Document d'Architecture d'Entreprise : Modernisation du Processus Électoral Étudiant (SmartVote)

Version	Date	Auteur(s)	Commentaires
0.1	27.12.2023	Elhadji Abdou BOYE	Version initiale

I. Introduction

I.1. Objectifs du document

Ce document vise à définir l'architecture d'entreprise pour la modernisation du processus électoral du Bureau des Étudiants à l'Université XYZ. Il servira de guide pratique pour l'équipe de développement, détaillant les architectures logique, métier, applicative, données, technique, solutions et intégration, ainsi que les bonnes pratiques associées.

I.2. Diagramme de contexte

- Description :
  - SmartVote : Le système SmartVote est représenté au centre du diagramme. C'est le système que nous sommes en train de concevoir.
  - Acteurs Principaux :
    - Administrateur : Un utilisateur avec des privilèges d'administration pour gérer les élections, les candidats, et consulter les résultats.
    - Candidat : Un étudiant se portant candidat à une élection.
    - Électeur : Un étudiant éligible pour voter dans une élection.
  - Interactions :

- Gérer les Élections : L'Administrateur peut créer, modifier et supprimer des élections.
- Gérer les Candidats : L'Administrateur peut ajouter, modifier et supprimer des candidats pour une élection.
- Consulter les Résultats : L'Administrateur peut consulter les résultats des élections et générer des rapports statistiques.
- S'Inscrire comme Candidat : Le Candidat peut s'inscrire en tant que candidat en fournissant des informations personnelles.
- Consulter les Élections : Le Candidat et l'Électeur peuvent consulter la liste des élections et voir les détails d'une élection.
- Voter dans une Élection : L'Électeur peut voter en sélectionnant un candidat lors d'une élection.

- Flux d'Informations :
  - Des flèches indiquent les flux d'informations entre le système et les acteurs. Par exemple, l'Administrateur peut fournir des informations pour créer ou modifier une élection.

Ce diagramme de contexte donne une vue d'ensemble des interactions entre le système SmartVote et ses principaux acteurs dans son environnement externe.

I.3. SmartVote en chiffres

- Modules du Logiciel :
  - Gestion des Élections :
    - Description : Ce module gère la création, la modification et la suppression des élections. Il inclut également des fonctionnalités pour définir les dates de début et de fin, le type d'élection, etc.

- Estimation des Données : Chaque élection aura des métadonnées telles que le titre, les dates, le type, et pourra être associée à plusieurs candidats et électeurs.

- Gestion des Candidats :
  - Description : Ce module permet d'ajouter, de modifier et de supprimer des informations sur les candidats participant à une élection. Il gère également les programmes d'études des candidats.
  - Estimation des Données : Chaque candidat aura des données personnelles (nom, prénom, date de naissance) et des informations spécifiques à la candidature (programme d'études).
- Gestion des Électeurs :
  - Description : Ce module prend en charge l'inscription des électeurs, la vérification de leur éligibilité et la gestion de leurs informations.
  - Estimation des Données : Les électeurs auront des données d'identification et des informations sur leur statut de vote.
- Votes et Résultats :
  - Description : Ce module traite les votes émis par les électeurs, les associe aux candidats correspondants et génère les résultats des élections.
  - Estimation des Données : Les votes seront enregistrés avec des horodatages, et les résultats contiendront le nombre de votes, le pourcentage, etc.

- Estimation des Données :
  - Utilisateurs :
  - Administrateurs : 5
  - Candidats (par élection) : 10
  - Électeurs (par élection) : 500
  - Taille de la Base de Données :

    Base de données principale : 500 Mo (estimation initiale, susceptible de croître avec le temps)

- Volume de Données :

Votes par élection : 5000 (estimation basée sur 500 électeurs par élection)

- Trafic Applicatif :

Taux de demande simultanée : 100 requêtes par seconde.

Remarques :

Ces estimations sont des valeurs initiales et peuvent être ajustées en fonction de l'évolution du système et du nombre d'utilisateurs.

La gestion des performances doit être prise en compte pour garantir une expérience utilisateur fluide, en particulier lors des périodes d'élections où le trafic peut augmenter de manière significative.

II. Architecture logique du SmartVote

La plateforme SmartVote est conçue pour gérer les demandes en orchestrant une logique métier complexe, en accédant à des bases de données spécialisées et en retournant des réponses adaptées aux divers besoins des utilisateurs. Cette architecture logique garantit une flexibilité et une adaptabilité maximales aux différentes interfaces et aux évolutions futures.

- Composants de la Plateforme SmartVote :
  - Composants de Présentation :
    - Description : Ces composants sont responsables de la gestion de l'interface utilisateur et de la consommation des services distants. Ils assurent une expérience utilisateur fluide et intuitive.
    - Technologies : HTML, CSS, JavaScript, React (pour les applications web), React Native (pour les applications mobiles).
  - Logique de Domaine ou Métier :

- Description : Il s'agit du cœur de la plateforme SmartVote, où la logique métier est implémenté. Cette logique gère les processus liés aux élections, y compris la gestion des candidatures, le processus de vote, le dépouillement des votes, et la déclaration des résultats.
- Technologies : Java (ou un langage équivalent), Spring Framework.

- Logique d'Accès aux Bases de Données :
  - Description : Ces composants sont responsables de l'accès aux bases de données où sont stockées les informations cruciales telles que les données des candidats, des électeurs, les votes, et les résultats électoraux.
  - Technologies : Hibernate (pour la persistance des données), PostgreSQL (ou un autre système de gestion de base de données relationnelle).
- Logique d'Intégration :
  - Description : Cette composante englobe un canal de messagerie basé sur des répartiteurs de messages. Elle assure la communication asynchrone entre les différents modules de la plateforme SmartVote, favorisant ainsi la résilience des services en cas de défaillance partielle.
  - Technologies : Message Brokers (comme Apache Kafka), RESTful APIs.

II.1. Principes d'architecture retenues

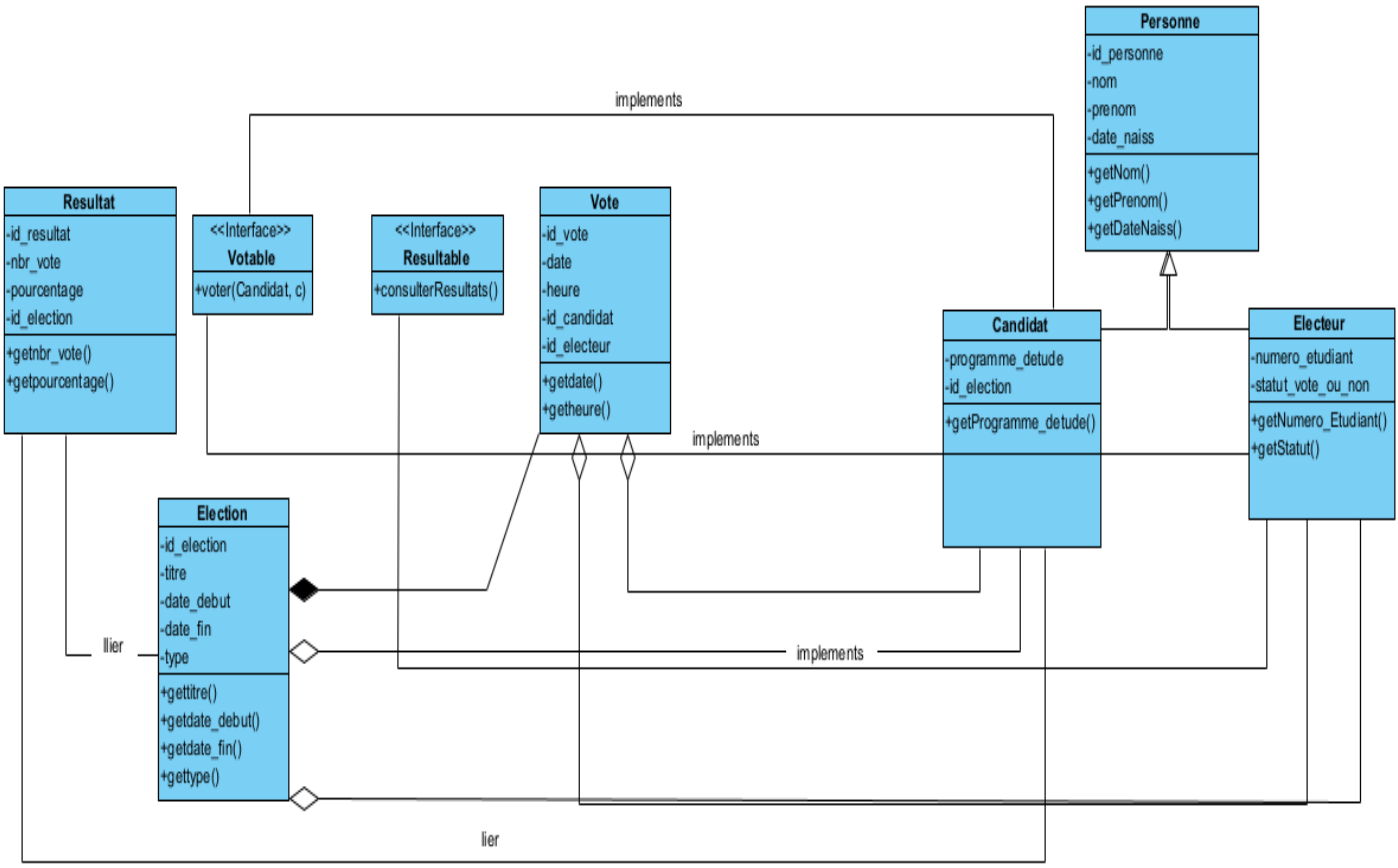
- Architecture Orientée Services (SOA) :
  - Justification : L'utilisation d'une architecture orientée services permet une modularité accrue, facilitant l'ajout de nouvelles fonctionnalités et la maintenance des services existants de manière indépendante.
- Architecture Robuste Pilotée par les Évènements :

- Justification : En adoptant une architecture robuste basée sur les événements, SmartVote peut maintenir sa fonctionnalité même en cas de défaillance partielle, assurant une expérience utilisateur continue.

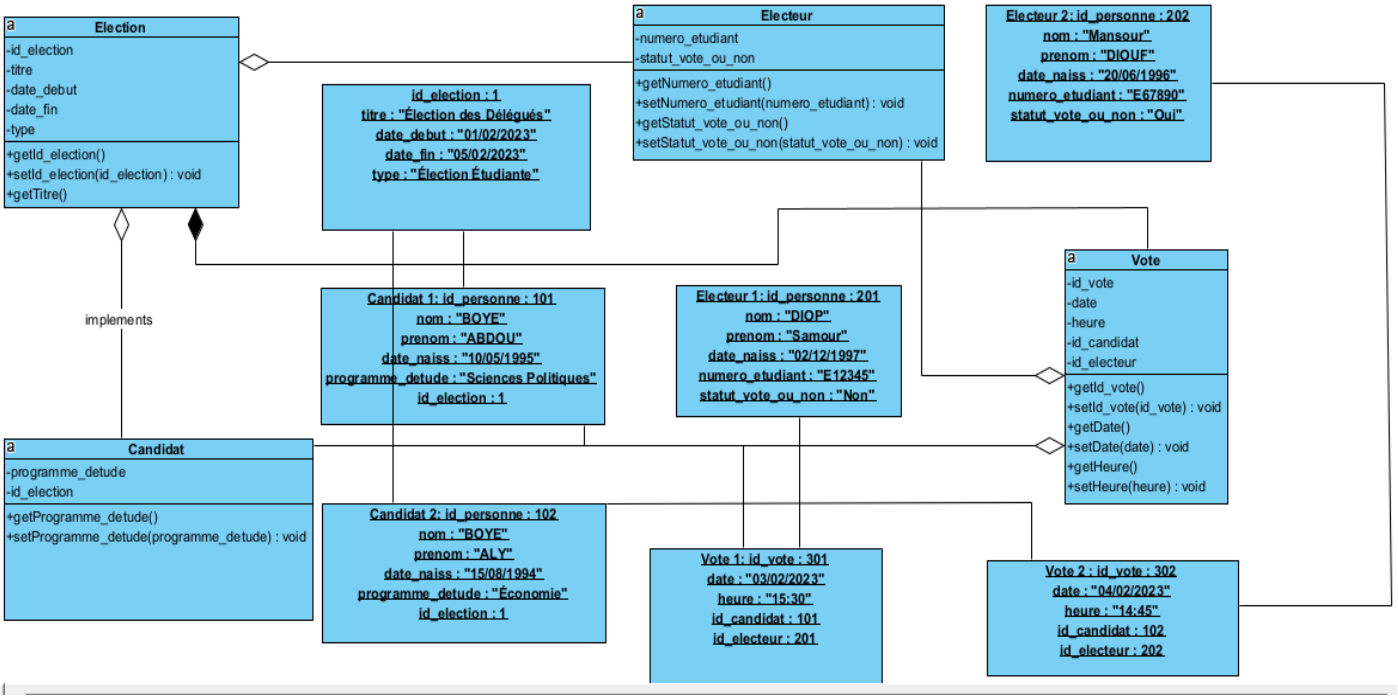
- Gestion des Domaines Opérationnels par Équipes Dédiées :
  - Justification : Cette approche favorise une responsabilité claire et une expertise spécialisée dans chaque domaine opérationnel, optimisant ainsi le développement et la maintenance.
- Facilité de Maintenance dans le Temps :
  - Justification : L'architecture est conçue pour être facilement compréhensible et modifiable, garantissant une maintenance aisée au fil du temps.
- Évolution Continue de l'Application :
  - Justification : SmartVote est prévu pour évoluer avec le temps pour s'adapter aux changements dans les règles métier et les exigences de l'université.
- Intégration Continue et Déploiement Continu :
  - Justification : Ces pratiques permettent des mises à jour rapides et fréquentes, assurant la disponibilité des dernières fonctionnalités et corrections de bugs.
- Adoption Progressive des Nouvelles Technologies :
  - Justification : SmartVote tirera parti des nouvelles technologies au fur et à mesure de son évolution, évitant ainsi des migrations complètes coûteuses.

II.2. Objets métiers du SmartVote

- **Diagramme de classe**

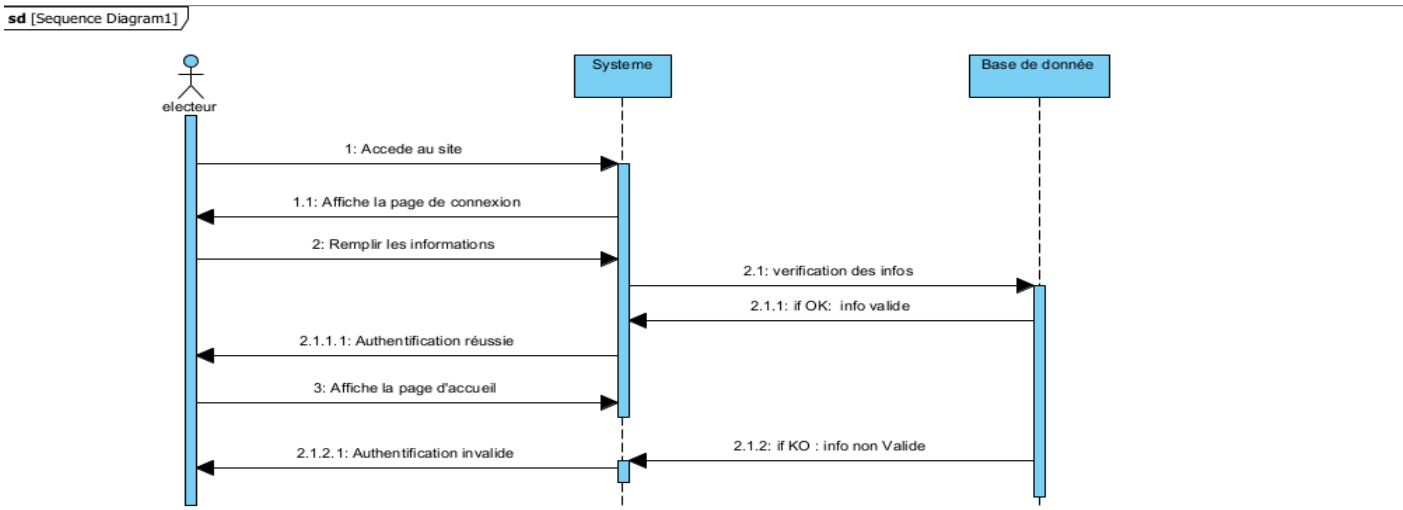


➢ Diagramme Objets

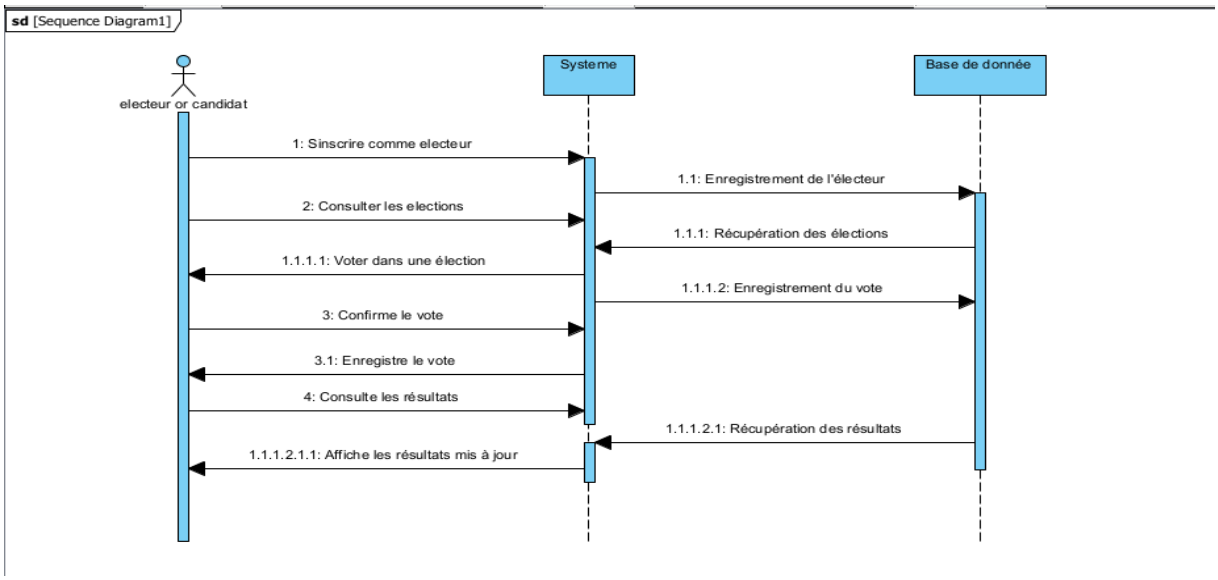


➢ Diagramme de Séquence

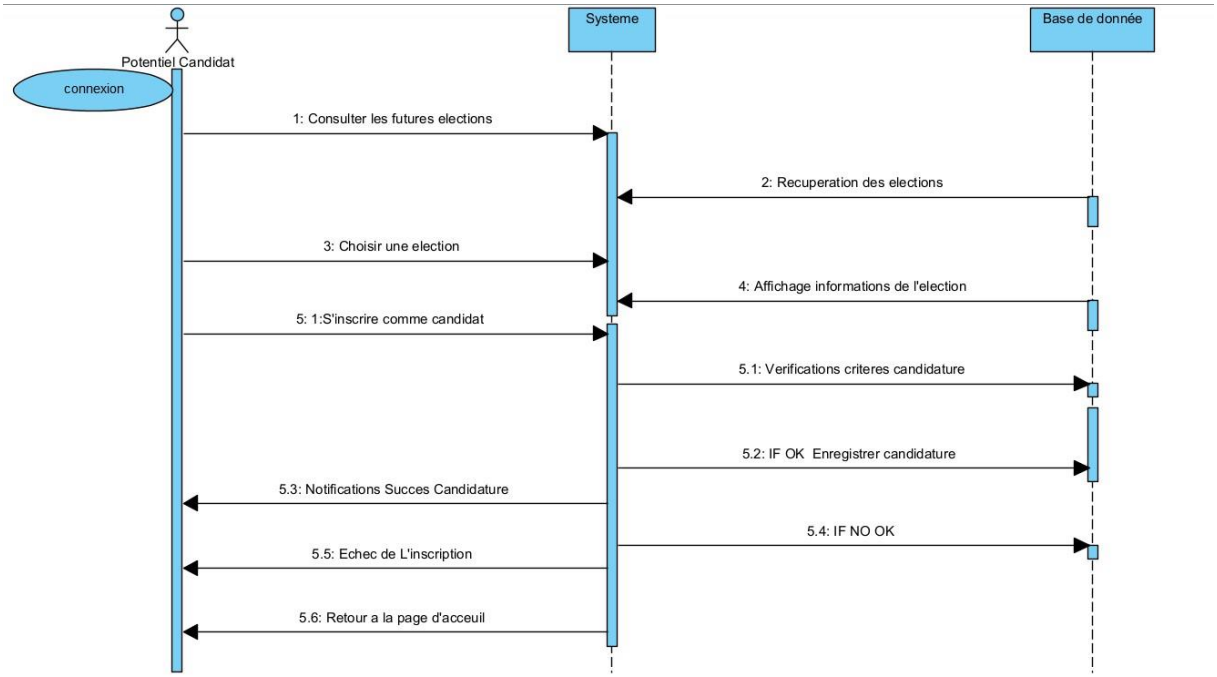
○ Connexion



- **Voter**



- **Candidater**



- Contrats d'Operations
  - **Premier Cas**
    - Nom : Voter à une élection (id_electeur)
    - Responsabilité : Enregistrer le vote d'un électeur
    - Préconditions :

L'élection existe et n'as pas encore été clôturée
L'électeur remplit les critères établis pour être électeur
L'électeur n'as pas encore voter pour cette élection

- Post-Conditions :

Un vote a été enregistré pour une élection donnée.
L'électeur en question ne peut plus voter pour cette élection.
Notifications succès du vote

- **Deuxième Cas**
  - Nom : Candidater à une élection (id_candidat)
  - Responsabilité : Enregistrer un candidat dans une élection
  - Préconditions :

L'élection existe et n'as pas encore débutée

La date de dépôt des candidatures n'est pas dépassée

Le candidat remplit les critères d'éligibilités pour être candidat

- Post-Conditions :

Un candidat a été enregistré pour l'élection donnée.

La liste de candidat est mise à jour

Notifications succès de la candidature

II.3. Domaines métiers du SmartVote

- Description Textuelle du Modèle du Domaine :
  - Élection :
    - Attributs :

      Titre

      Date de début

      Date de fin

      Type (par exemple, élection présidentielle, élection du conseil étudiant)

    - Associations :

      Candidat (Un Candidat peut participer à plusieurs Élections)

      Électeur (Un Électeur peut participer à plusieurs Élections)

  - Candidat :
    - Attributs :

      Programme d'études

      Nom

      Prenom

      Date de naissance

CIN

- Associations :

Élection (Un Candidat peut participer à plusieurs Élections)

- Électeur :
  - Attributs :

    Numéro d'étudiant

    Nom

    Prenom

    Date de naissance

    Cin

  - Associations :

    Élection (Un Électeur peut participer à plusieurs Élections)

    Vote (Un Électeur peut émettre plusieurs Votes)

- Vote :
  - Attributs :

    Date

    Heure

  - Associations :

    Candidat (Un Vote est associé à un Candidat)

    Électeur (Un Vote est émis par un Électeur)

    Élection (Un Vote est associé à une Élection)

- Résultat :
  - Attributs :

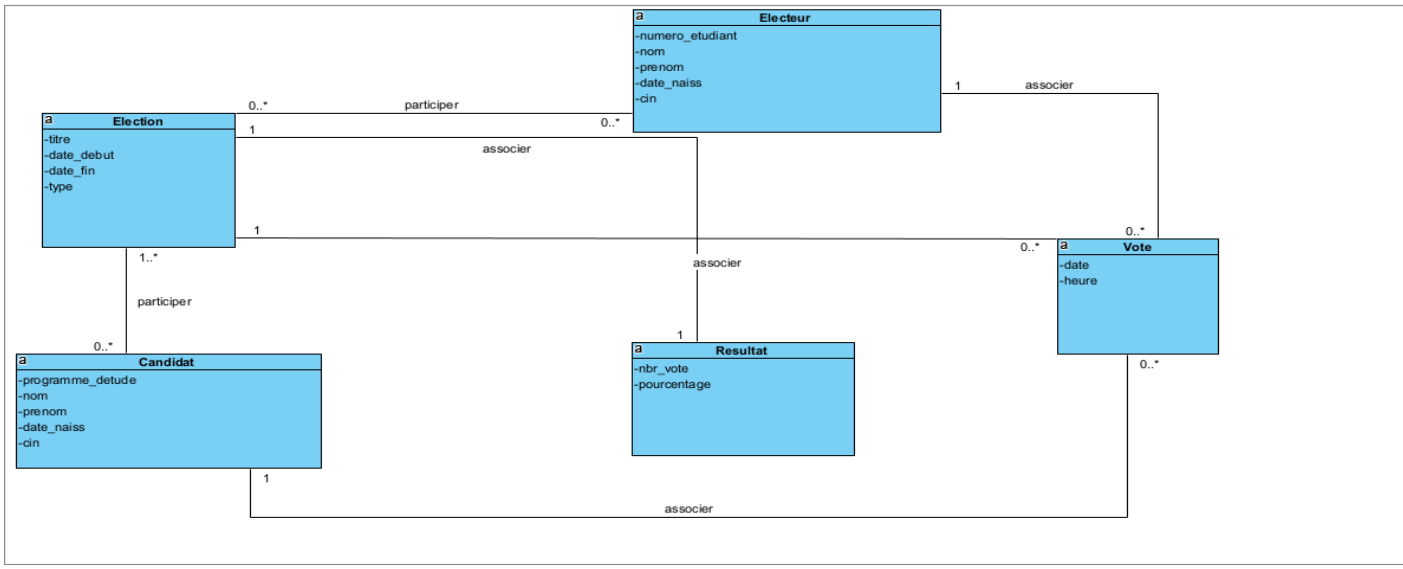
Nombre de votes

Pourcentage

- Associations :

  Élection (Un Résultat est associé à une Élection)

➢ Diagramme du Modèle du Domaine :



Ce diagramme illustre les entités principales (Élection, Candidat, Électeur, Vote, Résultat) et leurs relations dans le domaine métier de SmartVote. Les associations entre les entités reflètent les relations logiques entre elles.

III. Services du SmartVote

Le modèle de services détaille chaque service, ses responsabilités, et les relations entre eux.

➢ Pseudocode

```
public class Electeur extends Personne {

    private int numero_etudiant;
    private int statut_vote_ou_non;

    public void getNumero_etudiant() {
```

```
          // TODO - implement Electeur.getNumero_etudiant
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param numero_etudiant
      */
     public void setNumero_etudiant(int numero_etudiant) {
          this.numero_etudiant = numero_etudiant;
     }

     public void getStatut_vote_ou_non() {
          // TODO - implement Electeur.getStatut_vote_ou_non
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param statut_vote_ou_non
      */
     public void setStatut_vote_ou_non(int statut_vote_ou_non) {
          this.statut_vote_ou_non = statut_vote_ou_non;
     }

} public class Election {


     private int id_election;
     private int titre;
     private int date_debut;
     private int date_fin;
     private int type;

     public void getId_election() {
          // TODO - implement Election.getId_election
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param id_election
      */
     public void setId_election(int id_election) {
          this.id_election = id_election;
     }

     public void getTitre() {
          // TODO - implement Election.getTitre
```

```
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param titre
     */
    public void setTitre(int titre) {
        this.titre = titre;
    }

    public void getDate_debut() {
        // TODO - implement Election.getDate_debut
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param date_debut
     */
    public void setDate_debut(int date_debut) {
        this.date_debut = date_debut;
    }

    public void getDate_fin() {
        // TODO - implement Election.getDate_fin
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param date_fin
     */
    public void setDate_fin(int date_fin) {
        this.date_fin = date_fin;
    }

    public void getType() {
        // TODO - implement Election.getType
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param type
     */
    public void setType(int type) {
        this.type = type;
    }
```

```
}
public class Personne {

    private int id_personne;
    private int nom;
    private int prenom;
    private int date_naiss;

    public void getId_personne() {
        // TODO - implement Personne.getId_personne
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param id_personne
     */
    public void setId_personne(int id_personne) {
        this.id_personne = id_personne;
    }

    public void getNom() {
        // TODO - implement Personne.getNom
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param nom
     */
    public void setNom(int nom) {
        this.nom = nom;
    }

    public void getPrenom() {
        // TODO - implement Personne.getPrenom
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param prenom
     */
    public void setPrenom(int prenom) {
        this.prenom = prenom;
    }

    public void getDate_naiss() {
```

```
          // TODO - implement Personne.getDate_naiss
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param date_naiss
      */
     public void setDate_naiss(int date_naiss) {
          this.date_naiss = date_naiss;
     }

}
public interface Resultable {

     void consulterResultat();

}
public class Resultat {

     private int id_resultat;
     private int nbr_vote;
     private int pourcentage;
     private int id_election;

     public void getId_resultat() {
          // TODO - implement Resultat.getId_resultat
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param id_resultat
      */
     public void setId_resultat(int id_resultat) {
          this.id_resultat = id_resultat;
     }

     public void getNbr_vote() {
          // TODO - implement Resultat.getNbr_vote
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param nbr_vote
      */
     public void setNbr_vote(int nbr_vote) {
          this.nbr_vote = nbr_vote;
```

```
    }

    public void getPourcentage() {
        // TODO - implement Resultat.getPourcentage
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param pourcentage
     */
    public void setPourcentage(int pourcentage) {
        this.pourcentage = pourcentage;
    }

}
public interface Votable {

    /**
     *
     * @param Candidat
     * @param c
     */
    void voter(int Candidat, int c);

}
public class Vote {

    private int id_vote;
    private int date;
    private int heure;
    private int id_candidat;
    private int id_electeur;

    public void getId_vote() {
        // TODO - implement Vote.getId_vote
        throw new UnsupportedOperationException();
    }

    /**
     *
     * @param id_vote
     */
    public void setId_vote(int id_vote) {
        this.id_vote = id_vote;
    }

    public void getDate() {
        // TODO - implement Vote.getDate
```

```
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param date
      */
     public void setDate(int date) {
          this.date = date;
     }

     public void getHeure() {
          // TODO - implement Vote.getHeure
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param heure
      */
     public void setHeure(int heure) {
          this.heure = heure;
     }

}
public class Candidat extends Personne {

     private int programme_detude;
     private int id_election;

     public void getProgramme_detude() {
          // TODO - implement Candidat.getProgramme_detude
          throw new UnsupportedOperationException();
     }

     /**
      *
      * @param programme_detude
      */
     public void setProgramme_detude(int programme_detude) {
          this.programme_detude = programme_detude;
     }

}
```

IV. Architecture fonctionnelle

Les cas d'utilisation incluent l'enregistrement des candidatures, le vote, le dépouillement, la déclaration des résultats, etc. Les contrats d'opération détailleront ces processus.

V. Architecture de solution

Type de service	Technologie/Brique/Composant	Solutions	Liens
Métiers	Microservices (Spring Boot, Java)	Modularité, Indépendance, Maintenance, Résilience	Spring | Microservices

Utilitaires	Base de Données NoSQL (MongoDB)	Stockage flexible des données, Évolutivité	MongoDB: La Plateforme De Données D'application | MongoDB
Processus	File d'Attente (Apache Kafka)	Gestion asynchrone des tâches, Résilience	Apache Kafka
Informations	Base de Données Relationnelle (PostgreSQL)	Intégrité des données, Requêtes complexes	PostgreSQL: The world's most advanced open source database
Accès	API RESTful (Spring Framework)	Communication flexible, Intégration avec d'autres services	Getting Started | Building a RESTful Web Service (spring.io)
Présentations	Interface Utilisateur Web (React)	Expérience utilisateur moderne, Interactivité	React

VI. Architecture de données

VI.1. Catalogue des données métiers

Numero	Nom	Type	Nature	Description

1	Id_personne	Integer	Clé primaire	Identifiant unique de la personne dans la base de données.
2	Nom	Varchar (50)	Attribut	Nom de la personne.
3	Prenom	Varchar (50)	Attribut	Prénom de la personne.
4	Date_naiss	Date	Attribut	Date de naissance de la personne.
5	Numero_etudiant	Integer	Attribut	Numéro d'étudiant pour les étudiants (Électeur et Candidat).
6	Statut_vote_ou_non	Boolean	Attribut	Statut de vote de l'électeur (Vrai si a voté, Faux sinon).
7	Id_election	Integer	Clé étrangère	Identifiant unique de l'élection liée.
8	Programme_detude	Varchar (100)	Attribut	Programme d'études du Candidat.
9	Id_vote	Integer	Clé primaire	Identifiant unique d'un vote dans la

				base de données.
10	Date	Date	Attribut	Date du vote.
11	Heure	Time	Attribut	Heure du vote.
12	Id_candidat	Integer	Clé étrangère	Identifiant unique du candidat lié.
13	Id_electeur	Integer	Clé étrangère	Identifiant unique de l'électeur lié.
14	Id_resultat	Integer	Clé primaire	Identifiant unique du résultat dans la base de données.
15	Nbr_vote	Integer	Attribut	Nombre total de votes pour un candidat.
16	Pourcentage	Float	Attribut	Pourcentage de votes pour un candidat.

- Tableau du Dictionnaire des Données - Données Maîtres

Numéro	Nom	Type	Nature	Description
1	Id_personne	Int	Clé Primaire	Identifiant unique de la personne.
2	Nom	Varchar (50)	Attribut	Nom de la personne.

3	Prenom	Varchar (50)	Attribut	Prénom de la personne.
4	Date_naiss	Date	Attribut	Date de naissance de la personne.
5	Id_election	Int	Clé Étrangère	Identifiant de l'élection associée au candidat.
6	Programme_detude	Varchar (50)	Attribut	Programme d'études du candidat.
7	Numero_etudiant	Int	Attribut	Numéro d'étudiant de l'électeur.
8	Statut_vote_ou_non	Int	Attribut	Statut de vote de l'électeur (0 ou 1).
9	Id_electeur	Int	Clé Étrangère	Identifiant unique de l'électeur lié.
10	Date	Date	Attribut	Date du vote.

- Précision des Données Maîtres :

- Définition : Les données maîtres comprennent les informations fondamentales sur les personnes, les élections, les candidats et les votes.
- Validation Rigoureuse : Les données personnelles comme le nom, prénom et date de naissance doivent être validées à l'entrée pour garantir la cohérence et la précision.
- Gestion des Erreurs : Des mécanismes doivent être en place pour détecter et corriger les erreurs, avec des alertes en temps réel pour des informations sensibles.
- Contrôles d'Intégrité : Les contrôles d'intégrité garantissent la cohérence des relations, par exemple, entre les candidats, les élections et les votes.
- Audit et Suivi : Les modifications dans les données maîtres doivent être traçables via des journaux d'audit pour identifier toute modification inappropriée.
- Formation Continue : Le personnel responsable de la gestion des données doit recevoir une formation continue pour maintenir la qualité des données.

➢ Tableau du Dictionnaire des Données - Données Secondaires

Numéro	Nom	Type	Nature	Description
1	Id_election	Int	Clé Primaire	Identifiant unique de l'élection.
2	Titre	Varchar (50)	Attribut	Titre de l'élection.
3	Date_debut	Date	Attribut	Date de début de l'élection.

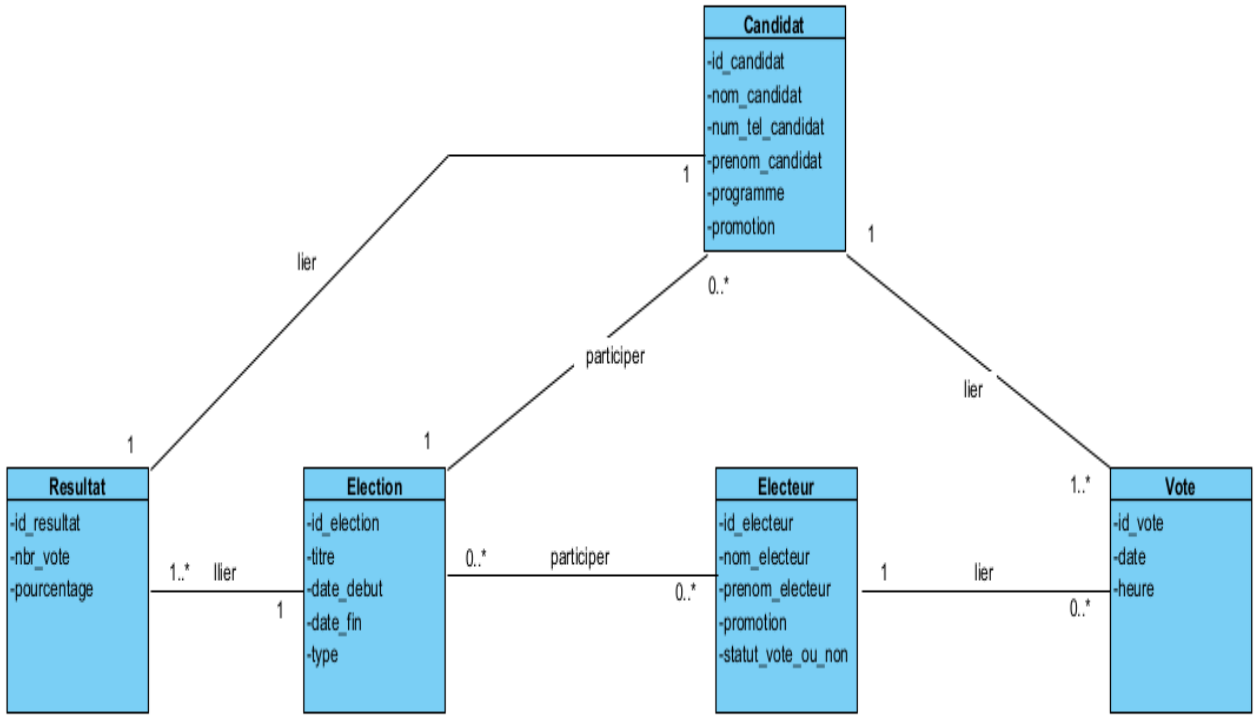
4	Date_fin	Date	Attribut	Date de fin de l'élection.

- Précision des Données Secondaires :
  - Dépendance aux Données Maîtres : Les données secondaires, telles que les détails de l'élection, dépendent des données maîtres pour garantir leur précision.
  - Processus de Dérivation Clair : La dérivation des données secondaires, comme les détails de l'élection, doit suivre des processus clairs, documentés et transparents.
  - Mise à Jour Synchronisée : Toute mise à jour des données maîtres (par exemple, modification du titre de l'élection) doit être synchronisée avec les données secondaires.
  - Contrôles de Qualité : Des contrôles de qualité spécifiques aux données secondaires doivent être en place pour détecter toute anomalie ou inexactitude.
  - Réconciliation Régulière : Des processus de réconciliation réguliers entre les données maîtres et secondaires permettent d'identifier et de résoudre tout écart.
  - Documentation Détaillée : Les règles et processus de dérivation des données secondaires doivent être documentés en détail pour assurer la compréhension et la conformité.

En suivant ces principes, la précision des données maîtres et secondaires est assurée, contribuant à l'intégrité globale du système SmartVote.

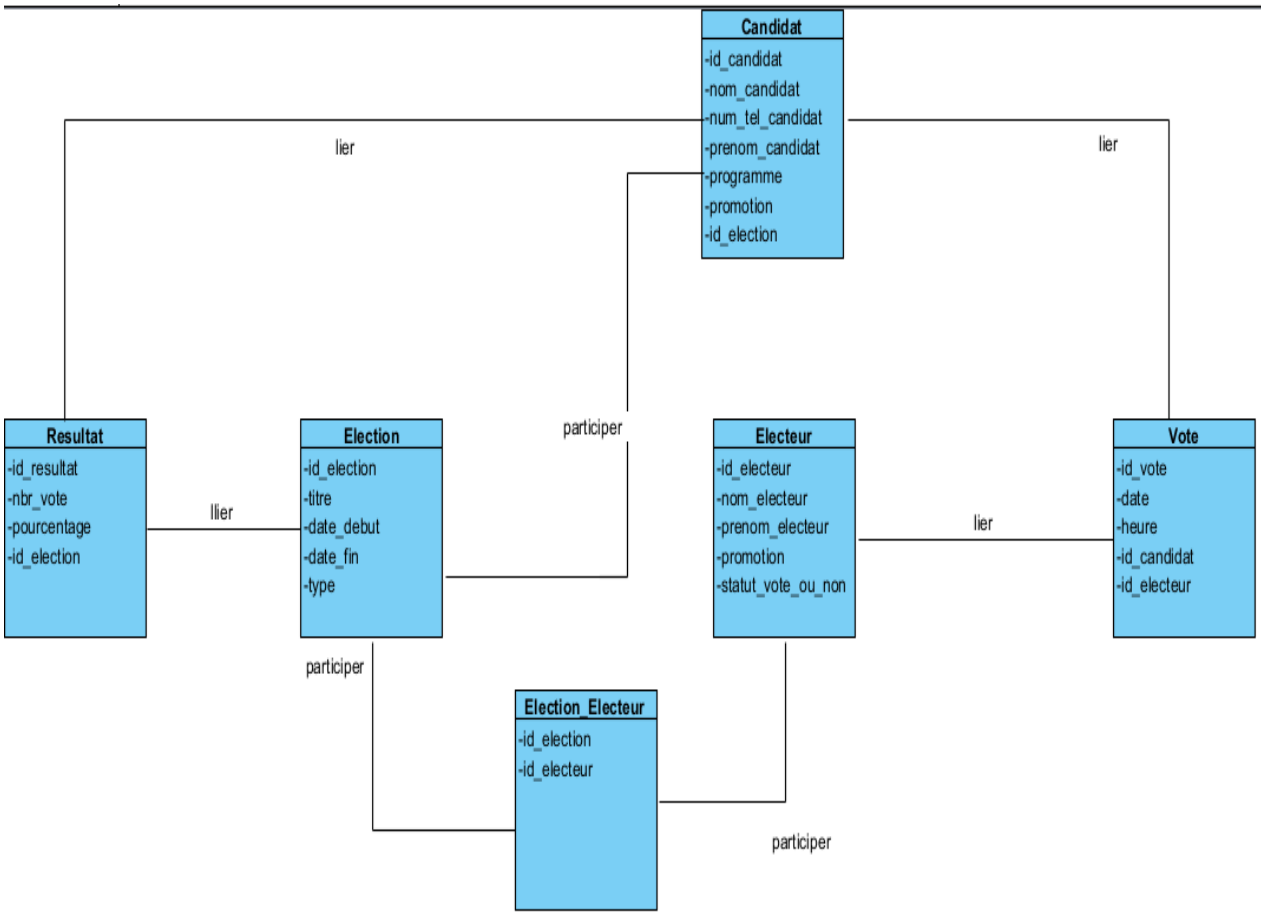
VI.2. Modèle conceptual des données

Le modèle conceptuel des données illustre les entités et leurs relations.



VI.3. Modèle physique des données

Le modèle physique détaille comment les données seront stockées, y compris les tables, les relations et les contraintes.



VII. Architecture de sécurité

- Système d'Authentification :
  - Méthode d'Authentification : Authentification basée sur les identifiants (nom d'utilisateur/mot de passe) pour les utilisateurs (administrateurs, candidats, électeurs).
  - Stockage des Informations d'Authentification : Les informations d'authentification seront stockées de manière sécurisée, en utilisant des algorithmes de hachage pour les mots de passe.
- Algorithmes de Codage des Données :
  - Transmission des Données : Toutes les communications entre les clients et les serveurs (notamment lors des votes, de l'inscription, etc.) seront chiffrées en utilisant

le protocole HTTPS (SSL/TLS) pour assurer la confidentialité des données en transit.

- Stockage des Données : Les données sensibles telles que les informations d'authentification, les résultats des élections, etc., seront stockées de manière sécurisée en utilisant des algorithmes de chiffrement robustes. Par exemple, AES (Advanced Encryption Standard) pour le chiffrement des bases de données.

- Stratégies de Sécurité Supplémentaires :
  - Protection contre les Attaques par Force Brute : Mettre en place des mécanismes de verrouillage de compte après un certain nombre de tentatives infructueuses.
  - Audit des Activités : Mettre en œuvre des journaux d'audit pour enregistrer les activités des utilisateurs, facilitant ainsi la détection des comportements suspects.
  - Gestion des Sessions : Utiliser des mécanismes de gestion de sessions sécurisés pour éviter les attaques de session.
  - Principe du Moindre Privilège : Accorder aux utilisateurs uniquement les privilèges nécessaires à l'exécution de leurs tâches.

Remarques :

Ces choix sont conformes aux bonnes pratiques de sécurité en matière d'authentification et de protection des données.

La mise en œuvre précise des mécanismes de sécurité dépendra des technologies spécifiques utilisées dans le cadre du développement de l'application.

VIII. Architecture d'intégration

- Contrat de Service :
  - Le contrat de service définit les spécifications des services exposés par le système. Pour ce projet, nous adopterons une approche basée sur la description de services à l'aide de la

Web Services Description Language (WSDL). La WSDL est un standard XML utilisé pour décrire les services Web, notamment leurs opérations, leurs entrées et sorties.

- Utilisation de la WSDL :
  - Description des Services : Chaque service, tel que la gestion des élections, l'inscription des candidats, le vote des électeurs, etc., sera décrit dans un fichier WSDL correspondant.
  - Spécification des Opérations : Chaque opération offerte par un service sera clairement spécifiée, y compris les données d'entrée requises et les résultats attendus.
  - Format des Messages : La WSDL permet de spécifier le format des messages échangés entre les clients et les services. Cela garantit une compréhension mutuelle des données.
  - Protocoles de Communication : Le contrat inclura des informations sur les protocoles de communication pris en charge, tels que SOAP (Simple Object Access Protocol) pour les services Web.
- Avantages de l'Approche WSDL :
  - Interopérabilité : La WSDL favorise l'interopérabilité entre les différentes parties du système, car elle fournit une norme compréhensible de tous.
  - Découverte Automatique : Les clients peuvent découvrir et comprendre les services disponibles sans une interaction humaine directe, facilitant ainsi l'intégration.
  - Facilité de Maintenance : Les modifications ultérieures dans les services peuvent être gérées de manière plus fluide grâce à la clarté des contrats.
  - Documentation Automatique : Les fichiers WSDL peuvent être utilisés pour générer automatiquement la documentation du service, ce qui facilite la compréhension et l'utilisation pour les développeurs tiers.

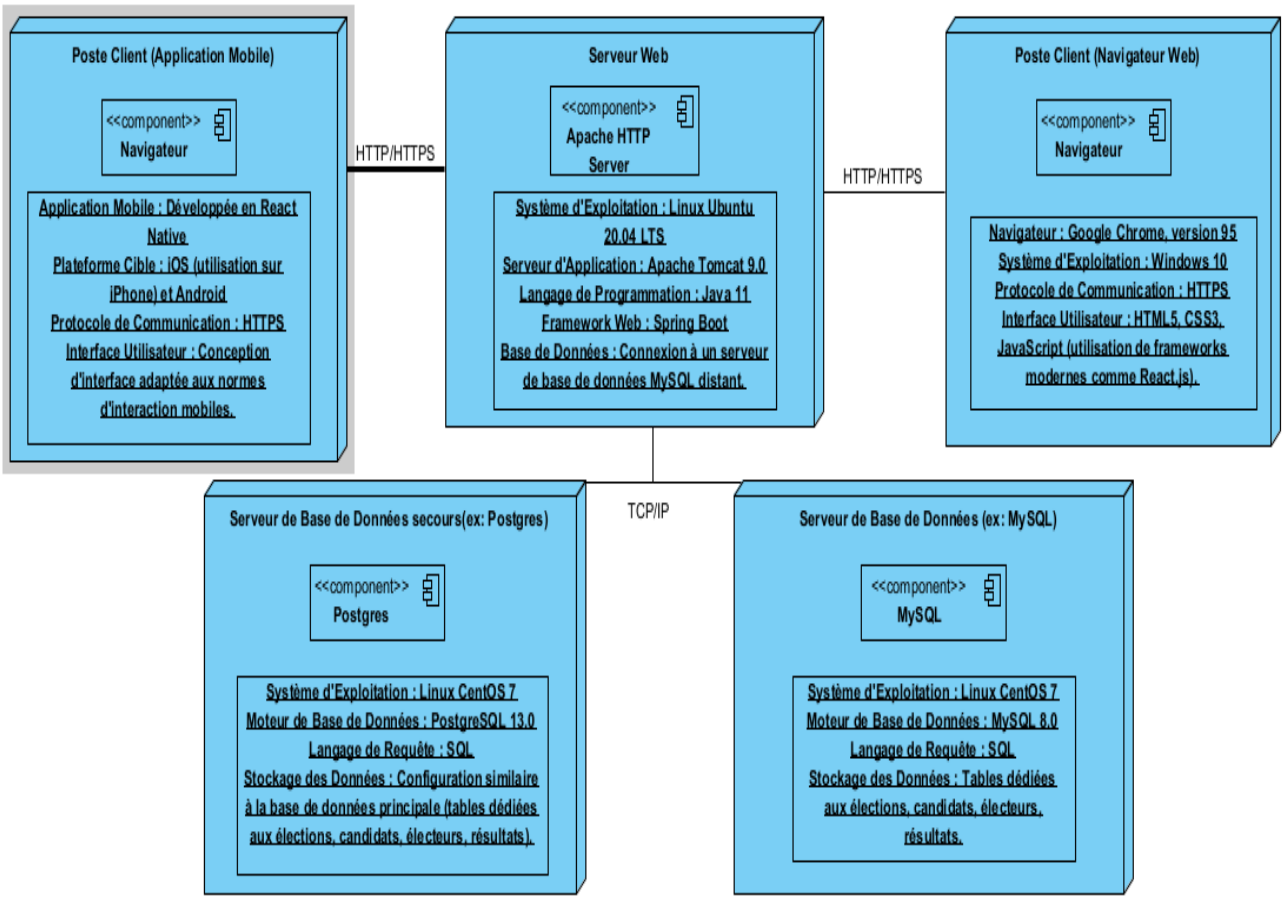
Références :

Web Services Description Language (WSDL) - Wikipedia.

Remarque :

L'adoption de standards comme la WSDL assure une meilleure extensibilité et pérennité du système, facilitant également son intégration avec d'autres systèmes tiers si nécessaire.

IX. Architecture de déploiement



Explications :

- Serveur Web :

Héberge l'application web de gestion des élections.

Utilise un serveur d'application (ex : Apache Tomcat) pour exécuter l'application.

Accède à une base de données pour stocker et récupérer les données.

- Serveur de Base de Données :

  Stocke les informations sur les élections, candidats, électeurs, et résultats.

  Utilise un moteur de base de données (ex : MySQL) pour gérer les données de manière efficace.

- Poste Client (Navigateur Web) :

  Utilise un navigateur web pour accéder à l'application web.

  Interagit avec l'interface utilisateur pour s'inscrire, consulter les élections, et voter.

- Poste Client (Application Mobile) :

  Utilise une application mobile pour accéder à l'application.

  Interagit avec l'interface utilisateur mobile pour s'inscrire, consulter les élections, et voter.

- Plan de Reprise d'Activité (PRA) en Cas de Panne :

  En cas de panne du serveur de base de données principal, le système peut être basculé vers le serveur de base de données de secours. Les données sont répliquées en temps réel entre les deux serveurs pour assurer une continuité d'activité minimale en cas de défaillance.

Remarques :

- Les serveurs sont configurés avec des mesures de sécurité standard, y compris des pare-feux, des mises à jour régulières, et des accès restreints.
- Les communications entre le serveur web et la base de données sont cryptées pour assurer la confidentialité des données.
- Les postes clients accèdent à l'application via une connexion sécurisée (HTTPS) pour garantir la sécurité des informations personnelles des utilisateurs.
- Les technologies choisies visent à assurer la stabilité, la performance et la sécurité de l'ensemble du système.
- La réplication continue des données entre les deux serveurs garantit une cohérence des données en cas de basculement vers le serveur de secours.
- PostgreSQL a été choisi comme alternative pour sa robustesse, sa gestion avancée des transactions, et sa capacité à gérer des charges de travail élevées.
- Des tests réguliers de basculement sont effectués pour garantir la disponibilité et la fiabilité du serveur de base de données de secours.

Cette architecture permet une gestion centralisée des élections tout en offrant une accessibilité aux utilisateurs via des navigateurs web et des applications mobiles.