

Projet 3 - PopBazar

Plateforme gestion de petites annonces

20% de la note finale

Remise le mardi le 13 mai

I. Description du projet

PopBazar est un projet de développement web dont l'objectif est de concevoir une plateforme de petites annonces fonctionnelle en architecture **MVC** (Modèle-Vue-Contrôleur) à l'aide de **PHP** et **MySQL**.

Vous êtes amenés à développer, à partir d'une structure de base fournie, un site web qui permet à des utilisateurs :

- De créer, consulter, modifier, supprimer et marquer des annonces comme vendues
- De filtrer les annonces par statut ou par catégorie
- De s'inscrire, se connecter, et gérer leur profil
- D'interagir avec une base de données via un code structuré et sécurisé

Ce projet met l'accent sur les **bonnes pratiques de développement** (séparation des responsabilités, validation côté serveur, sécurité minimale).

Ce projet peut être réalisé en équipe de deux.

Objectifs d'apprentissage

À la fin de ce projet, vous serez en mesure de :

1. Développer une application web complète en architecture MVC

- Comprendre et appliquer la séparation des responsabilités entre le Modèle, la Vue et le Contrôleur
- Intégrer les composants MVC dans une structure de projet cohérente

2. Interagir avec une base de données relationnelle (MySQL)

- Lire, insérer, modifier et supprimer des données à l'aide de requêtes SQL préparées

3. Concevoir une interface utilisateur dynamique et cohérente

- Afficher des données dynamiques dans les vues à partir du backend
- Organiser et filtrer les informations affichées (par catégorie, statut, pagination, etc.)

4. Gérer l'authentification et la sécurité de base

- Implémenter un système d'inscription et de connexion avec gestion de session
- Protéger les pages privées accessibles uniquement aux utilisateurs connectés

5. Valider et assainir les données saisies par l'utilisateur

- Appliquer des règles de validation côté serveur
- Afficher des messages d'erreur clairs dans l'interface utilisateur

6. Travailler avec Git de manière professionnelle

- Utiliser un dépôt distant (GitHub) pour collaborer
- Travailler en branches, fusionner du code, résoudre des conflits
- Documenter et organiser son projet dans un README

7. Appliquer une démarche de développement structuré

- Décomposer un projet en étapes fonctionnelles réalisables
- Tester régulièrement les fonctionnalités développées
- Corriger les erreurs de logique ou d'affichage de manière autonome

Soutien et outils autorisés

Afin de vous aider à compléter ce projet de manière autonome, tout en respectant les bonnes pratiques d'apprentissage, vous pouvez vous appuyer sur les ressources suivantes:

Documentation et références

- Documentation officielle de PHP : <https://www.php.net/>
- Documentation SQL (MySQL/MariaDB) : <https://dev.mysql.com/doc/>
- Références HTML/CSS/JS :
 - <https://developer.mozilla.org/fr/>
 - <https://www.w3schools.com/>
- StackOverflow

Outils d'assistance

- VS Code
- PhpMyAdmin
- Google Chrome
- WAMP (ou MAMP/LAMP/XAMPP) pour l'exécution locale du projet

Utilisation de l'intelligence artificielle (IA)

- Vous êtes autorisés à utiliser des outils d'IA (ex. ChatGPT) **à condition de ne pas copier-coller du code généré sans compréhension.**
- Vous pouvez poser des questions à l'IA pour :
 - Clarifier un concept (ex. : "C'est quoi MVC ?")
 - Débloquer une erreur

- Vous **ne devez pas** utiliser l'IA pour :
 - Générer entièrement vos modèles ou contrôleurs sans modification
 - Copier du code sans comprendre ce qu'il fait
 - Déléguer la structure complète de votre application

Le but est que **vous développiez vos propres compétences**. Toute utilisation abusive de l'IA sera considérée comme du plagiat.

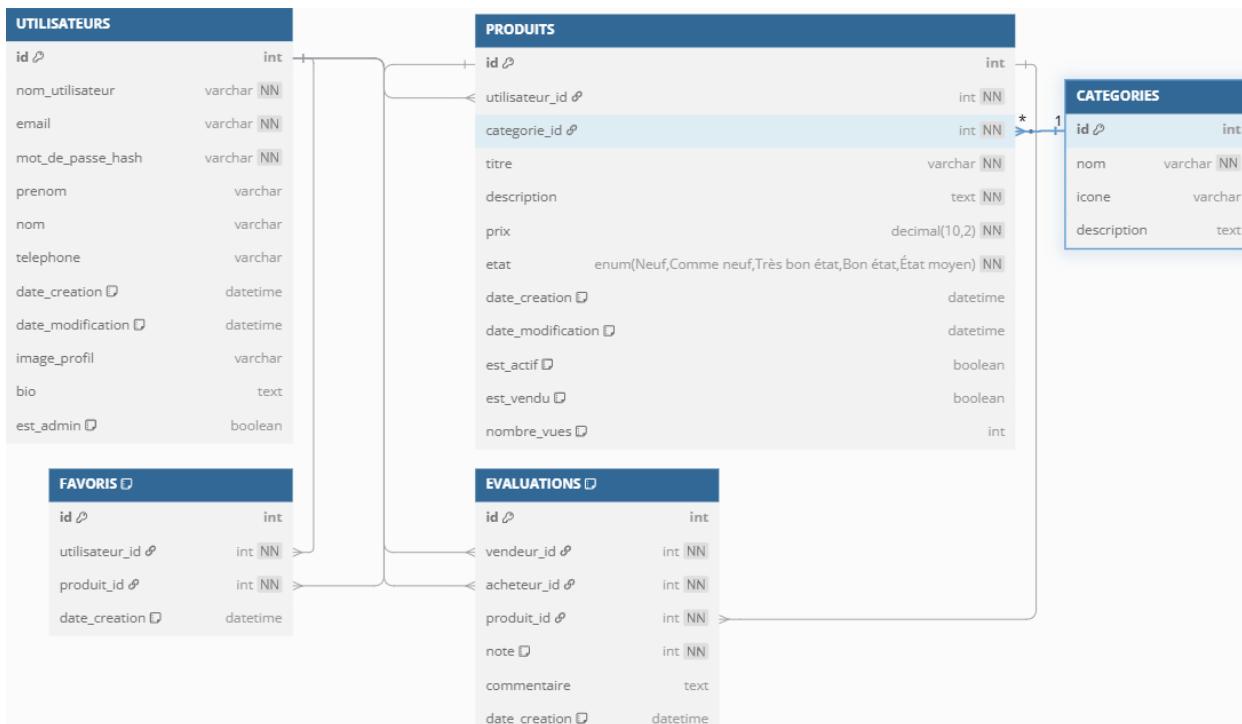
Technologies utilisées

- Environnement de développement : WAMP (Windows, Apache, MySQL, PHP)
- PHP : Version 7.4 ou supérieure pour le traitement côté serveur
- MySQL : Gestion de la base de données
- phpMyAdmin : Interface d'administration de la base de données
- HTML5/CSS3 : Structure et présentation des pages
- Bootstrap 5 : Framework CSS pour un design responsive et moderne

Structure de la base de données

La base de données est structurée de la façon suivante:

- Nous n'utiliserons véritablement que les tables **utilisateurs**, **produits**, et **categories** pour ce projet.



II. Mise en place

Installation

Téléchargez le projet [projet3](#) sur [Lea](#) contenant la structure de base du projet et décompressez-le dans le dossier `C:\wamp64\www\popbazar` sur votre ordinateur. Ce dossier comprend :

- L'arborescence des dossiers structuré selon l'architecture MVC
- Les fichiers de configuration de base
- Le script SQL pour initialiser la base de données
- `utils/` contient des fichiers de fonctions utilitaires prêtes à l'emploi

Les modèles des pages html dont les squelettes se trouvent dans `views` proviennent d'un modèle. Vous pouvez vous y référer au besoin. Le modèle original se trouve sur [github](#) à cette adresse:

https://github.com/DominicTremblay/template_popbazar

Créer un hôte virtuel

Créer un hôte virtuel à l'aide de [Wamp](#) :

1. Cliquer gauche sur l'icône verte WampServer dans la zone de notification
2. Ouvrir la gestion des hôtes virtuels. Dans le menu :
 - Aller dans "Vos VirtualHosts"
 - Cliquer sur "Gestion des VirtualHost"
3. Remplire le formulaire :
 - Nom du VirtualHost : `popbazar.local`
 - Chemin complet vers le dossier : `C:/wamp64/www/popbazar/public`

4. Redémarre les services WAMP

Après avoir démarrer **Wamp**, pointez le navigateur à **popbazar.local**. Vous devriez voir la page d'accueil de l'application.

Git/GitHub

Vous devez utiliser **git/GitHub** pour votre projet. Vous devez donc créer un dépôsitoire privé sur **GitHub** pour les membres de votre équipe. Un fichier **.gitignore** se trouve déjà dans le projet. Référez-vous aux instructions de configuration en cas de besoin.

Création de la base de données

- a. Créer une base de données intitulée **popbazar** dans **phpMyAdmin**.
- b. Copier-coller le script **popbazar.sql** dans l'onglet SQL de **phpMyAdmin** pour créer les tables et peupler la base de données.

Mots de passe des utilisateurs créés:

- Pour l'utilisateur admin (admin@popbazaar.com) : admin123
- Pour l'utilisateur bleponge (bob.leponge@exemple.com) : password123
- Pour l'utilisateur pletoile (patrick.letoile@exemple.com) : password123

Note: À chaque fois que vous roulez ce script, il détruira et recréera les tables et les données. Toutes les données créées au préalable seront détruites et remplacées.

Structure du projet

Ouvrez le projet dans VS Code. Le projet est structuré selon l'architecture MVC.

```
/popbazar
|
├ .env                                ← Fichier de configuration d'environnement
├ .env.php                             ← Chargeur de variables d'environnement
├ .htaccess                            ← Redirection vers /public/index.php
├ routes.php                           ← Déclaration des routes
├ Routeur.php                          ← Classe de routage principale
├ config/                             ← Fichiers de configuration
│   └ bd.php                            ← Paramètres de connexion à la base de données
│   └ Database.php                      ← Classe d'abstraction PDO
|
└ controllers/                         ← Contrôleurs MVC
    └ AccueilController.php
    └ AnnonceController.php
    └ CategorieController.php
    └ ErreurController.php
    └ FavorisController.php
    └ ProfilController.php
    └ UtilisateurController.php
|
└ models/                             ← Modèles
    └ Annonce.php
    └ Categorie.php
    └ Utilisateur.php
```

```
└─ utils/                                ← Fonctions utilitaires réutilisables
    ├─ index.php
    ├─ Session.php
    └─ Validation.php

    |

    └─ views/                               ← Vues HTML
        ├─ accueil.view.php
        ├─ erreur.view.php
        ├─ partials/                            ← Entête, pied de page, etc.
        ├─ annonces/                           ← Pages liées aux annonces
        ├─ favoris/                            ← Pages de favoris
        └─ utilisateur/                         ← Pages de connexion, inscription, profil, etc.

    |

    └─ public/                               ← Point d'entrée du site (accessible publiquement)
        └─ index.php                          ← Lance l'application (inclut le routeur)
        └─ css/                                ← Fichiers de styles CSS
        └─ images/                             ← Images du site
```

Clarification sur les fichiers à modifier

Pour éviter toute confusion, voici la liste des fichiers que vous **devez modifier**, et ceux qui sont **déjà fournis** et ne doivent pas être recréés.

Fichiers à modifier ou compléter

Vous devez **implémenter ou adapter** le code dans les fichiers suivants :

- Les **méthodes des modèles** (/models/): pour interagir avec la base de données
- Les **méthodes des contrôleurs** (/controllers/): pour gérer la logique métier et transmettre les données aux vues

- Les **vues dynamiques** (/views/) : vous devez rendre dynamiques les fichiers .view.php à partir des squelettes statiques fournis

Fichiers déjà fournis – Ne pas modifier sauf indication contraire

- Le **routeur** (Routeur.php) est déjà prêt à l'emploi
- Le fichier des **routes** (routes.php) est initialement structuré. Vous devez seulement **ajouter des routes** à l'intérieur, pas changer la structure.
- Les fichiers utilitaires dans /utils/ (Session.php, Validation.php, etc.) sont fournis. Vous pouvez les consulter ou utiliser leurs fonctions, mais **ne devez pas les reprogrammer**.
- Le fichier de **connexion à la base de données** (config/bd.php) est également prêt à être utilisé.

Organisation recommandée

Vous devez suivre la structure du projet telle qu'elle a été fournie (modèle MVC). Toute fonctionnalité doit être répartie logiquement entre :

- **Modèle** : traitement et accès aux données
- **Contrôleur** : traitement de la requête, coordination
- **Vue** : affichage HTML dynamique

III. Développement des fonctionnalités

Étape 1 – Affichage des annonces

Voir les annonces par catégorie

Récit utilisateur

- **En tant que visiteur**, je veux consulter les annonces disponibles dans une catégorie spécifique afin de voir uniquement ce qui m'intéresse.

Objectif

Vous devez permettre l'affichage des annonces publiques selon leur catégorie :

- Lorsque l'utilisateur clique sur une catégorie (ex. Jeux vidéo), il doit voir toutes les annonces actives associées à cette catégorie.
- La page doit afficher les informations essentielles de chaque annonce.

Consignes

- Vous devez récupérer les annonces filtrées par catégorie dans la base de données.
- La logique est déclenchée via une URL de type `/categories/{id}/annonces`.
- Le nom de la catégorie peut aussi être affiché dans la vue.

Résultats attendus

- **Un visiteur** peut naviguer par catégorie et voir des annonces publiques.

Voir ses propres annonces

Récit utilisateur

- **En tant qu'utilisateur connecté**, je veux voir la liste de mes annonces afin de pouvoir les modifier, les désactiver ou les supprimer.

Objectif

Vous devez créer une page qui affiche les annonces de l'utilisateur actuellement connecté :

- Cette page n'est accessible qu'après authentification
- On doit y afficher toutes les annonces de cet utilisateur, peu importe leur statut (actif ou non)
- On y retrouvera plus tard des actions comme "Modifier", "Supprimer", "Marquer comme vendu"

Consignes

- Utilisez la session pour connaître l'utilisateur courant.
- Filtrez les annonces selon l'**utilisateur_id**
- Affichez dans la vue : titre, prix, état, si l'annonce est active ou vendue.

Résultats attendus

- Un **utilisateur connecté** a sa propre section "Mes annonces" pour consulter ses publications.

Filtrage : Toutes / Actives / Vendues

Récit utilisateur

- **En tant qu'utilisateur connecté**, je veux pouvoir filtrer mes annonces entre toutes, celles encore actives et celles déjà vendues, afin de mieux gérer mes publications.

Objectif

Vous devez permettre à l'utilisateur de **filtrer dynamiquement l'affichage de ses propres annonces** à l'aide de boutons ou onglets.

- Le filtre peut s'effectuer à l'aide de **query strings** dans l'URL :
`/mes-annonces?filtre=actives`, `/mes-annonces?filtre=vendues`
- Trois états doivent être gérés :
 - **Toutes** (par défaut)
 - **Actives uniquement (est_actif = 1)**
 - **Vendues uniquement (est_vendu = 1)**

💡 Indications

- Vous pouvez utiliser une logique conditionnelle dans la méthode du contrôleur
- Adaptez la requête selon le filtre sélectionné
- La vue doit mettre en évidence le filtre actif (ex. : bouton surligné)
- Affichez aussi les **compteurs** à côté de chaque filtre :
(Toutes 10 | Actives 7 | Vendues 3)

Résultat attendu

- L'utilisateur connecté peut **changer dynamiquement l'affichage de ses annonces** en fonction de leur statut. La page réagit au filtre et reflète les bons chiffres.

Étape 2 – Affichage d'une annonce individuelle

Récit utilisateur

- **En tant qu'utilisateur**, je veux pouvoir cliquer sur une annonce pour consulter tous ses détails, afin de mieux comprendre ce qui est offert avant de décider si ça m'intéresse.

Objectif

Vous devez mettre en place une page qui affiche tous les détails d'une annonce spécifique, incluant :

- Le titre
- La description complète
- Le prix
- L'état du produit
- Le nom de la catégorie
- La date de publication
- Active ou inactive

Consignes

- Cette page est publique : elle peut être consultée par n'importe quel visiteur.
- L'URL de la page doit identifier l'annonce par son identifiant, par exemple : [/annonces/42](#)
- Le contrôleur doit extraire une seule annonce à partir de son ID.
- Le modèle doit effectuer une requête avec jointure si nécessaire pour obtenir la catégorie liée.
- La vue fournie contient une structure HTML que vous devez rendre dynamique.

À valider

- Si l'ID de l'annonce n'existe pas, la page doit rediriger vers une page d'erreur (ex. 404).
- Si l'annonce n'est pas active et que l'utilisateur n'est pas le propriétaire, un message d'erreur indiquant que l'annonce n'est pas active doit être affichée.

Résultats attendus

- L'utilisateur voit une page détaillée d'une annonce, bien structurée, avec toutes les infos importantes clairement présentées.

Étape 3 – Ajouter une annonce

Récit utilisateur

- **En tant qu'utilisateur connecté**, je veux pouvoir remplir un formulaire pour ajouter une nouvelle annonce, afin de proposer un objet à vendre à la communauté.

Objectif

Vous devez permettre aux utilisateurs **connectés** d'accéder à un **formulaire d'ajout** d'annonce, puis de **soumettre les données** pour qu'elles soient enregistrées en base de données.

Deux actions sont nécessaires :

1. Afficher le formulaire d'ajout
2. Traiter les données soumises (si valides) pour créer une nouvelle annonce

Consignes

- Le formulaire contient des champs pour :
 - Titre, description, prix, état, catégorie
- L'utilisateur doit être **connecté** pour accéder à cette page

- Lors de la soumission, l'annonce doit être enregistrée avec :
 - Le bon `utilisateur_id`
 - Le champ `est_actif` par défaut à `true`
 - Le champ `est_vendu` par défaut à `false`
- Une fois l'annonce créée, vous pouvez rediriger vers la page "**Mes annonces**" ou **la page de l'annonce**.

Validation minimale

- Vérifiez que tous les champs obligatoires sont remplis.
- Aucune logique de validation avancée n'est requise à cette étape (sera vue plus tard).

Résultats attendus

- Un utilisateur connecté peut accéder à la page "**Ajouter une annonce**", soumettre un formulaire valide, et retrouver sa nouvelle annonce dans la section "**Mes annonces**".
- Un **visiteur** sera redirigé à la page de connexion avec un message flash qui indique "*Vous devez être connecté pour créer une annonce*".

Étape 4 – Modifier une annonce

Récit utilisateur

- **En tant qu'utilisateur connecté**, je veux pouvoir modifier les informations d'une de mes annonces, afin de corriger une erreur ou mettre à jour les détails de ce que je vends.

Objectif

Vous devez permettre à l'utilisateur **connecté** d'accéder à un **formulaire prérempli** avec les informations de l'annonce à modifier, puis de **soumettre les changements**.

Consignes

- Le lien "**Modifier**" est disponible seulement dans la section "**Mes annonces**".
- L'utilisateur peut **modifier uniquement ses propres annonces**.
- Les champs du formulaire sont les mêmes que pour l'ajout :
 - Titre, description, prix, état, catégorie
- L'utilisateur ne doit pas pouvoir modifier le statut **est_vendu** ici.
- Après la modification, rediriger vers la page de l'annonce ou vers "**Mes annonces**".

Points importants

- Un **visiteur** ne devrait pas pouvoir accéder à la page de modification.
- Le système doit s'assurer que l'utilisateur est bien **propriétaire** de l'annonce qu'il souhaite modifier.

Résultats attendus

- Un **utilisateur connecté** peut cliquer sur **Modifier** depuis "**Mes annonces**", voir un formulaire prérempli, faire des changements, et les voir appliqués dans sa liste d'annonces.
- Un **visiteur** sera redirigé à la page de connexion avec un message flash qui indique "*Vous devez être connecté pour modifier une annonce*".
- Un **utilisateur connecté** qui accède au formulaire de modifications pour une annonce dont il **n'est pas le propriétaire** sera redirigé à ses annonces avec un message flash qui indique "*Vous n'êtes pas autorisé à modifier cette annonce.*"

Étape 5 – Supprimer une annonce

Récit utilisateur

- **En tant qu'utilisateur connecté**, je veux pouvoir supprimer l'une de mes annonces si je ne veux plus qu'elle soit visible sur la plateforme.

Objectif

Vous devez permettre à l'utilisateur **connecté** de **supprimer une annonce** dont il est propriétaire.

Consignes

- Le bouton "**Supprimer**" est accessible uniquement depuis "**Mes annonces**".
- Avant de supprimer, vous pouvez afficher une **fenêtre de confirmation (modale)**.
- Une fois confirmé, la requête doit **retirer l'annonce de la base de données**.
- Après suppression, rediriger l'utilisateur vers "**Mes annonces**" avec un message flash qui indique "Announce supprimée avec succès.".

Précautions

- Il faut s'assurer que l'annonce existe avant de pourvoir l'effacer.
- L'utilisateur ne peut **supprimer que ses propres annonces**

Résultats attendus

- Depuis la section "**Mes annonces**", l'utilisateur peut cliquer sur **Supprimer**, confirmer l'action, et voir l'annonce supprimée définitivement.
- Si un utilisateur tente de supprimer une annonce qui ne lui appartient pas, il doit être redirigé vers "**Mes annonces**" avec un message flash qui indique "L'annonce n'a pas pu être supprimée."

Note: La façon de tester le cas où une annonce n'a pu être supprimée, c'est un utilisant **Postman** ou **Insomnia** (ex. requête POST à <http://popbazar.local/annonces/8/supprimer>).

Étape 6 – Marquer une annonce comme vendue

Récit utilisateur

- **En tant qu'utilisateur connecté**, je veux pouvoir marquer une annonce comme vendue afin que les autres sachent qu'elle n'est plus disponible.

Objectif

Vous devez permettre à l'utilisateur **connecté** de marquer **une de ses annonces** comme vendue. Cela consiste à **mettre à jour un champ dans la base de données**, sans supprimer l'annonce.

Consignes

- L'option "**Marquer comme vendue**" est accessible depuis "**Mes annonces**".
- L'action peut être déclenchée par :
 - Un bouton
 - Une modale de confirmation
 - Un formulaire POST discret
- Vous devez mettre à jour la colonne **est_vendu** de l'annonce à 1 (**true**).
- L'annonce vendue doit continuer à apparaître dans la section "Mes annonces", avec une **indication visuelle** (texte ou badge "VENDUE").

Précautions

- L'utilisateur peut **marquer comme vendue uniquement ses propres annonces**.
- Un **visiteur** ou un **autre utilisateur** ne doit pas pouvoir accéder à cette fonctionnalité.

Résultat attendu

- L'utilisateur clique sur "Marquer comme vendue", confirme l'action, et voit que l'annonce est maintenant identifiée comme vendue, sans être supprimée.

Étape 7 – Affichage du profil utilisateur

Récit utilisateur

- En tant qu'utilisateur connecté, je veux pouvoir consulter mon profil pour voir les informations liées à mon compte et savoir depuis quand je suis membre.

Objectif technique

Vous devez créer une page permettant à l'utilisateur connecté de voir ses propres informations.

La page doit afficher :

- Prénom
- Nom
- Nom d'utilisateur
- Courriel
- Date d'inscription (affichée sous une forme conviviale, par exemple : "Membre depuis janvier 2024")
- (Optionnel) Bio, téléphone, image de profil s'ils sont disponibles

Consignes

- Les informations doivent être extraites de la base de données à partir de l'**id** de l'utilisateur connecté (session).
- La vue **profil.view.php** est déjà fournie. Elle doit être rendue dynamique.
- Vous pouvez utiliser la fonction utilitaire prévue pour formater la date de création en une phrase lisible.
- Le lien vers "Mon profil" devrait être accessible dans la barre de navigation quand l'utilisateur est connecté.

Résultat attendu

- L'utilisateur accède à une page claire qui affiche ses informations personnelles. Il comprend rapidement à quel compte il est connecté et peut vérifier les données associées à son profil.

Étape 8 – Pagination des annonces

Récit utilisateur

- **En tant qu'utilisateur**, je veux pouvoir naviguer entre plusieurs pages de résultats d'annonces, afin de ne pas être submergé par une trop longue liste et accéder facilement à toutes les annonces.

Objectif

Vous devez ajouter la **pagination** aux pages qui affichent une liste d'annonces, notamment :

- Annonces par **catégorie**
- Annonces dans "**Mes annonces**"

Indications

- Affichez un nombre fixe d'annonces par page (ex. 9)
- Ajoutez des liens "**Précédent**" et "**Suivant**", ainsi que le numéro des pages
- Utilisez la **query string** dans l'URL pour passer l'information de pagination :
ex : `/categories/4/annonces?page=2`
- Vous devez adapter la requête SQL pour ajouter **LIMIT** et **OFFSET**
- Calculez le nombre total de pages selon le nombre d'annonces disponibles

À prévoir

- Si aucun paramètre `page` n'est présent dans l'URL, utilisez la valeur par défaut 1

Résultat attendu

- L'utilisateur peut naviguer entre les différentes pages d'annonces, que ce soit dans une **catégorie** ou dans **ses propres annonces**, avec un affichage cohérent et fluide.

Étape 9 – Authentification : inscription et connexion

Récits utilisateurs

- **En tant que nouvel utilisateur**, je veux pouvoir créer un compte pour publier des annonces.
- **En tant qu'utilisateur inscrit**, je veux pouvoir me connecter pour accéder à mes annonces et en publier de nouvelles.

Objectif

Vous devez mettre en place deux fonctionnalités essentielles :

1. **Inscription** : permet à un utilisateur de créer un compte
2. **Connexion** : permet à un utilisateur existant de se connecter à son compte

Inscription

- Formulaire déjà fourni avec les champs suivants :
 - Prénom, nom, nom d'utilisateur, courriel, mot de passe
- Le mot de passe doit être **haché** avant d'être inséré dans la base de données
- Après inscription réussie :
 - L'utilisateur peut être redirigé vers "**Mes annonces**" ou être automatiquement connecté
- Validations:
 - Tous les champs sont requis
 - Les nom et prenom doivent comporter entre 2 et 50 caractères
 - Le nom d'utilisateur doit avoir entre 4 et 50 caractères
 - Le email doit être validé
 - Le mot de passe doit contenir:
 - Au moins 8 caractères
 - Au moins une lettre majuscule
 - Au moins un chiffre
 - Au moins un caractère spécial
 - Le mot de passe et la confirmation du mot passe doivent être identiques.

Connexion

- Formulaire avec **nom d'utilisateur** et **mot de passe**
- Vous devez vérifier que :
 - Le nom d'utilisateur existe
 - Le mot de passe correspond au hachage en base
- Si les informations sont valides, **enregistrer l'utilisateur dans la session**
- Si elles sont invalides, **afficher un message d'erreur "Mauvaise combinaison d'email et de mot de passe"**
- Validations:
 - Tous les champs sont requis
 - Validation du email

Session

- Une fois connecté, l'utilisateur doit être identifié tout au long de sa navigation
- Le contenu de la barre de navigation doit changer selon que l'utilisateur est connecté ou non
- Par exemple : afficher un lien "Mes annonces" ou "Se déconnecter"

Résultat attendu

- Un utilisateur peut s'inscrire, se connecter, et rester connecté
- Un utilisateur non connecté ne peut pas accéder à des pages protégées comme **ajouter une annonce** ou **mes annonces**

Déconnexion

Récit utilisateur

- **En tant qu'utilisateur connecté**, je veux pouvoir me déconnecter facilement, afin de sécuriser mon compte lorsque j'ai terminé ma session.

Objectif technique

Vous devez permettre à un utilisateur connecté de **se déconnecter** à tout moment.

Indications

- Ajouter un lien "**Déconnexion**" dans la barre de navigation visible uniquement lorsque l'utilisateur est connecté.
- Ce lien peut pointer vers une route dédiée ([`/deconnexion`](#)) qui :
 - Vide la session
 - Redirige vers la page d'accueil ou la page de connexion

Résultat attendu

- L'utilisateur clique sur "**Déconnexion**", est redirigé, et son compte est complètement déconnecté (il n'a plus accès aux pages protégées).

Étape 10 – Validation des champs de formulaire

Récit utilisateur

En tant qu'utilisateur, je veux être averti si je fais une erreur en remplissant un formulaire, afin de corriger l'information avant que mes données soient envoyées ou enregistrées.

Objectif technique

Vous devez valider les **champs de formulaire soumis par l'utilisateur**, que ce soit lors :

- De l'ajout d'une annonce
- De la modification d'une annonce
- De l'inscription

Indications générales

- Vérifiez que les champs obligatoires **ne sont pas vides**
- Validez les formats :
 - Courriel valide
 - Mot de passe d'une certaine complexité (longueur, majuscules, chiffre, symbole.)
 - Prix numérique et positif
- Affichez des **messages d'erreur clairs et visibles** dans les vues si une validation échoue
- Conservez les données déjà remplies dans le formulaire si une erreur survient

Validation côté serveur (obligatoire)

- Même si du JavaScript est utilisé côté client, les **vérifications principales doivent se faire en PHP**
- Utilisez les méthodes de la classe **Validation**.

Résultat attendu

- L'utilisateur est guidé s'il oublie un champ ou entre une valeur invalide. Il comprend ce qui doit être corrigé et peut soumettre à nouveau une fois les erreurs corrigées.

IV. Tableau récapitulatif des étapes du projet PopBazaar

Étape	Fonctionnalité	Description
1A	Affichage des annonces par catégorie (visiteur)	Le visiteur peut voir les annonces actives filtrées par catégorie.
1B	Mes annonces (utilisateur connecté)	L'utilisateur connecté peut voir toutes ses annonces, qu'elles soient actives ou non.
1C	Filtres Toutes / Actives / Vendues	L'utilisateur peut filtrer ses annonces par statut dans la section "Mes annonces".
2	Pagination	Ajoute la pagination à la liste des annonces (dans les catégories et dans "Mes annonces").
3	Détails d'une annonce	Permet d'afficher les détails d'une annonce via son identifiant.
4	Ajouter une annonce	L'utilisateur connecté peut publier une nouvelle annonce via un formulaire.
5	Modifier une annonce	L'utilisateur peut modifier ses propres annonces via un formulaire prérempli.
6	Supprimer une annonce	L'utilisateur peut supprimer définitivement ses annonces.
7	Marquer comme vendue	Permet de marquer une annonce comme vendue sans la supprimer.
8	Authentification : inscription et connexion	Permet à un utilisateur de créer un compte ou se connecter pour accéder aux fonctionnalités réservées.
8.1	Déconnexion	Permet à l'utilisateur de se déconnecter proprement.
9	Affichage du profil utilisateur	L'utilisateur connecté peut voir les informations liées à son compte (nom, courriel, date d'inscription, etc.).

10	Validation des champs	Valide les champs des formulaires (ajout/modification/inscription), côté serveur, avec affichage des messages d'erreur dans les vues.
----	-----------------------	---

V. Fonctionnalités additionnelles facultatives

Les fonctionnalités suivantes ne sont **pas obligatoires**, mais peuvent être réalisées si vous souhaitez **pousser plus loin votre projet**, ou enrichir votre portfolio.

Vous pouvez en choisir une ou plusieurs, selon votre rythme et vos intérêts.

1. Ajouter une annonce aux favoris

Permettre à un utilisateur connecté de marquer une annonce comme **favori** afin de la retrouver plus tard.

- Icône ou bouton " ❤️ " dans l'annonce détaillée
- Section "Mes favoris" dans le profil
- Stockage dans une table **favoris** liant utilisateurs et annonces

2. Noter un vendeur (évaluation)

Permettre à un utilisateur ayant marqué une annonce comme "vendue" de **laisser une évaluation** sur le vendeur.

- Évaluation sur 5 étoiles avec commentaire (facultatif)
- Affichage des évaluations reçues dans le profil du vendeur
- Prévention des doublons (une seule évaluation par transaction)

3. Modifier les informations de son profil

Permettre à l'utilisateur connecté de modifier ses informations personnelles :

- Prénom, nom, courriel
- Bio, numéro de téléphone
- Image de profil (optionnel, fichier ou URL)

Un formulaire similaire à l'inscription pourrait être utilisé.

4. Recherche dans les annonces

Permettre aux visiteurs de **rechercher** des annonces via un champ de recherche dans la barre de navigation.

- Rechercher par mots-clés dans le titre ou la description
- Combiner la recherche avec un filtre de catégorie (facultatif)

5. Ajouter une image par annonce

Permettre d'**ajouter une image** lors de la création ou modification d'une annonce.

- Affichage de l'image dans la fiche détaillée
- Option de suppression ou remplacement de l'image
- Sauvegarde dans un dossier public/images

VI. Pondération et échéance

- Ce projet compte pour 20 % de la note finale, mais plus important encore, il vous prépare à l'examen final.
- Date de remise: le mardi 13 mai en fin de journée.
 - Une pénalité de 10 % par jour de retard, incluant les jours de congé, sera imposée, et ce, jusqu'à concurrence de 5 jours. (Après ce délai, la note attribuée est zéro.)
 - Fichier à remettre (pénalité jusqu'à 40 % si consigne non respectée)
 - Remettre le fichier compressé `projet3_nom_famille.zip` via LÉA. Vous devrez vous assurer que tous les fichiers nécessaires sont présents sans quoi vous serez pénalisés. **Le dossier `.git` doit être obligatoirement présent** pour que je puisse voir votre historique de git.
 - **Alternativement**, poster sur LÉA l'adresse du dépôsitoire GitHub de votre projet. **Vous devez vous assurer que j'ai été invité** (mon username GitHub: **DominicTremblay**) pour que je puisse y accéder.

VII. Barème de correction

- Code () 90 %
 - Fiabilité, validité, conformité aux spécifications
 - Qualité du code MVC (structure, clarté, répartition des rôles)
 - Bonne utilisation des vues dynamiques
 - Validation, contrôle d'accès
 - Présentation finale (navigation, clarté, style)
 - Liste des commits Git
- Style (commentaires, normes de programmation, clarté et disposition du code) 10%

VIII. Consigne

Style et conventions

- Vous êtes libres d'utiliser la convention de nommage ***snake case*** ou ***camel case***, à condition que son utilisation soit cohérente à travers l'application. Vous ne devez pas alterner entre les deux.

Documentation

1. Créer un fichier **README.md** avec le contenu suivant:
 - a. Nom du projet : Expliquer brièvement son objectif.
 - b. Auteurs : Indiquer les étudiants responsables du projet
 - c. Technologies utilisées : Expliquer les outils et langages employés.
 - d. Installation et exécution : Instructions claires pour exécuter le projet.
 - e. Fonctionnalités principales : Explication des modules clés.

2. Commentaires dans le Code

- a. Chaque fichier doit inclure :
 - Un en-tête expliquant le fichier et son rôle.

Ex fictif.:

```
/**  
  
gestion.php - Gestion des locataires  
  
Ce fichier permet d'afficher, ajouter et modifier des locataires.  
  
@author Dom  
@date 2024-02-10  
  
*/
```

- b. Des commentaires de fonction décrivant les paramètres, les valeurs renvoyées et le fonctionnement.

Ex. :

```
/**  
 * Ajoute un locataire dans la base de données.  
 * @param string $nom Nom du locataire  
 * @param string $email Email du locataire  
 * @return bool Retourne true si l'ajout est réussi, false sinon  
 */  
  
function ajouterLocataire($nom, $email) {...}
```

IX. Annexe I – Workflow Git/GitHub

1. Créer un dépôt central sur GitHub

- Un des membres crée un **nouveau dépôt GitHub** privé.
- Il invite les autres membres comme **collaborateurs** (paramètres → Collaborators).

2. Cloner le dépôt sur chaque poste de travail

Chaque membre exécute :

```
git clone https://github.com/nom-utilisateur/nom-du-depot.git
```

Vous devez tous **travailler à partir du même dépôt distant**.

3. Créer une branche par fonctionnalité

- Ne jamais coder directement dans main.
- Avant de commencer une tâche, créez une branche :

```
git checkout -b fonctionnalite/nom-de-la-fonction
```

Exemples :

- fonctionnalite/ajouter-annonce
- fonctionnalite/connexion-utilisateur
- correction/pagination
- amelioration/style-accueil

4. Commits réguliers et clairs

Pendant que vous codez :

```
git add .
git commit -m "Implémenter le formulaire d'ajout d'annonce"
```

- Commitez **souvent** et utilisez des messages explicites pour chaque étape.

5. Pousser la branche vers GitHub

Lorsque vous avez terminé une partie :

- `git push origin fonctionnalite/nom-de-la-fonction`

6. Faire une Pull Request (PR)

Depuis GitHub, ouvrez une **pull request** vers la branche `main` :

- Expliquez ce que la branche ajoute
- Assignez vos partenaires pour qu'ils relisent

7. Revue de code entre coéquipiers

- Avant d'approuver une PR, **lisez le code de votre coéquipier**
- Posez des questions si besoin, corrigez ensemble

8. Fusionner dans main

- Après validation, **fusionnez la branche** dans `main` depuis GitHub, **pas localement**.

9. Synchroniser votre code

Avant de démarrer une nouvelle fonctionnalité, **mettez toujours à jour votre code local :**

```
git checkout main  
git pull origin main
```

10. Résolution de conflits

Si vous avez des **conflits Git** :

1. Lisez attentivement les sections <<<<<, =====, >>>>>
2. Gardez le code souhaité, supprimez les marqueurs
3. Sauvegardez, puis :

```
git add .  
git commit -m "Résolution de conflit dans AnnonceController"  
git push
```

Bonnes pratiques

- Ne travaillez **jamais à deux sur la même branche**.
- Toujours **tester le code** avant de faire une PR.
- Si vous devez travailler côté à côté : communiquez bien et poussez/pullez régulièrement.