INDEX

Description générale

Le projet **NetSoc** consiste à réaliser un réseau social minimal couplé à une régie publicitaire et exploitant un réseau de sites externes. Votre site web disposera d'un système d'authentification : il sera possible de s'inscrire, de se connecter, de se déconnecter, de supprimer son compte.

L'inscription consiste à fournir un triplet **login**, **mail** et **mot de passe**. Une fois connecté, l'utilisateur disposera d'une **page personnelle** et d'une **page nouveauté**.

La **page personnelle** lui servira entre autres à changer ses paramètres personnels :

Login	Prénom	A promouvoir
Mot de passe	Nom de famille	Site web
Mail	Date de naissance	Image site web
Avatar	Ville	Description du site

Cette **page personnelle** permettra également d'avoir des **conversations** : une **conversation** est un message d'ouverture, éventuellement suivis de **commentaires**. Il y a donc un formulaire pour lancer une nouvelle conversation et chaque conversation contient un formulaire pour envoyer un commentaire.

Pour finir, cette **page personnelle** contient, sauf si l'on est l'utilisateur dont c'est la page, un bouton **abonnement** permettant à l'utilisateur connecté de s'abonner aux publications de la page personnelle en question.

La **page nouveauté** est une page constitué des conversations des utilisateurs auxquels l'utilisateur actuellement connecté est **abonné**. Ces conversations sont triées par ordre ante-chronologique du dernier commentaire reçu (ou du début de la conversation si il n'y en a pas)

Toutes les trois conversations sur cette **page nouveauté**, une **publicité** sera insérée, cette publicité sera choisie en fonction des goûts détectés de l'utilisateur connecté.

Les conversations et commentaires doivent être accompagnés des logins et avatars de leurs auteurs, ainsi que de la date d'écriture.

Il sera possible à un utilisateur **d'exporter son bouton d'abonnement** à l'extérieur du réseau social à l'aide d'un code fournit par le réseau social et insérable sur un site tiers. Ce bouton sera constitué d'un script et son exécution provoquera la remonté des informations des visiteurs vers le réseau social – même si ceux là n'y ont pas de compte.

Les informations ainsi ramassées permettront de procéder ensuite à du ciblage publicitaire.

Une séparation API/Page vous est proposée dans ce sujet. Vous devez en respecter le formalisme ainsi qu'implémenter des tests unitaires avec couverture de test associés.

Pour cela, vous utiliserez PHPUnit. Son principe est très simple : via la commande phpunit, vous pourrez établir :

- Une liste des dossiers contenant le code à prendre en considération (--whiteslist)
- Un dossier de sortie des conclusions d'analyse (--coverage-html)
- Un script de mise en place initial (--bootstrap fichier_bootstrap.php)
- L'option –strict-coverage établit que les tests dit « risqués » sont ignorés dans la couverture.
- Et enfin le dossier contenant les tests.

Les fichiers de tests seront considéré s'ils respectent le format *Test.php. (Remplacez * par n'importe quoi). De plus, dans ces fichiers, vous devrez déclarer des classes PHP étendant le type TestCase, exemple :

```php
use PHPUnit\Framework\TestCase;

// Sera appelée par requête POST
class XTestCase extends TestCase
{
	public function testX()
	{
		$this->assertSame(4, 5);
	}
}
```

Remarquez également la politique de nommage de la classe XtestCase et de la fonction membre testX. Vous êtes invités à respecter ce format.

La fonction membre assertSame provoquera la notification d'une erreur si ses paramètres ne correspondent pas.

Attention : vous allez réaliser un site web dynamique avec un contenu utilisateur riche. Cela signifie que vous devez **contrôler** tout ce qui entre dans votre base de donnée : messages, fichiers… Imaginez un seul instant par exemple qu'un utilisateur entre du HTML, ou pire, du javascript comme message : il serait affiché chez les autres utilisateurs. **Cette situation ne doit pas arriver.**

Plusieurs API vous seront demandées dans ce projet. Vous devez respecter un formalisme qui nous permettra de faciliter votre évaluation.

Vous allez devoir fournir un fichier .htaccess permettant de créer des URL de type /api/action redirigeant vers les fichiers porteurs des fonctions afin de faciliter l'utilisation externe et le test de vos interfaces.

Pour chacune de ces API, une fonction PHP correspond. Le nom de cette fonction PHP devra être accessible via l'URL. Par exemple la fonction update_profile sera incarnée par l'URL /api/update_profile et ses paramètres seront un JSON envoyé sur l'entré standard dont les champs décodés seront les paramètres de la fonction.

Les fonctions de récupération utiliseront la méthode GET. Les fonctions d'ajout de nouveau utiliseront la méthode POST. Les fonctions de modification utiliseront la méthode PUT. Les fonctions de suppression utiliseront la méthode DELETE.

API d'authentification

Ce projet commence par la réalisation d'une API d'authentification. Une API est une interface pour logiciel. Votre API, contenue dans le fichier account.php comportera les fonctions suivantes :

```
// Sera appelée par requête POST
// /api/account
// "login" valant le pseudonyme à inscrire
// "pass" valant le mot de passe du compte
// "mail" valant le mail associé au compte
// Inscrit en base de donnée ces valeurs
// et connecte l'utilisateur
function register($login, $pass, $mail)
```

```
// Sera appelée par requête PUT
// /api/account
// "login" valant le pseudonyme à inscrire
// "pass" valant le mot de passe du compte
// Inscrit en cookie les valeurs de connexion
// Permet donc de se connecter si valeurs vide
function login($login, $pass)
```

```
// Exploite les cookies (variable $_COOKIE)
// Regarde si les valeurs de connexion sont dans les
// cookies. Renvoi les valeurs utilisateur si oui.
// La fonction est accessible via une requête GET
// /api/account
function is_logged()
```

```
// Sera appelée par requête DELETE. Détruit le compte
// actuellement connecté
// /api/account
function unregister()
```

Les informations des paramètres seront passé en **body**, formaté en JSON. Les champs auront le même nom que les variables indiqués ici. Le retour sera également un JSON, disposant au moins d'un champ **success** pouvant valoir vrai ou faux. **is_logged** renverra **login**.

Faites attention avec la fonction **unregister** : votre base de donnée doit rester cohérente ! Par exemple, les messages employant l'identifiant $id comme identifiant de l'auteur ne supporteront pas la suppression de cet utilisateur. Plusieurs choix s'offrent à vous pour gérer ce problème, à vous d'y réfléchir.

Le fichier **account.php** doit pouvoir fonctionner seul : un script l'incluant et appelant ces fonctions doit pouvoir fonctionner. Par ailleurs, c'est ce que fera votre fichier **test_account.php**, qui testera que les fonctions tournent bien. Ce fichier exploitera évidemment une base de donnée différente de celle du site.

Vos fonctions exploiteront une base de donnée SQL pour stocker les informations que vous avez besoin de stocker. Voici quelques exemples de requêtes, utilisant MySQL. Vos tests unitaires utiliseront SQLite. Vos fonctions de manipulation de bibliothèques devront donc être neutre MySQL/SQLite.

```php
$Database = new mysqli("localhost", "login", "pass", "bdd_name");
$login = "Login";
// Inscrit l'utilisateur "Login" dans la base de donnée
$Database->query("INSERT INTO user (login) VALUES ('$login')");
// Récupère l'identifiant généré
$id = $Database->insert_id;

// Affiche tous les utilisateurs enregistrés
$user_query = $Database->query("SELECT login FROM user");
while (($user = $user_query->fetch_assoc())!= NULL)
{
  assert($user["login"] == $login);
}

// Affiche toutes les informations de l'utilisateur
$user_query = $Database->query("SELECT * FROM user WHERE id = $id");
$user = $user_query->fetch_assoc();
assert($user["login"] == $login);

// Modifie le pseudonyme de l'utilisateur "Login"
$Database->query("UPDATE user SET login = 'pseudo' WHERE id = $id");
$user_query = $Database->query("SELECT * FROM user WHERE id = $id");
assert($user_query->fetch_assoc() == NULL);

// Supprime l'utilisateur "pseudo"
$Database->query("DELETE FROM user WHERE id = $id");

// Deconnexion d'avec la base de donnée
unset($Database);
```

Les requêtes SQL servent à transmettre et à récupérer des informations depuis une base de données SQL. Elles peuvent être complexes, comportent de nombreuses options. N'hésitez pas à vous renseigner au besoin, les exemples ci-dessus sont triviaux.

La loi française vous **interdit** de stocker les mots de passe directement dans votre base de donnée. Vous **devez** « hasher » les mots de passe : ils s'agit d'une transformation à sens unique générant des identifiants supposés unique. La fonction **hash** en php vous permet de le faire. Différents algorithmes existent : renseignez-vous sur l'Internet.

Page d'authentification

Réalisez maintenant une ou plusieurs pages disposant des formulaires de connexion/déconnexion/inscription. Ces pages devront exploiter votre API d'authentification en permettant d'effectuer toutes les taches rendues possibles par celle-ci.

Les formulaire d'inscription et de connexions ne doivent pas être affiché si l'utilisateur est connecté. Le formulaire de déconnexion ne doit pas être affiché si l'utilisateur n'est pas connecté.

Insérez ces pages comme vous le souhaitez dans l'architecture de votre site.

C'est cette page qui s'occupera des paramètres POST, GET et des COOKIES et appellera les fonctions de l'API d'authentification.

Vous devriez utiliser Javascript pour les interactions navigateur→serveur afin d'améliorer l'expérience utilisateur.

API du profil

Vous allez maintenant réaliser l'API du profil de l'utilisateur. La fonction **get_profile** permettant de récupérer les informations de l'utilisateur connecté. Cette API sera dans le fichier **profile.php**.

```
// Renvoi les valeurs utilisateur si connecté, via GET
// /api/profile
function get_profile()
```

Pour pouvoir éditer les informations utilisateurs, il faudra disposer des fonctions suivantes :

```
// Le tableau $informations respectant ce format :
// {
//     "avatar" : $_FILES['tmp_name'],
//     "login" : "login",
//     "password" : "password"
// }
// La seule information non éditable étant l'adresse
// mail. Si un champ est vide où n'existe pas, il
// n'est pas modifié.
// /api/profile, via PUT
function update_profile($id, array $informations)
```

L'avatar, si il est précisé, doit être un fichier PNG. Si il ne fait pas la bonne taille, vous devez le redimensionner (voir PHP GD). Il sera transmit encodé en base64.

API de la page personnelle

Vous allez maintenant écrire l'API de la page personnelle. Cette page est l'endroit où les conversations vont être affiché, accompagné de leurs commentaires, ainsi que les caractéristiques de l'utilisateur.

Votre API sera située dans le fichier profile.php. Ce fichier doit pouvoir être utilisé seul. A vous de vous arranger pour qu'il fonctionne en incluant de façon intelligente les accès à la base de données situé dans authentication.php.

```
// Sera appelée en cas de paramètres GET
// $id_user est l'identifiant de la personne dont
// on veut la page. Si id_user n'est pas précisé,
// On récupère celle de la personne connectée.
// Ce paramètre ne passe pas par JSON mais via l'URL
// /api/wall/ ou /api/wall/id
function get_wall($id_user = -1)
```

La fonction get_wall permet d'obtenir la page personnelle d'une personne en particulier. Si aucune personne n'est spécifiée, alors c'est la page de la personne connectée qui sera servie. Il est possible qu'une pagination soit nécessaire (Voir SQL LIMIT). Dans ce cas, vous pourrez étendre votre fonction get_wall avec des paramètres optionnels pour le faire. Votre API attendra les valeurs pour compléter son appel via un JSON au format libre. Le format de retour est sur la page suivante.

```
// Sera appelée par requête POST
// L'id transitera par URL, son absence indique la
// page de l'utilisateur courant. Le message via
// le body du POST en texte brut
// /api/wall/ ou /api/wall/id_user
function post_conversation($id_user, $message)
```

```
// Sera appelée par requête PUT.
// L'id transitera par URL, cette fois il s'agit
// de l'id du message, non optionel.
// Le message passera par le body, en texte brut.
// /api/wall/id_conversation
function post_comment($id_conversation, $message)
```

Et bien sur, il vous faudra également vous pencher sur la capacité à suivre un autre utilisateur.

```
// Sera appelée par requête POST
// Via /api/follow/id_user, avec id_user valant
// l'id de la personne a qui on s'abonne
function follow($id_user)
```

```
// Sera appelée par requête DELETE
// Via /api/follow/id_user, avec id_user valant
// l'id de la personne de qui on se désabonne
function unfollow($id_user)
```

L'arborescence en retour de l'appel à get_wall est la suivante :

```
[ // Tableau des conversations
      {
            "id_message" : id_message,
            "id_author" : id_author,
            "login_author" : "login_author",
            "avatar_author" : "avatar_url", // Pas de base64 !
            "message" : "message",
            "commentaries" : [ // Tableau des commentaires
                               // Dans l'ordre chronologique
                  {
                        "id_comment" : id_comment,
                        "id_author" : id_author,
                        "login_author" : "login_author",
                        "avatar_author" : "avatar_url",
                        "message" : "message"
                  }
            ]
      }
]
```

Page personnelle et de profil

Réalisez maintenant la page qui va afficher les données récupérable via l'API de la page personnelle. L'affichage espéré est l'écriture des champs caractérisant l'utilisateur dont c'est la page personnelle ainsi qu'une suite de conversations par ordre ante-chronologique de leur commentaire le plus récent.

Les logins des utilisateurs ayant posté doit mener à leur page personnelle, et leurs avatars doit être visible. En l'absence d'avatar, un avatar par défaut doit être affiché.

Concernant les données personnelles, cette page offre, si l'utilisateur est sur sa propre page, un simple formulaire – ou une simple série de formulaires - comprenant un élément particulier : le champ de mise en ligne de fichier pour l'avatar de l'utilisateur.

API de la page nouveauté

Vous allez maintenant écrire l'API de la page nouveauté. Cette page est l'endroit où les conversations vont être affiché, accompagné de leurs commentaires, cela en fonction des abonnements de la personne connectée.

Votre API sera située dans le fichier news.php. Ce fichier doit pouvoir être utilisé seul. A vous de vous arranger pour qu'il fonctionne en incluant de façon intelligente les accès à la base de données situé dans authentication.php.

```
// GET, sur /api/feed/
// Une page peut-être précisé dans l'URL.
function get_news($page = -1)
```

La fonction get_news_page permet d'obtenir la page des news de l'utilisateur actuellement connecté en fonction de ses abonnements. Si aucun abonnement n'a encore été pris, alors un tableau vide est renvoyé. Le format des news est le même que celui des conversations, étant donné que les nouveautés sont des conversations…

Une autre fonction que vous ne pouvez pas encore écrire sera à réaliser :

```
// GET, sur /api/ads/ ou /api/ads/code
// Une publicité est renvoyée.
// Le paramètre in indique que la pub est pour le
// netsoc lui même (pas de code) et non pour
// l'exterieur.
function get_ads($in = true)
```

A l'heure actuelle, vous pouvez implémenter une version vide de celle-ci : vous l'appellerez toutes les trois publications de nouvelles. Le format sera le suivant :

```
{
      "alt" : "picture_alt",
      "link_title" : "link_title",
      "link_target" : "url",
      "picture" : "ads_picture_url" // Pas de base64 !
}
```

Cette fonction renverra une publicité jugée pertinente pour l'internaute.

Apparence générale

Votre site web doit être beau et autant que possible, capable de s'adapter à une résolution de téléphone portable, portrait comme paysage, autant qu'à un ordinateur disposant d'un écran FULL HD.

N'hésitez pas à utiliser en particulier les transitions CSS.

Vous pouvez trouver de l'inspiration sur internet, par exemple sur

https://codemyui.com/tag/pure-css/

Javascript

Aujourd'hui, il est habituel de voir les sites être rafraîchi non plus au rythme des changements de pages et des formulaires mais en temps réel.

Ce mécanisme, **Ajax, ou** plus anciennement **XmlHTTPRequest** ne vous est pas inconnu. Il consiste à demander au langage qui s'exécute dans le navigateur (le « front »), le JavaScript, d'aller chercher des informations auprès de la base de donnée, enrobé dans une API écrite en PHP (le « back ») et de modifier **a chaud** le HTML. La fonction **fetch** apparue récemment permet également d'effectuer cette tâche.

L'utilisation de cette technique est obligatoire dans le cadre de ce projet.

Réseau des sites extérieurs

Sur la page personnelle d'un utilisateur, si cette page est la sienne, l'utilisateur devra pouvoir exporter un code à insérer dans son propre site. Ce code ne fera rien de lui-même : il provoquera la recherche d'un script accessible depuis le réseau social et qu'il exécutera.

Ce script sera situé dans **/api/telescreen**, à la condition de réaliser un GET. Il disposera de paramètres URL permettant d'identifier à qui appartient ce code.

Ce script distant provoquera l'affichage d'un bouton « Je m'abonne » sur le site tiers. Il collectera également toutes les informations imaginables sur l'utilisateur actuellement en visite sur ce site tiers :

- Son adresse IP
- Ses cookies sur ce domaine
- Les thèmes abordés par le site visité, à l'aide du parcours du site *et de la description fournie par le propriétaire du site via le champ Description du site indiqué sur le NetSoc.*

Ces informations seront envoyées par le script vers le réseau social qui les enregistrera et tentera d'établir une relation avec un utilisateur enregistré. Qu'un utilisateur existe ou non, ces informations seront conservées.

La cible du script sera **/api/telescreen**, via la méthode POST.

L'objectif de proposer ce bouton « Je m'abonne » aux sites tiers est, de manière publique, de les aider « à promouvoir leur page NetSoc, former une communauté » ainsi que des variantes de cette propagande.

L'objectif caché est de permettre la collecte d'informations sur les visiteurs en utilisant leurs sites comme relais : en apprenant des choses sur ces visiteurs, il sera possible plus tard de leur servir de la publicité ciblée contre de l'argent.

Régie publicitaire

A partir des informations dont vous disposez sur les utilisateurs et visiteurs divers d'internet, vous allez pouvoir réaliser deux tâches :

La première est l'affichage de publicités ciblées sur la page nouveauté, en exploitant du contenu directement accessible sur le NetSoc (Typiquement, des conversations d'utilisateurs inconnus mais dont les thèmes recoupent les intérêts de l'utilisateur)

La seconde est de rendre disponible via l'API incarné par le fichier **ads.php** disposant d'un accès au fichier **news.php** et à sa fonction **get_ads** un moyen de générer une publicité pour un site tiers disposant d'un code spécifique. Cette publicité sera pour un utilisateur du réseau NetSoc dont le champ « A promouvoir » est vrai et exploitera le champ « Image site web » pour constituer une publicité de bonne taille menant sur le site web de la personne (et non pas sur sa page NetSoc).

Vous vous arrangerez pour que les clics sur cette publicité soient comptabilisés. Vous indiquerez via une table de votre base de donnée que l'utilisateur dont la publicité a été faite vous doit désormais 2 Francs. Vous verserez à l'utilisateur du site porteur de votre publicité 1 Franc. Le Franc restant est pour vous, en tant que NetSoc.

Les utilisateurs débités comme crédités doivent pouvoir lire le statut de leur compte sur leur page personnelle.