

Projet : Vote Cryptographique

Etape 1: Système centralisé, intègre, non anonyme

Abdelkader Gouaich

2023


Table des matières

1	Introduction	2
2	Analyse des besoins	2
2.1	Gestion des Votants	3
2.2	Gestion des Élections	4
3	Conception	6
3.1	Architecture du serveur	6
3.2	Le pattern design command	7
4	Conception détaillée	9
4.1	Objets de base	9
4.2	Couche de persistance	11
4.3	Tests	12
4.4	Développement du serveur de vote	13
5	Annexe	13
5.1	Installation de SQLite	13
5.2	installation de glib	14

1 Introduction

Notre objectif est de réaliser un premier système de vote électronique qui aura les propriétés suivantes:

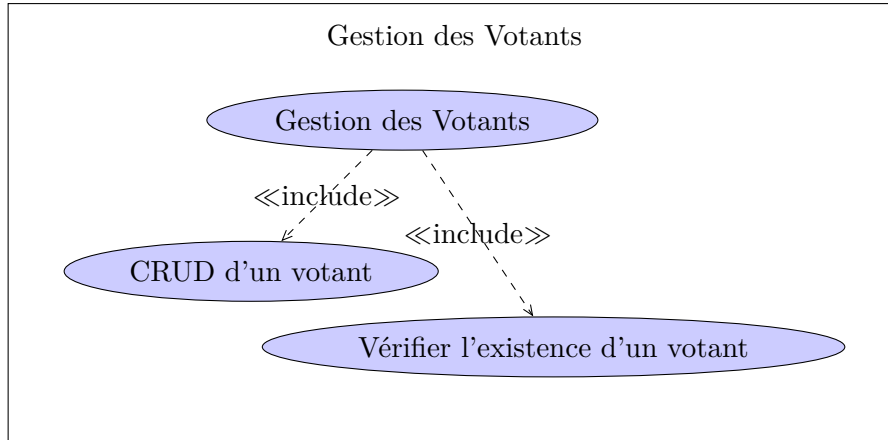
- **Intégrité** : Assurer l'impossibilité de modifier, supprimer ou dupliquer les votes de manière indétectable.
- **Exhaustivité** : Tous les votes valides doivent être correctement comptés.
- **Vérifiabilité** : Permettre aux électeurs et aux observateurs de confirmer que les votes ont été correctement comptabilisés.
- **Accessibilité** : Faciliter la participation des électeurs, quel que soit leur emplacement.
- **Éligibilité** : Seuls les électeurs légitimes doivent pouvoir participer à l'élection.
- **Validité** : Les votes invalides doivent être facilement détectables et écartés.

 **A faire** : Nous avons besoin de la librairie 'glib2' et 'sqlite3'. Reportez vous aux annexes pour installer ces librairies sur vos machines.

2 Analyse des besoins

Nous allons détailler dans ce qui suit les principales fonctionnalités de notre système de vote.

2.1 Gestion des Votants

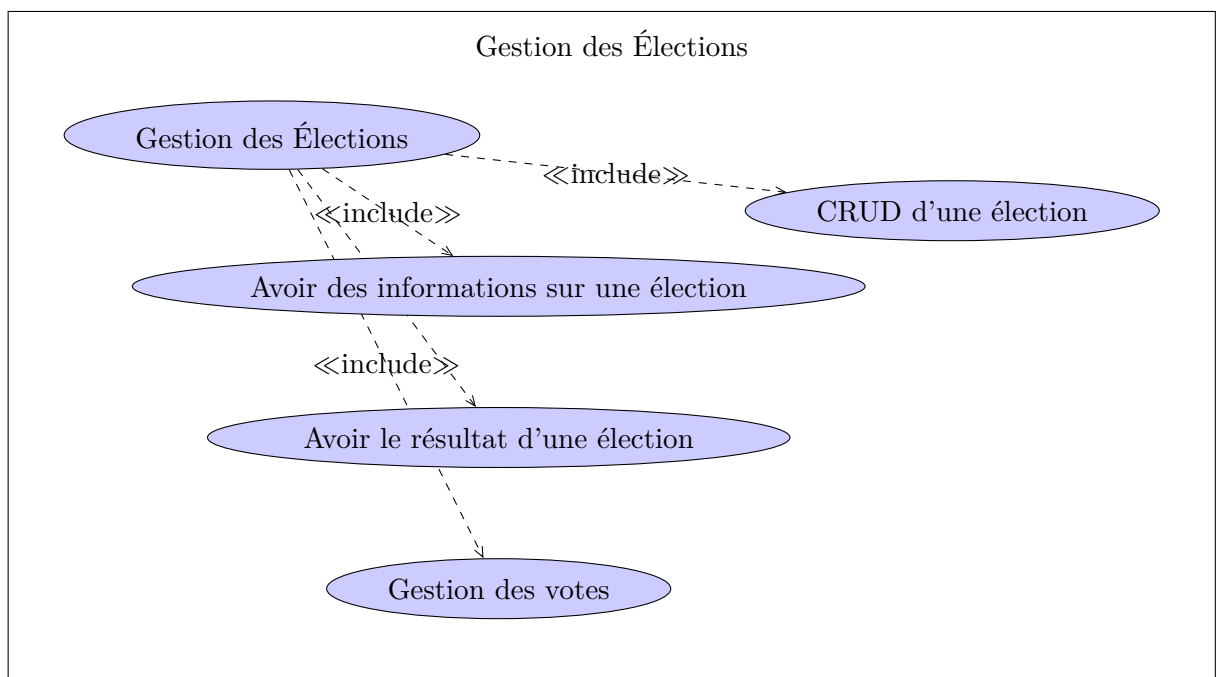


Le sous-système de gestion des votants permet de réaliser les fonctionnalités suivantes:

Fonctionnalité	Description
CRUD Électeur <i>Création d'un Électeur</i>	<ul style="list-style-type: none">- Permet aux administrateurs d'ajouter de nouveaux électeurs dans le système.- Un électeur sera identifié par son numéro d'identification.- Assure la validation des données saisies pour éviter les entrées incorrectes ou dupliquées.
<i>Lecture des informations</i>	<ul style="list-style-type: none">- Fournit une fonctionnalité de consultation des informations des votants enregistrés.- Permet la recherche et le filtrage basés sur divers critères comme le nom, l'adresse, ou le numéro d'identification.
<i>Mise à Jour d'un Électeur</i>	<ul style="list-style-type: none">- Autorise la modification des informations d'un votant existant.- Les modifications peuvent inclure la mise à jour de l'adresse, du statut d'éligibilité, ou d'autres informations personnelles.

Fonctionnalité	Description
<i>Suppression d'un Électeur</i>	<ul style="list-style-type: none"> - Permet de retirer un votant du système. - Cette fonctionnalité est généralement protégée par des mesures de sécurité pour prévenir les suppressions accidentelles ou malveillantes.
Vérification de l'Existence d'un Électeur	<ul style="list-style-type: none"> - Permet de vérifier si un individu est enregistré dans le système en tant que votant. - Utilise le numéro d'identification comme critère de recherche. - Fournit une réponse indiquant si le votant existe ou non dans la base de données.

2.2 Gestion des Élections



CRUD d'une Élection

Cette fonctionnalité permet de réaliser les opérations CRUD sur une élection comprenant:

- **Création** : Permet aux administrateurs d'ajouter une nouvelle élection

dans le système. Cela inclut la définition de ses paramètres tels que la date de début, la date de clôture et la question posée. Dans un premier temps nous allons considérer uniquement des élections à choix binaire qui représente la réponse par oui ou non à une question posée.

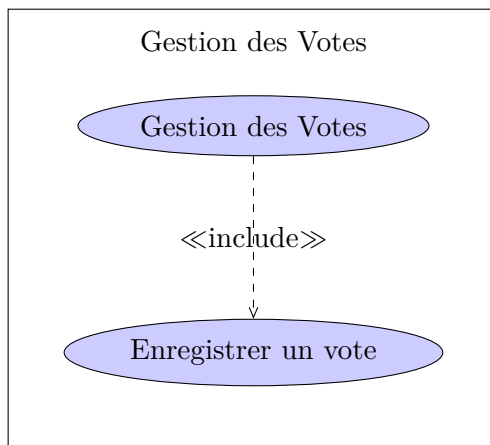
- **Lire** : Offre la possibilité de consulter les détails des élections enregistrées.
- **Mettre à jour** : Autorise la modification des détails d'une élection, comme changer les dates ou modifier le texte de la question posée.
- **Supprimer** : Permet de retirer une élection du système; cette fonctionnalité sera utile pour les élections annulées ou obsolètes.

Demander des Informations sur une Élection :

Cette fonctionnalité permet de demander et de recevoir des informations détaillées sur une élection spécifique. Elle peut inclure des informations telles que les dates de début et de clôture et la question posée.

Avoir le Résultat d'une Élection :

Après la clôture du vote, cette fonctionnalité permet de consulter les résultats de l'élection. Les résultats sont généralement présentés de manière claire et transparente, montrant le nombre de voix obtenues pour les deux options Oui et Non en réponse à la question posée.



Gérer les Votes des Électeurs :

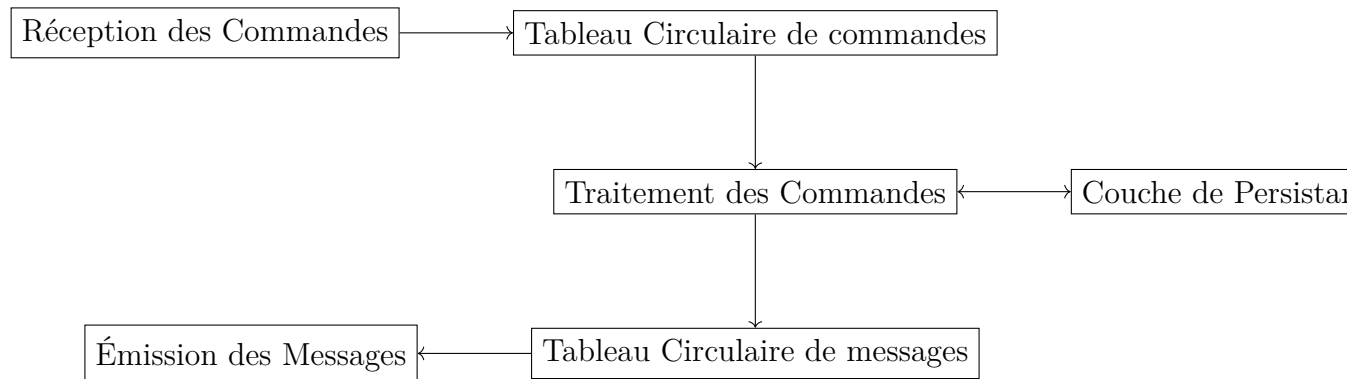
Cette fonctionnalité est au coeur du processus électoral car elle permet

d'enregistrer le vote exprimé par un électeur. Elle doit en outre assurer la validité du vote et son intégrité en évitant par exemple les votes multiples ou hors période.

Elle aussi doit garantir la confidentialité et la sécurité des votes, en s'assurant que chaque électeur ne vote qu'une seule fois et que son choix reste secret.

3 Conception

3.1 Architecture du serveur



Nous proposons d'organiser notre serveur autour des activités suivantes :

- Une activité de réception des commandes, qui seront placées dans un tableau circulaire dédié aux commandes.
- Une ou plusieurs activités de traitement des commandes.
- Une activité d'émission des messages vers l'extérieur.

L'activité de traitement de commandes aura accès à une couche de persistance pour sauvegrader les informations.

Cette architecture va permettre de définir plusieurs activités de traitement (workers) par des threads indépendants. Nous devons pour cela assurer la synchronisation entre toutes ces activité en utilisant les mécanismes vus dans les précédentes séances.

3.2 Le pattern design command

Nous proposons d'utiliser le design pattern "command" pour représenter les principales fonctions de notre système.

Ce pattern design repose sur les principes suivants:

- Une fonction du système est représentée avec une structures qui encapsule les paramètres de l'appel de la fonction.
- L'émetteur de la commande aura la charge de créer la structure et de renseigner les paramètres
- Le receveur de la commande aura en charge l'exécution de la commande et éventuellement l'émission d'un message pour renseigner le résultat
- La mise en correspondance entre l'émetteur et le receveur se fait par un composant logiciel intermédiaire (middleware) qui va orchestrer la réception des commandes et leurs exécutions par les receveurs.

☞ A voir : `common/include/messages.h`.

Voici par exemple les commandes représentant les cas d'utilisation pour la gestion des électeurs. Chaque fonction est représentée par une structure C qui encapsule les paramètres.

```
#ifndef MESSAGE_H
#define MESSAGE_H

#include "protocol.h"

typedef struct
{
    char identifiant[ENTITY_ID_SIZE];
} AjoutElecteurCmd;

typedef struct
{
    char identifiant[ENTITY_ID_SIZE];
} SupprimeElecteurCmd;

typedef struct
{
```

```

    char identifiant[ENTITY_ID_SIZE];
} EstPresentCmd;

typedef enum
{
    NOP = 0,
    AJOUT_ELECTEUR,
    SUPPRIME_ELECTEUR,
    EST_PRESENT
} CommandType;

/--
typedef struct
{
    CommandType type;
    char signature[256]; // la signature de la commande
    union
    {
        AjoutElecteurCmd ajoutElecteur;
        SupprimeElecteurCmd supprimeElecteur;
        EstPresentCmd estPresent;
    } commande;
} Commande;
#endif

```

En C, il est d'usage de regrouper toutes les commandes avec un type `union` et une énumération permettant de savoir que quelle commande il s'agit. Une fonction de traitement typique va réaliser les instructions suivantes en pseudo code:

...

```

Commande cmd;
switch (cmd.type) {

    case AJOUT_ELECTEUR: {
        AjoutElecteurCmd* p_cmd = &cmd.ajoutElecteur;
        traitementAjoutCommande(p_cmd);
        break;
    }
}

```



```

    }

    // ... autres cas possibles ...

}

```

✎ **A faire** : complétez le fichier ‘messages.h’ pour inclure les autres cas d’utilisation

✎ **A faire** : préparez le fichier ‘serveur_vote.c’ avec la mise en place l’architecture de traitement proposée sans réaliser les fonctions de traitement spécifiques à chaque commande. Les fonctions de traitement spécifiques pourraient simplement afficher les paramètres des commandes reçues.

4 Conception détaillée

4.1 Objets de base

Electeur
- numeroID : char[256]

Election
- identifiant : char[256] - question : char[256] - dateDebut : Date - dateFin : Date - status : Enum canceled, active, closed

Vote
- idVotant : char[256] - idElection : char[256] - timestamp : DateTime - bulletin : char[256] - hashValidation : char[256]

Notre système repose sur les objets décrits par les diagrammes UML ci-dessus. Il forment la base d'un système de vote électronique, où les électeurs (**Electeur**) participent à des élections (**Election**) en soumettant leurs votes (**Vote**).

4.1.1 Electeur

L'objet **Electeur** représente une entité participant au processus de vote. Chaque électeur est caractérisé par :

- **numeroID** : Un identifiant unique pour chaque électeur, représenté sous forme d'une chaîne de caractères de 256 caractères maximum. Cet identifiant peut être utilisé pour authentifier l'électeur dans le système de vote.

4.1.2 Election

L'objet **Election** représente un événement de vote spécifique. Chaque élection est définie par :

- **identifiant** : Un identifiant unique pour l'élection, sous forme d'une chaîne de 256 caractères. Cet identifiant sert à distinguer chaque élection des autres.
- **question** : La question ou la proposition soumise au vote, limitée à 256 caractères. Cela définit le sujet ou la décision sur laquelle les électeurs doivent voter. Nous allons pour l'instant considérer que des questions à choix binaire (oui/non)
- **dateDebut** : La date et l'heure de début de l'élection. Cela indique quand les électeurs peuvent commencer à voter.
- **dateFin** : La date et l'heure de fin de l'élection. Cela indique quand le vote se termine et aucune autre soumission de vote n'est acceptée.
- **status** : L'état actuel de l'élection, qui peut être **canceled** (annulée), **active** (active) ou **closed** (clôturée). Cela indique si l'élection est en cours, terminée ou annulée.

4.1.3 Vote

L'objet **Vote** représente l'action de voter d'un électeur dans une élection spécifique. Chaque vote est caractérisé par :

- **idVotant** : L'identifiant de l'électeur qui a voté, correspondant à son `numeroID` dans l'objet `Electeur`.
- **idElection** : L'identifiant de l'élection dans laquelle le vote a été effectué, correspondant à l'`identifiant` dans l'objet `Election`.
- **timestamp** : La date et l'heure exactes à laquelle le vote a été soumis. Cela permet de suivre quand chaque vote a été effectué.
- **bulletin** : Le choix ou la réponse donnée par l'électeur, représentée sous forme d'une chaîne de 256 caractères. Cela peut être, par exemple, un choix parmi plusieurs options ou un oui/non. Pour notre système le choix sera écrit en claire, mais nous prévoyons que le choix soit crypté pour garantir l'anonymat du vote.
- **hashValidation** : Un hash ou une signature numérique utilisée pour valider l'intégrité et l'authenticité du vote. Cela assure que le vote n'a pas été altéré après sa soumission.

☞ **A voir** : L'implémentation des objets métier est proposée dans le fichier ``protocol.h``.

4.2 Couche de persistance

Nous allons utiliser une base de données embarquée `sqlite` comme couche de persistance de notre application.

Voici une description des tables `Electeur`, `Election`, et `Vote` basées sur nos objets métier.

4.2.1 Table `Electeur`

- **id** (INTEGER, PRIMARY KEY) : Un identifiant unique pour chaque électeur.
- **numeroID** (BLOB) : Un identifiant sous forme de BLOB, probablement utilisé pour stocker des informations cryptées ou binaires.

4.2.2 Table `Election`

- **id** (INTEGER, PRIMARY KEY) : Un identifiant unique pour chaque élection.

- **identifiant** (BLOB) : Un identifiant sous forme de BLOB pour l'élection.
- **question** (TEXT, CHECK(length(question) <= 256)) : La question ou le sujet de l'élection, avec une longueur maximale de 256 caractères.
- **dateDebut** (TEXT) : La date de début de l'élection.
- **dateFin** (TEXT) : La date de fin de l'élection.
- **status** (TEXT, CHECK(status IN('canceled', 'active', 'closed')))) : L'état de l'élection, pouvant être 'canceled', 'active', ou 'closed'.

4.2.3 Table Vote

- **id** (INTEGER, PRIMARY KEY) : Un identifiant unique pour chaque vote.
- **idVotant** (INTEGER) : L'identifiant de l'électeur qui a voté, faisant référence à la table **Electeur**.
- **idElection** (INTEGER) : L'identifiant de l'élection dans laquelle le vote a été effectué, faisant référence à la table **Election**.
- **timestamp** (TEXT) : La date et l'heure du vote.
- **ballot** (BLOB) : Le bulletin de vote sous forme de BLOB.
- **hashValidation** (TEXT, CHECK(length(hashValidation) <= 256)) : Un hash pour la validation du vote, avec une longueur maximale de 256 caractères.

4.2.4 Relations

- La table **Vote** a une relation avec la table **Electeur** via **idVotant**.
- La table **Vote** a également une relation avec la table **Election** via **idElection**.

🔗 **A faire** : Le gestionnaire d'accès à la couche de persistance est déjà réalisé. Vous pouvez consulter les fichiers 'bd.h' et 'bd.c' pour comprendre son fonctionnement.

4.3 Tests

Le répertoire **test** contient les tests unitaires de notre applications:

- **test_electeur.c** présente un test du cas d'utilisation des électeurs
- **test_election.c** présente un test du cas d'utilisation des élections

- `test_vote.c` présente un test du cas d'utilisation de gestion du vote

✎ **A faire :** Consultez le code de ces fichiers

✎ **A faire :** Compilez (avec 'make') et exécutez les programmes de test

4.4 Développement du serveur de vote

✎ **A faire :** Complétez le fichier 'serveur_vote.c' pour réaliser le code des fonctions de traitement des commandes

✎ **A faire :** Réalisez un programme de test complet de votre serveur de vote (création d'électeurs, élections et votes). Vous devez soumettre des commandes et voir le résultat de vos commandes une fois traitées par le serveur.

5 Annexe

5.1 Installation de SQLite

Voici les étapes à suivre pour installer SQLite dans une distribution Ubuntu:

- Exécutez la commande suivante :

```
sudo apt update
```

- Pour l'installer sqlite3, exécutez :

```
sudo apt install sqlite3
```

- Pour développer des applications utilisant SQLite en C vous devez installer les bibliothèques de développement SQLite:

```
sudo apt install libsqlite3-dev
```

- Après l'installation, vous pouvez vérifier la version de SQLite installée et confirmer qu'elle fonctionne correctement:

```
sqlite3 --version
```

Vous êtes maintenant prêt à développer des applications utilisant SQLite. Vous pouvez inclure SQLite dans vos programmes C en utilisant `#include <sqlite3.h>` et lier la bibliothèque lors de la compilation.

Lorsque vous compilez un programme C qui utilise SQLite, assurez-vous de lier la bibliothèque SQLite. Par exemple :

```
gcc mon_programme.c -lsqlite3 -o mon_programme
```

5.2 installation de glib

Glib est une bibliothèque qui offre des structures de données comme des listes, arbres et tables de hachage.

Vous devez installer les bibliothèques de développement GLib qui incluent les en-têtes et les fichiers nécessaires pour compiler des programmes utilisant GLib.

Voici les étapes à suivre :

1. Ouvrir un Terminal :

- Vous pouvez ouvrir un terminal en appuyant sur **Ctrl + Alt + T** ou en cherchant “Terminal” dans les applications de votre système.

2. Mettre à Jour la Liste des Paquets :

- Il est recommandé de mettre à jour la liste des paquets de votre système avant d’installer de nouveaux paquets. Exécutez la commande suivante :

```
sudo apt update
```

3. Installer les Bibliothèques de Développement GLib :

- Les bibliothèques de développement GLib contiennent les fichiers d’en-tête et les bibliothèques statiques/dynamiques nécessaires pour le développement. Installez-les en exécutant :

```
sudo apt install libglib2.0-dev
```

4. Vérifier l’Installation :

- Après l’installation, vous pouvez vérifier si les bibliothèques GLib sont correctement installées en utilisant **pkg-config**, un outil qui aide à compiler des applications et des bibliothèques. Exécutez :

```
pkg-config --modversion glib-2.0
```

- Cette commande devrait retourner la version de GLib installée.

5. Compilation de Programmes Utilisant GLib :

- Lorsque vous compilez un programme qui utilise GLib, vous devez inclure les drapeaux de compilation appropriés. `pkg-config` peut vous aider à obtenir ces drapeaux. Par exemple, pour compiler un programme `mon_programme.c` qui utilise GLib, utilisez :

```
gcc `pkg-config --cflags --libs glib-2.0` mon_programme.c -o mon_programme
```