que la table **t\_card** récupère quant à elle les clés étrangères des tables **t\_order**, **t\_user** et **t\_collection**. Cela nous permet de lier une carte à une commande lorsqu'elle est passée, à un utilisateur ainsi qu'à une collection.

Enfin, la table **t\_order** récupère la clé étrangère de la table **t\_user**, ce qui nous permet de lier une commande à un utilisateur. Cela nous sera utile lors d'une réalisation de commande puisque même si elle contient plusieurs cartes appartenant au même utilisateur, il ne s'agira que d'une seule et même commande.

## 2.5 Stratégie de test

Afin d'effectuer des vérifications testant le fonctionnement de nos différentes fonctionnalités, il est nécessaire d'établir une stratégie qui permettra de réaliser ces tests selon différents scénarios en anticipant le résultat escompté.

Nom du test	Scénario	Résultat attendu
Ajout d'une carte avec succès.	<p>Un utilisateur connecté clique sur l'onglet <i>Ajouter une carte</i>. Il renseigne les informations demandées et télécharge une image au format JPG.</p> <p>Il clique ensuite sur le bouton <i>Ajouter une carte</i> lorsqu'il est satisfait des informations renseignées.</p>	<p>La validation des champs s'effectue sans rencontrer d'erreurs et un message affiche : <i>Votre carte a bien été ajoutée.</i></p> <p>L'utilisateur peut consulter sa carte en cliquant en parcourant le tableau des cartes mises en vente.</p>
Ajout d'une carte sans respecter le format de l'image.	<p>Un utilisateur connecté clique sur l'onglet <i>Ajouter une carte</i>. Il renseigne les informations demandées, mais envoie un fichier au format PNG.</p> <p>Il clique ensuite sur le bouton <i>Ajouter une carte</i> lorsqu'il est satisfait des informations renseignées.</p>	<p>La validation des champs s'effectue et repère que le fichier envoyé n'est pas au format attendu.</p> <p>Une erreur contextuelle affiche : <i>Merci de n'envoyer que des images au format JPG.</i></p> <p>Les informations renseignées dans les champs qui n'ont pas déclenché d'erreur sont retournés à l'utilisateur.</p>
Accès à une page non autorisée.	Un utilisateur non connecté essaye d'accéder à la page <i>Ajouter une carte</i> via l'URL sans qu'il ne dispose des droits pour y accéder.	Une page d'erreur 403 s'affiche et empêche l'utilisateur d'accéder à la page web. Celle-ci l'invite à rejoindre l'accueil ou à se connecter.
Tri des cartes affichées sur la page d'accueil.	Un utilisateur connecté souhaite trier les cartes par ordre alphabétique à partir du nom. Dans ce but, il clique sur l'icône au-dessus de la colonne permettant de réaliser le tri.	Les cartes s'affichent correctement par ordre alphabétique croissant ou décroissant après le clique sur l'icône.
Filtrage des cartes affichées sur la page d'accueil selon leur état.	Un utilisateur souhaite filtrer les cartes affichées sur la page d'accueil selon leur état. Il clique sur le bouton	Seules les cartes renseignées comme étant dans un état Neuf s'affichent dans le tableau

	<p>plus de filtres et indique un état Neuf sur le filtre concerné qui est correctement apparu. Il clique ensuite sur le bouton Rechercher.</p>	des cartes en vente de la page d'accueil.
Crédits mis en attente lors d'une transaction	<p>Un utilisateur connecté clique sur le bouton Acheter de la carte qui l'intéresse depuis le tableau de la page d'accueil. Après avoir cliqué sur l'onglet Mon panier il constate que la carte a correctement été ajoutée à son panier. Il confirme l'achat en cliquant sur le bouton <i>Confirmer la commande</i>.</p>	Les crédits sont bien débités du compte de l'acheteur mais ne sont pas crédités sur le compte du vendeur avant que l'acheteur ne confirme la réception sur son profil.
Réussite de l'utilisation du manuel d'installation	<p>Un collègue ou un enseignant disponible télécharge l'application depuis le dépôt git et consulte le manuel d'installation présent dans le dossier sous forme de README.md. Il essaie ensuite de le reproduire méticuleusement en l'adaptant à son environnement si nécessaire.</p>	La personne qui a suivi les instructions du manuel d'installation a réussi à lancer l'application dans son propre environnement.

## 2.5.1 Logiciels et outils supplémentaires

- **Docker**, plateforme permettant de lancer des applications dans des conteneurs. C'est grâce à cet outil que seront stockés la base de données, le site web ainsi que PHPMyAdmin.
- **PHPMyAdmin**, application Web de gestion pour les systèmes de base de données MySQL. Grâce à cet outil, il sera facile de visualiser la base de données et d'y ajouter des données manuellement si besoin.
- **ChatGPT**, modèle de langage. Consultation pour la compréhension d'erreurs ou la recherche d'informations.
- **DBMain**, logiciel pour la conception du MCD et du MLD de la base de données.

## 2.6 Risques techniques

- Système de transaction de points de crédits :

N'ayant jamais réalisé ce type de fonctionnalité, il est difficile de prévoir sa difficulté exacte et le temps nécessaire pour la réaliser.

- Requête **SQL** permettant d'utiliser les filtres :

La requête permettant d'utiliser les filtres prévoit d'être construite dynamiquement selon les choix de l'utilisateur. J'ai déjà rencontré des difficultés à la réalisation de ce type de requête, notamment pour les sécuriser.

## 3 Réalisation

### INTRO + VERSIONS OUTILS

#### 3.1 Choix de la structure du projet

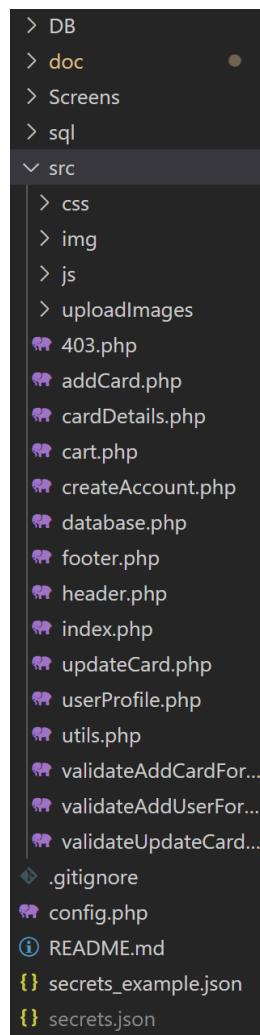


Figure 22 : Structure du projet

Pour ce TPI, il a été décidé au préalable avec le chef de projet de ne pas travailler avec une architecture **MVC**. Le **framework Laravel** n'étant pas enseigné durant la formation et aucune architecture **MVC** satisfaisante à reproduire n'ayant été trouvée durant le module P\_Appro2, nous avons estimé qu'il était plus prudent de travailler avec une architecture moins complexe tout en essayant de se rapprocher le plus possible d'une architecture professionnelle.

## 3.2 Base de données

### 3.2.1 Importation du fichier .SQL

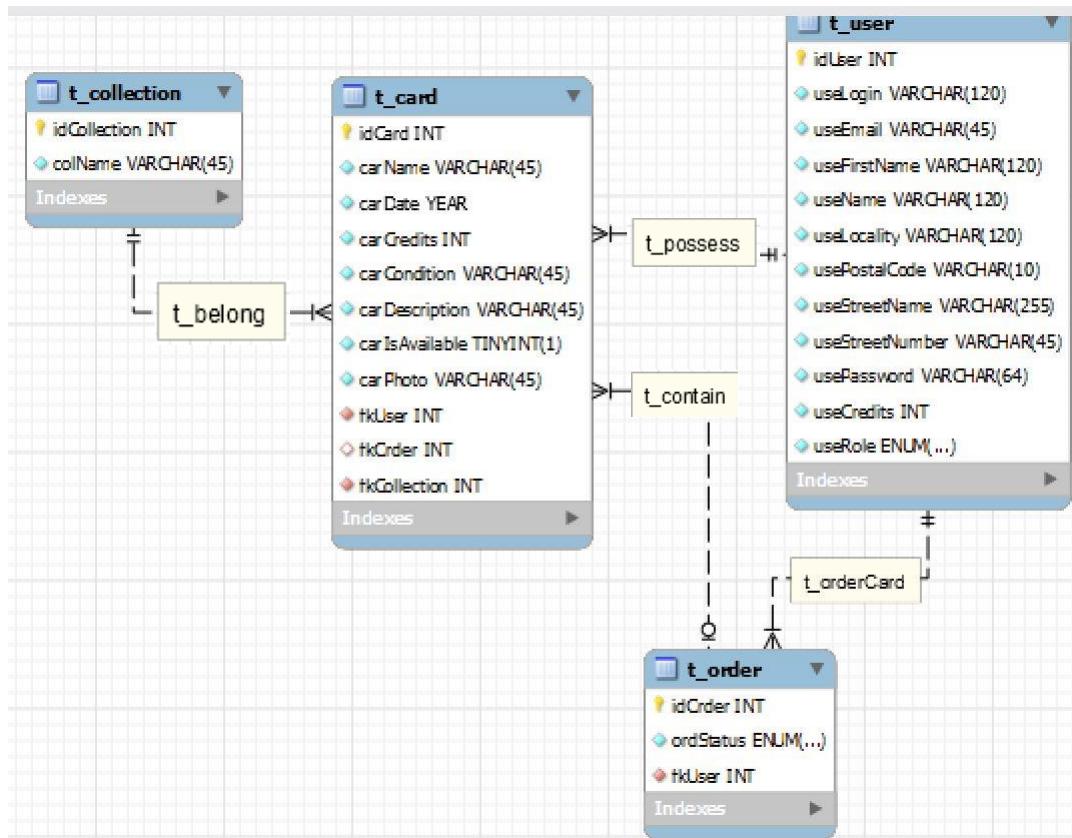


Figure 23 : MLD sur MySQL Workbench

Pour la génération et l'importation du fichier **SQL**, le programme **MySQL Workbench** a été préféré à **DB-MAIN** car il propose une définition beaucoup plus fine des types pour les attributs ce qui permet de générer un fichier **SQL** déjà optimisé pour notre application.

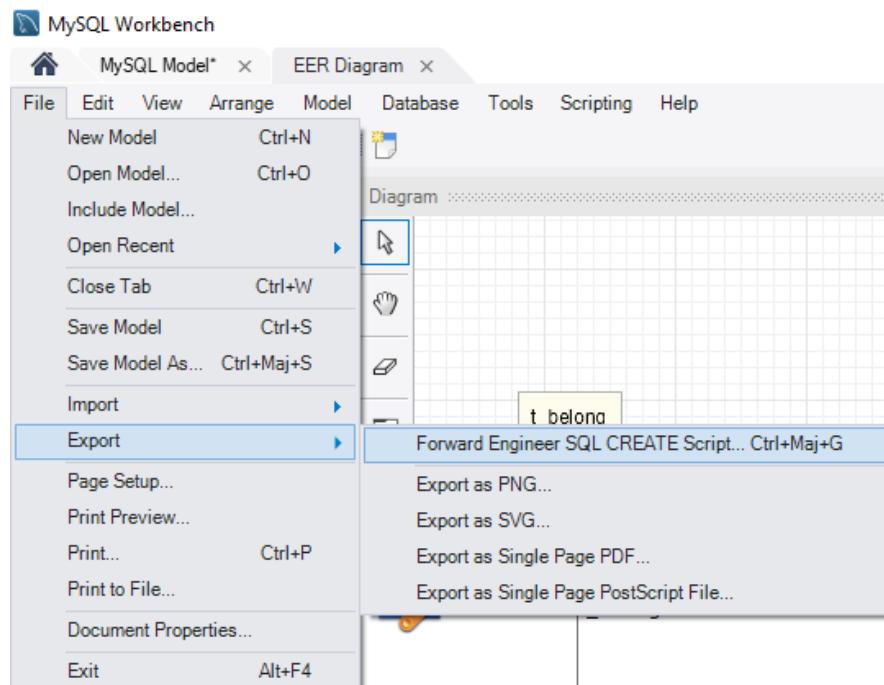


Figure 24 : Export du code .SQL depuis le MLD

Nous exportons le code correspondant à notre **MLD** dans un fichier **SQL** avant de l'importer dans **PHPMyAdmin**.



Figure 25 : Importation du fichier .SQL dans phpMyAdmin

Une fois notre fichier **SQL** généré et adapté à nos besoins nous l'importons sur **PHPMyAdmin**.

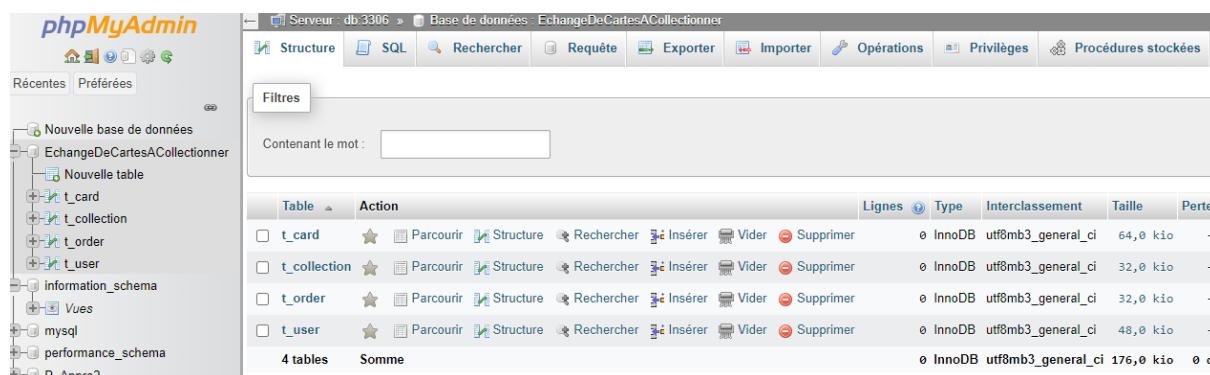


Figure 26 : Interface de la base de données sur phpMyAdmin

Nous pouvons constater que la base de données a correctement été importée sur **phpMyAdmin**.

### 3.2.2 MPD

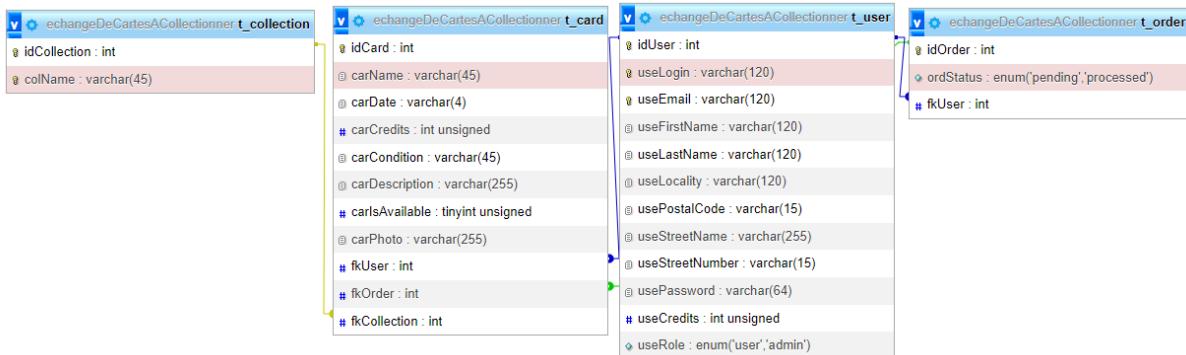


Figure 27 : Vue du MPD de l'application sur phpMyAdmin

A présent que notre base de données a été importée sur **phpMyAdmin**, nous pouvons avoir une vue d'ensemble sur la logique de notre base de données via notre **MPD**.

### 3.2.3 Tables et type des attributs

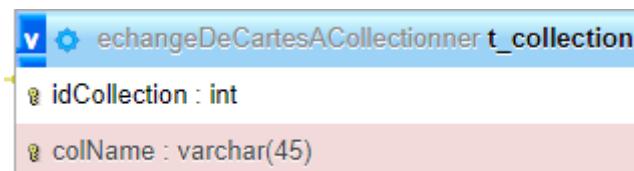


Figure 28 : table *t\_collection* du MPD

Nom	Type	Attributs	Null	Unique	Valeur Par défaut	Extra
idCollection	int	-	Non	Oui	Aucun(e)	Auto-Increment
colName	varchar(45)	-	Non	Oui	Aucun(e)	-

Dans la table **t\_collection** :

- **idCollection**, en tant que clé primaire, est en **auto-increment**.
- **ColName** est unique car il n'est pas possible d'enregistrer plusieurs fois la même collection.

v echangeDeCartesACollectionner t_card	
#	idCard : int
	carName : varchar(45)
	carDate : varchar(4)
#	carCredits : int unsigned
	carCondition : varchar(45)
	carDescription : varchar(255)
#	carlsAvailable : tinyint unsigned
	carPhoto : varchar(255)
#	fkUser : int
#	fkOrder : int
#	fkCollection : int

Figure 29 : table t\_card du MPD

Nom	Type	Attributs	Null	Unique	Valeur Par défaut	Extra
idCard	int	-	Non	Oui	Aucun(e)	Auto-Increment
carName	varchar(45)	-	Non	Non	Aucun(e)	-
carDate	varchar(4)	-	Non	Non	Aucun(e)	-
carCredits	int	unsigned	Non	Non	Aucun(e)	-
carCondition	varchar(45)	-	Non	Non	Aucun(e)	-
carDescription	varchar(255)	-	Non	Non	Aucun(e)	-
carlsAvailable	tinyint	unsigned	Non	Non	1	-
carPhoto	varchar(255)	-	Non	Non	Aucun(e)	-
fkUser	int	-	Non	Non	Aucun(e)	-
fkOrder	int	-	Oui	Non	NULL	-
fkCollection	int	-	Non	Non	Aucun(e)	-

Dans la table **t\_card** :

- **idCard**, en tant que clé primaire, est en **auto-increment**.
- **CarDate** est de type **varchar(4)** car elle représente l'année de création de la carte et doit donc représenter une valeur réaliste.
- Les attributs de **carCredits** sont **unsigned** car le montant de crédits ne peut pas être négatif.
- **carlsAvailable** est de type **tinyint** car il nous faut uniquement définir un état disponible ou indisponible, la valeur peut passer de 0 à 1 mais dès sa création, la carte a une valeur par défaut à 1 car elle est disponible sur le marché tant que personne ne l'a achetée.

- **fkOrder** peut être **NULL** car tant qu'une carte n'a pas été achetée, elle n'appartient à aucune commande.

v echangeDeCartesACollectionner t_user	
idUser : int	
useLogin : varchar(120)	
useEmail : varchar(120)	
useFirstName : varchar(120)	
useLastName : varchar(120)	
useLocality : varchar(120)	
usePostalCode : varchar(15)	
useStreetName : varchar(255)	
useStreetNumber : varchar(15)	
usePassword : varchar(64)	
# useCredits : int unsigned	
useRole : enum('user','admin')	

Figure 30 : table t\_user du MPD

Nom	Type	Attributs	Null	Unique	Valeur Par défaut	Extra
idUser	int	-	Non	Oui	Aucun(e)	Auto-Increment
useLogin	varchar(120)	-	Non	Oui	Aucun(e)	-
useEmail	varchar(120)	-	Non	Oui	Aucun(e)	-
useFirstName	varchar(120)	-	Non	Non	Aucun(e)	-
useLastName	varchar(120)	-	Non	Non	Aucun(e)	-
useLocality	varchar(120)	-	Non	Non	Aucun(e)	-
usePostalCode	varchar(15)	-	Non	Non	Aucun(e)	-
useStreetName	varchar(255)	-	Non	Non	Aucun(e)	-
useStreetNumber	varchar(15)	-	Non	Non	Aucun(e)	-
usePassword	varchar(64)	-	Non	Non	Aucun(e)	-
useCredits	int	unsigned	Non	Non	Aucun(e)	-
useRole	enum('user', 'admin')	-	Non	Non	user	-

Dans la table **t\_user** :

- **idUser**, en tant que clé primaire, est en **auto-increment**.
- **useLogin** est unique car un utilisateur peut utiliser la valeur de cet attribut pour se connecter à son compte via le login.
- **useEmail** est unique car un utilisateur peut utiliser la valeur de cet attribut pour se connecter à son compte via le login.
- **usePassword** est de type varchar(64) car le mot de passe sera hashé.
- Les attributs de **useCredits** sont **unsigned** car la valeur des crédits ne peut pas être négatifs.
- **useRole** est de type **enum('user', 'admin')**, ce qui nous permet de définir le rôle de l'utilisateur lors de la création du compte. Le rôle admin n'était définissable que par l'administrateur lui-même via l'interface de **phpMyAdmin**, la valeur par défaut de cet attribut est **user**.

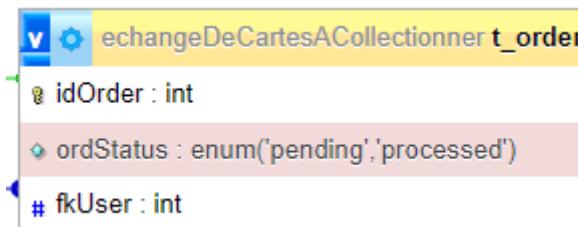


Figure 31 : table *t\_order* du MPD

Nom	Type	Attributs	Null	Unique	Valeur Par défaut	Extra
idOrder	int	-	Non	Oui	Aucun(e)	Auto-Increment
ordStatus	enum('pending', 'processed')	-	Non	Non	Aucun(e)	-
fkUser	int	-	Non	Non	Aucun(e)	-

Dans la table **t\_order** :

- **IdOrder**, en tant que clé primaire, est en **auto-increment**.
- **ordStatus** est de type **enum('pending', 'processed')** ce qui nous permet de gérer l'état de la commande. Tant qu'un utilisateur n'achète pas de cartes, la commande n'existe pas. Dès qu'il achète une carte, le statut passe à **pending**. Et lorsque l'acheteur confirme la réception, le statut passe à **processed**.

Nous avons volontairement prévu des champs pouvant accueillir des longues chaînes de caractères en imaginant une éventuelle internationalisation.

### 3.3 Connexion à la base de données

```

class Database
{
    private static $instance = null;
    private $connector;

    private function __construct()
    {
        include(__DIR__ . "/utils.php");
        $configs = include(__DIR__ . "/../config.php");
        try {
            $this->connector = new PDO(
                $configs['dns'],
                $configs['user'],
                getPassword()
            );
        } catch (PDOException $e) {
            die('Erreur : ' . $e->getMessage());
        }
    }

    public static function getInstance()
    {
        if (self::$instance == null) {
            self::$instance = new self();
        }

        return self::$instance;
    }
}

```

Figure 32 : classe Database permettant la connexion à la base de données en utilisant le pattern Singleton

Toutes les fonctions qui effectuent des opérations en base de données grâce aux requêtes SQL se trouvent dans la classe **Database** du fichier database.php ([voir figure 31](#)). Afin de ne créer qu'une seule instance de la classe **Database** lorsqu'elle est appelée sur les différentes pages de l'application via le fichier header.php, nous avons implémenté un pattern Singleton.

Tout ce qui a trait à la gestion de la connexion à la base de données passe par cette classe **Database**. La propriété **\$instance** est une variable statique permettant de stocker l'instance unique de la classe **Database** dont nous avons besoin. Dans le constructeur, nous incluons le fichier utils.php ainsi que le fichier config.php permettant respectivement de stocker les fonctions utilitaires et charger les configurations de la base de données.

```

$configs = array(
    'db'      => "mysql",
    'host'    => "localhost",
    'port'    => "6033",
    'dbname'  => "echangeDeCartesACollectionner",
    'charset' => "utf8",
    'user'    => 'root',
);

$configs["dns"] =
$configs["db"] .
":host=" . $configs["host"] .
";dbname=" . $configs["dbname"] .
";charset=" . $configs["charset"] .
";port=" . $configs["port"];
|  

return $configs;

```

Figure 33 : tableau de configuration pour la base de données

Dans le fichier config.php, nous créons un tableau **\$configs** contenant les informations de configuration pour la base de données. Dans notre cas, il s'agit de :

- Du type de base de données.
- De l'hôte.
- Du port.
- Du nom de la base de données.
- Du codage de caractères.
- Du nom de l'utilisateur.

Nous ajoutons ensuite une nouvelle clé « **dns** » au tableau dont la valeur correspond à la concaténation de toutes les informations nécessaires à la configuration. Cette procédure nous permet maintenant d'appeler les valeurs de **\$configs[]** dans le constructeur de la classe **Database**.

```

function getPassword()
{
    $readJSONFile = file_get_contents(__DIR__ . "/../secrets.json");

    $array = json_decode($readJSONFile, TRUE);

    return $array["password"];
}

```

Figure 34 : fonction getPassword() se trouvant dans le fichier utils.php

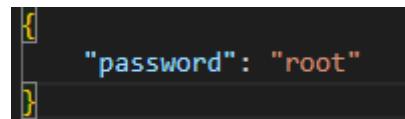


Figure 35 : Contenu de la page secrets.json contenant le mot de passe pour la BD

La méthode **getPassword()**, se trouvant dans le fichier `utils.php`, permet d'obtenir le mot de passe de la base de données qui est stocké de façon sécurisée dans un fichier `secrets.json`. Nous utilisons la fonction **file\_get\_contents()** pour lire le contenu du fichier `secrets.json` puis nous le décodons grâce à la méthode **json\_decode()** tandis que le second argument **TRUE** nous permet d'obtenir un tableau associatif plutôt qu'un objet. De cette façon, nous pouvons extraire la valeur correspondante à la clé « **password** » du tableau **\$array[]** grâce à la méthode que nous appelons dans le constructeur de la classe **Database**. Dans notre cas, le fichier `secrets.json` contient uniquement le mot de passe pour la connexion à la DB.

Enfin, la méthode **getInstance()** permet d'obtenir l'instance unique de la classe **Database**. Elle vérifie si l'instance de la classe n'a pas encore été créée et, si c'est le cas, elle crée une nouvelle instance qui pourra être appelée autant que nécessaire. Si l'instance existe déjà, la méthode retourne l'instance existante.

Ce procédé nous permet de garantir qu'il n'y aura qu'une seule connexion active à la base de données de façon globale dans le système afin d'éviter d'éventuels problèmes liés aux ressources. Il nous suffit ensuite d'implémenter le code permettant d'utiliser le pattern Singleton dans notre fichier `header.php`.

```
require_once 'database.php';
$db = Database::getInstance();
```

Figure 36 : implémentation du code permettant d'utiliser le pattern Singeton dans le fichier header.php

Nous chargeons le fichier `database.php` permettant d'accéder à la classe **Database** puis nous créons une nouvelle instance de la classe en utilisant la méthode **getInstance()**. De cette façon, il ne nous reste plus qu'à inclure le fichier `header.php` sur toutes les pages où nous avons besoin d'une connexion à la base de données.

### 3.4 Gestion de l'authentification

```

if (isset($_POST['login']) && isset($_POST['password'])) {
    $user = $db->CheckAuth($_POST['login'], $_POST['password']);
    if ($user == null) {
        $_SESSION['connexionError'] = 'Erreur de connexion';
    } else if ($user != null) {
        if (isset($_SESSION['connexionError'])) {
            unset($_SESSION['connexionError']);
        }
        $_SESSION['userConnected'] = $user['useRole'];
        $_SESSION['idUser'] = $user['idUser'];
        $_SESSION['useLogin'] = $db->getOneUser($_SESSION['idUser'])['useLogin'];
        $_SESSION['useCredits'] = $db->getOneUser($_SESSION['idUser'])['useCredits'];
        // Si le panier n'existe pas en session, on va en créer un
        if (!isset($_SESSION['panier'])) {
            $_SESSION['panier'] = [];
            header('Location: index.php');
        }
    }
}
}

```

Figure 37 : Processus d'authentification

Afin d'identifier si l'utilisateur est connecté ou non, nous implémentons dans le fichier `header.php` la méthode permettant de démarrer une session **session\_start()** avant d'effectuer une vérification des champs **login** et **password**. S'ils ont été soumis via le formulaire, nous effectuons une comparaison des valeurs entrées par l'utilisateur dans ces deux champs par rapport aux valeurs enregistrées dans la base de données grâce à la méthode **checkAuth()** de la classe **Database**.

Si la méthode **checkAuth()** ne renvoie pas d'utilisateur valide, une variable de session **connexionError** est définie et affichera un message d'erreur sur l'écran d'accueil. Sinon plusieurs valeurs sont stockées dans des variables de session. Pour la gestion des rôles, nous utilisons la variable de session **\$\_SESSION['userConnected']** dans laquelle nous stockons la valeur du rôle (user/admin) de l'utilisateur connecté (**\$user['useRole']**).

```

public function CheckAuth($useLogin, $password)
{
    $query = "
        SELECT *
        FROM t_user
        WHERE useLogin = :useLogin
        OR userEmail = :useLogin
    ";

    $replacements = ['useLogin' => $useLogin];
    $req = $this->queryPrepareExecute($query, $replacements);
    $user = $this->formatData($req)[0];

    if (password_verify($password, $user['usePassword'])) {
        return $user;
    } else {
        echo 'Erreur de connexion';
    }
}

```

La méthode **checkAuth()** a pour but de gérer l'authentification des utilisateurs. Elle reçoit deux paramètres :

- **\$useLogin** qui représente le nom d'utilisateur ou l'email soumis lors de la tentative de connexion.
- **\$password** qui représente le mot de passe soumis lors de la tentative de connexion.

Grâce à cette fonction, nous préparons une requête SQL afin de récupérer les données de l'utilisateur à partir de la table *t\_user*. Celle-ci vérifie si le nom d'utilisateur **useLogin** ou l'email **useEmail** correspond à la valeur soumise **:useLogin**. Si la requête renvoie un résultat, cela signifie qu'un utilisateur correspondant aux informations soumises a été trouvé.

Pour la vérification du mot de passe, nous utilisons la fonction **password\_verify()** afin de nous assurer que le mot de passe soumis correspond au mot de passe qui a été hashé et stocké dans la base de données lors de la création du compte. Lorsque nous comparons **\$password** à **\$user['usePassword']** deux résultats sont possibles :

- 1) Le mot de passe est correct et la méthode renvoi les données de l'utilisateur.
- 2) Le mot de passe ne correspond pas et un message d'erreur s'affiche.

**Erreur de connexion**

```
//Fonction permettant de préparer, de binder et d'exécuter une requête
private function queryPrepareExecute($query, $binds)
{
    $req = $this->connector->prepare($query);
    foreach ($binds as $bind => $value) {
        $req->bindValue($bind, $value);
    };
    $req->execute();
    return $req;
}
```

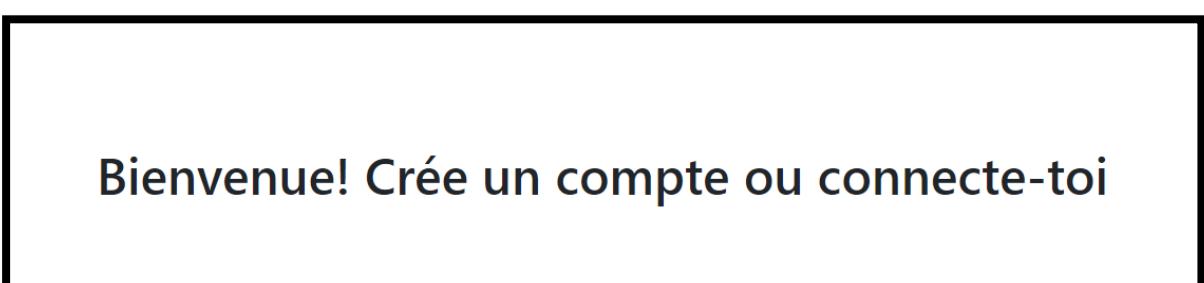
Dès lors que nous avons besoin de remplacer des valeurs en utilisant une requête SQL, nous appelons la méthode **queryPrepareExecute()**. Celle-ci nous permet de nous prémunir des injections SQL car les valeurs sont correctement échappées et traitées en tant que données brutes tout en étant séparées de la logique de la requête SQL. Ainsi, même si une valeur fournie par l'utilisateur contient du code SQL, il est traité comme une donnée et non comme une commande SQL.

### 3.5 Gestion des rôles

#### 3.5.1 Affichage de la page d'accueil selon le rôle de l'utilisateur

```
if (!isset($_SESSION['userConnected'])) {
    echo '<div style="display: flex; justify-content: center; align-items: center; height: 100vh;">
        <h1 style="font-size: 3em;">Bienvenue! Crée un compte ou connecte-toi</h1>
    </div>';
    include("footer.php");
    exit();
}
```

Figure 38 : Vérification de la connexion de l'utilisateur via la variable de session 'userConnected'



Si l'utilisateur n'est pas connecté, bloquons l'accès au contenu de la page index.php du site grâce à une vérification effectuée au niveau de la variable de session **\$\_SESSION['userConnected']**. Dans notre cas, l'utilisateur ne possède pas de compte et la clé **userConnected** n'existe donc pas. Nous affichons un message de bienvenue à l'utilisateur en l'invitant à se connecter ou à créer un compte puis nous bloquons l'accès au contenu de la page en arrêtant l'exécution du script.

Filtres													
Nom		Plus de filtres		Rechercher									
<input type="text"/>													
<b>Liste des cartes</b>													
Montrer 10 entrées													
Nom      Date de création      Crédits      Etat      Possesseur      Collection      Options													
Salamèche      2023      50      Neuf      User      Pokémons <button>Détails</button> <button>Modifier</button> <button>Supprimer</button>													
Bulbizarre NIV.13      2007      60      Occasion      Admin      Pokémons <button>Détails</button> <button>Acheter</button>													

Lorsque la valeur de **\$\_SESSION['userConnected']** est *user*, le contenu de la page d'accueil s'affiche normalement. L'utilisateur a la possibilité d'utiliser les filtres, les tris, d'afficher les détails d'une carte et d'acheter une carte du moment qu'il n'en est pas le possesseur. Il a également la possibilité de modifier ou supprimer une carte du moment qu'il en est le possesseur.

Filtres

Nom  Plus de filtres Rechercher

Liste des cartes

Montrer  entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Bulbizarre NIV.13	2007	60	Occasion	Admin	Pokémon	<input type="button" value="Détails"/> <input type="button" value="Modifier"/> <input type="button" value="Supprimer"/>
Démolosse	2002	30	Neuf	Admin2	Pokémon	<input type="button" value="Détails"/> <input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Acheter"/>

Lorsque la valeur de `$_SESSION['userConnected']` est admin, le contenu de la page d'accueil est le même que celui d'un utilisateur standard hormis le fait qu'il a la possibilité de supprimer et modifier toutes les cartes en vente en tout temps.

### 3.5.2 Navigation du header selon le rôle de l'utilisateur

```
if (!isset($_SESSION['userConnected'])) {
    // Affichage pour les utilisateurs non connectés
?>
<ul class="navbar-nav ml-auto">
    <li class="nav-item mt-2">
        <a class="nav-link" href="index.php">Accueil</a>
    </li>
    <li class="nav-item mt-2">
        <a class="nav-link" href="createAccount.php">Créer un compte</a>
    </li>
</ul>
```

Echange de cartes à collectionner    Accueil    Crée un compte

De la même façon, nous limitons la navigation à travers le site via le header. Un utilisateur non connecté ne pourra se rendre que sur la page d'accueil ou sur la page de création de compte.

```
// Affichage pour les utilisateurs connectés
if ($_SESSION['userConnected'] == 'user' || $_SESSION['userConnected'] == 'admin') {
?>
    <ul class="navbar-nav ml-auto">
        <li class="nav-item mt-2">
            <a class="nav-link" href="index.php">Accueil</a>
        </li>
        <li class="nav-item mt-2 textnowrap">
            <a class="nav-link" href="addCard.php">Ajouter une carte</a>
        </li>
        <li class="nav-item mt-2 textnowrap">
            <a class="nav-link" href="userProfile.php?idUser=<?php echo $_SESSION["idUser"]; ?>">Profil</a>
        </li>
        <li class="nav-item mt-2 textnowrap">
            <a class="nav-link" href="cart.php">Panier</a>
        </li>
    <?php } ?>
    </ul>
```

## Echange de cartes à collectionner    Accueil    Ajouter une carte    Profil    Panier

Lorsque la valeur de **\$\_SESSION['userConnected']** est **admin** ou **user**, l'utilisateur a la possibilité d'accéder aux pages : Accueil, Ajouter une carte, Profil et Panier.

### 3.5.3 Affichage du formulaire de login selon le rôle de l'utilisateur

```
<?php if (isset($_SESSION['userConnected'])) and
$_SESSION['userConnected'] == ('user' or 'admin')) { ?>
    <form class="hstack gap-3 mb-0" action="" method="post">
        <div class="me-2 text-white">
            Bienvenue <?php echo $_SESSION['useLogin'] ?><br>
            Crédits : <?php echo intval($_SESSION['useCredits']) ?>
        </div>
        <button class="btn btn-outline-danger mx-1" type="submit" name="logout">
            Déconnexion
        </button>
    </form>
<?php } else { ?>
    <form class="hstack gap-3 mb-0" action="" method="post">
        <input class="form-control" type="text" name="login" id="login" placeholder="Login">
        <input class="form-control" type="password"
name="password" id="password" placeholder="Mot de passe">
        <button class="btn btn-outline-success" type="submit">
            Connexion
        </button>
    </form>
<?php } ?>
```




Lorsque la valeur de **\$\_SESSION['userConnected']** est **null**, le formulaire de login apparaît dans le header en attente des informations de l'utilisateur.

Bienvenue Admin

Crédits : 100

[Déconnexion](#)

Lorsque la valeur de **`$_SESSION['userConnected']`** est `admin` ou `user`, le formulaire de login est remplacé par un message de bienvenue suivi du login de l'utilisateur, du total de crédits disponibles sur son compte ainsi qu'un bouton de déconnexion. Pour afficher le login de l'utilisateur, on récupère la valeur stockée dans la variable de session **`$_SESSION['useLogin']`** tandis que pour afficher le total de crédits on récupère la valeur stockée dans la variable de session **`$_SESSION['useCredits']`** en effectuant une conversion grâce à la méthode **`intval()`** car ce champ est de type int.

### 3.5.4 Déconnexion

```
// Déconnexion de l'utilisateur et redirection vers la page d'accueil
if (isset($_POST['logout'])) {
    session_unset();
    session_destroy();
    header('Location: index.php');
    exit;
}
```

Lorsque l'utilisateur clique sur le bouton de déconnexion, nous effectuons une vérification de la variable **`$_POST['logout']`** pour définir si l'utilisateur a bien cliqué dessus. Si c'est le cas, nous appelons la méthode **`session_unset()`** qui supprime toutes les variables de session pour vider les données de l'utilisateur avant d'appeler la méthode **`session_destroy()`** afin de détruire la session en cours. Une fois toutes les données de session supprimées et l'ID de session réinitialisé, nous effectuons une redirection vers la page `index.php` avant d'interrompre l'exécution du script.

```
if (!isset($_SESSION['userConnected']) || $_SESSION['userConnected'] != ('user' or 'admin')) {
    header('HTTP/1.0 403 Forbidden', true, 403);
    require_once(__DIR__ . "/403.php");
    exit;
}
```

Finalement, afin de renforcer la sécurité de notre application, nous utilisons une redirection vers une page 403 lorsque l'un utilisateur non connecté tente d'accéder à une page ou une ressource dont il ne dispose pas des droits. Nous appliquons ce procédé sur toutes les pages affichant du contenu.

<localhost:3000/addCard.php>Echange de cartes à collectionner - <localhost:3000/addCard.php>

**HTTP403 - Vous n'avez pas les droits pour accéder à cette page  
Connectez-vous ou retournez à la page d'accueil**

Si l'utilisateur n'est pas identifié par la session comme étant un *user* ou un *admin*, alors il est redirigé vers une page d'erreur 403. Celle-ci l'invite à se connecter ou à retourner sur la page d'accueil.

### 3.6 Création de compte utilisateur

**Créer un compte**

Login :	<input type="text"/>
Email :	<input type="text"/>
Prénom :	<input type="text"/>
Nom :	<input type="text"/>
Localité :	<input type="text"/>
Code Postal :	<input type="text"/>
Nom de la rue :	<input type="text"/>
Numéro :	<input type="text"/>
Mot de passe :	<input type="text"/>
<input type="button" value="Créer un compte"/>	

Lorsqu'un utilisateur clique sur l'onglet « Crée un compte » il est redirigé sur la page `createAccount.php` qui affiche l'interface ci-dessus. Pour pouvoir s'inscrire, il doit renseigner les champs demandés. Dès lors qu'il clique sur le bouton « Crée un compte » les données sont soumises à des contraintes de validation via le fichier `validateAddUserForm.php` qui est inclus sur la page `createAccount.php`.

Afin d'analyser le fonctionnement de ce formulaire de validation des données, nous indiquons les restrictions appliquées à chaque champ dans un tableau mais étudions la logique du code pour un seul champ (**login**) pour l'exemple.

## Contraintes du formulaire de validation des données pour la création de compte

Champ	Type	Contrainte 1	Contrainte 2	Contrainte 3
<b>login</b>	text	Obligatoire	1 à 120 caractères de tout type	Valeur unique
<b>email</b>	text	Obligatoire	Doit être un email	Valeur unique
<b>firstName</b>	text	Obligatoire	1 à 120 caractères sans chiffres et sans caractères qui soient des symboles non-alphanumériques	-
<b>lastName</b>	text	Obligatoire	1 à 120 caractères sans chiffres et sans caractères qui soient des symboles non-alphanumériques	-
<b>locality</b>	text	Obligatoire	1 à 120 caractères sans chiffres et sans caractères qui soient des symboles non-alphanumériques	-
<b>postalCode</b>	text	Obligatoire	1 à 15 caractères sans caractères qui soient des symboles non-alphanumériques	-
<b>streetName</b>	text	Obligatoire	1 à 120 caractères sans chiffres	-
<b>streetNumber</b>	text	Obligatoire	1 à 15 caractères sans caractères qui soient des symboles non-alphanumériques	-
<b>password</b>	password	Obligatoire	-	-

## Validation des données pour le champ login

```
const ERROR_LOGIN_REQUIRED = "Veuillez renseigner le champ login";
const ERROR_LOGIN = "Le champ doit avoir un nombre de caractères
entre 1 et 120 tous caractères compris";
const ERROR_LOGIN_EXISTS = "Ce login est déjà utilisé";

const REGEX_VARCHAR120 = '/^(?!.*\n.*$)(?!.\n)(?!.{121}).{1,120}$/us';
```

Tout d'abord, nous déclarons au moyen de constantes les différents messages d'erreur que nous voulons afficher dans le cas où les contraintes de validation du champ ne sont pas respectées. Nous pouvons ainsi stocker les messages d'erreurs de façon centralisée et utiliser ces constantes dans différentes parties du code ce qui facilite la maintenance du message d'erreur dans le cas où une modification est nécessaire ultérieurement.

Puis, si nécessaire, nous déclarons une constante contenant une expression régulière – pour ce champ, **REGEX\_VARCHAR120** – dans le but de vérifier si la chaîne de caractères du champ **login** correspond à la règle de la figure .

```
function validateAddUserForm($db)
```

```
$login = $_POST['login'] ?? '';
```

Nous déclarons ensuite la fonction **validateAddUserForm()** pour permettre la validation des données. Une fois cela fait, nous attribuons la valeur du champ **\$\_POST['login']** à la variable **\$login**. Si le champ **login** n'est pas présent dans le tableau, la variable sera définie par défaut comme une chaîne vide.

```
$errors = [];
```

Nous créons la variable **\$errors** et lui assignons la valeur initiale d'un tableau vide. Cela nous permet de stocker toutes les erreurs qui pourraient survenir dans ce tableau. Au début de l'exécution du code, il n'y a donc aucune erreur dans le tableau. L'idée étant de parcourir ce tableau au fur et à mesure de la validation et lorsqu'une erreur survient, elle est ajoutée dans notre tableau en étant associée à la clé correspondante qui identifie les champs concernés par les erreurs.

```
// le champ login :
// - est obligatoire
// - peut comporter 1 à 120 caractères de tout type
// - doit avoir une valeur unique
if (!$login) {
    $errors['login'] = ERROR_LOGIN_REQUIRED;
} elseif (!preg_match(REGEX_VARCHAR120, $login)) {
    $errors["login"] = ERROR_LOGIN;
} elseif ($existingUser = $db->getUserByLogin($login)) {
    $errors['login'] = ERROR_LOGIN_EXISTS;
}
```

Il ne nous reste plus qu'à définir la validation du champ **login** selon nos besoins. Pour cette application, nous avons décidé de le soumettre aux trois conditions suivantes :

- 1) **If (!\$login)** vérifie si le champ **login** est vide ou false. Si c'est le cas, cela signifie que le champ n'a pas été renseigné par l'utilisateur et une erreur est ajoutée au tableau **\$errors** avec la clé **login** et la valeur **ERROR\_LOGIN\_REQUIRED**. Un message d'erreur indiquant que ce champ est obligatoire sera affiché à l'utilisateur.
- 2) **Elseif ( !preg\_match(REGEX\_VARCHAR120, \$login))** vérifie grâce à l'expression régulière **REGEX\_VARCHAR120** si la valeur du champ **login** correspond bien à une chaîne de caractère d'une longueur allant de 1 à 120 caractères. Si la valeur ne correspond pas à la règle, une erreur est ajoutée au tableau **\$errors** avec la clé **login** et la valeur **ERROR\_LOGIN**. Un message d'erreur indiquant à l'utilisateur que le champ ne peut comporter seulement 1 à 120 caractères lui sera affiché.
- 3) **Elseif (\$existingUser = \$db->getUserByLogin(\$login))** vérifie si un utilisateur avec la même valeur de login existe déjà dans la base de données. Si la valeur renournée est différente de **null**, la valeur du champ **login** n'est pas unique et une erreur est ajoutée au tableau **\$errors** avec la clé **login** et la valeur **ERROR\_LOGIN\_EXISTS**. Dans ce cas, un message indiquant que le login qu'il a saisi est déjà utilisé lui sera affiché.

```
public function getUserByLogin($login)
{
    $query = "SELECT * FROM t_user WHERE useLogin = :login";
    $bind = array('login' => $login);
    $req = $this->queryPrepareExecute($query, $bind);
    $existingUser = $this->formatData($req);

    return $existingUser[0];
}
```

Pour permettre la vérification de la troisième condition, nous appelons la méthode **getUserByLogin()** de la classe **Database**. Celle-ci exécute une

requête SQL permettant de récupérer un utilisateur en fonction de son login. Grâce à cette méthode, nous comparons donc si la valeur du champ **login** soumise par l'utilisateur existe déjà dans la colonne **useLogin** de la table **t\_user**.

```
// On commence par désinfecter les données saisies par l'utilisateur
// ainsi on se protège contre les attaques de types XSS
$userData = filter_input_array(
    INPUT_POST,
    [
        'login' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
        'email' => FILTER_SANITIZE_EMAIL,
        'firstName' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
        'lastName' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
        'locality' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
        'postalCode' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
        'streetName' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
        'streetNumber' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
        'password' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
    ]
);
```

Le tableau **\$userData[]** stockera toutes les données de l'utilisateur est ensuite créée. La fonction **filter\_input\_array()** va désinfecter les données saisies et reçoit deux paramètres :

- 1) **INPUT\_POST** qui indique que les données doivent être récupérées à partir de la méthode **POST** et que les valeurs soumises par l'utilisateur via le formulaire avec cette méthode seront utilisées pour le filtrage.
- 2) **Un tableau associatif** qui nous permet de spécifier le champ à filtrer et le type de filtrage à appliquer à chaque champ.

Pour le champ **login** nous utilisons le filtre **FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS** afin de supprimer ou échapper tous les caractères spéciaux présents dans la valeur qui est soumise. Ce filtrage nous permet de nous protéger contre les attaques de type **XSS** en garantissant que les caractères spéciaux ne sont pas interprétés de manière malveillante lors de l'affichage ou l'utilisation de cette valeur. De cette façon, nous sécurisons la valeur du champ en éliminant les caractères spéciaux qui pourraient être exploités pour injecter du code ou compromettre la sécurité de l'application.

```
return ["userData" => $userData, "errors" => $errors];
```

Finalement, la fonction **validateAddUserForm()** retourne un tableau contenant les deux clés **userData** et **errors** qui contiennent le résultat de la validation du formulaire après avoir parcouru le tableau des erreurs.

## Logique d'ajout d'un utilisateur

```

include("header.php");
include_once(__DIR__ . "/validateAddUserForm.php");

$errors = [];

if (isset($_SESSION['userConnected'])) {
    header('Location: index.php');
}

if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    $result = validateAddUserForm($db);      davassayah, il y a 2 semaines • Corrections applic
    $errors = $result["errors"];
    $userData = $result["userData"];

    // Si aucune erreur de validation
    // Cela signifie que les données sont propres et validées
    // Nous pouvons insérer les données en BD
    if (count($errors) === 0) {
        $users = $db->addUser($userData);
        $_SESSION['successMessage'] = "Compte créé avec succès";
        header('Location: index.php');
    } else {
        if ($_POST) {
            $errorMessage = "Merci de bien remplir tous les champs marqués comme obligatoires";
        }
    }
}

```

Sur la page `createAccount.php`, nous incluons notre formulaire de validation `validateAddUserForm.php` afin de pouvoir effectuer la validation comme nous venons de le voir lorsque l'utilisateur clique sur le bouton « Créer un compte ». Nous créons une variable **\$errors** en lui attribuant une valeur de tableau vide qui nous servira à stocker les éventuelles erreurs.

Nous vérifions si le formulaire a bien été soumis en utilisant la méthode HTTP POST afin de s'assurer que le code n'est exécuté que lorsque le formulaire est soumis. Si c'est le cas, nous appelons la fonction **validateAddUserForm(\$db)** afin d'effectuer la validation du formulaire. Nous stockons le résultat retourné par cette fonction dans la variable **\$result** qui est un tableau contenant deux clés :

- 1) **Errors** : les erreurs de validation.
- 2) **userData** : les données de l'utilisateur soumises dans le formulaire.

Nous récupérons ensuite les erreurs de validation et les données soumises par l'utilisateur validées à partir du tableau **\$result** et les assignons respectivement aux variables **\$errors** et **\$userData**. Enfin, nous vérifions le nombre total d'erreurs stockées dans **\$errors**. Si aucune erreur n'est stockée dans le tableau des erreurs, alors la méthode **\$db->addUser(\$userData )** est appelée pour ajouter

les données soumises par l'utilisateur dans le formulaire dans la base de données.

Dans le cas où toutes les informations ont bien été ajoutées en base de données, un message de succès s'affiche et l'utilisateur est redirigé sur l'écran d'accueil.

Dans le cas contraire, les informations ne sont pas ajoutées en base de données et les erreurs contextuelles s'affichent en fonction des champs.

```
public function addUser($user)
{
    $query = "
        INSERT INTO t_user (useLogin, useEmail, useFirstName, useLastName,
        useLocality, usePostalCode, useStreetName, useStreetNumber, usePassword)
        VALUES (:login, :email, :firstName, :lastName,
        :locality, :postalCode, :streetName, :streetNumber, :password);
    ";

    $replacements = [
        'login' => $user['login'],
        'email' => $user['email'],
        'firstName' => $user['firstName'],
        'lastName' => $user['lastName'],
        'locality' => $user['locality'],
        'postalCode' => $user['postalCode'],
        'streetName' => $user['streetName'],
        'streetNumber' => $user['streetNumber'],
        'password' => password_hash($user['password'], PASSWORD_BCRYPT),
    ];

    $response = $this->queryPrepareExecute($query, $replacements);
}
```

La méthode **addUser()** exécute une requête SQL permettant d'insérer en base de données les valeurs fournies dans le tableau **\$user**. Elle hash également le mot de passe avant de l'insérer dans la base de données grâce à la fonction **password\_hash()**. Ici, l'algorithme **PASSWORD\_BYCRYPT** basé sur Blowfish est utilisé pour sécuriser le mot de passe. Si la requête se déroule correctement, les données soumises par l'utilisateur via le formulaire sont insérées dans la table **t\_user**.

## Gestion de l'affichage du champ dans le formulaire d'inscription

```
p>
    <label for="login">Login :</label>
    <input type="text" name="login" id="login"
    value=<?php echo isset($_POST['login']) ? htmlspecialchars($_POST['login']) : ''; ?>>
    <span id="show-error">
        <?php echo (array_key_exists("login", $errors) && $errors["login"]) ?
            '<p style="color:red;">' . $errors["login"] . '</p>' : '';
    </span>
/p>
```

Sur cette image, nous pouvons voir comment sont construits les différents champs qui composent notre formulaire. Nous vérifions si des données ont été soumises via le champ de saisie **login** dans une requête **POST** et si sa valeur existe dans **\$\_POST**. Si c'est le cas, la fonction **htmlspecialchars()** est utilisée pour échapper les caractères spéciaux afin d'éviter les problèmes de sécurité liés aux attaques XSS. Si la clé login n'existe pas dans **\$\_POST**, la valeur du champ est une chaîne vide. Cela nous permet de retourner à l'utilisateur la dernière valeur qu'il a soumise dans le champ même en cas d'erreur.

Nous parcourons ensuite le tableau **\$errors** pour définir si la clé **login** est existante. Si c'est le cas, l'erreur provoque l'affichage d'un message contextuel correspondant sinon rien ne s'affiche.

Pour illustrer, nous créons un compte pour l'utilisateur Vendeur et observons les différents affichages possibles du formulaire de validation des données lors de la création d'un compte conformément au tableau des contraintes que nous avons établi en figure 12.

## Créer un compte

Merci de bien remplir tous les champs marqués comme obligatoires

Login :

Veuillez renseigner le champ login

Email :

Veuillez renseigner le champ email

Prénom :

Veuillez renseigner le champ prénom

Nom :

Veuillez renseigner le champ nom de famille

Localité :

Veuillez renseigner le champ localité

Code Postal :

Veuillez renseigner le champ code postal

Nom de la rue :

Veuillez renseigner le champ nom de la rue

Numéro :

Veuillez renseigner le champ numéro de la rue

Mot de passe :

Veuillez renseigner le champ mot de passe

Nous pouvons constater que lorsque nous laissons les champs vides, les bonnes erreurs contextuelles apparaissent pour les bons champs.

## Créer un compte

Merci de bien remplir tous les champs marqués comme obligatoires

Login :

Le champ doit avoir un nombre de caractères entre 1 et 120 tous caractères compris

Email :

Merci de renseigner une adresse email valide

Prénom :

Merci de saisir une chaîne de caractères entre 1 et 120 caractères ne contenant pas de chiffres.

Nom :

Merci de saisir une chaîne de caractères entre 1 et 120 caractères ne contenant pas de chiffres.

Localité :

Merci de saisir une chaîne de caractères entre 1 et 120 caractères ne contenant pas de chiffres.

Code Postal :

Merci de saisir une chaîne de caractères de 1 à 15 caractères maximum

Nom de la rue :

Merci de saisir une chaîne de caractères entre 1 et 120 caractères ne contenant pas de chiffres.

Numéro :

Merci de saisir une chaîne de caractères de 1 à 15 caractères maximum - chiffres et lettres compris - sans caractères spéciaux

Mot de passe :

Nous pouvons constater que lorsque nous ne respectons pas le nombre de caractères ou le type attendu les erreurs contextuelles s'affichent correctement et les valeurs sont renvoyées à l'utilisateur. Pour des raisons de sécurité, le mot de passe n'est pas retourné.

## Créer un compte

Merci de bien remplir tous les champs marqués comme obligatoires

Login :

Ce login est déjà utilisé

Email :

Cette adresse email est déjà utilisée

Prénom :

Nom :

Localité :

Code Postal :

Nom de la rue :

Numéro :

Mot de passe :

Nous pouvons constater que lorsque nous soumettons des valeurs déjà existantes dans la base de données dans les champs **login** et **email**, les bonnes erreurs contextuelles sont affichées et les valeurs sont retournées à l'utilisateur.

## Créer un compte

Login :	Vendeur
Email :	Vendeur@hotmail.com
Prénom :	Ven
Nom :	Deur
Localité :	Ventarlier
Code Postal :	1002
Nom de la rue :	Rue de la vente
Numéro :	12
Mot de passe :	.....

[Créer un compte](#)

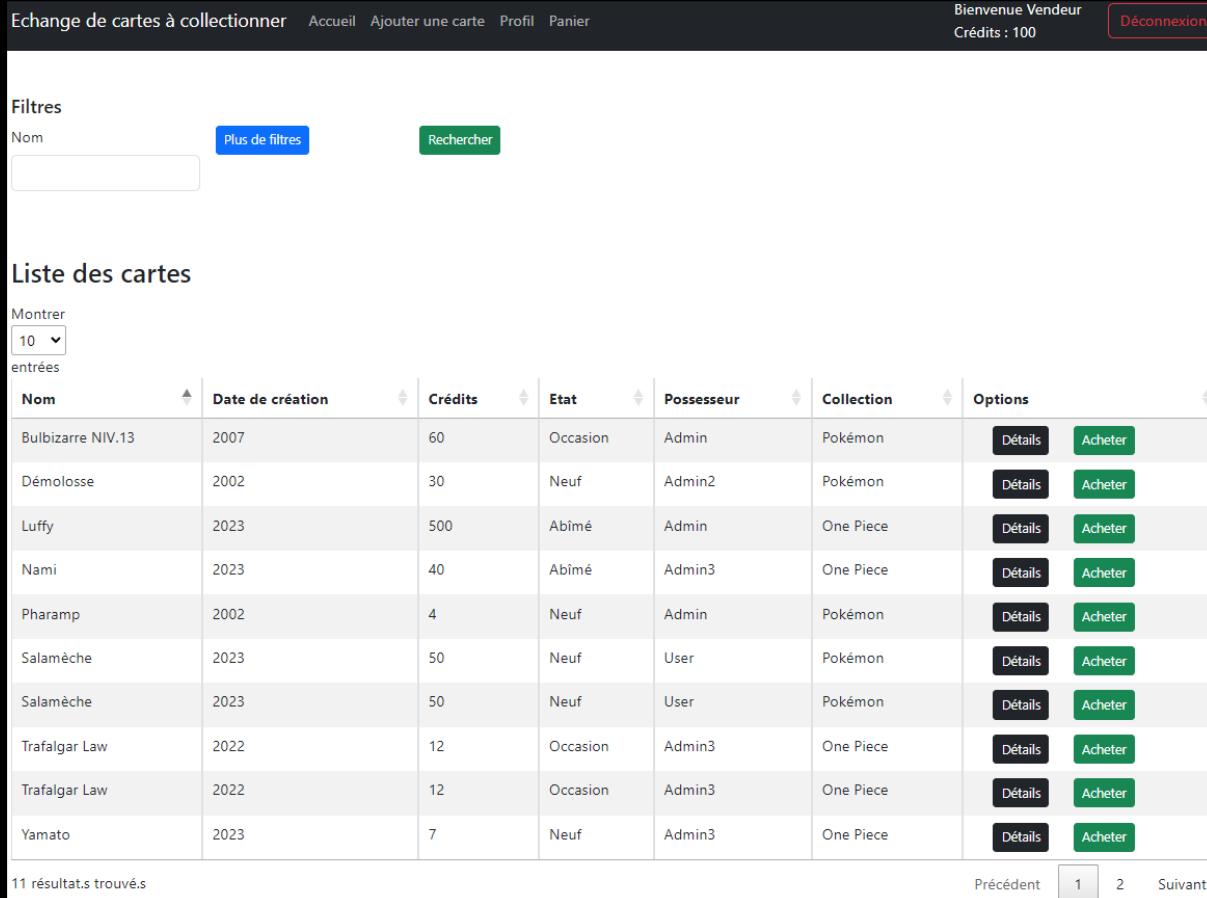
9 Vendeur

Vendeur@hotmail.com

## Compte créé avec succès

Dès lors que nous avons correctement rempli tous les champs et respecté toutes les contraintes de validation, notre compte est créé, inséré en base de données et la page d'accueil affiche un message de confirmation.

### **3.7 Affichage du tableau des cartes en vente**



Bienvenue Vendeur  
Crédits : 100 Déconnexion

Filtres  
Nom   Plus de filtres Rechercher

Liste des cartes

Montrer 10 entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Bulbizarre NIV.13	2007	60	Occasion	Admin	Pokémon	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Démolosse	2002	30	Neuf	Admin2	Pokémon	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Luffy	2023	500	Abîmé	Admin	One Piece	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Nami	2023	40	Abîmé	Admin3	One Piece	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Pharamp	2002	4	Neuf	Admin	Pokémon	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Salamèche	2023	50	Neuf	User	Pokémon	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Salamèche	2023	50	Neuf	User	Pokémon	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>
Yamato	2023	7	Neuf	Admin3	One Piece	<span style="border: 1px solid black; padding: 2px;">Détails</span> <span style="background-color: #28a745; color: white; border: 1px solid #28a745; padding: 2px;">Acheter</span>

11 résultats trouvés Précédent 1 2 Suivant

Une fois notre compte Vendeur créé, nous nous connectons et arrivons sur la page d'accueil. Par défaut, un compte créé est considéré comme user, nous n'avons donc aucune autre possibilité que d'utiliser les filtres, acheter une carte ou afficher les détails des cartes en vente.

```
<link rel="stylesheet"
      href="https://cdn.datatables.net/1.13.4/css/jquery.dataTables.css" />
```

```
<script src="https://cdn.datatables.net/1.13.4/js/jquery.dataTables.js"></script>
```

Pour l'affichage du tableau, nous utilisons Bootstrap pour la mise en page et le style ainsi que jQuery avec le plugin *DataTables* pour ajouter les fonctionnalités de recherche, de tri et de pagination du tableau.

```

$(document).ready(function() {
    $('#sortTable').DataTable({
        searching: false,
        language: {
            lengthMenu: "Montrer _MENU_ entrées",
            info: "_TOTAL_ résultat.s trouvé.s",
            paginate: {
                next: "Suivant",
                previous: "Précédent"
            }
        }
    });
}
);

```

Nous utilisons la fonction **`$(document).ready()`** pour exécuter le code jQuery ne fois que le DOM est prêt. Nous appelons ensuite **`$('#sortTable').DataTable()`** pour initialiser le plugin DataTables afin d'ajouter les fonctionnalités de tri, pagination et de recherche. Pour notre tableau, nous désactivons le champ de recherche par défaut du plugin pour pouvoir créer un nouveau champ de recherche fonctionnant avec nos filtres. Nous configurons également les éléments d'interface de langage manuellement afin qu'ils soient affichés en français et non en anglais.

**Liste des cartes**

Montrer  
10 ▾  
entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Zorro	2022	15	Occasion	Admin2	One Piece	<button>Détails</button> <button>Acheter</button>

11 résultat.s trouvé.s

Précédent 1 2 Suivant

Liste des cartes

Montrer  
10 entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Pharamp	2002	4	Neuf	Admin	Pokémon	<button>Détails</button> <button>Acheter</button>
Yamato	2023	7	Neuf	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Zorro	2022	15	Occasion	Admin2	One Piece	<button>Détails</button> <button>Acheter</button>
Démolosse	2002	30	Neuf	Admin2	Pokémon	<button>Détails</button> <button>Acheter</button>
Nami	2023	40	Abîmé	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Salamèche	2023	50	Neuf	User	Pokémon	<button>Détails</button> <button>Acheter</button>
Salamèche	2023	50	Neuf	User	Pokémon	<button>Détails</button> <button>Acheter</button>
Bulbizarre NIV.13	2007	60	Occasion	Admin	Pokémon	<button>Détails</button> <button>Acheter</button>

11 résultat.s trouvé.s

Précédent 1 2 Suivant

```
$cards = $db->getAllCards();
```

```
$collections = $db->getAllCollections();
```

Pour afficher les informations dans le tableau, nous récupérons les informations stockées en base de données en appelant les fonctions **getAllCards()** et **getAllCollections()** de la classe *Database*.

```
public function getAllCollections()
{
    $query = "SELECT * FROM t_collection";
    //appeler la méthode pour executer la requête
    $req = $this->querySimpleExecute($query);
    //appeler la méthode pour avoir le résultat sous forme de tableau
    $collections = $this->formatData($req);
    //retourne toutes les collections
    return $collections;
}
```

La méthode **getAllCollections()** exécute une requête SQL permettant de sélectionner toutes les lignes de la table **t\_collection**. Cela nous permet de récupérer toutes les collections de cartes en vente sur le site et de les afficher dans le tableau ainsi que pour les lister les collections à filtrer.

```
public function getAllCards()
{
    $query = "
SELECT
    t_card.*,
    t_collection.colName AS carCollectionName,
    t_user.useLogin AS carUserLogin
FROM t_card
LEFT JOIN t_collection ON t_collection.idCollection = t_card.fkCollection
LEFT JOIN t_user ON t_user.idUser = t_card.fkUser
WHERE t_card.carIsAvailable = 1
";
    //appeler la méthode pour executer la requête
    $req = $this->querySimpleExecute($query);
    //appeler la méthode pour avoir le résultat sous forme de tableau
    $cards = $this->formatData($req);
    //retourne toutes les cartes
    return $cards;
}
```

La méthode **getAllCards()** permet de récupérer toutes les cartes disponibles en exécutant une requête SQL qui sélectionne toutes les colonnes de la table **t\_card**. Grâce à elle, nous récupérons les informations concernant la collection et l'utilisateur associé à chaque carte et pouvons afficher uniquement les cartes considérées comme disponibles (lorsque **carIsAvailable** = 1).

```

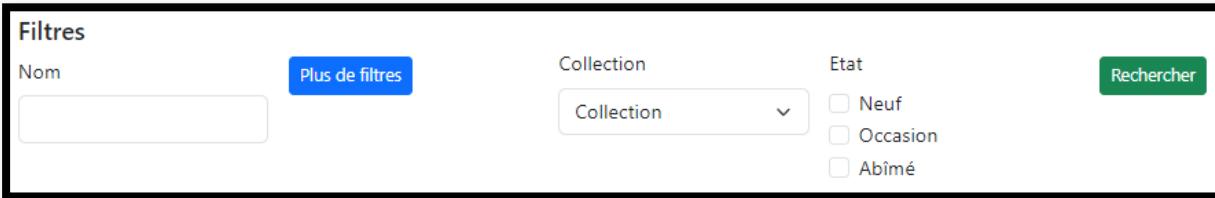
<?php foreach ($cards as $card) { ?>
<tr>
    <td><?php echo $card["carName"] ?></td>
    <td><?php echo $card["carDate"] ?></td>
    <td><?php echo $card["carCredits"] ?></td>
    <td>
        <?php if ($card["carCondition"] == "N") {
            echo "Neuf";
        } else if ($card["carCondition"] == "O") {
            echo "Occasion";
        } else if ($card["carCondition"] == "A") {
            echo "Abimé";
        } ?>
    </td>
    <td><?php echo $card["carUserLogin"] ?></td>
    <td><?php echo $card["carCollectionName"] ?></td>
    <td class="containerOptions">
        <?php if (isset($_SESSION['userConnected'])) { ?>
            <a class="btn btn-dark btn-sm"
                href="cardDetails.php?idCard=<?php echo $card["idCard"] ?>">Détails</a>
            <?php if ($_SESSION['userConnected'] == 'admin'
                || $_SESSION['userConnected'] == 'user'
                && $_SESSION['useLogin'] == $card['carUserLogin']) { ?>
                <a class="btn btn-warning btn-sm"
                    href="updateCard.php?idCard=<?php echo $card["idCard"]; ?>">Modifier</a>
                <a class="btn btn-danger btn-sm"
                    href="javascript:confirmDelete(<?php echo $card["idCard"] ?>)">Supprimer</a>
            <?php } ?>
            <?php if (($_SESSION['userConnected'] ==
                'admin' || $_SESSION['userConnected'] == 'user')
                && $_SESSION['useLogin'] != $card['carUserLogin']) { ?>
                <a class="btn btn-success btn-sm"
                    href="javascript:confirmBuy(<?php echo $card["idCard"] ?>)">Acheter</a>
            <?php } ?>
        <?php } ?>
    </td>
</tr>
<?php } ?>

```

Grâce à cette boucle `foreach`, nous parcourons ensuite les données qui nous intéressent pour afficher le nom, l'état, la date, le coût en crédits, le propriétaire et la collection d'une carte ainsi que les actions qu'un utilisateur peut ou non réaliser en fonction de son rôle et de ses droits dans le tableau

### 3.8 Filtres

```
// Afficher/Cacher Les filtres en fonction du bouton "Plus de filtres"
$('#more-filters-btn').click(function() {
    $('#filter-collection').toggleClass('d-none');
    $('#filter-condition').toggleClass('d-none');
});
```



Lorsque l'utilisateur souhaite filtrer les cartes du tableau, il a la possibilité de cliquer sur le bouton *Plus de filtres* ce qui déclenche la fonction `$('#more-filters-btn').click()`. Nous utilisons les méthodes `toggleClass()` pour ajouter ou supprimer la classe CSS **d-none** des éléments qui possèdent l'ID **filter-collection** et **filtrer-condition**. Cela nous permet d'afficher ou de cacher les filtres supplémentaires sur la collection et l'état des cartes en fonction du clique sur le bouton.

```
if (isset($_GET['submit'])) {
    $cards = $db->sortCards($_GET);
```

Afin d'effectuer une recherche via nos filtres, nous vérifions si le formulaire de filtrage a été soumis. Si c'est le cas, lorsque l'utilisateur clique sur le bouton Rechercher, la fonction `sortCards()` de la classe **Database** est exécutée et le filtrage a lieu selon les préférences renseignées par l'utilisateur.

```

public function sortCards($filters)
{
    $query = "
        SELECT
            t_card.*,
            t_collection.colName AS carCollectionName,
            t_user.useLogin AS carUserLogin
        FROM t_card
        LEFT JOIN t_collection ON t_collection.idCollection = t_card.fkCollection
        LEFT JOIN t_user ON t_user.idUser = t_card.fkUser
        WHERE t_card.carIsAvailable = 1
    ";

    $filter = false;
    $replacements = [];

    // Condition 1
    if (!empty($filters['search'])) {
        $filter = true;
        $query .= $this->addWhereOrAnd($filter);
        $query .= " t_card.carName LIKE :searchValue ";
        $replacements['searchValue'] = '%' . $filters['search'] . '%';
    }

    // Condition 2
    if (isset($filters['conditions'])) {
        $query .= $this->addWhereOrAnd($filter);
        $query .= " t_card.carCondition IN (" . $this->convertConditions($filters['conditions']) . ")";
        $filter = true;
    }

    // Condition 3
    if (isset($filters['idCollection']) and $filters['idCollection'] !== '') {
        $query .= $this->addWhereOrAnd($filter);
        $query .= " t_card.fkCollection = :idCollection";
        $replacements['idCollection'] = $filters['idCollection'];
        $filter = true;
    }

    $query .= " ORDER BY t_card.carName ASC";

    $req = $this->queryPrepareExecute($query, $replacements);
    $filters = $this->formatData($req);

    return $filters;
}

```

La fonction **sortCards()** permet de filtrer et de trier les cartes en fonction des critères qui sont spécifiés et stockés dans le tableau **\$filters** en construisant et exécutant une requête SQL dynamique dans le but de récupérer les résultats de la recherche filtrée.

Pour ce faire, nous sélectionnons d'abord toutes les colonnes de la table **t\_card** et effectuons des jointures avec les tables **t\_collection** et **t\_user**. Nous déclarons une variable **\$filter** et lui attribuons la valeur **false**. A chaque fois que l'un des filtres est renseigné par l'utilisateur, la valeur de **\$filter** passe à **true**. Cela nous permet de construire notre requête en fonction des différents filtres que nous appliquons selon les critères spécifiés dans le tableau **\$filters** :

- **Condition 1** : Si l'utilisateur a renseigné une valeur dans le champ destiné à rechercher une carte par le nom, une condition est ajoutée à la requête pour filtrer les cartes correspondant à la valeur renseignée par l'utilisateur.
- **Condition 2** : Si l'utilisateur a renseigné un état spécifique via le filtre par l'état, une condition est ajoutée à la requête pour permettre que les cartes affichées soient filtrées en fonction de la valeur correspondant à l'état renseignée par l'utilisateur. Pour convertir les états en chaîne de caractères à utiliser dans la requête SQL, nous utilisons la méthode **convertConditions()**. Selon les filtres qui ont été sélectionné par l'utilisateur, la méthode **addWhereOrAnd()** ajoute un **WHERE** ou un **AND** dans la requête de sorte à ce qu'elle puisse se construire dynamiquement.
- **Condition 3** : Si l'utilisateur a renseigné une collection spécifique via le filtre par la collection, une condition est ajoutée à la requête pour effectuer un filtrage en fonction de l'ID de la collection qui a été renseignée. Selon les filtres qui ont été sélectionné par l'utilisateur, la méthode **addWhereOrAnd()** ajoute un **WHERE** ou un **AND** dans la requête de sorte à ce qu'elle puisse se construire dynamiquement.

Les cartes sont ensuite triées par ordre croissant en fonction de leur nom et retourne les données filtrées et triées sous la forme du tableau **\$filters**.

```
private function convertConditions($conditions)
{
    $f = '';
    foreach ($conditions as $condition) {
        $f .= " " . $condition . ", ";
    }
    return substr($f, 0, -2);
}
```

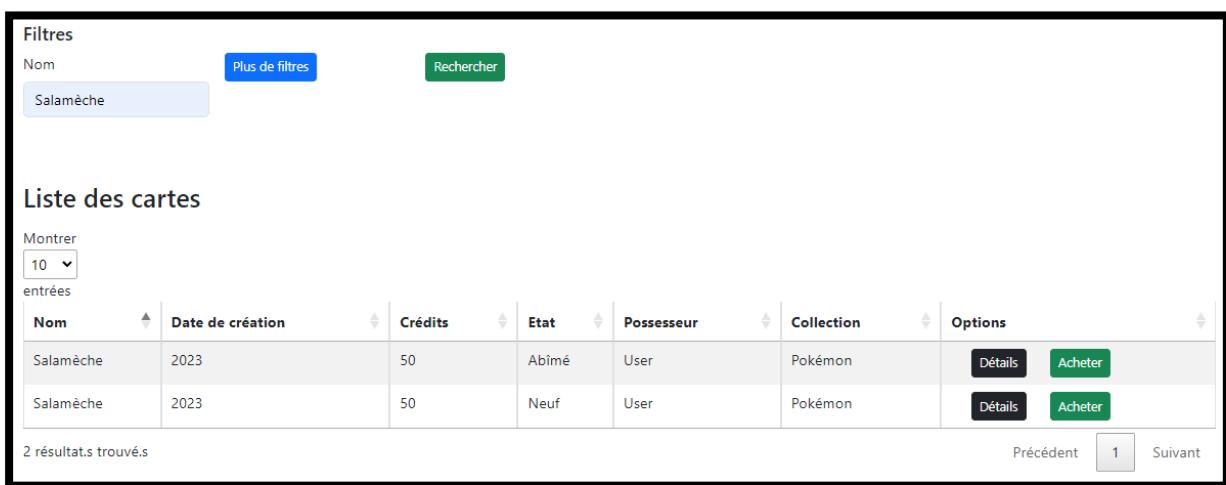
La méthode **convertConditions()** nous permet de convertir le tableau des états **\$conditions** en une chaîne de caractères prête à être utilisée dans une requête SQL. Elle parcourt le tableau en utilisant une boucle **foreach** et concatène la valeur entre '' (par exemple 'N' pour neuf) et ajoute une virgule et un espace puis retourne la chaîne de caractère formatée après avoir supprimé les deux derniers caractères de la chaîne. Cela nous permet d'obtenir une chaîne de valeurs séparées par des virgules que nous utilisons avec la clause **IN** dans la requête SQL de la méthode **sortCards()**. Il a été

nécessaire d'implémenter cette méthode pour gérer le cas où l'utilisateur sélectionne plusieurs états différents simultanément dans les filtres.

```
private function addWhereOrAnd($filter)
{
    $operator = "";
    if ($filter) {
        $operator .= " AND ";
    } else {
        $operator .= " WHERE ";
    }
    return $operator;
}
```

La méthode **addWhereOrAnd()** permet de générer un **AND** ou un **WHERE** lors de la construction de la requête SQL de la méthode **sortCards()** en fonction des filtres qui ont été sélectionné par l'utilisateur. A chaque fois que la valeur de la variable **\$filter** est **true**, la valeur renournée par la méthode sera **AND**. Si ce n'est pas le cas, la valeur renournée par la méthode sera **WHERE**.

Lors de la réalisation de la méthode **sortCards()**, nous avons constaté une certaine complexité qui ne nous a pas permis de bien sécurisé la requête SQL. Si le temps le permet, nous reviendrons sur ce point ultérieurement.



The screenshot shows a web application interface for filtering and listing cards. At the top left, there's a 'Filtres' section with a 'Nom' input field containing 'Salamèche', a 'Plus de filtres' button, and a 'Rechercher' button. Below this is a 'Liste des cartes' section. It includes a 'Montrer' dropdown set to '10 entrées'. The main area displays a table with two rows of card data:

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Salamèche	2023	50	Abîmé	User	Pokémon	<button>Détails</button> <button>Acheter</button>
Salamèche	2023	50	Neuf	User	Pokémon	<button>Détails</button> <button>Acheter</button>

At the bottom of the card list, it says '2 résultat.s trouvé.s' and has navigation buttons for 'Précédent', '1', and 'Suivant'.

**Filtres**

Nom	<input type="text" value="Salamèche"/>	<a href="#">Plus de filtres</a>	Collection	Etat	<a href="#">Rechercher</a>
			Collection	<input type="checkbox"/> Neuf <input type="checkbox"/> Occasion <input checked="" type="checkbox"/> Abîmé	

**Liste des cartes**

Montrer  entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Salamèche	2023	50	Abîmé	User	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>

1 résultat.s trouvé.s

Précédent  Suivant

**Filtres**

Nom	<input type="text"/>	<a href="#">Plus de filtres</a>	Collection	Etat	<a href="#">Rechercher</a>
			Pokémon	<input type="checkbox"/> Neuf <input type="checkbox"/> Occasion <input type="checkbox"/> Abîmé	

**Liste des cartes**

Montrer  entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Bulbizarre NIV.13	2007	60	Occasion	Admin	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Démolosse	2002	30	Neuf	Admin2	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Pharamp	2002	4	Neuf	Admin	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Salamèche	2023	50	Neuf	User	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Salamèche	2023	50	Neuf	User	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>

5 résultat.s trouvé.s

Précédent  Suivant

**Filtres**

Nom	<input type="text"/>	<a href="#">Plus de filtres</a>	Collection	Etat	<a href="#">Rechercher</a>
			One Piece	<input type="checkbox"/> Neuf <input checked="" type="checkbox"/> Occasion <input type="checkbox"/> Abîmé	

**Liste des cartes**

Montrer  entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>
Zorro	2022	15	Occasion	Admin2	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>

3 résultat.s trouvé.s

Précédent  Suivant

**Filtres**

Nom	<input type="text"/>	<a href="#">Plus de filtres</a>	Collection	Etat	<a href="#">Rechercher</a>
			Collection	<input checked="" type="checkbox"/> Neuf <input type="checkbox"/> Occasion <input type="checkbox"/> Abimé	

**Liste des cartes**

Montrer  entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Démolosse	2002	30	Neuf	Admin2	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Pharamp	2002	4	Neuf	Admin	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Salamèche	2023	50	Neuf	User	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Salamèche	2023	50	Neuf	User	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Yamato	2023	7	Neuf	Admin3	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>

5 résultat.s trouvé.s

Précédent  Suivant

**Filtres**

Nom	<input type="text"/>	<a href="#">Plus de filtres</a>	Collection	Etat	<a href="#">Rechercher</a>
			One Piece	<input type="checkbox"/> Neuf <input checked="" type="checkbox"/> Occasion <input type="checkbox"/> Abimé	

**Liste des cartes**

Montrer  entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Zorro	2022	15	Occasion	Admin2	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>

1 résultat.s trouvé.s

Précédent  Suivant

**Filtres**

Nom	<input type="text"/>	<a href="#">Plus de filtres</a>	Collection	Etat	<a href="#">Rechercher</a>
			Collection	<input checked="" type="checkbox"/> Neuf <input checked="" type="checkbox"/> Occasion <input type="checkbox"/> Abimé	

**Liste des cartes**

Montrer  entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Démolosse	2002	30	Neuf	Admin2	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Pharamp	2002	4	Neuf	Admin	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Salamèche	2023	50	Neuf	User	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Yamato	2023	7	Neuf	Admin3	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>
Bulbizarre NIV.13	2007	60	Occasion	Admin	Pokémon	<a href="#">Détails</a> <a href="#">Acheter</a>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>
Zorro	2022	15	Occasion	Admin2	One Piece	<a href="#">Détails</a> <a href="#">Acheter</a>

### 3.9 Mise en vente d'une carte

#### Ajout d'une carte

Nom :

Date de création :

Crédits :

Neuf     Occasion     Abîmé

Description :

Photo de la carte  
(format jpg) :

Aucun fichier choisi  
[Convertissez votre fichier au format jpg en cliquant ici](#)

Notre utilisateur Vendeur souhaite proposer l'une de ses cartes à la vente sur notre site. Pour ce faire, il doit se rendre sur la page Ajouter une carte. Il doit ensuite remplir un formulaire soumis à des contraintes de validations des données que nous définissons dans le fichier ValidateAddCardForm.php. Afin d'éviter des redondances, nous nous attardons uniquement sur les différences par rapport au formulaire de validation que nous avons précédemment analysé. Pour l'exemple, nous nous intéressons ici au champ **downloadImg** qui permet à l'utilisateur d'uploader une image qui sera enregistrée à la fois en local et en base de données.

## Contraintes du formulaire de validation des données pour l'ajout d'une carte

Champ	Type	Contrainte 1	Contrainte 2
<b>name</b>	text	Obligatoire	Chaîne de caractères de 1 à 45 caractères, pouvant contenir des tirets, des espaces et des apostrophes ainsi que des lettres accentuées.
<b>date</b>	text	Obligatoire	Chaîne de 4 caractères, uniquement des chiffres.
<b>credits</b>	text	Obligatoire	Chaîne de 1 à 3 caractères, uniquement des chiffres, dont le premier caractère n'est pas 0.
<b>condition</b>	radio	Obligatoire	-
<b>description</b>	textArea	Obligatoire	Chaîne de caractères qui n'excède pas 255 caractères.
<b>downloadImg</b>	file	Obligatoire	Seul le format jpg est accepté.
<b>collection</b>	select	Obligatoire	-

Nous partons du principe qu'une carte a une valeur maximum de 999 crédits et qu'elle ne peut pas en valoir 0.

```
$collections = $db->getAllCollections();

$errors = [];

if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    $result = validateAddCardForm($db);
    $errors = $result["errors"];
    $cardData = $result["cardData"];
    $imageData = addImages($_FILES, $db);

    // Si aucune erreur de validation
    // Cela signifie que les données sont propres et validées
    // Nous pouvons insérer les données en BD
    if (count($errors) === 0) {

        move_uploaded_file($imageData['fileTmpNameImg'], $imageData['uploadPathImg']);
        $cards = $db->addCard($_POST, $imageData, $_SESSION['idUser']);

        echo '<script>';
        echo 'window.onload = function() {';
        echo '    showSuccessMessage();';
        echo '}';
        echo '</script>';

    } else {
        if ($_POST) {
            $errorMessage= "Merci de bien remplir tous les champs en respectant les contraintes de validation";
        }
    }
}
```

Afin d'ajouter une carte, nous effectuons également une validation des données à la différence que cette fois-ci, nous devons également gérer le téléchargement de l'image avant d'ajouter les données en base de données. La méthode **move\_uploaded\_file()** déplace l'image du dossier d'envoi vers le dossier de réception et la méthode **addCard()** de la classe **Database** est appelée pour insérer les valeurs en base de données. Si l'ajout a réussi, nous affichons un message de succès et proposons à l'utilisateur d'ajouter une nouvelle carte ou de retourner à la page d'accueil. Sinon, nous lui affichons les erreurs contextuelles nécessaires.

```

function addImages($dataFiles, $db)
{
    $imageData = [];

    if (isset($dataFiles['downloadImg'])) {
        //Gestion du transfert de l'image
        //prends le dossier actuel
        $imageData["currentDirectory"] =.getcwd();
        //dossier vers lequel le fichier va être transféré
        $imageData["uploadDirectoryImg"] = "\img\photos";
        //Récupère le fichier
        $imageData["downloadImg"] = $dataFiles["downloadImg"];
        //Récupère le nom du fichier
        $imageData["fileNameImg"] = $dataFiles['downloadImg']['name'];
        //Récupère le nom temporaire du fichier
        $imageData["fileTmpNameImg"] = $dataFiles['downloadImg']['tmp_name'];
        //Reprends l'extension du fichier transféré
        $imageData["fileExtensionImg"] = strtolower(end(explode('.', $imageData["fileNameImg"])));
        //Définit l'extension du fichier après l'avoir récupérée
        $imageData["extensionImg"] = pathinfo($imageData["fileNameImg"], PATHINFO_EXTENSION);

        $nameId = $db->renameFile();
        $ImgNewName = "\Img_" . ($nameId + 1) . "." . $imageData["extensionImg"];
        $imageData["fileNameImg"] = $ImgNewName;

        //Définit le chemin final avec le nom du fichier où va être transférer le fichier en lui donnant un nom unique
        $imageData["uploadPathImg"] = $imageData["currentDirectory"] . $imageData["uploadDirectoryImg"] . $imageData["fileNameImg"];
    }

    return $imageData;
}

```

Afin de gérer le téléchargement de l'image, nous appelons la méthode **addImages()** de la page `addImages.php`. Celle-ci nous permet d'obtenir et de stocker toutes les informations nécessaires d'une image telle que le nom du fichier, le chemin de destination, la taille et l'extension dans le tableau **\$imageData**. Pour renommer le fichier de l'image téléchargé avant le téléchargement et l'insertion en base de données, nous appelons la méthode **renameFile()** de la classe **Database**. Nous le renommions ainsi toujours selon le modèle suivant `\img\photos\Img_21.jpg` qui correspond au chemin absolu du fichier suivi de l'**ID** de la carte qui est calculé à partir du dernier **ID** enregistré + 1.

```

public function renameFile()
{
    $query = "
        SELECT idCard FROM t_card ORDER BY idCard desc Limit 1";
    $req = $this->querySimpleExecute($query);
    $result = $this->formatData($req);
    return $result[0]['idCard'];
}

```

La fonction **renameFile()** récupère l'identifiant de la dernière carte enregistrée dans la table **t\_card** et retourne la valeur de cet **ID**. De cette façon, nous pouvons toujours renommer un fichier de façon unique selon le même modèle en partant du dernier **ID** enregistré pour une carte.

```
$imageData = addImages($_FILES, $db);
```

Nous définissons la variable **\$imageData** qui va récupérer les informations de l'image à uploader de la page addImages.php.

```
$downloadImg = $imageData['downloadImg']['size'] ?? '';
```

Nous vérifions ainsi si le fichier chargé a une valeur **size** existante. Si ce n'est pas le cas, nous considérons qu'aucun fichier n'a été téléchargé.

```
if (!$downloadImg) {
    $errors['downloadImg'] = ERROR_IMAGE_REQUIRED;
} elseif (!in_array($imageData['extensionImg'], ['jpg', 'JPG'])) {
    $errors['downloadImg'] = ERROR_IMAGE_EXTENSION;
}
```

```
const ERROR_IMAGE_REQUIRED = "Veuillez ajouter l'image de la carte";
const ERROR_IMAGE_EXTENSION = "Seul le format jpg est accepté";
```

Lors de validation des données du champ **downloadImg** nous vérifions si l'utilisateur a bien chargé l'image avant de soumettre le formulaire. Si ce n'est pas le cas, nous lui affichons une erreur contextuelle lui demandant de le faire. Si l'utilisateur a bien chargé une image, nous vérifions cette fois-ci le format du fichier afin de n'autoriser que les fichiers au format **JPG**. Si l'utilisateur tente de télécharger une image qui n'est pas au format **JPG**, une erreur contextuelle lui est affichée en lui indiquant qu'aucun autre format n'est pris en charge.

```
<p>
    <label for="downloadImg">Photo de la carte (format jpg) :</label>
    <br>
    <input type="file" name="downloadImg" id="downloadImg" />
    <br>
    <a href="https://convertio.co/fr/convertisseur-jpg/">
        Convertissez votre fichier au format jpg en cliquant ici</a>
    <span id="show-error">
        <?= array_key_exists("downloadImg", $errors) && $errors["downloadImg"] ?>
        <p style="color:red;">' . $errors["downloadImg"] . '</p>' : '' ?>
    </span>
</p>
<br>
```

Finalement, nous créons un formulaire permettant aux utilisateurs de télécharger une photo au format JPG pour une carte. Si le fichier n'est pas au bon format, une redirection vers un convertisseur en ligne est proposé.

## Ajout d'une carte

Merci de bien remplir tous les champs en respectant les contraintes de validation

Nom :

Veuillez renseigner le champ nom de la carte

Date de création :

Veuillez renseigner le champ date de la carte

Crédits :

Veuillez renseigner le champ credits de la carte

Neuf

Occasion

Abîmé

Veuillez renseigner le champ état de la carte

Description :

Veuillez renseigner le champ description de la carte

Photo de la carte

(format jpg) :

Aucun fichier choisi

[Convertissez votre fichier au format jpg en cliquant ici](#)

Veuillez ajouter l'image de la carte

Veuillez renseigner le champ collection de la carte

Nous constatons que lorsque nous ne renseignons aucun champ, les erreurs contextuelles indiquant les champs obligatoires à remplir s'affichent correctement.

## Ajout d'une carte

Merci de bien remplir tous les champs en respectant les contraintes de validation

Nom :

Merci de renseigner une chaîne de caractères valide, de 1 à 45 caractères pouvant contenir des tirets, des espaces et des apostrophes, ainsi que des lettres accentuées.

Date de création :

Merci de renseigner une chaîne de caractères qui sont uniquement des chiffres et uniquement 4 caractères.

Crédits :

Merci de renseigner une chaîne de caractères qui composée de 1 à 3 chiffres et dont le premier caractère n'est pas 0

Neuf  Occasion  Abîmé

Description :

Merci de renseigner une chaîne de caractères qui n'excède pas 255 caractères

Photo de la carte

(format jpg) :

python.exe

[Convertissez votre fichier au format jpg en cliquant ici](#)

Seul le format jpg est accepté

Nous constatons que lorsque nous ne respectons pas les contraintes de validation liées à la taille ou aux types de valeurs permises, les erreurs contextuelles en rapport s'affichent correctement. Les valeurs entrées par l'utilisateur lui sont retournées, hormis le fichier pour des raisons de sécurité.

```
public function addCard($card, $imgData, $idUser)
{
    $query = "
        INSERT INTO t_card (carName, carDate, carCredits, carCondition,
        carDescription, carPhoto, fkUser, fkCollection)
        VALUES (:name, :date, :credits, :condition,
        :description, :photo, :fkUser, :fkCollection);
    ";

    $replacements = [
        'name' => $card['name'],
        'date' => intval($card['date']),
        'credits' => intval($card['credits']),
        'condition' => $card['condition'],
        'description' => $card['description'],
        'photo' => $imgData['uploadDirectoryImg'] . $imgData['fileNameImg'],
        'fkUser' => $idUser,
        'fkCollection' => $card['collection']
    ];

    $response = $this->queryPrepareExecute($query, $replacements);
}
```

La méthode **addCard()** permet d'insérer les données d'une nouvelle carte dans la table **t\_card** de la base de données grâce à une requête **SQL**. Elle

prend en compte les informations de la carte, les données de l'image et l'**ID** de l'utilisateur.

### Ajout d'une carte

Nom :

Date de création :

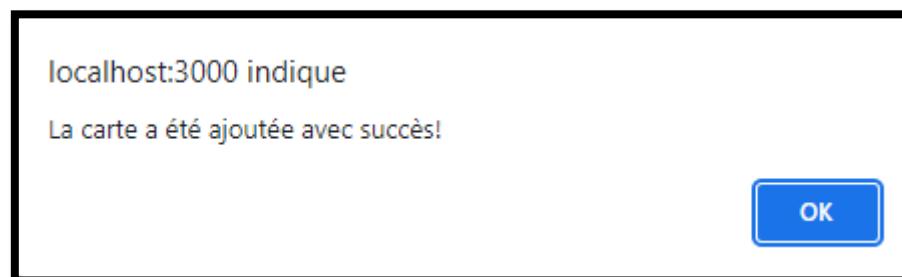
Crédits :

Neuf       Occasion       Abîmé

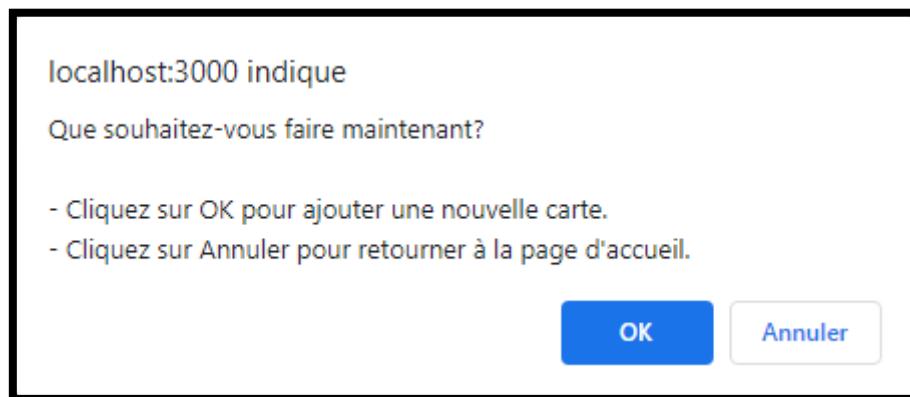
Description :

Photo de la carte  
(format jpg) :  
  
[Convertissez votre fichier au format jpg en cliquant ici](#)

L'utilisateur Vendeur ajoute maintenant sa première carte sur le site.



Les champs ont été rempli conformément aux contraintes de validation des données, la carte est donc correctement ajoutée.



N'ayant pas d'autres cartes à mettre en vente, il clique sur annuler et retourne à la page d'accueil.

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Pikachu	2023	5	Neuf	Vendeur	Pokémon	<button>Détails</button> <button>Modifier</button> <button>Supprimer</button>

Nous pouvons constater que la carte de l'utilisateur Vendeur apparaît bien dans le tableau des cartes en vente et qu'il dispose des droits de modification et de suppression étant donné qu'il s'agit de sa propre carte.

idCard	carName	carDate	carCredits	carCondition	carDescription	carIsAvailable	carPhoto	fkUser
47	Pikachu	2023	5	N	Test d'ajout de carte en vente	1	\img\photos\Img_47.JPG	9

Nous pouvons constater que la carte a bien été ajoutée en base de données. Le fichier a bien été renommé selon le modèle souhaité.

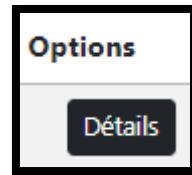
```




```

Finalement, nous pouvons constater que l'image a bien été téléchargée et renommée en local également.

### **3.10 Affichage des détails d'une carte**



Après avoir ajouté sa carte, l'utilisateur Vendeur souhaite maintenant consulter les détails de la carte qu'il vient de mettre en vente. Pour ce faire, il clique sur le bouton *Détails* correspondant à sa carte.

```
<a class="btn btn-dark btn-sm"
href="cardDetails.php?idCard=<?php echo $card["idCard"] ?>">Détails</a>
```

Pour que l'utilisateur puisse accéder aux informations d'une carte spécifique, nous générerons un lien hypertexte qui redirige vers la page *cardDetails.php* avec l'**ID** de la carte comme paramètre d'URL.

```
//Récupère les informations de la carte via son id qui se trouve dans l'url
$card = $db->getOneCard($_GET["idCard"]);           davassayah, il y a 7 jours •
```

Afin d'afficher les détails d'une carte spécifique, nous appelons la méthode **getOneCard()** de la classe **Database**.

```
public function getOneCard($id)
{
    //avoir la requête sql pour une carte (utilisation de l'id)
    $query = "
        SELECT
            t_card.*,
            t_user.useLogin AS carUserLogin
        FROM t_card
        LEFT JOIN t_collection ON t_collection.idCollection = t_card.fkCollection
        LEFT JOIN t_user ON t_user.idUser = t_card.fkUser
        WHERE t_card.idCard = :id
    ";
    //appeler la méthode pour executer la requête
    $bind = array('id' => $id);
    $req = $this->queryPrepareExecute($query, $bind);
    //appeler la méthode pour avoir le résultat sous forme de tableau
    $oneCard = $this->formatData($req);
    //retourne la carte
    return $oneCard[0];
}
```

La méthode **getOneCard()** permet de récupérer toutes les informations d'une carte spécifique à partir de son ID en effectuant une jointure avec la table **t\_collection**.

[localhost:3000/cardDetails.php?idCard=47](http://localhost:3000/cardDetails.php?idCard=47)

De cette façon, nous pouvons récupérer l'ID d'une carte spécifique et afficher toutes les informations que nous souhaitons pour celle-ci.

```


<h1>Informations de la carte : </h1>
    <div class="photo">
        <img height="600em" src=<?php echo $card['carPhoto'] ?>>
    </div>
    <?php
    echo '<div style="font-size: 1.7em; margin-bottom: 20px;">';
    echo "Nom de la carte : " . $card["carName"] . "<br>" .
        "Date de création : " . $card["carDate"] . "<br>" .
        "Crédits : " . $card["carCredits"] . "<br>";
    if ($card["carCondition"] == "N") {
        echo "Etat : Neuf";
    } else if ($card["carCondition"] == "O") {
        echo "Etat : Occasion";
    } else if ($card["carCondition"] == "A") {
        echo "Etat : Abimé";
    }
    echo "<br>";
    echo "Description : " . "<br>" . $card["carDescription"];
    echo '</div>' . "<br>";
?>
    <div class="actions" style="margin-bottom: 20px;">
        <?php
        if ($_SESSION['idUser'] == $card['fkUser']) {
            echo "Actions :" . "<br>" .
                '<button class="btn btn-warning btn-sm" onclick="location.href='updateCard.php?idCard=<?php echo $card["idCard"]'; ?>">
                    Modifier
                </button>
                <button class="btn btn-danger btn-sm" onclick="confirmDelete(<?php echo $card["idCard"]; ?>)">
                    Supprimer
                </button>
        <?php
        } else {
            echo "Possesseur : " . $card["carUserLogin"] . "<br>";
        }
        <p>
            <input type="submit" value="Acheter">
        </p>
    <?php
    }
}


```

Pour une carte en particulier, nous affichons l'image, le nom, la date de création, sa valeur en crédits, son état et sa description. Si l'utilisateur possède cette carte, il a la possibilité de la modifier ou de la supprimer. S'il ne la possède pas, le possesseur de la carte est affiché et il a la possibilité de l'acheter.

**Informations de la carte :**

Nom de la carte : Pikachu  
 Date de création : 2023  
 Crédits : 5  
 Etat : Neuf  
 Description :  
 Test d'ajout de carte en vente

Actions : [Modifier](#) [Supprimer](#)



Après avoir été redirigé vers la page `cardDetails.php` l'utilisateur Vendeur constate que toutes les informations qu'il a soumises sont conformes aux valeurs affichées.

### 3.11 Modification d'une carte



Afin de rendre la vente de sa carte plus attractive, il souhaite modifier les informations de sa carte. Pour ce faire, il clique sur le bouton **Modifier**.

```
<a class="btn btn-warning btn-sm"
  href="updateCard.php?idCard=<?php echo $card["idCard"]; ?>">Modifier</a>
```

Nous procédons de la même façon que pour afficher les détails d'une carte pour arriver sur la page de modification d'une carte spécifique en récupérant son ID via l'URL.

```
$errors = [];

if ($_SERVER['REQUEST_METHOD'] == 'POST') {

    $imageData = normalizeImgData($_FILES, $oneCard);
    $result = validateUpdateCardForm($imageData);
    $errors = $result["errors"];
    $cardData = $result["cardData"];

    if (count($errors) > 0) {
        // Si le compte des erreurs est supérieur à 0, on affiche les erreurs
        echo "Merci de vérifier que tous les champs sont bien remplis correctement et que l'extension du fichier est jpg";
    } else {
        if ($_POST) {
            // si le formulaire a été envoyé, alors on met à jour la carte
            if ($imageData['downloadImg'][name] != '') {
                // On supprime l'ancienne image
                deletePreviousImg($imageData);
                // Si une image a été sélectionnée, on la déplace et on met à jour la carte avec la nouvelle image
                move_uploaded_file($imageData['fileTmpNameImg'], $imageData['filePath']);
                $db->updateCardById($_GET['idCard'], $cardData);
            } else {
                // Sinon, on met à jour la carte sans changer l'image
                $db->updateCardById($_GET['idCard'], $cardData);
            }
            // On redirige vers la page d'accueil
            header('Location: index.php');
        } else {
            // Si le formulaire n'a pas été envoyé, on affiche un message d'erreur
            echo "Merci de remplir le formulaire.";
        }
    }
}
}
```

Pour la modification, nous procédons d'une façon quasiment similaire à celle de l'ajout d'une carte. La différence principale étant que nous appelons le formulaire de validation du fichier `validateUpdateCardForm.php`. Nous appelons également la méthode **normalizeImgData()** pour définir les données de l'image à mettre à jour.

Si aucune erreur ne survient lors de la validation des données, nous vérifions si une image a été sélectionnée. Si c'est le cas, nous récupérons ses informations, supprimons l'image déjà existante avec la fonction **deletePreviousImg()** et déplaçons la nouvelle image pour qu'elle la remplace. La méthode **updateCardById()** insère ensuite toutes les modifications réalisées par l'utilisateur en base de données.

Si l'image une image n'a pas été sélectionnée, les informations de la carte sont mises à jour sans gérer la gestion de l'image.

```
// Vérifie s'il n'y a que "downloadImg" comme erreur dans le tableau
if (count($errors) === 1 && isset($errors['downloadImg'])) {
    // Si c'est le cas, supprime l'erreur "downloadImg"
    $errors = [];
}
```

Dans la fonction **validateUpdateImageForm()**, nous spécifions que si la seule erreur survenue lors de la validation est qu'aucune image n'a été chargée par

l'utilisateur, alors il n'y a aucune erreur. Cela nous permet de gérer le cas où un utilisateur veut modifier les informations de sa carte sans changer d'image.

Les contraintes de validation des données pour la modification d'une carte sont donc les mêmes que celles pour l'ajout d'une carte, hormis pour le champ **downloadImg** n'est pas obligatoire.

```
/*
 * Fonction permettant de supprimer l'ancienne image sur le serveur
 * @param array $imageData | Contient les données de l'image à télécharger déjà normalisées
 */
function deletePreviousImg($imageData) {
    if (file_exists($imageData["filePath"]) and ($imageData["extensionImg"] == "jpg")) {
        unlink($imageData["filePath"]);
    }
}
```

Dans la fonction **deletePreviousImg()**, nous utilisons les fonctions **file\_exists()** et **unlink()** respectivement pour vérifier si une image a été téléchargée par l'utilisateur et, si c'est le cas, supprimer l'image déjà existante.

```
<p>
    <label for="name">Nom :</label>
    <input type="text" name="name" id="name" value=<?php echo isset($_POST['name']) ? htmlspecialchars($_POST['name']) : $oneCard['carName']; ?>>
    <span id="show-error">
        <?= array_key_exists("name", $errors) && $errors["name"] ?>
            <p style="color:red;">' . $errors["name"] . '</p>' : '' ?>
    </span>
</p>
```

Finalement, nous faisons en sorte que lorsque l'utilisateur arrive sur la page `updateCard.php`, les informations de la carte enregistrées en base de données lui soient retournées par défaut.

```

public function updateCardById($id, $card)
{
    $query = "
        UPDATE
            t_card
        SET
            carName = :name,
            carDate = :date,
            carCredits = :credits,
            carCondition = :condition,
            carDescription = :description,
            carPhoto = :imgPath,
            fkCollection = :collection
        WHERE
            idCard = :id
    ";

    $card["id"] = $id;
    $this->queryPrepareExecute($query, $card);
}

```

La méthode **updateCardByID()** permet de mettre à jour les informations d'une carte dans grâce à une requête SQL de modification dans la base de données. Elle prend l'ID de la carte spécifique et les nouvelles informations ajoutées par l'utilisateur afin de modifier les données de la table **t\_card** de la base de données.

### Modifier une carte

Nom :

Année de création :

Crédits :

Neuf     Occasion     Abimé

Pokémon

Description :

Photo de la carte  
(format jpg) :

Aucun fichier n'a été sélectionné  
[Convertissez votre fichier au format jpg en cliquant ici](#)



Nous constatons que les informations enregistrées en base de données sont bien retournées à l'utilisateur.

**Modifier une carte**Nom : 

Veuillez renseigner le champ nom de la carte

Année de création : 

Veuillez renseigner le champ date de la carte

Crédits : 

Veuillez renseigner le champ credits de la carte

 Neuf     Occasion     Abîmé

Collection ▾

Veuillez renseigner le champ collection de la carte

Description :

Veuillez renseigner le champ description de la carte

Photo de la carte

(format jpg) :

Choisir un fichier | Aucun fichier n'a été sélectionné

[Convertissez votre fichier au format jpg en cliquant ici](#)

Veuillez ajouter l'image de la carte



Lorsque nous ne renseignons pas les champs obligatoires, nous constatons que les bonnes erreurs contextuelles s'affichent. L'ajout d'une image est considéré comme obligatoire car il ne s'agit pas de la seule erreur existante qui a été stockée lors de la validation des données.

**Modifier une carte**Nom : 

Merci de renseigner une chaîne de caractères valide, de 1 à 45 caractères pouvant contenir des tirets, des espaces et des apostrophes, ainsi que des lettres accentuées.

Année de création : 

Merci de renseigner une chaîne de caractères qui sont uniquement des chiffres et uniquement 4 caractères.

Crédits : 

Merci de renseigner une chaîne de caractères qui composée de 1 à 3 chiffres et dont le premier caractère n'est pas 0

 Neuf     Occasion     Abîmé

Pokémon ▾

Merci de renseigner une chaîne de caractères qui n'excède pas 255 caractères

Photo de la carte

(format jpg) :

Choisir un fichier | Aucun fichier n'a été sélectionné

[Convertissez votre fichier au format jpg en cliquant ici](#)

Veuillez ajouter l'image de la carte



Nous constatons que lorsque les champs ne respectent pas les contraintes de validation spécifiques, les erreurs contextuelles correspondantes s'affichent correctement.

### Modifier une carte

Nom :

Année de création :

Crédits :

Neuf     Occasion     Abîmé

Pokémon

Description :

Photo de la carte (format jpg) :  
 Pikachu.jpg  
[Convertissez votre fichier au format jpg en cliquant ici](#)



L'utilisateur Vendeur modifie la valeur en crédits de sa carte à 12, indique une nouvelle description Petite souris et souhaite modifier l'image de sa carte.

### Informations de la carte :

Nom de la carte : Pikachu

Date de création : 2023

Crédits : 12

Etat : Neuf

Description :

Petite souris

Actions :



Après avoir soumis le formulaire, il est redirigé vers la page d'accueil si la modification a bien eu lieu. En cliquant sur l'affichage des détails de la carte qu'il vient de modifier, nous pouvons constater que les modifications ont bien été appliquées.

### 3.12 Suppression d'une carte

**Supprimer**

L'utilisateur Vendeur souhaite finalement supprimer la carte qu'il a mise en vente.

```
<a class="btn btn-danger btn-sm"
 href="javascript:confirmDelete(<?php echo $card["idCard"] ?>)">Supprimer</a>
```

Afin de gérer la suppression d'une carte, nous déclenchons une fonction JavaScript **confirmDelete()**. Celle-ci permet de confirmer et effectuer la suppression de la carte lors du clique sur le bouton Supprimer.

```
function confirmDelete(cardId) {
    if (confirm("Êtes-vous sûr de vouloir supprimer la carte?") === true) {
        window.location.href = window.location.href + '?idCard=' + cardId;
    }
}
```

La fonction **confirmDelete()** nous permet d'afficher une boîte de dialogue de confirmation pour la suppression de la carte. Si l'utilisateur clique sur ok, l'ID de la carte est ajouté à l'URL avant de le rediriger vers cette nouvelle URL.

```
if (isset($_GET['idCard']) and $id = $_GET['idCard']) {
    $db->deleteCardById($id);
    header('Location: index.php');
}
```

Sur la page *index.php*, nous vérifions et récupérons l'ID de l'URL et le comparons à l'ID enregistré en base de données. Si c'est le cas, la méthode **deleteCardById()** supprime la carte correspondant à cet ID dans la base de données puis il est redirigé vers la page d'accueil.

```
public function deleteCardById($id)
{
    $query = "
        DELETE FROM t_card
        WHERE idCard = :id
    ";
    $replacements = [ 'id' => $id];
    $this->queryPrepareExecute($query, $replacements);
}
```

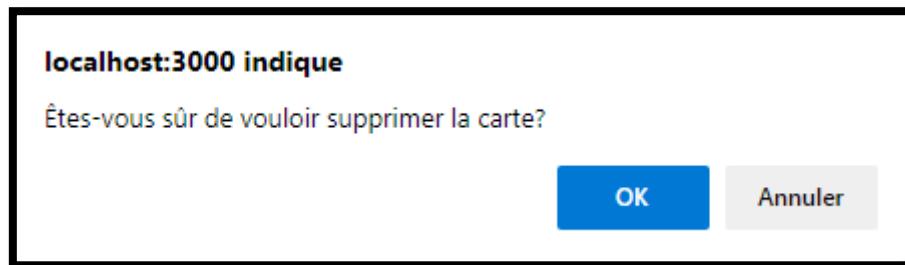
La méthode **deleteCardById()** permet de supprimer les enregistrements de la table **t\_card** lorsque la colonne **idCard** correspond à la valeur de l'ID en paramètre.

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Pikachu	2023	12	Neuf	Vendeur	Pokémon	<button>Détails</button> <button>Modifier</button> <button>Supprimer</button>
Bulbizarre NIV.13	2007	60	Occasion	Admin	Pokémon	<button>Détails</button> <button>Acheter</button>
Démolosse	2002	30	Neuf	Admin2	Pokémon	<button>Détails</button> <button>Acheter</button>
Luffy	2023	500	Abîmé	Admin	One Piece	<button>Détails</button> <button>Acheter</button>
Nami	2023	40	Abîmé	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Pharamp	2002	4	Neuf	Admin	Pokémon	<button>Détails</button> <button>Acheter</button>
Salamèche	2023	50	Abîmé	User	Pokémon	<button>Détails</button> <button>Acheter</button>
Salamèche	2023	50	Neuf	User	Pokémon	<button>Détails</button> <button>Acheter</button>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>

12 résultat.s trouvé.s

Précédent 1 2 Suivant

Nous pouvons constater qu'avant la suppression, la carte est disponible à la vente.



Dès lors que l'utilisateur clique sur le bouton OK la fenêtre de confirmation de suppression s'affiche.

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Bulbizarre NIV.13	2007	60	Occasion	Admin	Pokémon	<button>Détails</button> <button>Acheter</button>
Démolosse	2002	30	Neuf	Admin2	Pokémon	<button>Détails</button> <button>Acheter</button>
Luffy	2023	500	Abîmé	Admin	One Piece	<button>Détails</button> <button>Acheter</button>
Nami	2023	40	Abîmé	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Pharamp	2002	4	Neuf	Admin	Pokémon	<button>Détails</button> <button>Acheter</button>
Salamèche	2023	50	Abîmé	User	Pokémon	<button>Détails</button> <button>Acheter</button>
Salamèche	2023	50	Neuf	User	Pokémon	<button>Détails</button> <button>Acheter</button>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Trafalgar Law	2022	12	Occasion	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>
Yamato	2023	7	Neuf	Admin3	One Piece	<button>Détails</button> <button>Acheter</button>

11 résultat.s trouvé.s

Précédent 1 2 Suivant

Nous pouvons constater que la carte a bien été supprimée des cartes disponibles à la vente.

### **3.13 Système de transaction des points de crédits**

#### **3.13.1 Ajout d'une carte au panier**

```
if (!isset($_SESSION['panier'])) {
    $_SESSION['panier'] = [];
}
```

Afin de gérer l'ajout d'articles au panier, lorsqu'un utilisateur se connecte à l'application, un panier de session est automatiquement créé dans le cas où celui-ci n'existe pas déjà.



```
<a class="btn btn-success btn-sm"
 href="javascript:confirmBuy(<?php echo $card["idCard"] ?>)">Acheter</a>
```

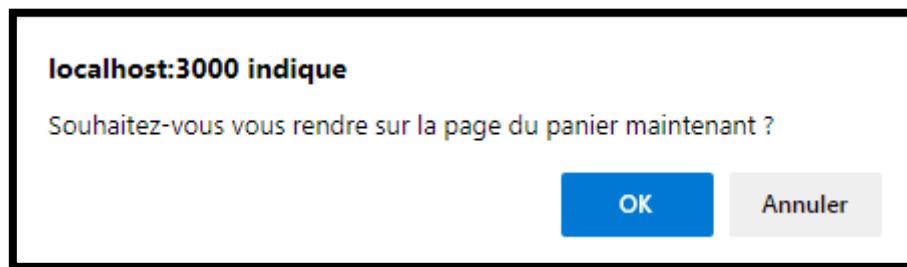
Dès lors que l'utilisateur clique sur le bouton acheter, nous appelons la fonction javaScript **confirmBuy()**.

```
function confirmBuy(cardId) {
    if (confirm("Êtes-vous sûr de vouloir ajouter cette carte au panier ?")) {
        // Ajoute la carte au panier en redirigeant vers la page index.php avec l'identifiant de la carte
        window.location.href = "index.php?idCardToAddInCart=" + cardId;
        if (confirm("Souhaitez-vous vous rendre sur la page du panier maintenant ?")) {
            // Redirige vers la page cart.php si l'utilisateur clique sur "Oui"
            window.location.href = "cart.php";
        }
    } else {
        // Redirige vers la page index.php sans ajouter la carte au panier
        window.location.href = "index.php";
    }
}
```

La fonction **confirmBuy()** permet de gérer l'ajout d'une carte au panier en une ou deux étapes en fonction des choix de l'utilisateur et le redirige en conséquence. Dès lorsqu'il clique sur le bouton acheter, une boîte de dialogue de confirmation apparaît :



Si l'utilisateur clique sur OK la carte est ajoutée au panier. Pour ce faire, nous redirigeons la page vers `index.php?idCardToAddInCart=` suivi de l'ID de la carte spécifique.

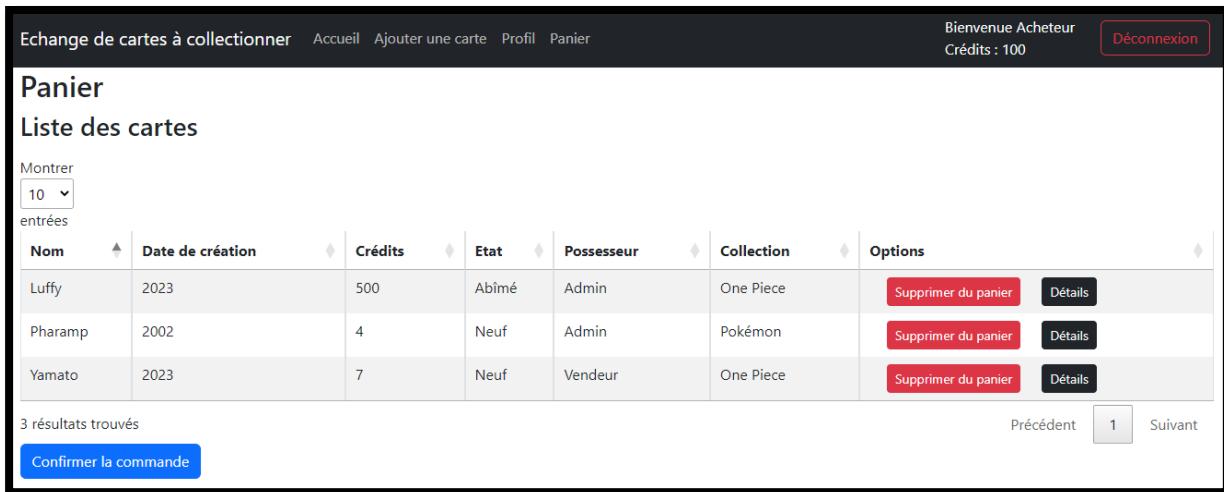


Une deuxième boîte de dialogue s'ouvre ensuite pour proposer à l'utilisateur d'accéder directement à son panier. S'il clique sur OK, la redirection vers le panier s'effectue, sinon une redirection vers la page d'accueil s'effectue.

Dans le cas où l'utilisateur clique sur Annuler lors de la première boîte de dialogue, la carte n'est pas ajoutée au panier et une redirection vers la page index.php a lieu.

Nous avons décidé que le statut de disponibilité d'une carte n'est modifié que lorsque l'acheteur confirme la commande et non pas dès l'ajout d'une carte au panier. Cela dans le but de ne pas pénaliser d'autres éventuels acheteurs en bloquant des articles dans un panier sans que l'utilisateur aille au bout du processus de commande.

### 3.13.2 Confirmation de la commande



The screenshot shows a user interface for managing a shopping cart. At the top, there are links for 'Accueil', 'Ajouter une carte', 'Profil', and 'Panier'. On the right, it says 'Bienvenue Acheteur' and 'Crédits : 100', with a 'Déconnexion' button. Below this, the word 'Panier' is highlighted in blue. A sub-header 'Liste des cartes' is displayed. A dropdown menu 'Montrer' is set to '10 entrées'. The main area is a table with the following columns: Nom, Date de création, Crédits, Etat, Possesseur, Collection, and Options. The table contains three rows of data:

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Luffy	2023	500	Abîmé	Admin	One Piece	<button>Supprimer du panier</button> <button>Détails</button>
Pharamp	2002	4	Neuf	Admin	Pokémon	<button>Supprimer du panier</button> <button>Détails</button>
Yamato	2023	7	Neuf	Vendeur	One Piece	<button>Supprimer du panier</button> <button>Détails</button>

Below the table, it says '3 résultats trouvés'. At the bottom left is a blue button labeled 'Confirmer la commande'. Navigation buttons 'Précédent', '1', and 'Suivant' are at the bottom right.

En dehors de la redirection proposée lors de l'achat d'une carte, l'utilisateur a accès en tout temps à la page `cart.php` via l'onglet Panier du header.

```
<?php foreach ($_SESSION['panier'] as $card) { ?>
    <tr>
        <td><?php echo $card["carName"] ?></td>
        <td><?php echo $card["carDate"] ?></td>
        <td><?php echo $card["carCredits"] ?></td>
        <td><?php if ($card["carCondition"] == "N") {
            echo "Neuf";
        } else if ($card["carCondition"] == "O") {
            echo "Occasion";
        } else if ($card["carCondition"] == "A") {
            echo "Abîmé";
        } ?></td>
        <td><?php echo $card["carUserLogin"] ?></td>
        <td><?php echo $card["carCollectionName"] ?></td>
        <td class="containerOptions">
            <!--Affiche différentes fonctionnalités selon que l'utilisateur
            soit connecté en tant qu'utilisateur ou en tant qu'admin-->
            <?php if (isset($_SESSION['userConnected']) &&
                $_SESSION['userConnected'] == ('user' or 'admin')) { ?>
                <?php if (isset($_SESSION['userConnected'])) [ ?>
                    <a class="btn btn-danger btn-sm"
                        href="javascript:confirmDeleteFromCart(<?php echo $card["idCard"] ?>)">Supprimer du panier</a>
                    <a class="btn btn-dark btn-sm" href="cardDetails.php?idCard=<?php echo $card["idCard"] ?>">Détails</a>
                

```

Nous affichons les détails des cartes présentes dans le panier d'un utilisateur grâce à un tableau en parcourant son panier de session `$_SESSION['panier']` à l'aide d'une boucle `foreach`. Pour chaque carte dans le panier, nous proposons à l'utilisateur d'accéder aux détails de la carte ou de la supprimer de son panier.

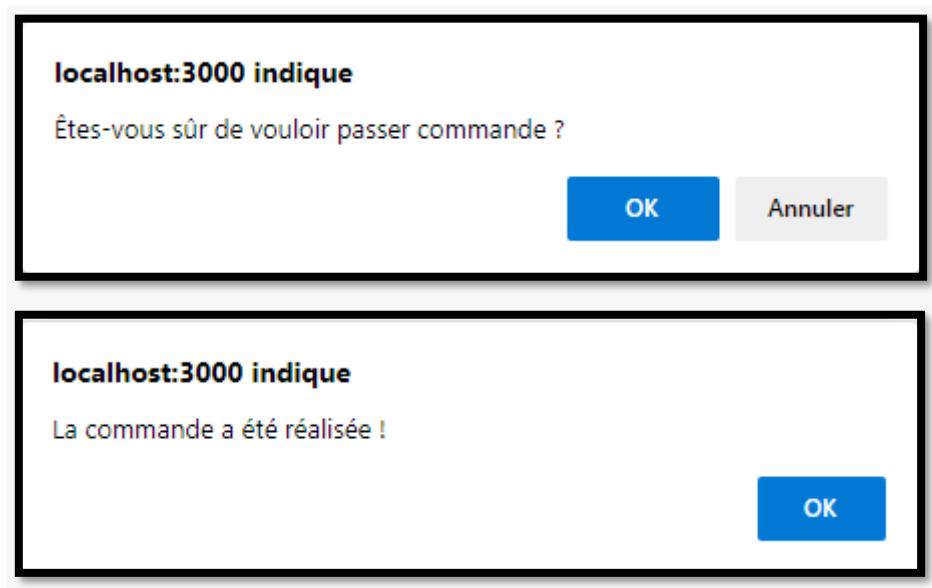
## Confirmer la commande

Lorsque l'utilisateur a ajouté au moins une carte à son panier, il a la possibilité de passer commande en cliquant sur le bouton Confirmer la commande.

```
<?php if (count($_SESSION['panier']) > 0) { ?>
    <a class="btn btn-primary" href="cart.php?buy=true"
       role="button" onclick="confirmOrder()">Confirmer la commande</a>
</php ?>
```

```
function confirmOrder() {
    if (confirm("Êtes-vous sûr de vouloir passer commande ?")) {
        alert("La commande a été réalisée !");
    }
}
```

Nous vérifions si le panier contient des éléments et affichons un bouton **Confirmer la commande** pointant vers la page `cart.php` avec le paramètre **buy=true** dans l'URL si c'est le cas. Nous déclenchons ensuite la fonction JavaScript **confirmOrder()** permettant d'afficher une confirmation de commande à l'utilisateur.



**Supprimer du panier**

A tout moment, l'utilisateur a la possibilité de supprimer une des cartes qu'il a ajouté au panier en cliquant sur le bouton **Supprimer du panier**. Lors du clique, nous déclenchons la fonction **confirmDeleteFromCart()**.

```
href="javascript:confirmDeleteFromCart(<?php echo $card["idCard"] ?>)">Supprimer du panier</a>
```

```
function confirmDeleteFromCart(cardId) {
    if (confirm("Êtes-vous sûr de vouloir supprimer la carte de votre panier?") === true) {
        window.location.href = "cart.php?idCard=" + cardId;
    }
}
```

La fonction **confirmDeleteFromCart()** affiche une boîte de dialogue demandant à l'utilisateur de confirmer la suppression d'une carte de son panier. Si l'utilisateur clique sur OK, la valeur de la condition passe à *true* et une redirection vers la page *cart.php* est effectuée en ajoutant l'URL de la carte spécifique grâce au paramètre **cardId**. De cette façon, la carte spécifique est supprimée du panier.

```
$error = null;
if (isset($_GET['buy']) and $_GET['buy'] == 'true') {
    $isOrderCreated = $db->createOrder($_SESSION['idUser'], array_keys($_SESSION['panier']));
    if ($isOrderCreated) {
        $_SESSION['useCredits'] = $db->getOneUser($_SESSION['idUser'])['useCredits'];
        $_SESSION['panier'] = [];
        header('Location: userProfile.php?idUser=' . $_SESSION["idUser"]);
    } else {
        $error = "L'utilisateur n'a pas suffisamment de credits pour passer la commande.";
    }
}
```

Nous vérifions si l'utilisateur a cliqué sur le bouton *Confirmer la commande* en récupérant la valeur de **buy**. Si la valeur est *true*, on considère que la commande a été passée et on appelle la méthode **createOrder()** de la classe **Database** pour nous permettre de créer une commande à partir du panier de session de l'utilisateur. Si lors de la création de la commande l'acheteur dispose d'assez de crédits, son total de crédits est débité du montant de la commande. Le montant de la transaction est mis en attente tant que l'acheteur ne confirme pas la réception des cartes depuis sa page de profil. Finalement, il est redirigé sur sa page de profil.

Si l'acheteur n'a pas suffisamment de crédits, un message d'erreur lui est affiché pour l'en informer et la commande n'est pas créée.

### 3.13.3 Création de la commande

```

public function createOrder($idBuyer, $cardIds)
{
    // Récupère les cartes et les organise par propriétaire
    $userCards = $this->getAndOrderCardsByOwner($cardIds);

    // Vérifie si l'utilisateur a suffisamment de crédits pour passer la commande
    if ($this->hasUserEnoughCredits($userCards) == false) {
        return false;
    }

    // Modifie la disponibilité des cartes
    $this->toggleAvailabilityOfCards($cardIds, 0);

    foreach ($userCards as $idCardOwner => $cards) {
        // Crée une transaction pour l'acheteur
        $transactionId = $this->createTransaction($idBuyer);

        // Pour chaque carte, assigne la carte à la commande,
        // met à jour les crédits de l'acheteur,
        // se met à jour les crédits de session et modifie la nouvelle valeur en DB
        foreach ($cards as $card) {
            $this->assignCardToOrder($card['idCard'], $transactionId);
            $this->substrCreditsOfBuyer($card['idCard'], $idBuyer);
            $_SESSION['useCredits'] = $this->getOneUser($idBuyer)['useCredits'];
        }
    }

    return true;
}

```

La méthode **createOrder()** reçoit deux paramètres :

- 1) **\$idBuyer** correspondant à l'ID de l'acheteur.
- 2) **\$cardIds** correspondant à un tableau où sont stockés les identifiants des cartes à commander.

Pour commencer, nous appelons la méthode **getAndOrderCardsByOwner()** afin de récupérer les informations des cartes correspondant aux ID dans le but de les organiser par propriétaire.

Ensuite, nous vérifions si l'acheteur dispose d'assez de crédits sur son compte pour créer la commande en appelant la méthode **hasUserEnoughCredits()**. Si l'acheteur ne dispose pas d'assez de crédits, la valeur renournée par la méthode est **false**.

Dans le cas où l'acheteur a suffisamment de crédits, nous appelons la méthode **toggleAvailabilityOfCards()** pour modifier le statut de disponibilité

des cartes spécifiques à la commande. Concrètement, la valeur du champ **carIsAvailable** de la table **t\_card** passe de 1 à 0.

Une fois fait, nous parcourons les différentes cartes par propriétaire stockées dans **\$userCards** grâce à une boucle **foreach**. Pour chaque propriétaire, nous créons une nouvelle transaction en appelant la méthode **createTransaction()**. Tandis que pour chaque carte de chaque propriétaire, nous réalisons trois actions :

- 1) Nous associons la carte à la commande en utilisant l'ID de la carte et l'ID de la transaction en appelant la méthode **assignCardToOrder()**.
- 2) Nous mettons à jour les crédits de l'acheteur en utilisant l'ID de la carte et l'ID de l'acheteur en appelant la méthode **substrCreditsOfBuyer()**.
- 3) Nous mettons à jour la variable de session **\$\_SESSION['useCredits']** avec la nouvelle valeur des crédits de l'acheteur.

Finalement, la méthode **createOrder()** retourne la valeur **true** pour indiquer que la commande a été créée avec succès.

```
public function getAndOrderCardsByOwner($cardIds)
{
    $userCards = []; // Tableau vide pour stocker les cartes classées par propriétaire

    foreach ($cardIds as $idCard) {
        $query = "SELECT * FROM t_card WHERE idCard = :id";
        $bind = array('id' => $idCard);

        $req = $this->queryPrepareExecute($query, $bind);

        $card = $this->formatData($req);

        // Vérifie si le propriétaire de la carte existe déjà dans le tableau $userCards
        if (!isset($userCards[$card[0]['fkUser']])) {
            // Si le propriétaire n'existe pas, initialise un tableau vide pour stocker ses cartes
            $userCards[$card[0]['fkUser']] = [];
        }

        // Ajoute la carte à la liste des cartes du propriétaire correspondant
        array_push($userCards[$card[0]['fkUser']], $card[0]);
    }

    return $userCards;
}
```

La méthode **getAndOrderCardsByOwner()** permet de récupérer les informations des cartes spécifiques grâce à leur ID avant de les regrouper par propriétaire. Elle exécute une requête SQL permettant de sélectionner les informations d'une carte de la table **t\_card** en utilisant l'**ID** de la carte. Après quoi, nous vérifions si le propriétaire de la carte existe déjà dans **\$userCards** grâce au champ **fkUser**. Si le propriétaire n'existe pas dans le tableau, la méthode initialise un tableau vide et ajoute et stock les cartes du propriétaire en utilisant la fonction **array\_push()**.

```

public function hasUserEnoughCredits($userCards)
{
    $totalCardsCredits = 0; // Variable pour stocker le total des crédits des cartes

    foreach ($userCards as $idCardOwner => $cards) {
        foreach ($cards as $card) {
            // Ajoute les crédits de chaque carte au total des crédits des cartes
            $totalCardsCredits = $totalCardsCredits + $card['carCredits'];
        }
    }

    // Vérifie si le nombre de crédits de l'utilisateur moins le total des crédits des cartes est inférieur à 0
    if (($_SESSION['useCredits'] - $totalCardsCredits) < 0) {
        return false;
    }

    return true;
}

```

La méthode **hasUserEnoughCredits()** permet de vérifier si l'utilisateur dispose d'assez de crédits pour passer une commande. Pour ce faire, nous comparons la différence entre le nombre de crédits de l'utilisateur et le montant total du coût en crédits des cartes. Afin d'effectuer ce calcul, nous parcourons les cartes classées par propriétaire grâce à une boucle **foreach**. Pour chaque carte, nous ajoutons les crédits d'une carte au total des crédits **\$totalCardsCredits** en les additionnant. Enfin, pour effectuer notre vérification, nous soustrayons le nombre de crédits total des cartes **\$totalCardsCredits** aux crédits de l'utilisateur **\$\_SESSION['useCredits']**. Si le résultat de cette soustraction donne une valeur inférieure à 0, l'utilisateur n'a pas assez de crédits et la méthode retourne **false**. Si elle est supérieure à 0, la méthode retourne **true**.

```

public function toggleAvailabilityOfCards($cardIds, $isCardAvailable = 1)
{
    // Convertit le tableau d'identifiants de cartes en une chaîne séparée par des virgules
    $cardIds = implode(',', $cardIds);

    $query = "
        UPDATE t_card
        SET carIsAvailable = :isCardAvailable
        WHERE idCard IN ($cardIds);
    ";

    $replacements = [ 'isCardAvailable' => $isCardAvailable];

    $this->queryPrepareExecute($query, $replacements);
}

```

La méthode **toggleAvailabilityOfCards()** permet de modifier le statut de disponibilité des cartes en mettant à jour la colonne **carIsAvailable** de la table **t\_card**. Elle récupère les ID des cartes spécifiques à rendre indisponibles et effectue une requête SQL qui permet de faire passer la valeur de la colonne de 1 à 0.

```

public function createTransaction($idBuyer)
{
    $query = "
    INSERT INTO t_order (fkUser)
    VALUES (:fkUser);
    ";

    $replacements = [ 'fkUser' => $idBuyer];

    // Exécute la requête préparée avec les valeurs fournies
    $this->queryPrepareExecute($query, $replacements);

    // Retourne l'ID de la dernière transaction insérée dans la base de données
    return $this->connector->lastInsertId();
}

```

La méthode **createTransaction()** crée une nouvelle transaction en insérant une nouvelle entrée dans la table **t\_order** en associant l'ID de l'acheteur **fkUser** à la transaction dans la base de données et permet de retourner l'ID de la dernière transaction créée.

```

public function assignCardToOrder($idCard, $idOrder)
{
    $query = "
    UPDATE t_card
    SET fkOrder = :idOrder
    WHERE idCard = :idCard;
    ";

    $replacements = [
        'idCard' => $idCard,
        'idOrder' => $idOrder,
    ];

    // Exécute la requête préparée avec les valeurs fournies
    $this->queryPrepareExecute($query, $replacements);
}

```

La méthode **assignCardToOrder()** permet d'assigner une carte à une commande en mettant à jour la colonne **fkOrder** de la table **t\_card** avec l'ID de la commande de la carte appartenant à la commande.

```

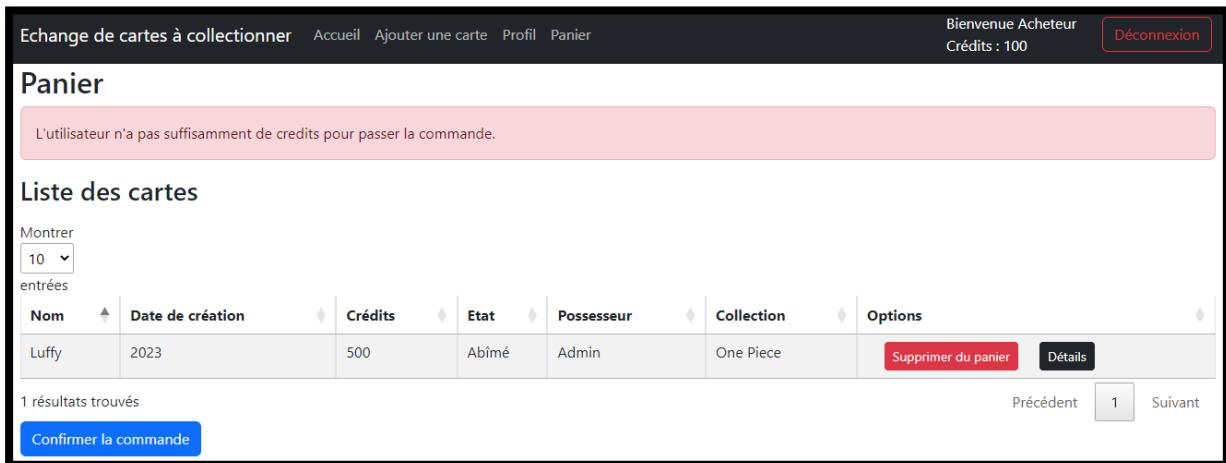
public function substrCreditsOfBuyer($idCard, $idUser)
{
    $query = "
UPDATE t_user
SET useCredits = (SELECT SUM(t_user.useCredits -
(SELECT t_card.carCredits FROM t_card WHERE t_card.idCard = :idCard)))
WHERE t_user.idUser = :idUser;
";

    $replacements = [
        'idCard' => $idCard,
        'idUser' => $idUser,
    ];

    $this->queryPrepareExecute($query, $replacements);
}

```

La méthode **substrCreditsOfBuyer()** permet de mettre à jour le nombre de crédits de l'acheteur en soustrayant les crédits de la carte achetée de son total de crédits dans la colonne **useCredits** de la table **t\_user** de la base de données grâce à une requête SQL.



The screenshot shows a web application interface for managing a shopping cart. At the top, there's a navigation bar with links for 'Accueil', 'Ajouter une carte', 'Profil', and 'Panier'. On the right side of the header, it says 'Bienvenue Acheteur' and 'Crédits : 100', with a 'Déconnexion' button. Below the header, the word 'Panier' is displayed. A pink message box contains the text: 'L'utilisateur n'a pas suffisamment de credits pour passer la commande.' Underneath, there's a section titled 'Liste des cartes' with a table showing one item:

Montrer	10	entrées	Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Luffy	2023	500	Abîmé	Admin	One Piece	<a href="#">Supprimer du panier</a>	<a href="#">Détails</a>		

Below the table, it says '1 résultats trouvés'. At the bottom left is a blue button labeled 'Confirmer la commande'. Navigation buttons 'Précédent' and 'Suivant' are at the bottom right, with the number '1' between them.

Nous constatons que lorsqu'un acheteur tente d'acheter des cartes qui ont une valeur totale plus élevée que le total de crédits sur son compte, la commande n'est pas créée au moment de la confirmation.

Echange de cartes à collectionner Accueil Ajouter une carte Profil Panier Bienvenue Acheteur Crédits : 100 Déconnexion

## Panier

### Liste des cartes

Montrer 10 entrées

Nom	Date de création	Crédits	Etat	Possesseur	Collection	Options
Pharamp	2002	4	Neuf	Admin	Pokémon	<button>Supprimer du panier</button> <button>Détails</button>
Yamato	2023	7	Neuf	Vendeur	One Piece	<button>Supprimer du panier</button> <button>Détails</button>
Zorro	2022	15	Occasion	Vendeur	One Piece	<button>Supprimer du panier</button> <button>Détails</button>

3 résultats trouvés

Précédent 1 Suivant

[Confirmer la commande](#)

Nous pouvons constater que l'utilisateur Acheteur est sur le point de passer une commande pour trois cartes différentes. Deux d'entre-elles appartiennent à l'utilisateur Vendeur et une autre appartient à l'utilisateur Admin. Afin de pouvoir passer la commande, il devra payer au total 22 crédits à Vendeur et 4 crédits à Admin soit 26 crédits en tout. Comme tous les utilisateurs, il dispose par défaut de 100 crédits sur son compte. Il devrait donc ne posséder plus que 74 crédits après avoir passé commande.

Bienvenue Acheteur  
Crédits : 74

[Déconnexion](#)

Liste des commandes

Montrer 10 entrées

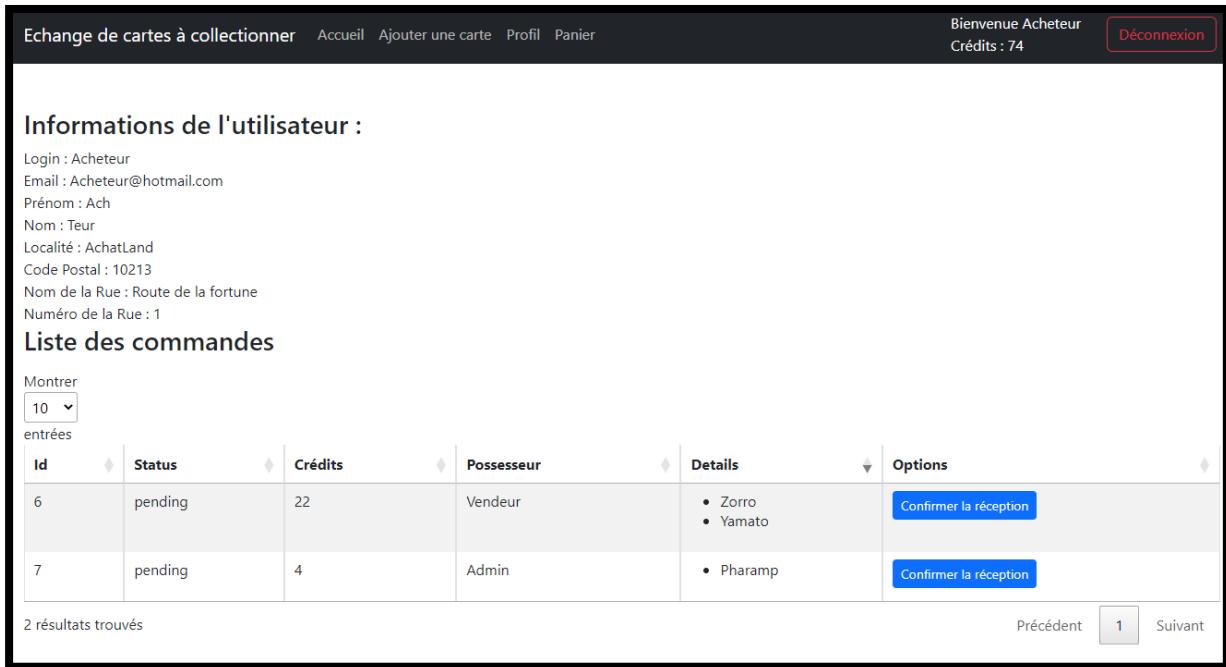
Id	Status	Crédits	Possesseur	Details	Options
6	pending	22	Vendeur	<ul style="list-style-type: none"> <li>Zorro</li> <li>Yamato</li> </ul>	<button>Confirmer la réception</button>
7	pending	4	Admin	<ul style="list-style-type: none"> <li>Pharamp</li> </ul>	<button>Confirmer la réception</button>

2 résultats trouvés

Précédent 1 Suivant

Nous pouvons constater qu'après réalisation de la commande, 26 crédits ont bien été déduits du compte de crédits de l'utilisateur Acheteur. Deux commandes distinctes ont bien été créées pour chacun des vendeurs et les cartes appartenant à un même vendeur sont bien regroupées dans la même commande.

### 3.14 Confirmation de réception d'une commande



**Informations de l'utilisateur :**

Login : Acheteur  
Email : Acheteur@hotmail.com  
Prénom : Ach  
Nom : Teur  
Localité : AchatLand  
Code Postal : 10213  
Nom de la Rue : Route de la fortune  
Numéro de la Rue : 1

**Liste des commandes**

Montrer  
 entrées

Id	Status	Crédits	Possesseur	Details	Options
6	pending	22	Vendeur	<ul style="list-style-type: none"> <li>Zorro</li> <li>Yamato</li> </ul>	<input type="button" value="Confirmer la réception"/>
7	pending	4	Admin	<ul style="list-style-type: none"> <li>Pharamp</li> </ul>	<input type="button" value="Confirmer la réception"/>

2 résultats trouvés

Précédent  Suivant

```
<li class="nav-item text-nowrap">
    <a class="nav-link" href="userProfile.php?idUser=<?php echo $_SESSION['idUser']; ?>">Profil</a>
</li>
```

Une fois une commande passée, un utilisateur a la possibilité de consulter l'historique de ses commandes ainsi que ses informations personnelles sur la page Profil.

```
$OneUser = $db->getOneUser($_GET["idUser"]);

//recuperer la liste des informations pour un utilisateur
public function getOneUser($id)

{
    //avoir la requête sql pour un utilisateur (utilisation de l'id)
    $query = "SELECT * FROM t_user WHERE idUser = :id";
    //appeler la méthode pour executer la requête
    $bind = array('id' => $id);
    $req = $this->queryPrepareExecute($query, $bind);
    //appeler la méthode pour avoir le résultat sous forme de tableau
    $OneUser = $this->formatData($req);
    //retourne l'utilisateur
    return $OneUser[0];
}
```

La méthode **getOneUser()** permet de récupérer les informations d'un utilisateur en particulier à partir de son ID en effectuant une requête SQL.

```
<h3>Informations de l'utilisateur : </h3>

<?php echo
"Login : " . $OneUser["useLogin"] . "<br>" .
"Email : " . $OneUser["useEmail"] . "<br>" .
"Prénom : " . $OneUser["useFirstName"] . "<br>" .
"Nom : " . $OneUser["useLastName"] . "<br>" .
"Localité : " . $OneUser["useLocality"] . "<br>" .
"Code Postal : " . $OneUser["usePostalCode"] . "<br>" .
"Nom de la Rue : " . $OneUser["useStreetName"] . "<br>" .
"Numéro de la Rue : " . $OneUser["useStreetNumber"] . "<br>" ?

```

Afin d'afficher les informations d'un utilisateur en particulier, nous utilisons la fonction **echo()** en les extrayant du tableau associatif **\$OneUser**.

```
$orders = $db->getAllOrdersById($_SESSION['idUser']);
```

```
<tbody>
<?php foreach ($orders as $order) { ?>
<tr>
    <td><?php echo $order['idOrder'] ?></td>
    <td><?php echo $order['ordStatus'] ?></td>
    <td><?php echo $order['ordCredits'] ?></td>
    <td><?php echo $order['ordCardsOwner'] ?></td>
    <td>
        <ul>
            <?php foreach ($order['ordCards'] as $card) { ?>
                <li><?php echo $card['carName'] ?></li>
            <?php } ?>
        </ul>
    </td>
    <td class="containerOptions">
        <?php if ($order['ordStatus'] == 'pending') [ ?>
            <button class="btn btn-primary btn-sm"
                   onclick="confirmOrderReceptionFromUser(<?php echo $_GET['idUser'] ?>,
[<?php echo $order['idOrder'] ?>)">
                Confirmer la réception
            </button>
            <?php ] ?>
        </td>
    </tr>
<?php } ?>
```

Afin d'afficher les informations relatives aux commandes passées par l'utilisateur de la session en cours, nous utilisons une boucle **foreach** pour générer un tableau affichant les détails de chaque commande, les cartes qui y sont associées ainsi qu'un bouton *Confirmer la réception* pour chaque commande dont la valeur de la colonne **ordStatus** est **pending**. Tant que l'utilisateur n'a pas confirmé avoir reçu la commande, le bouton reste affiché.

Dès lors qu'il confirme avoir reçu la commande, le statut de la commande passe de **pending** à **processed**. Dans le cas où aucune commande en attente n'est présente dans le tableau, le bouton *Confirmer la réception* n'est pas affiché.

```
function confirmOrderReceptionFromUser(userId, orderId) {
    if (confirm("Êtes-vous sûr de vouloir confirmer la réception de la commande ?")) {
        window.location.href = `userProfile.php?idUser=${userId}&idOrderToConfirm=${orderId}`;
    }
}
```

Lorsque l'utilisateur clique sur ce bouton, la méthode javaScript **confirmOrderReceptionFromUser()** est déclenchée. Celle-ci permet de gérer la confirmation de la réception d'une commande par l'utilisateur. Elle affiche une boîte de dialogue de confirmation qui, si elle est validée, fait passer en paramètres via l'URL et redirige l'utilisateur vers son profil afin d'actionner la confirmation de la commande.

```
if (isset($_GET['idOrderToConfirm'])) {
    $db->confirmOrderReception($_GET['idOrderToConfirm']);
    header("Location: userProfile.php?idUser={$_GET['idUser']}");
}
```

Sur la page *userProfile.php*, nous implémentons le code permettant de confirmer la réception d'une commande et de rediriger l'utilisateur vers la page de son profil une fois l'opération effectuée. Nous vérifions si le paramètre **idOrderToConfirm** se trouve dans l'URL. Si c'est le cas, nous appelons la méthode **confirmOrderReception()** afin de confirmer la réception de la commande et, le cas échéant, mettre à jour le statut de la commande dans la base de données.

```
public function confirmOrderReception($idOrder)
{
    // Requête pour calculer le total des crédits des cartes associées à la commande
    $query = "
        SELECT
            SUM(carCredits) as total
        FROM t_card
        WHERE t_card.fkOrder = :idOrder
    ";

    $replacements = [ 'idOrder' => $idOrder];

    // Exécute la requête préparée avec les valeurs fournies
    $req = $this->queryPrepareExecute($query, $replacements);

    // Récupère le total des crédits sous forme de tableau
    $totalOrderCredits = $this->formatData($req);

    // Requête pour mettre à jour le statut de la commande en tant que "processed"
    $query = "
        UPDATE t_order
        SET ordStatus = 'processed'
        WHERE idOrder = :idOrder;
    ";

    // Exécute la requête préparée avec les valeurs fournies
    $this->queryPrepareExecute($query, $replacements);

    // Requête pour mettre à jour les crédits de l'utilisateur en ajoutant les crédits de la commande
    $query = "
        UPDATE t_user
        SET useCredits = (SELECT SUM(t_user.useCredits + :creditsToAdd))
        WHERE idUser = (
            SELECT
                t_card.fkUser
            FROM t_card
            WHERE t_card.fkOrder = :idOrder
            LIMIT 1
        );
    ";

    $replacements = [
        'idOrder' => $idOrder,
        'creditsToAdd' => $totalOrderCredits[0]['total'],
    ];

    // Exécute la requête préparée avec les nouvelles valeurs de remplacements
    $this->queryPrepareExecute($query, $replacements);
}
```

La méthode **confirmOrderReception()** permet de mettre à jour les données liées à une commande qui a été confirmée comme reçue grâce à plusieurs requêtes SQL.

Pour ce faire, elle prend en paramètre **\$idOrder** qui permet de spécifier la commande dont l'on souhaite confirmer la réception. La première requête sélectionne la somme de crédits des cartes associées à la commande, la deuxième met à jour le statut de la commande à « **processed** » et la dernière crédite le totale des crédits de la commande au vendeur de la carte.

```

public function getAllOrdersByUserId($idUser)
{
    $query = "
        SELECT
            t_order.*,
            (SELECT SUM(t_card.carCredits) FROM t_card WHERE t_card.fkOrder = t_order.idOrder) AS ordCredits,
            (
                SELECT
                    t_user.useLogin
                FROM t_card
                LEFT JOIN t_user ON t_user.idUser = t_card.fkUser
                WHERE t_card.fkOrder = t_order.idOrder
                LIMIT 1
            ) AS ordCardsOwner
        FROM t_order
        WHERE t_order.fkUser = :idUser
    ";

    $replacements = [ 'idUser' => $idUser ];

    $req = $this->queryPrepareExecute($query, $replacements);

    $orders = $this->formatData($req);

    // Pour chaque commande, récupère les cartes associées
    foreach ($orders as $index => $order) {
        // Requête pour récupérer les cartes associées à la commande
        $query = "
            SELECT
                *
            FROM t_card
            WHERE t_card.fkOrder = :idOrder
        ";

        $replacements = [ 'idOrder' => $order['idOrder'] ];

        $req = $this->queryPrepareExecute($query, $replacements);

        // Associe les cartes récupérées à la commande
        $orders[$index]['ordCards'] = $this->formatData($req);
    }

    return $orders;
}

```

La méthode **getAllOrdersByUserId()** va permettre de récupérer toutes les commandes passées de l'utilisateur connecté. Pour détailler la commande, une query récupère les cartes connectées à la commande passée et sont ensuite ajoutées au tableau **\$orders** en utilisant l'index correspondant à cette dernière.

Id	Status	Crédits	Possesseur	Details	Options
1	pending	22	Vendeur	<ul style="list-style-type: none"> <li>• Zorro</li> <li>• Yamato</li> </ul>	<button data-bbox="1108 265 1224 294">Confirmer la réception</button>
2	pending	4	Admin	<ul style="list-style-type: none"> <li>• Pharamp</li> </ul>	<button data-bbox="1108 332 1224 361">Confirmer la réception</button>

Nous constatons que tant que l'utilisateur Acheteur n'a pas confirmé avoir reçu la commande de l'utilisateur Vendeur celle-ci est toujours affichée dans un statut **pending**.

Id	Status	Crédits	Possesseur	Details	Options
1	processed	22	Vendeur	<ul style="list-style-type: none"> <li>• Zorro</li> <li>• Yamato</li> </ul>	
2	pending	4	Admin	<ul style="list-style-type: none"> <li>• Pharamp</li> </ul>	<button data-bbox="1108 691 1224 720">Confirmer la réception</button>

Puis, nous constatons qu'après avoir confirmé la réception de la commande envoyée par l'utilisateur Vendeur. Le statut de la commande est passé à **processed** et le bouton de confirmation de la commande n'est plus affiché.



Enfin, nous pouvons constater que les 22 crédits qui étaient mis en attente ont bien été crédités à l'utilisateur Vendeur.

## 4 Tests

Nom du test	Scénario	Résultat attendu	Validé/Echec
Ajout d'une carte avec succès.	<p>Un utilisateur connecté clique sur l'onglet Ajouter une carte. Il renseigne les informations demandées et télécharge une image au format JPG.</p> <p>Il clique ensuite sur le bouton Ajouter une carte lorsqu'il est satisfait des informations renseignées.</p>	<p>La validation des champs s'effectue sans rencontrer d'erreurs et un message affiche : <i>Votre carte a bien été ajoutée.</i></p> <p>L'utilisateur peut consulter sa carte en cliquant en parcourant le tableau des cartes mises en vente.</p>	Validé
Ajout d'une carte sans respecter le format de l'image.	<p>Un utilisateur connecté clique sur l'onglet Ajouter une carte. Il renseigne les informations demandées, mais envoie un fichier au format PNG.</p> <p>Il clique ensuite sur le bouton Ajouter une carte lorsqu'il est satisfait des informations renseignées.</p>	<p>La validation des champs s'effectue et repère que le fichier envoyé n'est pas au format attendu.</p> <p>Une erreur contextuelle affiche : <i>Merci de n'envoyer que des images au format JPG.</i></p> <p>Les informations renseignées dans les champs qui n'ont pas déclenché d'erreur sont retournés à l'utilisateur.</p>	Validé
Accès à une page non autorisée.	Un utilisateur non connecté essaye d'accéder à la page Ajouter une carte via l'URL sans qu'il ne dispose des droits pour y accéder.	Une page d'erreur 403 s'affiche et empêche l'utilisateur d'accéder à la page web. Celle-ci l'invite à rejoindre l'accueil ou à se connecter.	Validé
Tri des cartes affichées sur la page d'accueil.	Un utilisateur connecté souhaite trier les cartes par ordre alphabétique à partir du nom. Dans ce but, il clique sur l'icône au-dessus de la colonne permettant de réaliser le tri.	Les cartes s'affichent correctement par ordre alphabétique croissant ou décroissant après le clique sur l'icône.	Validé

Filtrage des cartes affichées sur la page d'accueil selon leur état.	Un utilisateur souhaite filtrer les cartes affichées sur la page d'accueil selon leur état. Il clique sur le bouton <i>plus de filtres</i> et indique un état Neuf sur le filtre concerné qui est correctement apparu. Il clique ensuite sur le bouton Rechercher.	Seules les cartes renseignées comme étant dans un état Neuf s'affichent dans le tableau des cartes en vente de la page d'accueil.	Validé
Crédits mis en attente lors d'une transaction	Un utilisateur connecté clique sur le bouton Acheter de la carte qui l'intéresse depuis le tableau de la page d'accueil. Après avoir cliqué sur l'onglet Mon panier il constate que la carte a correctement été ajoutée à son panier. Il confirme l'achat en cliquant sur le bouton <i>Confirmer la commande</i> .	Les crédits sont bien débités du compte de l'acheteur mais ne sont pas crédités sur le compte du vendeur avant que l'acheteur ne confirme la réception sur son profil.	Validé
Réussite de l'utilisation du manuel d'installation	Un collègue ou un enseignant disponible télécharge l'application depuis le dépôt git et consulte le manuel d'installation présent dans le dossier sous forme de README.md. Il essaie ensuite de le reproduire méticuleusement en l'adaptant à son environnement si nécessaire.	La personne qui a suivi les instructions du manuel d'installation a réussi à lancer l'application dans son propre environnement.	Validé

#### 4.1.1

#### LISTE DES FONCTIONNALITÉS QUI DOIVENT FONCTIONNER

## 4.1.2

### 4.2 Erreurs restantes

S'il reste encore des erreurs:

- Description détaillée
- Conséquences sur l'utilisation du produit
- Actions envisagées ou possibles

### 4.3 Liste des documents fournis

Lister les documents fournis au client avec votre produit, en indiquant les numéros de versions

- le rapport de projet
- le manuel d'Installation (en annexe)
- le manuel d'Utilisation avec des exemples graphiques (en annexe)
- autres...

---

## 5 Conclusions

### 5.1 Bilan des fonctionnalités

### 5.2 Comparaison de la planification

### 5.3 Critiques / Finalité du projet

### 5.4 Difficultés particulières

### 5.5 Conclusion personnelle

Développez en tous cas les points suivants:

- Objectifs atteints / non-atteints
- Points positifs / négatifs
- Difficultés particulières
- Suites possibles pour le projet (évolutions & améliorations)

---

## 6 Lexique

Attaque XSS

Blowfish

jQuery

DOM

---

## 7 Table d'illustrations

FIGURE 1 : SCHEMA DE LA METHODE WATERFALL..... ERREUR ! SIGNET NON DEFINI.

---

## 8 Annexes

### 8.1 Résumé du rapport du TPI / version succincte de la documentation

#### 8.1.1 Situation de départ

#### 8.1.2 Mise en œuvre

#### 8.1.3 Résultats

### 8.2 Sources – Bibliographie

---

## 9 Bibliographie

### 9.1 Manuel d'Utilisation

### 9.2 Archives du projet

Media, ... dans une fourre en plastique