Rapport Projet PC3R - Implémentation d'une application AJAX

ShitPostLand - Un mini réseau pour shitposter

Dibassi Brahima, Said Mohammad Zuhair

27 Avril, 2023

Table des matières

1 Description L'application

ShitPostLand est un mini réseau social centré autour des shitposts. L'application permettra à ses utilisateurs de créer des comptes, de se connecter, de publier des messages, de commenter des messages, d'upvoter et downvoter des contenus. **ShitPostLand** est basé sur un système de capture de shitpost aléatoire récupéré depuis une API externe l'objectif étant de mettre en avant les meilleurs shitposts et utilisateurs.

1.1 Fonctionnalités Proposées

- Création de compte
- Suppression de compte
- Connexion avec session persistante
- Déconnexion
- Capture de shitpost aléatoire
- Upvote et downvote de shitpost
- Commentaire de shitpost
- Upvote et downvote de commentaire
- Affichage de Profil
- Affichage de shitpost capturé
- Système de Top des meilleurs shitposts
- Système de Top des meilleurs utilisateurs

1.2 L'API Externe

L'API externe utilisée est celle de thedailyshitpost.net cette API REST permet l'accés a un collection catalogue de shitpost sous différents formats (GIF, Image, Vidéo, Texte, etc. . .) dans le cadre de notre application nous avons choisi de récupérer des images et des GIFs.

1.3 Cas d'utilisation

ShitPostLand possède 2 types d'utilisateurs le visiteur et l'utilisateur connecté.

1.3.1 Visiteur

L'utilisateur non connecté ne peut pas directement interagir avec la communauté, il peut seulement se connecter, créer un compte ou observer les meilleurs posts et utilisateurs. Le visiteur peut également observer des posts aléatoires

1.3.2 Utilisateur connecté

L'utilisateur non connecté à lui accès à toutes les fonctionnalités de l'application.

1.4 Interface

- A remplir par des captures des fenêtres de l'application

2 Architecture

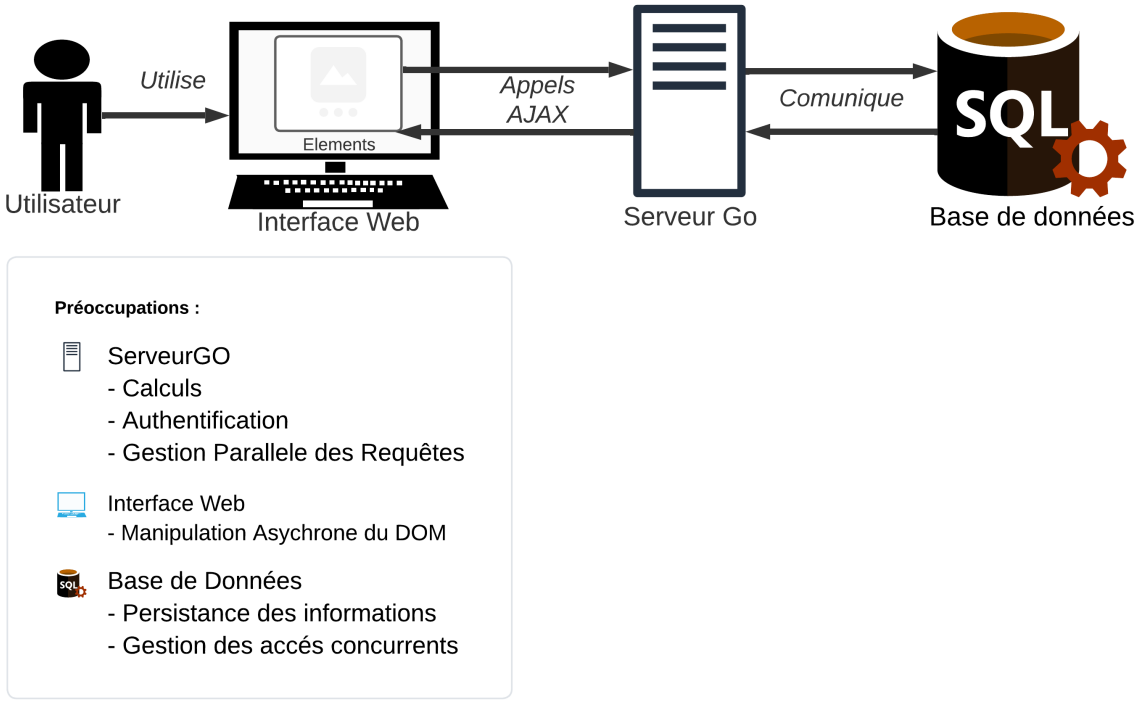

Figure 1: Vision Globale

2.1 Architecture Backend

Lors de la conception de l'architecture du backend nous avons choisi d'utiliser le langage GoLang pour la gestion du serveur HTTP et SQLite pour la base de données. Ces choix ont été motivés par la simplicité de GoLang et la facilité de déploiement et d'utilisation d'SQLite.

2.1.1 Serveur

Pour le serveur nous avons opté pour une architecture de type service, c'est-à-dire que chaque fonctionnalité de l'application est implémentée sous la forme d'un endpoint HTTP `app/api/*` qui est appelé par le client.

Sur chaque endpoint nous avons un handler qui va récupérer les données de la requête HTTP, les traiter et renvoyer une réponse au client. En s'inspirant du cours de PAF chacun des formats de requête et de réponse est encodé via une structure go et donc un type dans le fichier `inoutFormats.go`.

Ceci nous permet de réduire la vérification des JSON requêtes à une simple vérification du type.

Pour ne pas alourdir ce rapport, nous n'allons pas détailler la mise en œuvre en détail de chaque JSON mais nous vous invitons à consulter le code source du backend.

2.1.1.1 Les requêtes sont envoyées au format JSON et dependent du endpoint ciblée. Voici un exemple de requête pour le service `/api/login`:

```
{
    "login" : "admin",
    "mdp" : "admin"
}
```

2.1.1.2 Les réponses sont envoyées au format JSON et toute respectent la forme suivante:

```
{
    "Success": bool,
    "Message": string,
    "Result": interface{} // Peut être n'importe quel type JSON
}
```

2.1.1.3 Services Voici la liste exhaustives des services implémentés:

GET : Les services sans paramètres

- `/api/logout` : Déconnecte l'utilisateur.
- `/api/random_shitpost` : Récupère un shitpost aléatoire depuis l'API externe.

POST : Les services avec effets de creation

- `/api/create_account` : Crée un compte utilisateur.
- `/api/save_shitpost` : Capture un shitpost dans la base de données.
- `/api/post_comment`: Poste un commentaire sur un shitpost.
- `/api/post_comment_vote`: Applique un vote sur un commentaire.
- `/api/post_shitpost_vote`: Applique un vote sur un shitpost.

PUT : Les services avec paramètres

- `/api/login` : Connecte un utilisateur.
- `/api/get_public_profile` : Récupère le profil public d'un utilisateur.
- `/api/get_saved_shitpost` : Récupère un shitpost sauvegardé.
- `/api/get_comments` : Récupère les commentaires d'un shitpost.
- `/api/search` : Recherche de shitpost ou d'utilisateur.
- `/api/get_comment_list` : Récupère une liste de commentaires.
- `/api/get_saved_shitpost_list` : Récupère une liste de shitposts.
- `/api/get_top_users` : Récupère une liste des meilleurs utilisateurs.
- `/api/get_top_shitposts` : Récupère une liste des meilleurs shitposts.

DELETE : Les services avec effets de suppression

- `/api/delete_account` : Supprime un compte utilisateur et toute les données associées.

2.1.1.4 L'Authentication pour la mettre en place nous avons choisi d'utiliser des sessions persistantes, c'est-à-dire que lors de la connexion d'un utilisateur nous créons un token de session JWT crypté et signé avec une clé secrète.

Ce token est ensuite stocké dans un cookie HTTP qui est renvoyé à chaque requête par le client et dont l'intégrité est vérifiée par le serveur.

Toujours inspiré par le cours de PAF chaque endpoint nécessitant une authentification est représenté par le type `AuthServiceHandle` ceci permet lors du développement d'un nouveau service authentifié de forcer le passage par un middleware de vérification de l'authentification.

2.1.2 Base de données

2.2 Architecture Frontend

- Pourquoi React ETC
- A remplir
- une description du client: plan du site contenu des écrans, bien identifier a quels endroits apparaissent les appels aux différents composants du serveur (Genre Prendre les capture et pour certains images/Bouton de l'interfact dire les appels qui on lieu)
- Et la tu raconte ce que tu veux sur l'organisation du code, les librairies utilisées, les choix d'implémentation, etc...

3 Description de l'archive

Afin de lancer le projet en local, il faut se placer à la racine du projet et lancer la commande suivante.

```
bash run.sh
```

Et ensuite se rendre sur l'adresse suivante: http://localhost:25565/

3.1 Dépendances

Voici la liste des dépendances nécessaires pour lancer le projet:

3.1.1 Installation manuelle

- SQLite3 : `sudo apt install sqlite3` (Linux)
  ou https://www.sqlite.org/download.html (Windows)
  ou `brew install sqlite` (MacOS)
- GoLang : https://golang.org/doc/install
- NPM : `sudo apt install npm` (Linux)
  ou https://nodejs.org/en/download/ (Windows)
  ou `brew install node` (MacOS)

3.1.2 Installation automatique

- Sqlite3 Driver Pour Go
- jwt-go : Implémentation de JWT en Go

3.2 Code Source

Le code source du projet se trouve dans le dossier `Dev/` et est divisé en 2 sous-dossiers:

3.2.1 Source Frontend

Code source du client écrit en ReactJS.

- A remplir

3.2.2 Source Backend

Code source du serveur écrit en GoLang.

- `Helpers` :
  - `errors.go` : Code de gestion des erreurs.

  - `inoutFormats.go` : Contient les formats d'entrée et de sortie des requêtes JSON encodés en structure GoLang.
- `Database/` : Manipulation de la base de données SQLite.
- `auth.go` : Manipulation et vérification des tokens JWT.
- `handlers.go` : Handlers HTTP.
- `server.go` : Paramétrage du serveur HTTP.