Membres de l'équipe 01-02 :
Bernier Justine Pasquet Clément
Malki Basma Troeira Paul-Adrien
Groupe 1-2

Rapport de SAE - Site web d'e-commerce, Ressource R302

Rendu le 16 Janvier - Année académique 2023-2024

SAE 3.01 : Développement Efficace

Dans le cadre du projet SAE 3.01 "Développement Efficace", notre équipe, composée de Bernier Justine, Pasquet Clément, Troeira Paul-Adrien, et Malki Basma, a travaillé sur la conception d'un site web marchand. L'objectif était de mettre en pratique nos connaissances en développant une plateforme de service de vidéo à la demande (VOD) où les utilisateurs peuvent acheter virtuellement les films de leur choix. On demande alors à ce que ce site web soit en capacité de proposer des articles en vente et de pouvoir les acheter.
Ce rapport vise à détailler nos fonctionnalités, nos choix de conception, ainsi que le fonctionnement de notre plateforme, mettant en avant notre approche méthodique pour garantir une efficacité et une optimisation maximale dans le développement du site.

Mise en place de méthodes pour montrer que l'application est correcte

Dans ce contexte, on s'attend à ce que l'application exécute les fonctionnalités attendues. En d'autres termes, notre tâche consiste à mettre en place un site capable de présenter des articles à la vente et de permettre leur achat.

Fonctionnalités attendues :

1. **Créer utilisateur lors de l'inscription :**

- **Précondition :**
  - Champs pseudonyme remplis, adresse mail conforme, mots de passe identique au deuxième champ de confirmation du mot de passe.
  - Pseudonyme inexistant dans la base de donnée
- **Assertion :**

```php
$userCompared = $userModel->getUserByUsername($pseudo);

if (isset($userCompared)) {
    $session = session();
    $session->setFlashdata( data: 'error',  value: 'Votre pseudonyme est déjà pris.');

    // Redirection vers la page précédente (le formulaire)
    return redirect()->to(base_url( relativePath: '/accountCreate'));
}

if ($password != $passwordConfirmation) {
    $session = session();
    $session->setFlashdata( data: 'error',  value: 'Vos mots de passes ne correspondent pas');

    // Redirection vers la page précédente (le formulaire)
    return redirect()->to(base_url( relativePath: '/accountCreate'));
}
```

Ceci est l'équivalent d'assertion pour notre site en php. Si une des précondition n'est pas respectée l'utilisateur est redirigé vers une page avec un message d'erreur lui expliquant l'origine de l'erreur.

Si toutes les conditions sont réunies et valide les tests, une nouvelle entrée dans la table utilisateur est créée.

2. **Connecter :**
   - **Précondition :**
     - Champs pseudonyme et mots de passe remplis.
   - **Assertion :**

```php
public function connecter()
{
    // Récupérez les données du formulaire
    $pseudo = $this->request->getPost('pseudo');
    $password = $this->request->getPost('password');

    // Création d'une instance du modèle UserModel
    $userModel = new UserModel();

    $user = $userModel->getUserByUsername($pseudo);

    if ($user && password_verify($password, $user['password'])) {
        // Les informations de connexion sont correctes
        // Gestion de la connexion de l'utilisateur, en utilisant les sessions

        $session = session();
        $session->set('user', $user);

        // Redirection vers la page d'accueil
        return redirect()->to(base_url( relativePath: '/'));
    } else {
        // Les informations de connexion sont incorrectes
        // Redirection vers la page de connexion avec un message d'erreur
        $session = session();
        $session->setFlashdata( data: 'error', value: 'Identifiant et/ou mots de passe incorrects.');

        return redirect()->to(base_url( relativePath: '/accountConnection'));
    }
```

Vérification qu'un utilisateur avec ce pseudo existe et que le mot de passe entré correspond au mot de passe chiffré stocké dans la base de données.

Si toutes les conditions sont réunies et valident les tests, l'utilisateur est connecté sur sa session.

3. **Charger page d'un film précis :**
   - **Précondition :**

- L'utilisateur doit avoir sélectionné un film précis.

- **Assertion :**

```php
$film = $this->filmModel->find($id);
if ($film === null) {
    throw new \CodeIgniter\Exceptions\PageNotFoundException('Film not found');
}
```

Vérification que le film existe dans la BD.

Si toutes les conditions sont correctes, l'utilisateur accède à la page du film sélectionné.

4. **Aimer un film :**
   - **Précondition :**
     - L'utilisateur doit aimer un film précis.
   - **Assertion :**

```php
// Vérifiez d'abord si la clé 'user' existe dans la session
if ($session->has( key: 'user')) {
    $id_user = $session->get('user')['id_user'];

    // Appeler la nouvelle procédure stockée pour liker/déliker un film
    $this->db->query( sql: "CALL like_movie(?, ?)", [$id_film, $id_user]);

    return redirect()->to(base_url( relativePath: 'filmFocused/'.$id_film)); // Redirigez vers une autre page après le traitement.
} else {
    // Gérez le cas où la clé 'user' n'existe pas dans la session
    // Vous pouvez rediriger l'utilisateur vers une page de connexion ou effectuer d'autres actions nécessaires.
    // Par exemple :
    return redirect()->to(base_url( relativePath: '/accountConnection'));
}
```

Vérification que l'utilisateur est bien connecté. Ici, pas besoin de vérifier si le film existe bien dans notre base de données car la fonctionnalité aimé un film se fait sur la page du film sélectionné auparavant. si oui, le "like" est enregistré. Sinon, l'utilisateur est redirigé vers la page de connexion.

5. **Payer un film :**
   - **Précondition :**
     - L'utilisateur doit être connecté
     - L'utilisateur doit appuyer sur le bouton d'achat
   - **Assertion :**

```php
// Récupérez les données du formulaire
$id_film = $this->request->getPost('id_film');
$id_user = $this->request->getPost('id_user');

// Nous vérifions si l'utilisateur est connecté
if (empty($id_user)) {
    $session = session();
    $session->setFlashdata( data: 'error', value: 'L utilisateur n\'est pas connecté pas.');

    return redirect()->to(base_url( relativePath: '/accountConnection'));
}


$userModel = new UserModel();

$userCompared = $userModel->getUser($id_user);

// Nous vérifions si l'utilisateur existe bel et bien
if (!isset($userCompared)) {
    $session = session();
    $session->setFlashdata( data: 'error', value: 'L utilisateur n\'existe pas.');

    return redirect()->to(base_url( relativePath: '/'));
}
```

Vérification que l'utilisateur est bien connecté, et que l'utilisateur est dans la base données.

Si toutes les conditions sont réunies et valident les tests, l'achat est autorisé et ainsi le film fait partie du registre personnel de l'utilisateur. Dans son registre personnel, l'utilisateur à maintenant accès au film entier.

Montrer que l'application n'est pas coûteuse en temps d'exécution et en mémoire

Quand on parle d'une application "pas coûteuse", cela signifie qu'elle doit répondre rapidement aux demandes des utilisateurs. Par exemple, lorsqu'un client effectue une recherche, l'application doit la traiter et fournir des résultats aussi rapidement que possible. Nous allons donc tester notre site pour évaluer ses performances et ses limites.

Traitement des données, lors de la construction des données (normalisation), par le SGBD choisi, par des procédures ad hoc.

Pour le traitement de nos données nous avons conceptualisé notre modèle en identifiant les tables nécessaires pour stocker les produits et les utilisateurs. Nous avons également mis en place une relation qui nous permet de suivre quels produits sont liés à quels utilisateurs, y compris la possibilité de "liker".

Ces tables sont normalisées dans notre application à l'aide de "Model" avec notre Framework Code Igniter qui fait le lien entre la base de données et les objets logiques de l'application.

Cela a comme avantage d'optimiser les lectures et écritures dans la base de données ainsi que le stockage en mémoire / cache pour l'affichage du site.

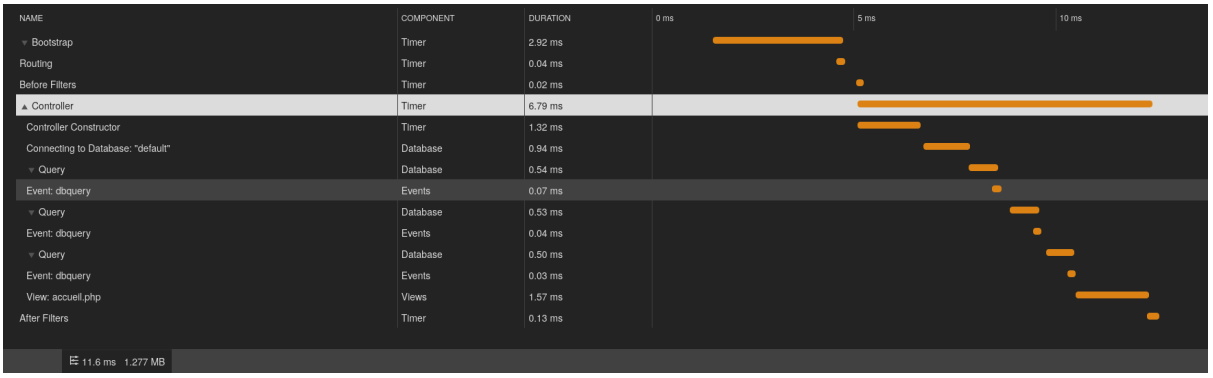
Nous avons mis en place plusieurs moyens pour rendre le plus propre possible notre base de données. En utilisant le SGBD MySQL, car celui-ci permet la mise en place d'une base de données relationnelle et il reste très accessible, simple d'utilisation et se couple très facilement avec le langage PHP, utilisé pour développer notre site Web. Nous avons aussi utilisé des procédures stockées pour les opérations d'achat.

Ci-dessous, se trouve les requêtes sql qui se chargent lorsque nous chargeons la page d'accueil.

Database (3 total Queries, 3 of them unique across 1 Connection)

Time	Query String
0.2 ms	SELECT * FROM `film` WHERE `est_serie` = 1
0.2 ms	SELECT * FROM `film` WHERE `est_serie` = 0 AND `genre` = 'Aventure'
0.1 ms	SELECT * FROM `film` WHERE `est_serie` = 0 AND `genre` = 'Science-fiction'

Ici on peut voir les différentes ressources demandées lors de l'affichage de la page et son temps d'exécution. Dans des conditions optimales du serveur, même la page d'accueil qui possède de nombreuses images par les vignettes des films met moins de 12 millisecondes à s'afficher et prend un peu plus d'un Mo en mémoire.



De même, nous pouvons remarquer que l'appel à la
BD prend très peu de temps, environ 0.5 ms par demande, ce qui est vraiment peu et montre le faible impact l'utilisation qu'une base de donnée peut avoir.

Peut être le fait d'héberger la base de données avec le serveur web sur le même permet de diminuer la durée nécessaire à la communication.

Performance

Nous avons utilisé Apache Benchmark pour obtenir les performances du site web par rapport aux nombres de connexions simultanées.

Les résultats se présentent sous cette forme. Le programme teste 100 connexions et en calcul le temps moyen par requête.

Avec cette méthode on peut tester d'augmenter le nombre d'utilisateur concurrent (ici à 1) pour savoir si notre site supporte une charge d'utilisateur

```
Concurrency Level:      1
Time taken for tests:   0.441 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      2503900 bytes
HTML transferred:       2457800 bytes
Requests per second:    226.90 [#/sec] (mean)
Time per request:       4.407 [ms] (mean)
Time per request:       4.407 [ms] (mean, across all concurrent requests)
Transfer rate:          5548.28 [Kbytes/sec] received
```

En changeant les arguments nous testons maintenant 1000 requêtes avec 100 connexions concurrentes :

```
Concurrency Level:      100
Time taken for tests:   4.224 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      25039000 bytes
HTML transferred:       24578000 bytes
Requests per second:    236.75 [#/sec] (mean)
Time per request:       422.392 [ms] (mean)
Time per request:       4.224 [ms] (mean, across all concurrent requests)
Transfer rate:          5788.97 [Kbytes/sec] received
```

On peut observer que le "Time per request" (4.224 ms) n'a pas augmenté avec ce nombre de requêtes. On va donc multiplier par 2 le nombre de requêtes concurrentes jusqu'à ce que le délai augmente de manière significative.

Requêtes Totales	Requêtes concurrente	Temps par requête moyen

100	1	4.407 ms
1000	100	4.224 ms
1000	200	4.228 ms
100 000	1000	4.467 ms

```
Concurrency Level:      1000
Time taken for tests:   446.727 seconds
Complete requests:      100000
Failed requests:        0
Total transferred:      2503900000 bytes
HTML transferred:       2457800000 bytes
Requests per second:    223.85 [#/sec] (mean)
Time per request:       4467.269 [ms] (mean)
Time per request:       4.467 [ms] (mean, across all concurrent requests)
Transfer rate:          5473.62 [Kbytes/sec] received
```

Le benchmark ne peut pas tester plus de 1000 requêtes concurrentes. Il permet uniquement de montrer que le site est parfaitement opérationnel avec 1000 utilisateurs concurrents effectuant 100 000 requêtes et ne montre aucune augmentation de délai. Ces résultats sont explicable notamment grâce au système de cache présent avec le FrameWork PHP utilisé.

Conclusion :

Notre équipe a travaillé de manière à développer une plateforme de vidéos à la demande répondant aux exigences du projet SAE 3.01. Nous avons mis en place des fonctionnalités telles que la création d'utilisateurs, la connexion, la recherche de films et l'achat de films.

Chacune a été développée avec des préconditions et des assertions pour assurer le fonctionnement de notre application. Notre projet a été conçu en pensant toujours à la manière dont on pourrait optimiser notre code mais aussi réduire la taille du contenu visuel, de sorte que même lorsque de nombreux clients réalisent de nombreuses requêtes, le temps d'attente ne soit pas rédhibitoire. Ainsi, nous avons réalisé.

Nous avons décidé de ne pas faire de tests en PHP et de plutôt opter pour des assertions qui permettent de trouver au plus tôt la cause d'un bug dans notre programme. Nous avons tout de même détaillé nos fonctionnalités, nos choix de conception et mis l'accent sur l'efficacité de notre application en termes de temps d'exécution et de gestion de la mémoire.

Dans l'ensemble, notre équipe a réussi à créer une boutique en ligne. Ce projet nous a permis d'apprendre en pratiquant à concevoir, administrer, optimiser, et conduire une application.