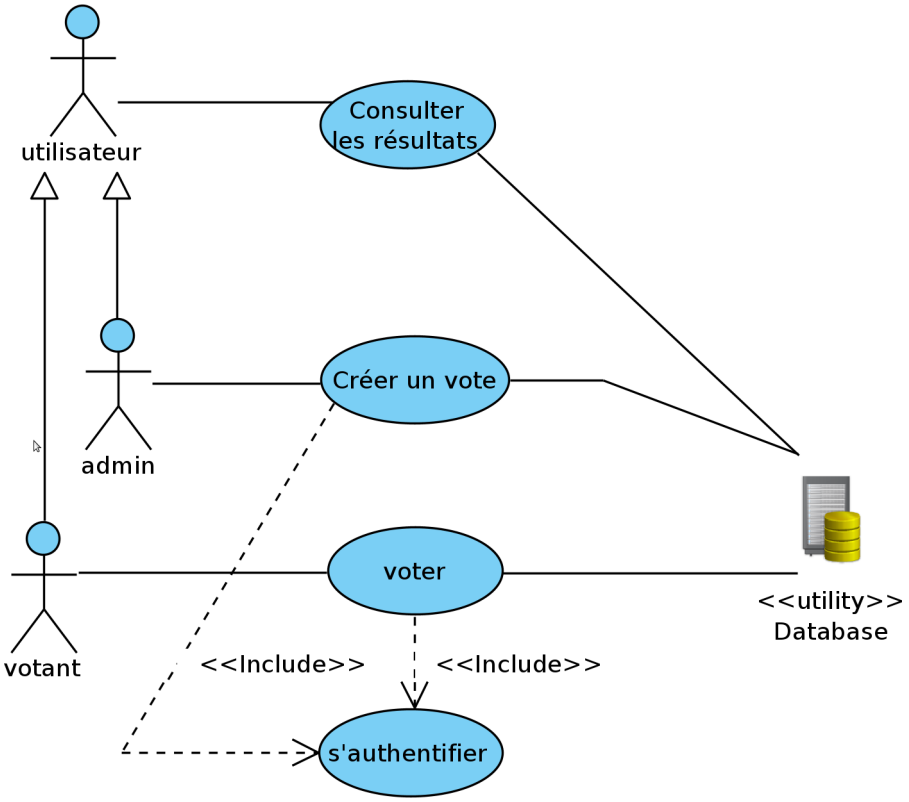
Projet Electrovote

Application WEB de gestion de votes

Table des matières

Objectifs

Construire une application en PHP qui permette de voter de manière unique et authentifié sur un sujet défini. Le diagramme des cas d'utilisation qui suit précise les besoins métier.



Répartition des tâches

Seul un administrateur peut concevoir un nouveau vote.
Le vote contient

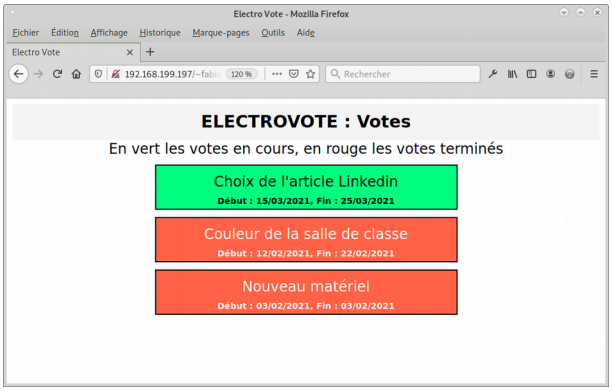
- Page d'accueil : liste des votes (en cours ou terminé)
- Page de vote : Intitulé du vote, choix possibles, possibilité de voter

L'administrateur génère des clefs de vote, une clef de vote est attribuée à un utilisateur, elle lui permet de voter.

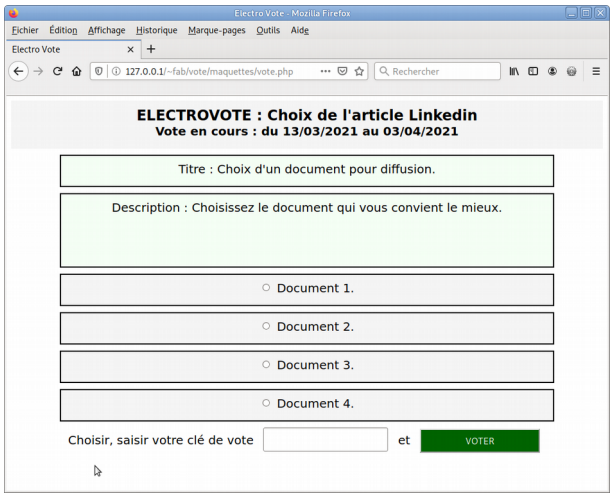
Maquettes de navigation

Vue utilisateur

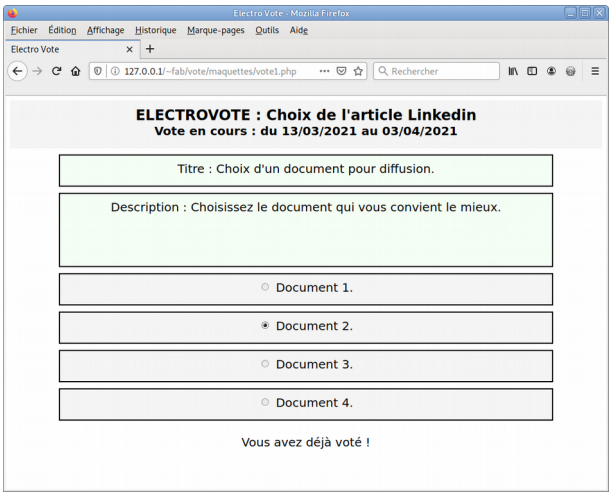
Vue d'accueil de l'utilisateur



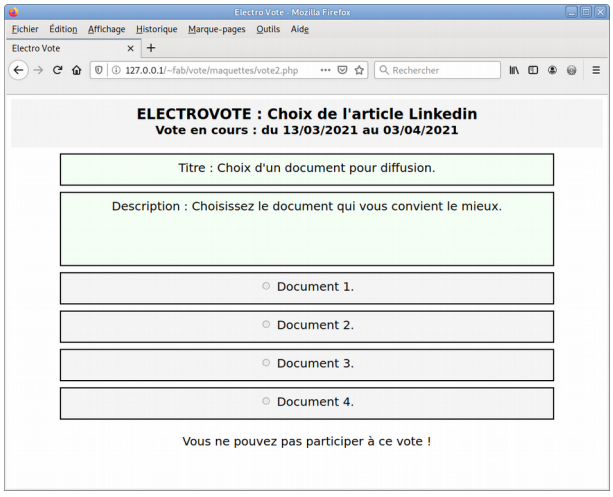
Vue pour utilisateur pouvant participer au vote



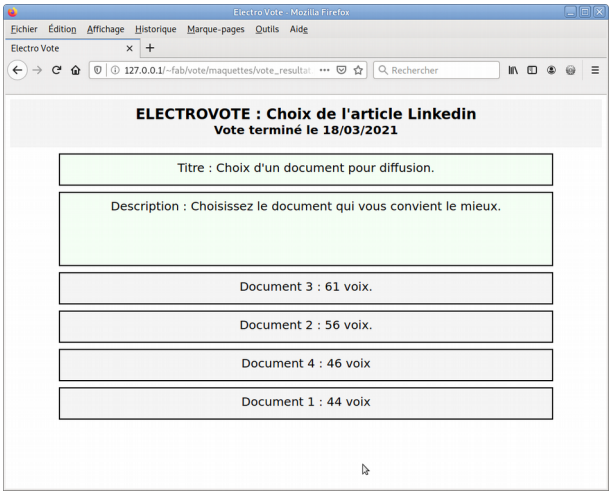
Une fois qu'il a voté ...



Vue publique d'un vote en cours



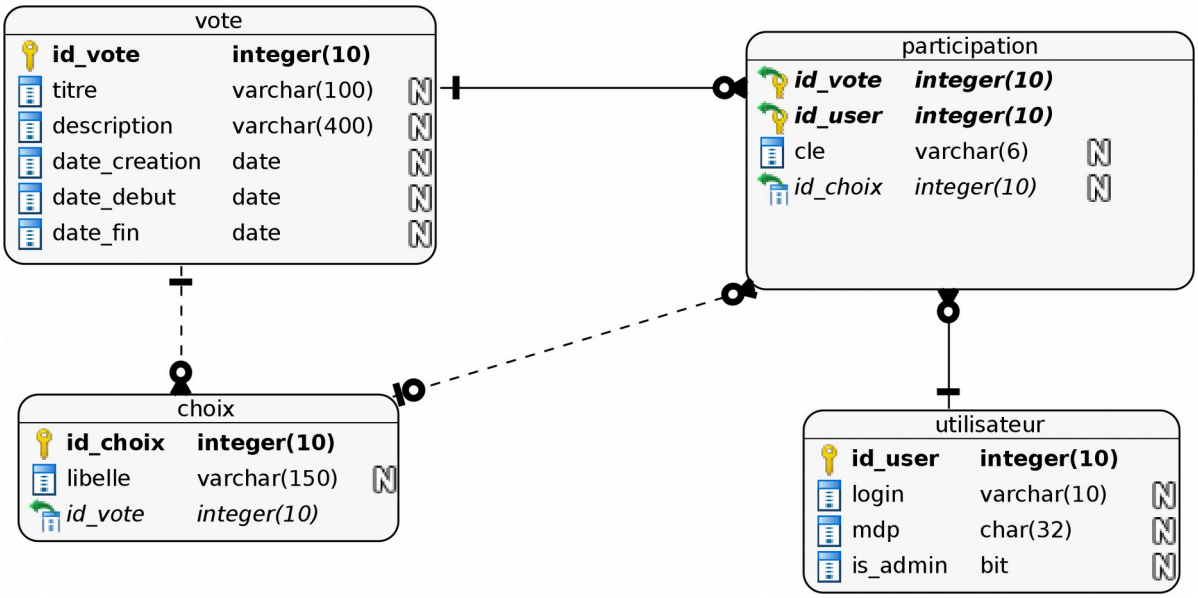
Vue publique d'un vote terminé



Vue administrateur

Modèle de données et base associée

Modèle de données.



Scripts de création

Nous créons la base de données et l'utilisateur lié.

```
create database vote;
create role vote_admin password 'admin' login;
grant all on database vote to vote_admin;
```

Puis les tables

begin transaction;

```
create table utilisateur
     (
     id_user         integer              primary key,
     login                varchar(10)          unique,
     mdp             char(32)             not null,
     is_admin        boolean              default false
     );

create table vote
     (
     id_vote              integer              primary key,
     titre                     varchar(100),
     description         varchar(400),
     date_creation  date,
     date_debut           date,
     date_fin             date
     );
```

```
create table choix
     (
     id_choix       integer             primary key,
     libelle        varchar(150),
     id_vote        integer             references vote
     );

create table participation
     (
     id_vote        integer        references vote,
     id_user        integer        references utilisateur,
     cle            varchar(6),
     id_choix       integer        references choix,
     primary key (id_vote, id_user)
     );

commit;
```

L’application vue en requêtes SQL

Ces requêtes reflètent ce que va faire l'application.

Peuplement des utilisateurs

```
begin transaction;

insert into utilisateur (id_user, login, mdp) values
     (0, 'admin', md5('admin')),
     (1, 'florent', md5('toto')),
     (2, 'charles', md5('tutu')),
     (3, 'antoine', md5('toto')),
     (4, 'tristan', md5('titi')),
     (5, 'bryan', md5('toto'));

update utilisateur set is_admin = true where id_user = 0;

commit;
```

Création d'un vote

```
begin transaction;

insert into vote ( id_vote, titre, description, date_creation,
                   date_debut, date_fin) values
     (1, 'Choix document linkedin',
     'Choisir le document sur le numérique responsable qui
     sera partagé sur Linkedin.',
     '23/03/2021', '23/03/2021', '30/03/2021');

insert into choix (id_choix, libelle, id_vote) values
     (10, 'document 1', 1),
     (12, 'document 2', 1),
     (11, 'document 3', 1),
     (13, 'document 4', 1);

commit;
```

Définition des participations

L'administrateur va définir qui sont les utilisateurs qui vont pouvoir participer à un vote donné.

```
begin transaction;

    -- inscription des utilisateurs autorisés à voter
    insert into participation (id_vote, id_user, cle) values
         (1, 2, 'du*tr4'),
         (1, 3, 'dty-gR'),
         (1, 4, 'Kjuht/');

    select titre, login
         from participation p
         inner join utilisateur u on u.id_user = p.id_user
         inner join vote on vote.id_vote = p.id_vote;

commit;
```

Action de voter

Un utilisateur autorisé pourra participer à un vote.

```
begin transaction;

    -- vote d'un utilisateur (antoine choisit le document 2)
    update participation
         set id_choix = 12
              where id_user = 3
              and id_vote = 1;

commit;
```

Troisième étape : authentification

Le principe de l'authentification repose sur 3 parties

1. Un échange entre formulaire et une page PHP qui va pouvoir traiter les valeurs passées au formulaire. Une vérification du couple login/mot de passe qui se fera par la suite grâce à une vérification dans une table d'une base de données.
2. La notion de session qui correspond à une reconnaissance du client par le serveur.
3. La mise en place d'une variable de session qui va permettre de savoir si un utilisateur qui accède à une page protégée a été authentifié.

Nous décomposons en 3 pages pour visualiser au mieux le procédé.

Formulaire

Il permet à l'utilisateur de saisir un login et un mot de passe

```
<form method="get" action="form_go.php">
     <input type="text" name="log">
     <input type="text" name="pass">
     <input type="submit">
</form>
```

et d'être ainsi reconnu

```
     if($_GET['log']=='admin' && $_GET['pass']=='pass')
          {
          echo 'OK, vous êtes autorisé.<br />';
          }
     else
          {
          echo 'N\'essayez pas d\'accéder au site.';
          }
```

Session

Chaque page qui sera sous session doit débuter par un session_start()
Le serveur (re)connaît ainsi le client qui l'interroge grâce au cookie de session du client et à l'id de session stocké sur le serveur.

```
<?php
     session_start();
```

Vérification de la variable de session

Création de la variable de session lors de la vérification du login.

```
<?php
     session_start();
     echo '<br />';
     if($_GET['log']=='admin' && $_GET['pass']=='pass')
          {
          echo 'OK, vous êtes autorisé.<br />';
          $_SESSION['login']='toto';
          echo '<a href="admin.php">Page protégée</a>';
          }
```

Utilisation de cette variable pour vérifier que l'on s'est authentifié auparavant.

```
<?php
     session_start();
     if(isset($_SESSION['login']))
          {
          echo 'Vous êtes autorisé.';
          }
```

Quatrième étape : Pages d'administration

Bibliothèques utiles

La génération automatique d'une clef sera utile.

```
function generer_cle()
     {
     $chaine = '!!@@##**2233445566778899aaaaaaaabbccddeeee
     eeeeeeeefghiiiiijkmmnnooooooppqrrrrsssttwxyzAAAAAAAAB
     BCCDDEEEEEEEEEFGHJKLMMNNPPQRRRRSSSTTWXYZ';
     $taille = strlen($chaine);
     $cle = $chaine[rand(0,$taille)].$chaine[rand(0,$taille)].
     $chaine[rand(0,$taille)].$chaine[rand(0,$taille)].
     $chaine[rand(0,$taille)].$chaine[rand(0,staille)];
     return $cle;
     }
```

Ajout d'un utilisateur

L'ajout repose sur deux pages (qui pourraient être réunies en une seule).
Il faut d'abord un formulaire qui permet à l'administrateur de saisir les données du nouvel utilisateur. Nous voyons ici les principes de base, il faudra tenir compte de la gestion des erreurs et de la sécurité pour être pleinement opérationnel, ainsi que de l'UX design.

```
<form action="ajout_util_go.php" method="get">
     <label>Login <input type="text" name="lo"></label>
     <label>Mot de passe <input type="text" name="mp"></label>
     <input type="submit" value="Enregistrer">
</form>
```

Il faut ensuite traiter les deux saisies dans la page suivante, d'abord le login.

```
<?php
     $login = $_GET['lo'];
```

Pour le mot de passe, pour des raisons de sécurité, on ne conserve que l'empreinte et pas le mot de passe en clair.

```
     $pass = md5($_GET['mp']);
```

On se connecte à la base.

```
     $conn = pg_connect('host=127.0.0.1 dbname=vote
                         user=vote_admin password=admin');
```

Puis on construit la requête SQL sous forme de chaîne de caractères. Nous avons 3 valeurs à passer :

- le login : $login
- le mot de passe : l'empreinte md5 du mot de passe saisie par l'administrateur.
- L'id_user, identifiant de la ligne

Nous n'avons pas à demander à l'administrateur de saisir l'identifiant. Une solution possible est de définir la colonne id_user en "serial", un mécanisme de séquence qui permet d'incrémenter automatiquement la clé primaire.
Nous allons ici partir du principe que la saisie est mono utilisateur et utiliser une requête imbriquée pour déterminer la prochaine valeur disponible pour la clé primaire. Attention, ce mécanisme nécessite qu'il y ait par défaut déjà au moins une ligne dans la table : le login de l'administrateur par exemple). L'idéal est de créer une transaction, nous verrons cela par la suite.
Pour construire la requête, nous charchons la valeur maximum de la clé primaire dans la table

```
select max(id_user) from utilisateur ;
```

Il suffit par la suite d'ajouter 1. Ce qui donne la construction suivante pour la requête.

```
    $chaine_req = 'insert into utilisateur (id_user, login, mdp)
values ((select max(id_user)+1 from utilisateur), '."'".$login."',
'".$pass."')";
```

On n'oublie pas de faire des var_dump() pour vérifier que la requête construite est correcte. On peut lancer la requête en direct sur la base pour vérifier que celle-ci s'exécute correctement.
Une fois certain que la chaîne est correcte, on envoie cette chaîne (requête SQL) au serveur de données Postgresql.

```
    $req = pg_query($chaine_req);
```

On peut, et on doit, vérifier que l'insertion s'est bien passée.

```
    if($req)
         {
         echo 'Enregistrement effectué.';
         }
    else
         {
         echo 'Enregistrement échoué';
         }
```

On ferme la connexion et on propose un retour à une autre page, menu ou saisie des utilisateurs.

```
    pg_close($conn);
?>
<a href="ajout_util.php">Retour à l'ajout d'utilisateur</a>
```

Sécurité des échanges et injection SQL

Avec le code précédent, il est facile d'injecter un code malin visant à détruire des données par exemple. En effet, l'utilisateur peut saisir à la place du login, un ensemble de commandes SQL en utilisant un " ;" entre chacune d'entre-elles.
Dans notre cas présent, il est facile de traiter le problème en simplement s'assurant que la taille de la chaîne saisie est bien inférieure à 10. Quant au mot de passe, il est passé à une fonction qui renvoie son empreinte, donc aucune inquiétude de ce côté.
Il existe plusieurs moyens pour traiter les chaînes saisies et éviter les injections SQL, l'important est de ne pas oublier en amont, lors du développement, de traiter ce points.

Exemple de traitement.

```
<?php
    // protection des valeurs
    $login = substr($_GET['lo'], 0, 10);
Cinquième étape : page de consultation
```

Ajout d'un vote

L'ajout du vote se fait en deux étapes, d'abord en renseignant le vote puis les choix.
Le formulaire est classique, il faut juste remarquer que nous pouvons passer un tableau comme paramètres.

```
<input type="text" name="ch[]" length="150">
<input type="text" name="ch[]" length="150">
<input type="text" name="ch[]" length="150">
<input type="text" name="ch[]" length="150">
<input type="text" name="ch[]" length="150">
<input type="text" name="ch[]" length="150">
<input type="text" name="ch[]" length="150">
<input type="text" name="ch[]" length="150">
```

L'utilisateur va saisir le vote puis les choix à concurrence de 8.
La page appelée "ajout_vote_go.php" va permettre de traiter les données saisies.

Pour traiter le tableau passé en paramètres, on utilise

```
$_GET['ch'][0]
$_GET['ch'][1]
…
$_GET['ch'][7]
```

Le traitement s'effectue de la même façon que pour l'utilisateur mais nous ajoutons une transaction pour s'assurer que toutes les requêtes seront effectuées ou aucune si une erreur survient pour conserver la base de données dans un état stable.

Le programme va donc envoyer à la base les requêtes suivantes

```
    pg_query('begin');
    // soit begin transaction
```

Puis

```
    pg_query($chaine_req);
    // soit par exemple
    // insert into vote( id_vote, titre,
    // description, date_creation, date_debut, date_fin)
    // values (1, 'Couleur',
    // 'Quelle est votre couleur préférée ?', '2021-04-05',
    // '2021-04-01', '2021-04-15')
```

Puis une deuxième fois, avec une nouvelle chaîne de caractères $chaine_req

```
    pg_query($chaine_req);
    // soit par exemple
    // insert into choix( id_choix, libelle, id_vote) values
    //   (1, 'Bleu', 1),
    //   (2, 'Noire', 1),
    // (3, 'Rouge', 1),
    // (4, 'Jaune', 1)
```

Puis enfin

```
    $req = pg_query('commit');
    // soit la validation de toutes les requêtes précédentes
```

Ajout d'une participation

La partie participation est un peu plus compliquée puisque nous allons devoir récupérer dans la base tous les utilisateurs et les participations existantes. Au début, il n'y aura aucune participation, donc tous les utilisateurs apparaîtront avec une case à cocher vierge pour un vote donné.
Pour interface, nous allons utiliser une liste déroulante ou figure les votes et utiliser ajax pour faire apparaître les participations des utilisateurs dès que l'on change la valeur de la liste déroulante.

Requête SQL

La première chose à faire est de définir la requête SQL qui fera le travail. Nous allons faire deux requêtes, la première consistant à récupérer les participations pour un vote donné, la deuxième consistant à récupérer les NON participations pour ce même vote donné. On pourra ensuite réunir les deux requêtes en une seule grâce à la clause SQL union.
La première requête est une jointure simple

```
select u.id_user, login, id_vote
    from utilisateur u
    inner join participation p on u.id_user = p.id_user
        where id_vote = 2
```

La deuxième requête va chercher les utilisateurs qui ne sont pas indiqués dans la première.

```
select id_user, login, 0
    from utilisateur
        where id_user not in
        (select u.id_user
            from utilisateur u
            inner join participation p on u.id_user = p.id_user
                where id_vote = 2)
```

On réunit les 2 requêtes et on classe le résultat pour faciliter la présentation.

```
-- Requête 1
union
-- Requête 2
    order by 3 desc, 2 asc
```

Liste déroulante

Il faut maintenant construire la liste déroulante à partir de la liste des votes existants, on les classera du plus récent au plus ancien.
L'exemple qui suit utilise la fonction pg_fetch_assoc() qui récupère une ligne après l'autre du résultat de la requête et la stocke dans un tableau associatif.

```
$chaine_req = 'select * from vote order by date_debut desc';
$req = pg_query($chaine_req);
// On lit la première ligne du résultat de la requête
$ligne = pg_fetch_assoc($req);
while($ligne)
     {
     // On remplit chaque option du select avec la ligne en cours
     echo '<option value="'.$ligne['id_vote'].'">';
     echo $ligne['titre'];
     echo '</option>';
     // on lit la ligne suivante du résultat de la requête
     // quand il n'y aura plus de ligne à lire, la fonction
     // retournera un booléen à false, cela permettra de sortir de
la boucle
     $ligne = pg_fetch_assoc($req);
     }
```

Grace à l'événement onchange sur le select, un appel sera fait à une fonction ajax_affect() qui va remplir le tableau qui suit avec le contenu du résultat de la requête union vue précédemment.

```
<select onchange="ajax_affect()">
```

Cette fonction récupère les affectations et les affiche dans le bloc sans que toute la page soit rechargée, un des principes d'ajax. Elle utilise une fonction (ajax() qui demande l'id d'un bloc de la page (ici "affect") et une url correspondant à la page qui va chercher les informations (ici "affectation_vote_ajax.php?id=2" par exemple).

```
function ajax_affect()
     {
     var d = document.getElementsByTagName('select')[0];
     u = 'affectation_vote_ajax.php?id='+d.value;
     b = 'affect';
     ajax(u,b);
     }
```

Il reste maintenant à tenir compte des modifications effectuées par l'utilisateur. Le plus simple est de supprimer toutes les affectations du vote concerné, de vérifier chaque case qui est cochée et de créer une affectation correspondante.

Ce travail est à faire mardi 6 avril. Le projet sera donné dans la foulée.

Un jeu d’essai est créé grâce à un script SQL

Sixième étape : page de vote

Septième étape : Installation et manuel

Test grande échelle

Établir un jeu d’essai conséquent