Université Cheikh Anta DIOP de Dakar

Faculté des Sciences et Techniques

Département Mathématiques et Informatique

Laboratoire d'Algèbre de Cryptologie
de Géométrie Algébrique et
Applications

Licence Transmission de Données et Sécurité de l'Information

Option : CI

Thème :

IMPLEMENTATION DU PROTOCOLE ZKP (ZERO-KNOWLEDGE PROOF) DANS LES SYSTEMES DE VOTE ELECTRONIQUE ET VERIFCATION PAR BLOCKCHAIN

Présenté et soutenu par :
Ousmane DEME
Cheikh Omar LY

Encadreurs :
Dr. TALL Amadou
M. KASSE Cherif

Membres du Jury :

Pr. Cheikh T. GUEYE

M. Cheikh KA

M. Moustapha MBAYE

M. Khaly DIENG

Année Académique 2023 – 2024

DEDICACES

Nous dédions ce mémoire à toutes les personnes qui croient en la force de la démocratie et en l'importance d'un processus électoral transparent et sécurisé. Que ce travail contribue à l'avancement des systèmes de vote électronique et à la promotion de la confiance dans les institutions démocratiques.

REMERCIMENTS

Tout d'abord, louange à **Allah** qui nous a accordé la santé et la force tout au long de ce parcours.

Nous tenons à exprimer notre profonde gratitude à nos encadreurs, **M. KASSE Chérif** et **Dr. TALL Amadou**, pour leur soutien, leurs conseils avisés et leur expertise tout au long de ce projet. Leur accompagnement a été précieux et a grandement contribué à la réalisation de ce mémoire.

Nous souhaitons également remercier toutes les personnes qui ont participé, de près ou de loin, à ce travail. Leur encouragement et leurs retours ont été essentiels pour enrichir notre réflexion et améliorer la qualité de ce projet.

Enfin, nous remercions nos familles et nos amis pour leur soutien inconditionnel et leur compréhension durant cette période de travail intense. Leur présence a été une source de motivation et de réconfort.

Table des matières

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

I.1 Contexte général sur le vote électronique

Les élections sont un pilier fondamental des systèmes démocratiques, permettant aux citoyens d'exprimer leurs choix et de participer à la gouvernance. Cependant, la participation électorale est en déclin, avec de nombreux citoyens ne se rendant aux urnes que lorsque les enjeux sont clairement définis. Dans ce contexte, l'introduction de modalités alternatives de vote, telles que le vote électronique, est souvent perçue comme une opportunité de renouveler l'engagement civique.

Le vote électronique représente une avancée technologique visant à remplacer les méthodes traditionnelles de vote papier dans certains contextes. Il offre une solution moderne aux défis logistiques et humains associés aux élections, tels que la complexité de l'organisation, les erreurs humaines et les coûts élevés. En facilitant la transmission et le dépouillement des votes, le vote électronique promet une plus grande rapidité et une meilleure efficacité dans le traitement des résultats.

Cependant, cette modernisation n'est pas sans défis. La transition vers des systèmes numériques a suscité des inquiétudes majeures concernant la sécurité et la fiabilité des processus électoraux. Des problématiques telles que le piratage informatique, la manipulation des résultats ou la perte de données ont souvent érodé la confiance des citoyens dans ces technologies. Dans ce contexte, l'équilibre entre transparence, confidentialité et robustesse devient une exigence fondamentale pour tout système de vote électronique.

I.2 Introduction au concept des Zero-Knowledge Proofs (ZKP)

Les Zero-Knowledge Proofs (ZKP), ou preuves à divulgation nulle de connaissance, sont des protocoles cryptographiques qui permettent à une partie de prouver qu'elle détient une certaine information sans en révéler le contenu. Introduit dans les années 1980, ce concept a révolutionné la cryptographie en offrant une méthode sécurisée et respectueuse de la confidentialité pour valider des informations sensibles.

Le fonctionnement des ZKP repose sur des principes mathématiques complexes, tels que les calculs probabilistes et les problèmes difficiles à résoudre informatiquement. Ces preuves trouvent aujourd'hui des applications variées, notamment dans les domaines de la blockchain, de l'authentification sécurisée et de la gestion des identités numériques. Appliquées au vote

électronique, elles pourraient permettre de garantir que chaque vote est valide tout en assurant que l'anonymat de l'électeur est parfaitement préservé.

Dans le cadre de ce mémoire, les ZKP constitueront également un prototype adaptable à divers types d'applications, au-delà du vote électronique. Ils seront étudiés pour des cas comme la gestion des clés cryptographiques et la protection des données sensibles, en s'appuyant sur des services de cloud computing tels qu'AWS (Amazon Web Services) pour garantir la sécurité et la scalabilité

I.3 Problématique

Comment intégrer les Zero-Knowledge-Proofs dans un système de vote électronique afin de garantir à la fois la confidentialité des votes individuels et la vérifiabilité globale du processus ? Cette question soulève plusieurs enjeux critiques :

Confidentialité : Protéger l'identité des électeurs et leurs choix, tout en empêchant tout traçage ou corrélation avec le vote émis.

Vérifiabilité : Permettre aux électeurs et aux observateurs de vérifier que les résultats sont exacts, sans compromettre la confidentialité.

Robustesse technique : Garantir la résilience du système face aux cyberattaques et assurer la fiabilité des résultats.

Gestion des données sensibles : Assurer la protection des clés cryptographiques et des informations confidentielles via des plateformes sécurisées comme AWS.

Acceptabilité : Fournir une solution intuitive, transparente et accessible à un large public, y compris ceux ayant peu de compétences techniques.

Ces défis techniques et éthiques nécessitent une exploration approfondie des solutions possibles, notamment à travers l'utilisation des ZKP comme technologie clé et le Cloud pour la gestion des données sensibles.

I.4 Objectifs du mémoire

Ce mémoire a pour but d'explorer et d'évaluer les possibilités d'intégration des Zero-Knowledge Proofs (ZKP) dans les systèmes de vote électronique et autres applications sensibles. Les objectifs principaux sont :

Analyser les fondements théoriques et les applications des ZKP : Comprendre leur fonctionnement et leur pertinence pour les systèmes électoraux.

Identifier les contraintes des systèmes de vote électronique et des plateformes cloud: Étudier les exigences liées à la confidentialité, la sécurité et la vérifiabilité dans des environnements décentralisés.

Proposer une architecture basée sur les ZKP : Développer un modèle de vote électronique répondant aux besoins identifiés et une gestion adaptée des données sensibles, intégrant des services tels qu'AWS.

Évaluer la faisabilité et les performances du modèle proposé : Mettre en place une simulation ou un prototype pour tester l'efficacité et la robustesse de la solution.

I.5 Structure du mémoire

Ce mémoire est organisé comme suit :

Chapitre 1 : Contexte et justification du sujets. Ce chapitre examine les travaux existants et les défis liés à ces technologies.

Chapitre 2 : Analyse des contraintes des systèmes de vote électronique modernes. Ce chapitre identifie les exigences techniques et les limites des approches actuelles.

Chapitre 3 : Proposition d'une architecture intégrant les **ZKP**. Une description détaillée du modèle est fournie, accompagnée d'une discussion sur sa conception.

Chapitre 4 : Validation expérimentale et analyse des résultats. Les performances de la solution proposée sont évaluées à travers des tests et simulations.

Conclusion et perspectives : Une synthèse des contributions du mémoire est présentée, suivie de recommandations pour les recherches futures et les améliorations possibles.

CONTEXTE ET JUSTIFICATION DU SUJET

II.1 État de l'art sur les systèmes de vote électronique

II.1.1 Présentation des moyens de vote traditionnels et électroniques

Les moyens de vote traditionnels incluent le vote papier, qui reste largement utilisé à travers le monde. Ce système repose sur des bulletins de vote que les électeurs remplissent et déposent dans des urnes. Il est apprécié pour sa simplicité et sa transparence mais souffre de problèmes tels que les erreurs humaines, la lenteur du dépouillement et la vulnérabilité aux fraudes locales.

Le vote électronique, quant à lui, s'appuie sur des dispositifs électroniques pour enregistrer, transmettre et dépouiller les votes. Il comprend les machines à voter, le vote en ligne et les systèmes hybrides. Ces technologies visent à améliorer l'efficacité, la rapidité et l'accessibilité, mais soulèvent des questions de sécurité, d'intégrité des données et de vérifiabilité.

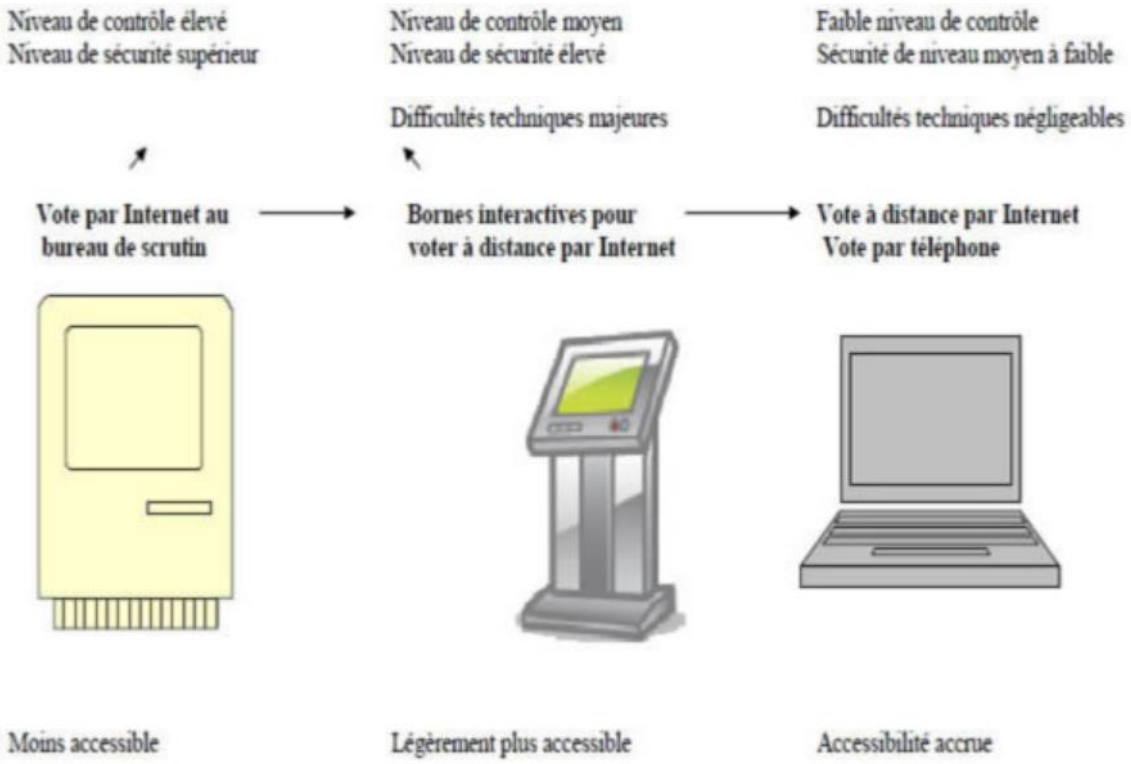


Figure 1 : Types de votes électroniques

II.1.2 Historique du vote électronique

Le vote électronique a commencé à émerger dans les années 1960 avec l'introduction des premières machines à voter. Dans les années 1990, l'essor d'Internet a permis le développement du vote en ligne. Depuis, des systèmes comme HELIOS et BELENIOS ont été conçus pour répondre aux besoins de sécurité et de transparence dans des environnements variés et on assiste même l'adoption de système de vote électronique a l'ensemble national au Sénégal en 2024 ce qu'ouvre une nouvelle ère au système de vote dans ce pays. Cependant, l'adoption de ces technologies reste limitée en raison de défis techniques, organisationnels et culturels.

II.2 Les environnements et défis du vote électronique

II.2.1 Environnements contrôlés et non contrôlés

Dans un environnement contrôlé, tel qu'un bureau de vote, les électeurs sont supervisés et les équipements sont physiquement sécurisés. Cela réduit les risques d'attaques externes mais ne garantit pas toujours la transparence et l'absence de fraudes internes.

Dans un environnement non contrôlé, comme le vote à distance, les électeurs utilisent leurs propres dispositifs. Cela pose des défis supplémentaires en matière de vérifiabilité, de protection contre les attaques cybernétiques et de prévention des fraudes. Ces contraintes soulignent l'importance d'outils cryptographiques robustes pour renforcer la confiance des Électeurs.

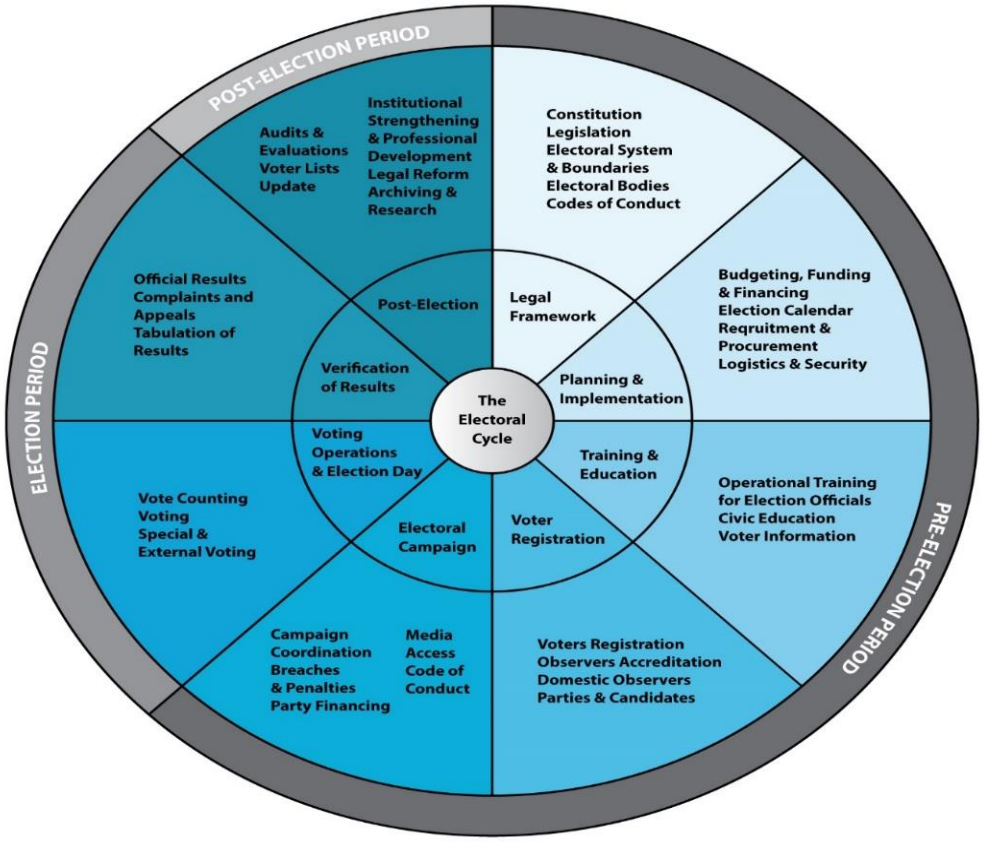


Figure 2 : Processus des votes électoraux

Réf : https://www.eces.eu/template/b.jpg

HELIOS : Ce système open-source permet le vote en ligne avec une vérifiabilité individuelle et universelle. Il repose sur des méthodes cryptographiques pour garantir l'intégrité des votes tout en préservant leur anonymat. Cependant, ses limites incluent une dépendance aux dispositifs des utilisateurs, qui peuvent être compromis, et des vulnérabilités potentielles aux attaques par analyse des données.

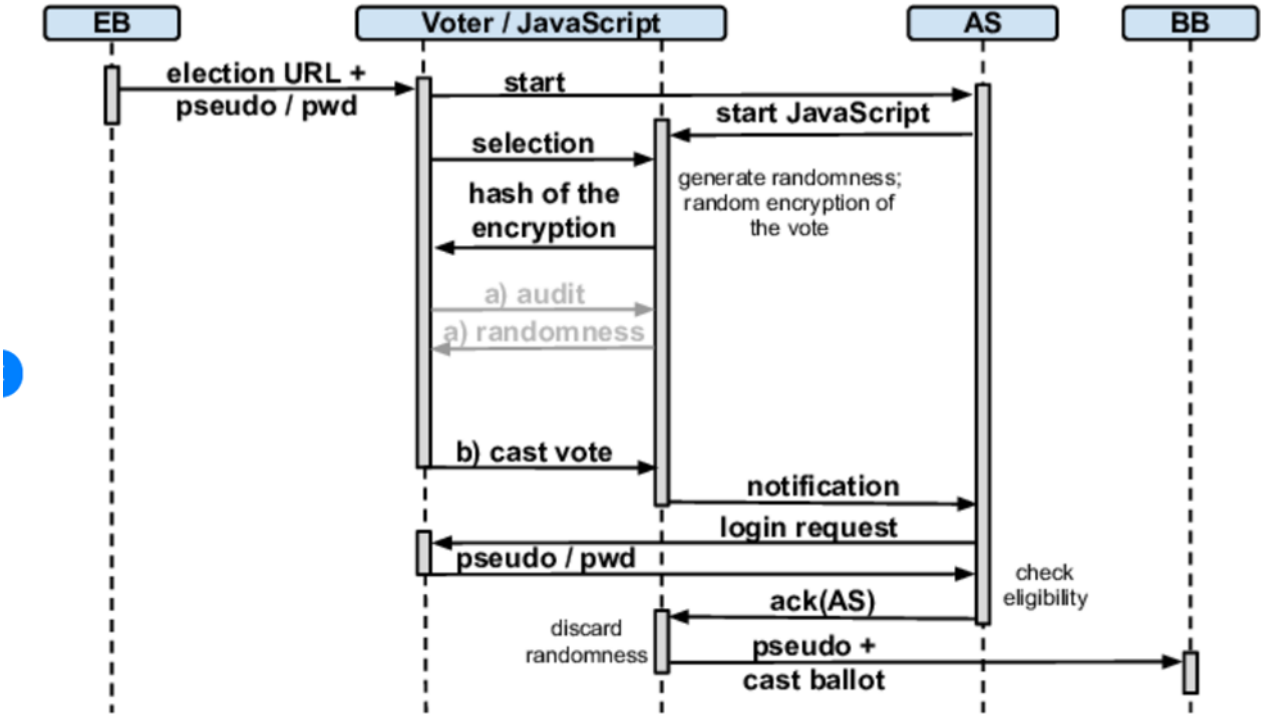


Figure 3 : Fonctionne système de vote HELIOS

Reference: https://www.researchgate.net/figure/Helios-Voting-Scheme_fig2_262933509

Un aperçu du système Hélios est donné dans la Figure 4. Au début de l'élection, le générateur d'élection Hélios envoie à l'électeur un e-mail d'invitation contenant un lien vers le site Web de l'élection ainsi que ses données de connexion éphémères

- ***BELENIOS*** : Également open-source, il se concentre sur la vérifiabilité et l'anonymat. Ce système intègre des outils avancés pour prévenir les manipulations électorales. Cependant, son adoption est freinée par des problèmes d'accessibilité, une complexité d'utilisation pour les non-experts et une capacité limitée à gérer de grands électorats. Belenios propose un système de vote vérifiable. Chaque votant peut s'assurer que son bulletin est bien dans l'urne, n'importe quel tiers peut vérifier que le résultat proclamé correspond aux bulletins dans l'urne et que ceux-ci proviennent d'électeurs légitimes. Le secret du vote est assuré à travers le partage de la clé de déchiffrement entre plusieurs autorités (par exemple des membres du comité électoral) avec un système de seuil (par exemple 3 parmi 5 suffisent à déchiffrer).

II.2.2 Problématiques spécifiques au Sénégal

Au Sénégal, les systèmes de vote électronique doivent répondre à des enjeux de :

- ***Transparence*** : Garantir la confiance des électeurs, particulièrement dans un contexte où les suspicions de fraude électorale sont récurrentes.
- ***Participation*** : Assurer une large accessibilité, y compris pour les électeurs situés dans des zones rurales ou peu connectées.
- ***Simultanéité*** : Permettre un dépouillement rapide et précis tout en garantissant la vérifiabilité des résultats.

II.3 Enjeux de l'utilisation des ZKP dans le vote électronique

Confidentialité des votes : Les preuves à divulgation nulle de connaissance (ZKP) permettent de garantir que le contenu d'un vote reste confidentiel tout en vérifiant son authenticité. En utilisant des méthodes comme les zk-SNARKs, les systèmes peuvent prouver qu'un vote a été correctement pris en compte sans révéler son contenu. Cela réduit les risques de violation de la vie privée, même dans des environnements non contrôlés.

Vérifiabilité et absence de fraude : Les ZKP assurent que chaque vote est correctement pris en compte sans que le contenu du vote soit révélé. Cela renforce la vérifiabilité universelle et l'intégrité du processus électoral. Par exemple, un électeur peut vérifier que son vote a été enregistré et comptabilisé correctement sans que personne d'autre ne puisse associer le vote à son identité.

II.4 Etude comparative des ZKP

Il existe plusieurs types de ZKP, mais voici les deux principaux :

- ZKP interactives : Le prouveur et le vérificateur échangent plusieurs messages. Ce type est moins pratique pour les systèmes distribués comme les blockchains.
- ZKP non-interactives (NIZK) : Il s'agit d'un échange unique, ce qui les rend plus efficaces dans les systèmes automatisés.

Utilisation de Java ou de Python :

- Java est un peu moins utilisée pour les ZKP, mais il existe quelques bibliothèques qui permettent d'implémenter des preuves succinctes et des opérations cryptographiques.
- Python est un excellent choix pour des projets de ZKP grâce à ses nombreuses bibliothèques de cryptographie et à son accessibilité pour le prototypage rapide.

Bibliothèques Python pour les ZKP :

- **PyCryptodome** : Bibliothèque de cryptographie qui peut être utilisée pour implémenter des concepts de base en cryptographie, bien que ce ne soit pas une bibliothèque dédiée aux ZKP.
- **Zokrates-python** : Une bibliothèque de Python qui peut interagir avec ZoKrates pour générer et vérifier des preuves zk-SNARK sur la blockchain Ethereum.
- **PySNARK** : Bibliothèque Python dédiée aux zk-SNARKs qui permet de générer et de vérifier des preuves. Elle est conçue pour ceux qui veulent utiliser les ZKP en Python sans devoir utiliser à des langages de bas niveau.

II.4.1 Comparaison et Choix Optimal pour le Projet de Vote Numérique

Critère	**PyCryptodome**	**Zokrates-python**	**PySNARK**
Adaptation au ZKP	Faible	Très élevé	Élevée
Complexité	Faible	Élevée	Moyenne
Intégration Blockchain	Limitée	Très élevé (Ethereum)	Limitée
Facilité d'utilisation	Très simple	Complexe	Simple à modérée
Performance	Excellente pour cryptographie de base	Excellente pour ZKP sur blockchain	Bonne pour des systèmes internes

Tableau 1 : Comparaison ZKP

- **Recommandation finale**

Pour un projet de vote numérique :

- **Si le vote doit être décentralisé et transparent sur une blockchain**, **Zokrates-python** est le meilleur choix, car il permet une intégration fluide avec Ethereum et des zk-SNARK optimisés pour la blockchain.
- **Si le vote est en local ou ne nécessite pas de vérification publique sur la blockchain**, **PySNARK** peut être plus simple à implémenter et offrir les fonctionnalités nécessaires pour générer et vérifier des ZKP en Python.

En conclusion, **Zokrates-python** est la solution la plus efficace pour un système de vote basé sur blockchain, tandis que **PySNARK** offre une alternative plus légère et plus simple pour des environnements locaux ou des prototypes académiques.

Figure 4 ZKP

II.4.2 Comparaison de Web3.js et Web3.py pour les Systèmes Basés sur ZKP et Blockchain

Critère	Web3.js	Web3.py
Langue	JavaScript (ou TypeScript)	Python
Communauté et écosystème	Très grand, soutenu dans les projets Défi	Large, mais souvent utilisé pour les scripts back end
Performances	Plus rapide dans des applications frontales grâce à JavaScript natif	Performant pour les scripts back end, mais plus prêt pour l'interface
Interopérabilité	Compatible avec les Framework front end (React, Angular, Vue)	Compatible avec les scripts back end et les Framework Python (Django, Flask)
Adéquation au ZKP	Adaptée aux interactions directes avec les contrats ZoKrates (via un front end)	Pratique pour le traitement en back end, avec Zokrates-python ou PySNARK
Prise en charge de TypeScript	Oui	Non, Python seulement
Simplicité d'intégration avec les ZKP	Plus facile pour les D'Apps basés sur le front end où les preuves sont générées ou vérifiées via un smart contract	Plus adapté pour les systèmes back end qui traitent des preuves en local puis envoient le résultat à la blockchain

Tableau 2 : Comparaison Web.py et Web3.js

- **Recommandations pour un système de vote sécurisé**

Pour un système de vote numérique avec des ZKP et une blockchain, la solution optimale dépendra de l'architecture cible.

1. **Système de Vote Interactif et en Temps Réel**
    - Utiliser **Web3.js** si l'objectif est de proposer une interface de vote en temps réel et intuitive pour les électeurs via une application web.
    - Les ZKP peuvent être intégrés côté client pour générer des preuves avant l'envoi du vote au contrat intelligent.
    - **Scénarios recommandés** : Vote d'une petite à moyenne échelle avec une forte interaction frontale (par exemple, élections internes, sondages d'entreprise).
2. **Système de Vote Massif et Décentralisé**
    - Utilisez **Web3.py** pour un back end sécurisé qui peut traiter et vérifier les votes en batch.
    - Python facilite le traitement sécurisé des données et permet d'agréger des votes avant de les enregistrer sur la blockchain, notamment ainsi les coûts de transaction.
    - **Scénarios recommandés** : Vote à grande échelle nécessitant un dépouillement sécurisé et automatisé, où la confidentialité et la scalabilité sont cruciales (par exemple, élections locales ou nationales).

Si on décide d'utiliser Web3.py, l'architecture se divise en deux parties principales :

1. **Le Back end (Python, Flask/FastAPI + Web3.py) :** Interagir avec la blockchain, vérifier les preuves à connaissance nulle, et gérer les transactions.
2. **Front end (React/Vue/Angular) :** Permet à l'utilisateur de voter, de visualiser l'état des votes, et de recevoir des confirmations, tout en masquant la complexité de l'interaction blockchain.

Cependant à la place de Flask on peut utiliser Django car il présente des avantages considérables en termes de gestion de la sécurité surtout dans le domaine du vote numérisé.

II.4.3 Avantages de Django pour un Système de Vote

- **Sécurité** : Gestion intégrée des sessions utilisateur et middleware de sécurité, ce qui est essentiel pour protéger un système de vote.
- **Structure robuste** : Django propose une structure modulaire adaptée aux projets complexes, facilitant la maintenance à long terme.
- **Gestion des Bases de Données** : Vous pouvez facilement utiliser la base de données de Django pour enregistrer les données des électeurs ou les logs des votes si nécessaire pour des audits internes.

- **Support des Permissions et de l'Autorisation** : Avec Django REST Framework, vous pouvez définir des permissions pour restreindre l'accès aux End points de l'API.

II.4.4 Pour le front end

Critère	React	Vue.js	Angular
Adaptation avec Django	Très bien adapté, flexible	Bien adapté, simple à intégrer	Bien adapté pour des projets complexes
Interaction avec Web3.py	Bonne, surtout pour temps réel	Bonne, simple à configurer	Excellente, surtout pour projets structurés
Complexité du projet	Applications dynamiques	Projets modulaires et légers	Projets d'envergure et complexes
Expérience Utilisateur	Très interactive, personnalisable	Simple et intuitif	Expérience structurée et rigoureuse
Apprentissage et Flexibilité	Moyen à facile	Facile	Modéré à difficile (TypeScript requis)

Tableau 3 : Front-end pour ZKP

- **Conclusion :**

- **Pour une interface utilisateur interactive et dynamique avec une intégration simple à Django et Web3.py** : **React** est souvent le choix recommandé. Il est flexible, dispose d'un large écosystème, et fonctionne particulièrement bien avec Django REST API et les mises à jour en temps réel.
- **Pour un projet léger et modulaire** où l'on veut une **intégration simple et rapide** avec Django, **Vue.js** est un excellent choix. Il est facile à apprendre, offre une bonne interactivité, et s'intègre sans difficulté avec Django et Web3.py via des API.
- **Pour des projets plus complexes ou de plus grande envergure** avec des **exigences élevées de structuration et de gestion d'état** (comme une plateforme de vote avec des

contrôles et des permissions rigoureux), **Angular** est très bien adapté. Il offre une architecture claire et une gestion robuste de l'état avec NgRx, même si la courbe d’apprentissage est plus élevée.

Chaque Framework a ses avantages spécifiques, mais pour une approche plus simple et flexible, **React** ou **Vue.js** est souvent préféré dans un contexte de développement rapide avec Django et Web3.py.

II.5 Avantages supplémentaires des ZKP dans le contexte du vote électronique

- ***Résistance aux attaques*** : Les ZKP peuvent empêcher des adversaires malveillants d'extraire des informations sensibles, même en cas d'accès partiel au système.
- ***Interopérabilité*** : Ces preuves peuvent être intégrées à différents types de systèmes électoraux, qu'il s'agisse de vote en ligne ou en présentiel.
- ***Adaptabilité*** : Les ZKP peuvent être personnalisées pour répondre aux besoins spécifiques de chaque contexte électoral.

Ce chapitre met en évidence les besoins croissants en sécurité, transparence et anonymat dans les systèmes de vote électronique, ainsi que les avantages potentiels des ZKP pour relever ces défis. Ces technologies représentent une évolution majeure dans la conception de systèmes électoraux modernes, permettant de concilier anonymat des votes et vérifiabilité des résultats.

SPÉCIFICATION ET ANALYSE DES BESOINS

III.1 Spécification des besoins fonctionnels

III.1.1 Identification des acteurs

Pour assurer un fonctionnement optimal et répondre aux exigences d'un système de vote électronique sécurisé, nous identifions trois catégories principales d'acteurs :

- **Administrateurs**

  Les administrateurs jouent un rôle central dans la gestion et la supervision du processus électoral. Leurs responsabilités incluent :

  - **Création et configuration de l'élection** : définition du calendrier électoral, ajout des candidats et électeurs, et configuration des règles spécifiques de l'élection.
  - **Gestion des candidatures** : validation ou rejet des candidatures soumises selon des critères préétablis.
  - **Publication des résultats** : gestion de la transparence via la publication des résultats et des preuves cryptographiques associées.

- **Électeurs**

  Les électeurs sont les utilisateurs finaux du système, et leurs droits doivent être protégés tout au long du processus. Leurs activités incluent :

  - **Participation anonyme et sécurisée** : garantir que le vote reste confidentiel tout en permettant une vérifiabilité indépendante.
  - **Vérification du vote** : après avoir voté, l'électeur doit pouvoir vérifier que son vote a été enregistré sans compromettre sa confidentialité.

- **Observateurs**

  Les observateurs assurent la transparence du système. Ils peuvent :

  - Accéder aux preuves cryptographiques pour vérifier la conformité des votes sans voir leur contenu.
  - Auditer le processus électoral afin de détecter tout comportement frauduleux ou manipulation des résultats.

III.1.2 Description des fonctionnalités clés

Les fonctionnalités du système doivent répondre aux objectifs de confidentialité, de transparence et de sécurité tout en offrant une expérience utilisateur fluide.

- **Dépôt de candidature**
    - **Soumission** : Les autorités compétentes confirment l'éligibilité des candidats, puis transmettent la liste des candidats aux administrateurs.
    - **Publication** : Les administrateurs publient la liste officielle des candidats validés pour la mettre à disposition des électeurs.
- **Vote électronique sécurisé**
    - **Authentification** : Chaque électeur s'authentifie via un système sécurisé basé sur un identifiant unique (ex., numéro d'électeur et mot de passe à usage unique).
    - **Sélection d'un candidat** : L'électeur choisit son candidat parmi une liste.
    - **Génération des preuves cryptographiques (ZKP)** : Une preuve de connaissance nulle est générée pour garantir que le vote a été correctement soumis sans divulguer le choix.
    - **Enregistrement dans la blockchain** : Le vote est chiffré et stocké dans une blockchain publique pour garantir son intégrité.
- **Visualisation des résultats**
    - Les résultats sont agrégés et présentés sous forme de graphiques et tableaux compréhensibles.
    - Les preuves cryptographiques des votes (via les ZKP) sont publiées pour que les observateurs puissent vérifier leur authenticité.

III.2 Analyse des activités

L'analyse des activités repose sur une modélisation des processus principaux. Chaque processus est décrit en détail et accompagné de diagrammes UML pour une meilleure compréhension.

III.2.1 Processus principaux

- **Dépôt de candidature**

Ce processus est destiné aux candidats souhaitant participer à l'élection.

  - **Étape 1** : L'autorité compétente envoie la liste des candidats
  - **Étape 2** : L'administrateur reçoit une alerte pour examiner la soumission.
  - **Étape 3** : La liste finale des candidats validés est publiée.

- **Processus de vote électronique sécurisé**

Ce processus est au cœur du système et doit garantir à la fois la confidentialité et la vérifiabilité des votes.

  - **Étape 1** : L'électeur s'identifie via une interface sécurisée (identifiant et mot de passe).
  - **Étape 2** : Une fois authentifié, l'électeur accède à une liste des candidats et sélectionne son choix.
  - **Étape 3** : Le système génère une preuve cryptographique (via ZKP) confirmant que l'électeur a voté conformément aux règles sans révéler son choix.
  - **Étape 4** : Le vote est chiffré et transmis à la blockchain où il est stocké de manière immuable.

- **Visualisation des résultats**

Ce processus intervient après la fin de l'élection.

  - **Étape 1** : Les résultats bruts sont calculés par le système en agrégeant les votes stockés dans la blockchain.
  - **Étape 2** : Les preuves cryptographiques (ZKP) de chaque vote sont publiées pour garantir leur validité.
  - **Étape 3** : Les résultats finaux sont présentés sous forme de graphiques interactifs accessibles aux électeurs et observateurs.

III.2.2 Diagrammes UML

L'UML est un outil de modélisation qui guide la création et la notation de nombreux types de diagrammes, y compris les diagrammes comportementaux, les diagrammes d'interaction et les diagrammes de structure.

III.2.2.1.1 Diagramme de de classe

- **Entités et Relations pour le MCD**

  Pour les différentes entités on a **Electeur, Vote, Candidat, Election, Session vérification**

1. **Electeur**

   - **ID Electeur** *(PK)* : Identifiant unique pour chaque électeur.
   - **Nom**
   - **Prénom**
   - **Email** : Email pour l'authentification et la communication.
   - **Mot_de_Passe** : Mot de passe chiffré pour la sécurité.
   - **Cle_Publique** : Clé publique de l'électeur utilisée pour valider le vote grâce aux ZKP.
   - **Cle_Privee** *(Optionnelle)* : Clé privée (stockée de manière sécurisée) si nécessaire pour l'interaction avec le block Chain.

2. **Vote**

   - **ID_Vote** *(PK)* : Identifiant unique pour chaque vote.
   - **Electeur** *(FK vers Electeur)* : Lien vers l'électeur ayant voté.
   - **Candidat** *(FK vers Candidat)* : Lien vers le candidat choisi.
   - **Date_Heure** : Date et heure du vote.
   - **Hash_Vote** : Hachage du vote pour garantir l'intégrité des données (généré avec les fonctions de la blockchain).
   - **ZKP_Vote** : Preuve ZKP validant le vote sans divulguer les détails du vote.
   - **Transaction_ID** : Identifiant de la transaction dans la blockchain pour la traçabilité et la validation.

3. **Candidat**

- **ID_Candidat** *(PK)* : Identifiant unique du candidat.
- **Nom** : Nom du candidat
- **Parti** : Nom du parti politique associé au candidat.
- **Description** : Brève description ou biographie du candidat.

4. **Election**

- **ID_Election** *(PK)* : Identifiant unique de l'élection.
- **Nom** : Nom de l'élection (ex. : "Election Présidentielle 2024").
- **Date_Debut** : Date de début de l'élection.
- **Date_Fin** : Date de fin de l'élection.
- **Statut** : Statut actuel de l'élection (ouverte, fermée, etc.).
- **Adresse_Blockchain** : Adresse blockchain unique pour stocker les données de l'élection.

5. **Session_Verification**

- **ID_Session** *(PK)* : Identifiant unique de la session de vérification.
- **Election** *(FK vers Election)* : Lien vers l'élection concernée.
- **Date_Heure** : Date et heure de la session de vérification.
- **Type_Verification** : Type de vérification (ZKP, déchiffrement des résultats).
- **Resultat** : Résultat de la session de vérification.
- **Verification_Hash** : Hachage de la vérification pour assurer l'intégrité et la traçabilité sur la blockchain

- **Relations**

- **Electeur - Vote** : *1, n* — Un électeur peut effectuer plusieurs votes (si autorisé).
- **Candidat - Vote** : *1, n* — Un candidat peut recevoir plusieurs votes.
- **Election - Vote** : *1, n* — Une élection peut avoir plusieurs votes associés.
- **Election - Session_Verification** : *1, n* — Une élection peut avoir plusieurs sessions de vérification.

- **Considérations pour l'Implémentation avec Django**

- Utiliser les modèles Django pour mapper chaque entité.

- Intégrer des bibliothèques de blockchain comme `web3.py` pour les transactions et des bibliothèques de ZKP comme `pycryptodome` pour les preuves cryptographiques.
- Exploiter l'`AbstractBaseUser` de Django pour le modèle Electeur avec un système d'authentification personnalisé.

Sécurisée et évolutive, en assurant une traçabilité grâce à l'intégration de la blockchain et des preuves ZKP

- **Analyse des Entités et Relations**
  - On a défini plusieurs entités principales : **Electeur**, **Vote**, **Candidat**, **Election**, et **Session_Verification**. Commence par bien comprendre les liens entre ces entités (MCD - modèle conceptuel de données).
  - Établis un schéma de base de données en partant des relations définies : par exemple, un **électeur** peut avoir plusieurs **votes** et chaque **vote** est associé à un **candidat** et une **élection**. Visualiser ce modèle peut te donner une vue d'ensemble du système.
- **Création des Modèles dans Django**
  - Pour chaque entité, on crée un modèle Django correspondant. Voici quelques recommandations spécifiques :
    - **Electeur** : Exploite AbstractBaseUser de Django pour personnaliser l'authentification. Cela te permettra de gérer les champs spécifiques, comme l'email et le mot de passe chiffré.
    - **Vote** et **Candidat** : Implémente une relation de clé étrangère pour lier chaque vote à un électeur et un candidat, en assurant que chaque vote est tracé avec un **Hash_Vote** et un **Transaction_ID** de blockchain.
  - **Élections** et **Sessions de Vérification** : Crée des champs pour gérer le statut de l'élection, les dates de début et de fin, et les sessions de vérification (inclus le hachage pour assurer l'intégrité).
- **Intégration de la Blockchain**
  - Pour assurer la transparence et la sécurité, chaque vote doit être enregistré sur la blockchain. Utilise web3.py pour interagir avec la blockchain Ethereum.

- Configure le champ **Adresse_Blockchain** dans le modèle Election pour stocker les informations spécifiques à chaque élection.
- Pour les **votes** et les **sessions de vérification**, stocke un hachage unique (par exemple avec **Hash_Vote** et **Verification_Hash**) pour assurer que les données n'ont pas été altérées.

- **Implémentation des Preuves à Connaissance Zéro (ZKP)**

  - Les ZKP sont utilisés pour valider le vote sans divulguer les détails. On utilisera une bibliothèque comme pycryptodome, ou des technologies comme Circom, Arkworks ou Zokrates pour gérer ces preuves. Cela permettra de renforcer la confidentialité, en vérifiant la validité des votes sans révéler l'identité de l'électeur ou de son choix.

- **Sécurité et Gestion des Clés**

  - On met en place un stockage sécurisé pour les clés privées si nécessaire AWS sera l'outil idéal. La clé publique de chaque électeur sera utilisée pour vérifier les votes, mais la clé privée, requise pour la blockchain, doit être gérée avec soin. Ainsi on utilisera des outils comme **Django cryptography** ou le AWS pour chiffrer les données sensibles, en particulier les clés.

- **Tests et Validation**

  - Une fois la base de données et les interactions blockchain mises en place, on effectue des tests unitaires pour chaque modèle et fonction.
  - On teste aussi les fonctionnalités de traçabilité et de vérification des votes via blockchain pour t'assurer de l'intégrité du système.

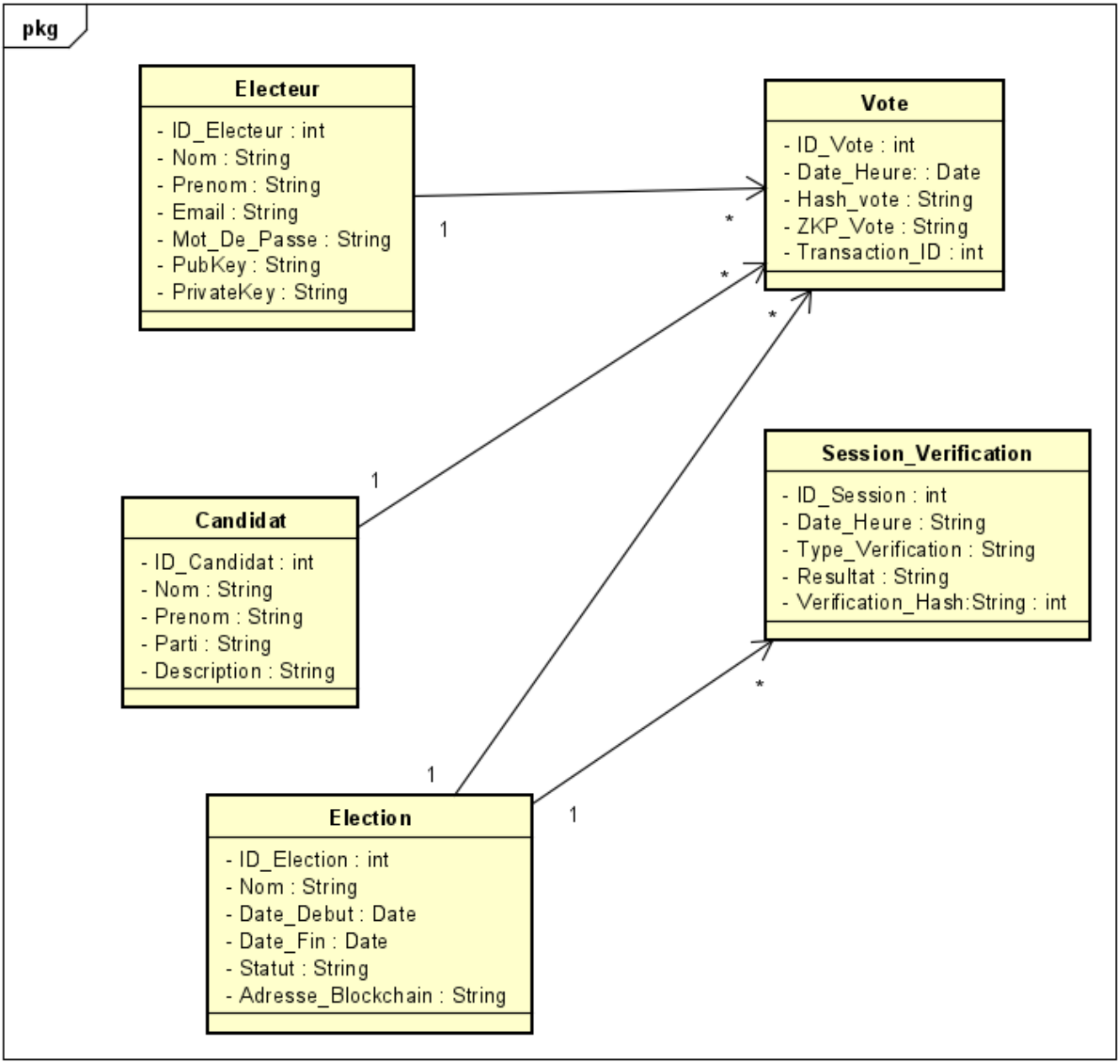


Figure 5 : Diagramme de classe

III.2.2.1.2 Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation montre les interactions entre les **acteurs** (administrateurs, électeurs, observateurs) et le **système de vote électronique**.

- **Acteurs**
    - **Administrateur :** Un acteur avec des privilèges élevés, responsable de la gestion de l'élection.
    - **Électeur :** Un utilisateur final, qui participe au vote.
    - **Observateur :** Une personne qui surveille et vérifie la transparence du processus électoral.
    - **Système de vote :** L'entité responsable de la gestion des interactions entre les acteurs

- **Administrateur**
    - **Configurer l'élection** : L'administrateur configure la date, le type d'élection, les candidats et les règles de l'élection. Il peut spécifier les conditions de sécurité et les paramètres techniques du système de vote.
    - **Gérer les candidatures** : L'administrateur examine et valide ou rejette les candidatures des candidats. Il peut aussi modifier ou supprimer des candidatures existantes.
    - **Gérer les électeurs** : L'administrateur peut ajouter, supprimer ou modifier les informations des électeurs dans la base de données.

- **Électeur**
    - **S'authentifier** : L'électeur se connecte au système à l'aide de ses identifiants (nom, mot de passe, etc.) et, le cas échéant, d'une authentification à deux facteurs.
    - **Voter** : Une fois authentifié, l'électeur peut sélectionner un candidat parmi ceux proposés.
    - **Vérifier son vote** : L'électeur peut consulter et s'assurer que son vote a bien été pris en compte dans le système, tout en garantissant son anonymat grâce aux Zero-Knowledge Proofs (ZKP).

- **Recevoir une confirmation** : Après avoir voté, l'électeur reçoit une confirmation que son vote a bien été enregistré, sans divulguer son choix.

- **Observateur**
  - **Accéder aux résultats** : L'observateur peut accéder aux résultats finaux des élections après leur publication par le système.
  - **Vérifier les preuves ZKP** : L'observateur peut consulter les preuves cryptographiques générées lors du vote pour s'assurer qu'aucun vote n'a été modifié et que les votes sont authentiques sans dévoiler le contenu de ces derniers.

- **Système de vote :**
  - **Gérer les votes** : Enregistrer les votes dans une blockchain sécurisée et les associer aux preuves cryptographiques.
  - **Publier les résultats** : Agréger les votes et les présenter sous forme de résultats détaillés.
  - **Fournir des preuves ZKP** : Générer des preuves ZKP pour chaque vote et permettre leur vérification par les observateurs.

- **Scénarios typiques**

- **L'administrateur configure une nouvelle élection**
  - L'administrateur remplit un formulaire pour définir les paramètres de l'élection.
  - Le système crée les espaces nécessaires pour les candidatures et l'inscription des électeurs.

- **L'électeur vote**
  - L'électeur se connecte et sélectionne un candidat.
  - Le système génère une preuve ZKP et enregistre le vote dans la blockchain.

- **L'observateur vérifie les résultats**

  L'observateur consulte les résultats et vérifie que les preuves cryptographiques des votes sont conformes

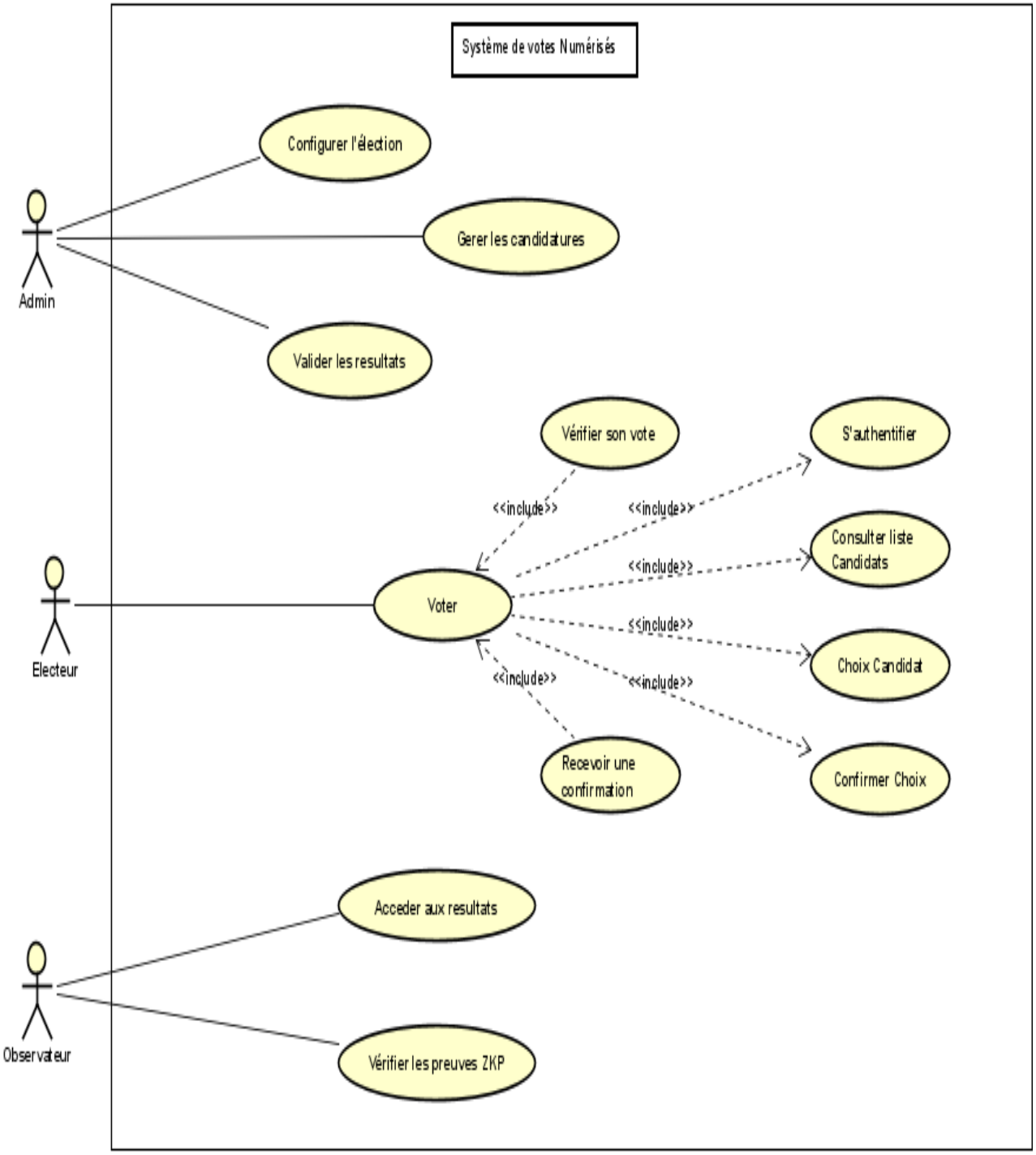


Figure 6 : Diagramme de cas d'utilisation

III.2.2.1.3Diagramme de Séquence

Un diagramme de séquence est un type de diagramme d'interaction, car il décrit comment et dans quel ordre plusieurs objets fonctionnent ensemble. Ces diagrammes sont utilisés à la fois par les développeurs logiciels et les managers d'entreprises pour analyser les besoins d'un nouveau système ou documenter un processus existant. Les diagrammes de séquence sont parfois appelés diagrammes d'événements ou scénarios d'événements.

- **Participants**
  - **Électeur** : L'utilisateur qui vote.
  - **Système de vote** : Le système qui gère le processus de vote.
  - **Blockchain** : Le système de stockage décentralisé où les votes sont enregistrés.
  - **ZKP (Zero-Knowledge Proofs)** : La méthode utilisée pour garantir l'intégrité du vote tout en préservant l'anonymat.
- **Processus détaillé de vote**

- **L'électeur se connecte au système**
  - L'électeur entre son identifiant et son mot de passe.
  - Le système vérifie l'authenticité de l'électeur.
  - Si l'identification est réussie, le système affiche la liste des candidats.

- **L'électeur sélectionne un candidat**
  - L'électeur choisit un candidat et confirme son choix.
  - Le système génère une preuve ZKP associée à ce vote. Cette preuve garantit que l'électeur a fait un choix valide sans révéler le candidat sélectionné.

- **Le vote est enregistré**
  - Le vote, accompagné de la preuve ZKP, est transmis au système de blockchain.
  - Le système de blockchain enregistre le vote de manière immuable et vérifie la validité de la preuve ZKP.

- **Confirmation du vote**

- Le système affiche une confirmation à l'électeur, indiquant que son vote a bien été enregistré et qu'il peut vérifier son vote ultérieurement grâce à un identifiant unique.
- Le vote est maintenant verrouillé dans la blockchain.

- **Scénarios supplémentaires :**

- **Vérification des résultats par l'observateur :**

  - L'observateur consulte les résultats finaux après la clôture de l'élection.
  - Il peut vérifier les résultats et s'assurer que tous les votes sont valides grâce aux preuves ZKP.
  - Les résultats sont affichés sous forme de graphiques ou de tableaux interactifs, accompagnés des preuves cryptographiques pour garantir la transparence du processus.

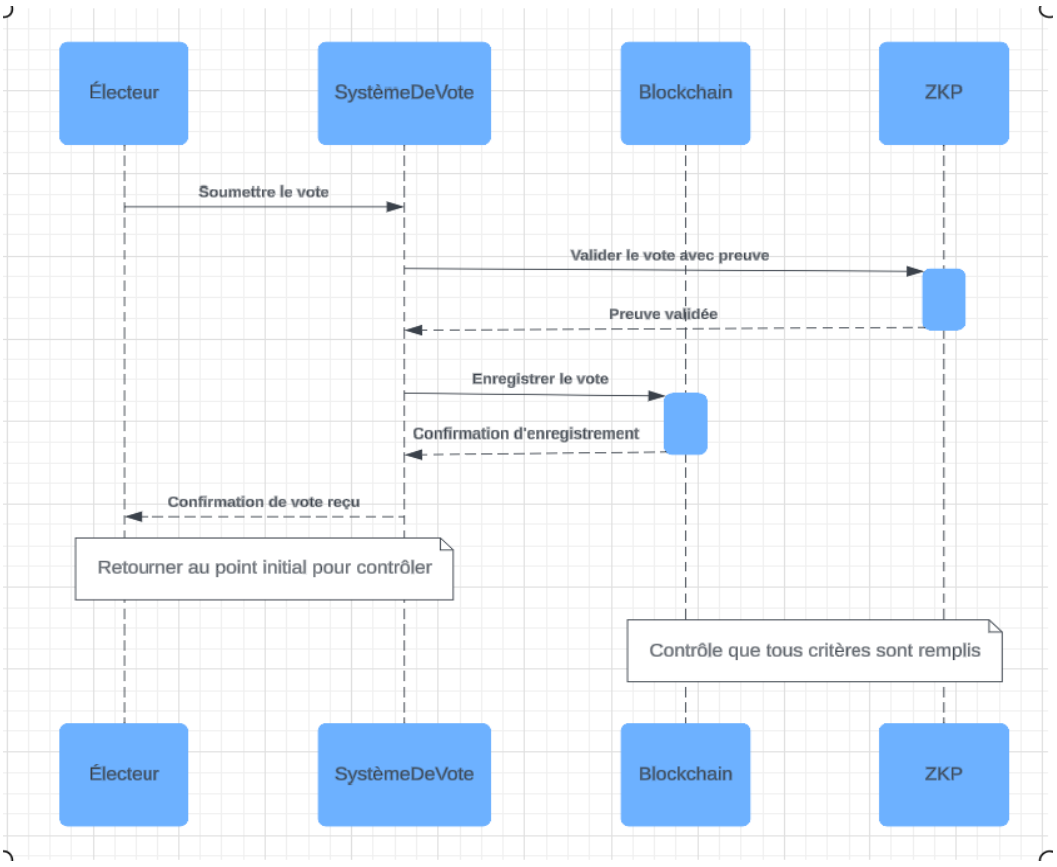


Figure 7 : Diagramme de séquence

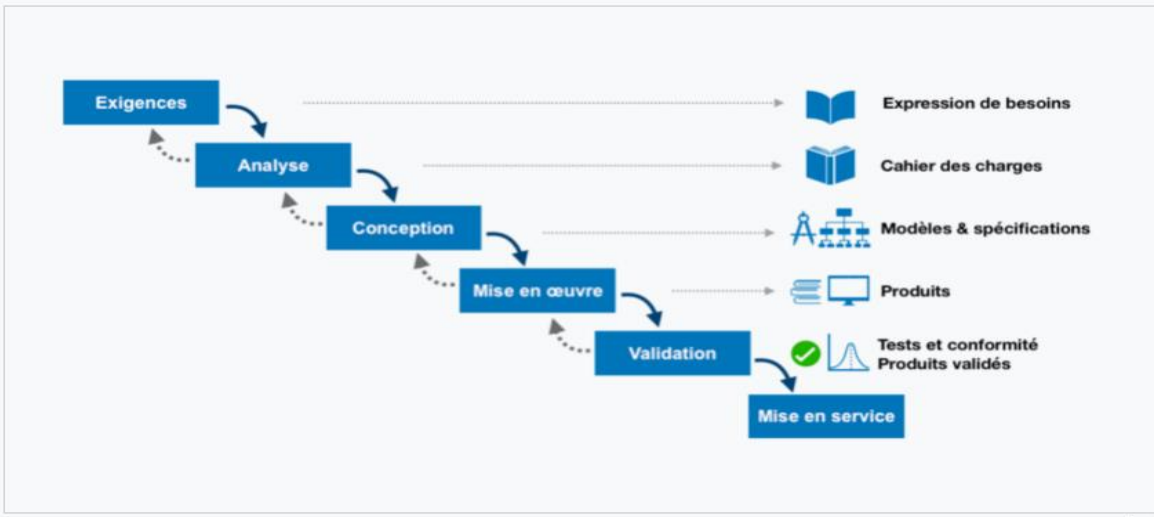
CONCEPTION DÉVELOPPEMENT ET MISE EN ŒUVRE DU SYSTÈME

IV.1 Méthodologies de développement

IV.1.1 Présentation des modèles traditionnels

Les modèles traditionnels de développement logiciel sont caractérisés par une approche structurée et linéaire, où chaque phase doit être complétée avant de passer à la suivante. Parmi les principaux modèles, nous retrouvons :

- **Modèle en cascade** : Ce modèle suit une progression séquentielle, débutant par l'analyse des besoins, suivie de la conception, du développement, des tests, du déploiement et de la maintenance. Bien que rigoureux, il manque de flexibilité pour gérer des changements imprévus.



Modèle en cascade générique avec les principaux livrables. La détection de défauts en aval nécessite des retours vers les étapes précédentes, jusqu'aux exigences si celles-ci sont erronées.

Figure 8:Modèle en Cascade

- **Modèle en V** : Extension du modèle en cascade, il intègre une validation et une vérification à chaque étape, en reliant les phases de conception et de test de manière systématique. 3 phases sont généralement associées à cette méthode :

  - La phase de conception : analyse et définition des besoins
  - La phase de réalisation : développement du produit
  - La phase de validation : tests et recettage

  Ce type de méthode requiert de valider chaque étape avant de passer à la suivante, ainsi qu'un engagement sur le planning très précis.

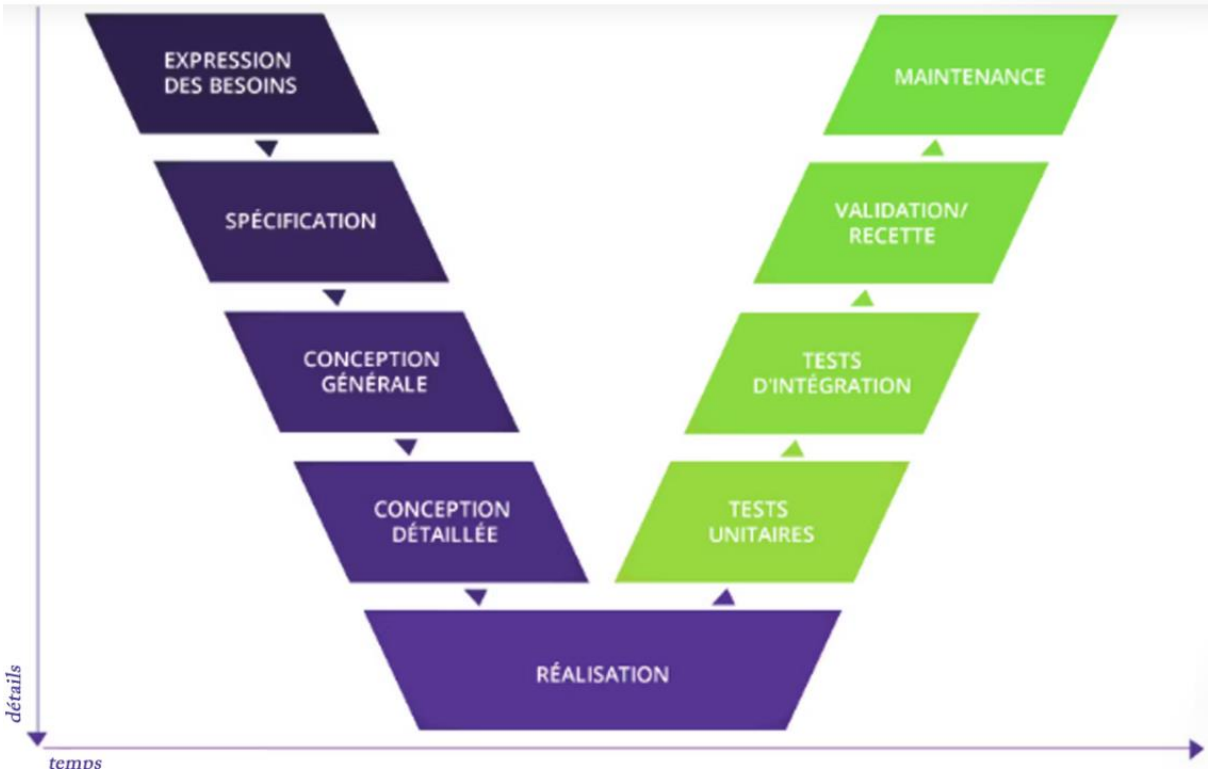


Figure 9: Modèle en V

- **Modèle en Y** : Ce modèle met l'accent sur la réutilisation des composants logiciels et la gestion des variations. Il est idéal pour des projets ayant des exigences similaires mais des variations fonctionnelles.

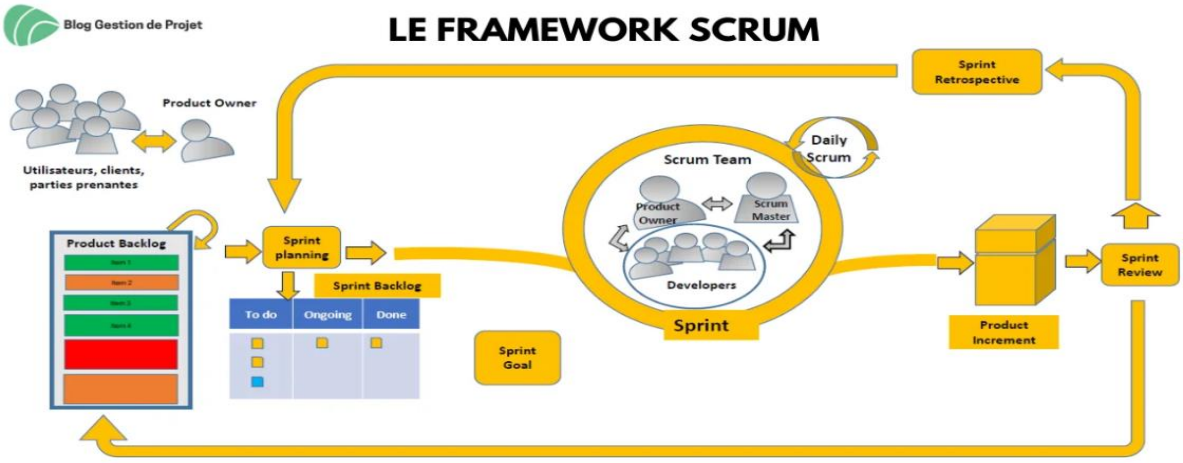


Figure 10:Modèle en Y

IV.1.2 Présentation des méthodes agiles

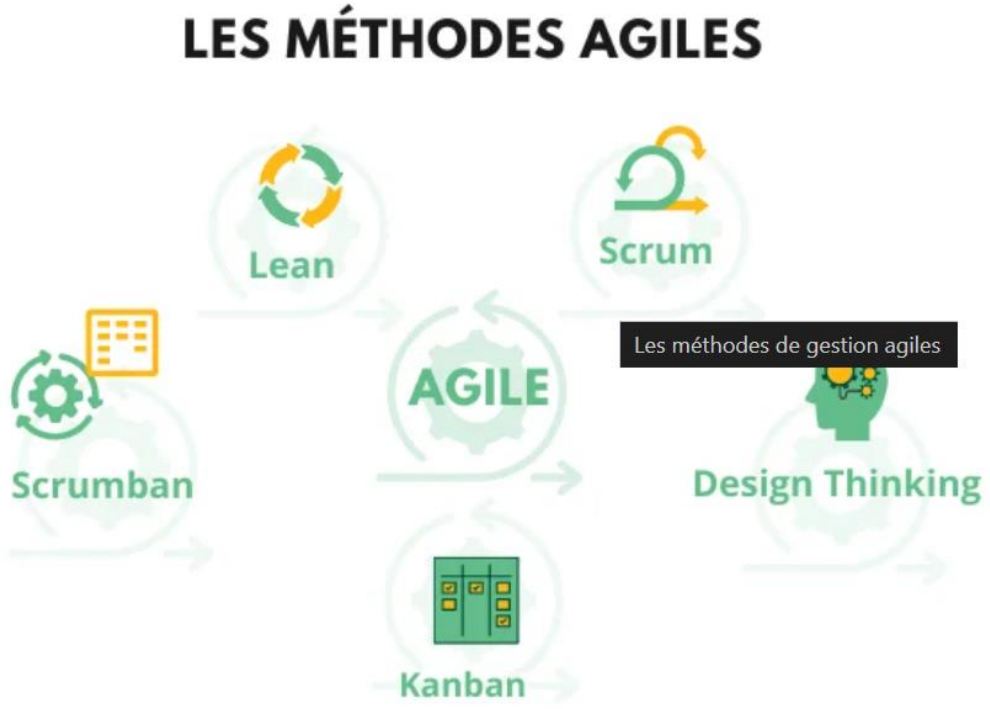
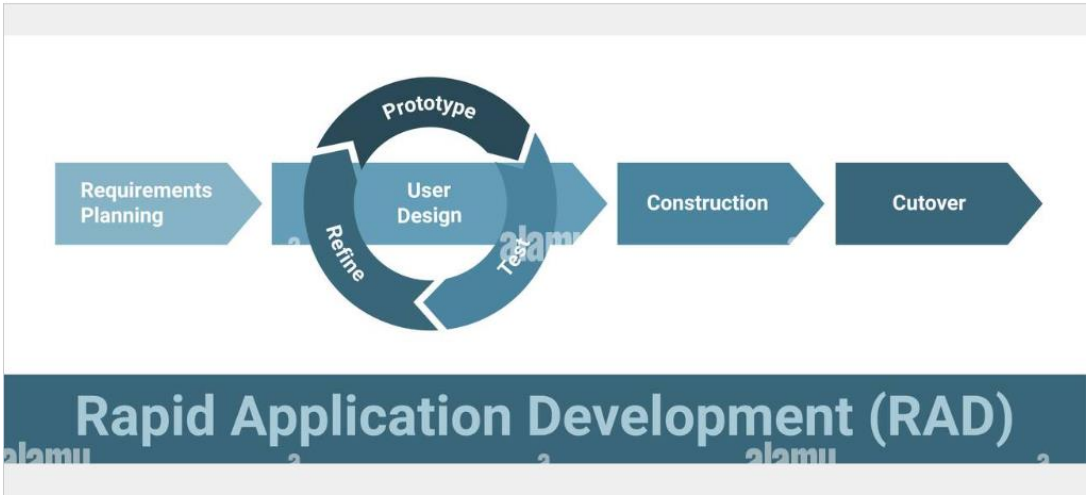


Figure 11:Les Méthodes agiles

Les méthodes agiles sont basées sur une approche itérative et incrémentale, permettant une adaptation rapide aux changements et une collaboration accrue avec les parties prenantes. Les principales méthodes incluent :

- **Scrum** : Un Framework axé sur des cycles de travail courts appelés "sprints". Scrum favorise la transparence, l'inspection et l'adaptation grâce à des réunions quotidiennes (stand-ups) et à des revues de sprint.

- **RAD (Rapid Application Développent)** : Cette méthode met l'accent sur le prototypage rapide et les itérations pour livrer des produits fonctionnels en un temps réduit.



IV.1.3 Justification de l'utilisation de Scrum

Scrum est une méthodologie agile qui se concentre sur la livraison rapide de produits fonctionnels. Elle est particulièrement adaptée aux projets complexes, comme un système de vote électronique, pour plusieurs raisons :

- **Flexibilité** : Scrum permet de s'adapter rapidement aux changements de besoins, ce qui est crucial dans un environnement dynamique où les exigences peuvent évoluer.
- **Collaboration** : La méthodologie encourage une communication constante entre les membres de l'équipe et les parties prenantes, garantissant que le produit final répond aux attentes.
- **Itérations courtes** : Les sprints (cycles de développement) permettent de livrer des fonctionnalités rapidement et d'obtenir des retours fréquents, ce qui améliore la qualité du produit.

IV.1.4 Organisation de l’équipe et des cycles du Framework

L’équipe Scrum est généralement composée de :

- **Scrum Master** : Facilite le processus Scrum et aide l’équipe à surmonter les obstacles. Il est responsable de la mise en œuvre de la méthodologie et veille à ce que l'équipe respecte les principes Scrum.
- **Product Owner** : Représente les parties prenantes et définit les priorités des fonctionnalités. Il est responsable de la gestion du backlog produit et s'assure que l'équipe travaille sur les tâches les plus importantes.
- **Développeurs** : Responsables de la mise en œuvre des fonctionnalités. Ils travaillent en collaboration pour concevoir, développer et tester le produit.

Les cycles de développement, appelés sprints, durent typiquement de 1 à 4 semaines et incluent les étapes suivantes :

- **Sprint Planning** : Planification des tâches à réaliser durant le sprint. L'équipe discute des éléments du backlog produit et sélectionne ceux qui seront traités pendant le sprint.
- **Daily Stand-ups** : Réunions quotidiennes pour discuter des progrès et des obstacles. Chaque membre de l'équipe partage ce qu'il a accompli, ce qu'il prévoit de faire et s'il rencontre des difficultés.
- **Sprint Review** : Présentation des fonctionnalités développées à la fin du sprint. L'équipe montre ce qui a été réalisé et recueille des retours des parties prenantes.
- **Rétrospective** : Analyse des points à améliorer pour les sprints suivants. L'équipe discute de ce qui a bien fonctionné et de ce qui peut être amélioré dans le processus.

IV.2 Conception du Système

IV.2.1 Conception générale

IV.2.1.1 Définition de la conception du système

La conception du système est le processus de définition de l'architecture, des composants, des modules et des interfaces d'un système. Elle vise à établir une structure qui répond aux besoins fonctionnels et non fonctionnels identifiés lors de l'analyse des besoins. Une bonne conception est essentielle pour garantir que le système est évolutif, maintenable et performant.

IV.2.1.2 Architecture générale du système

Le système repose sur une architecture 3-Tiers classique qui s'articule autour des éléments suivants :

- **Front end**

Le front end est la partie visible de l'application, celle avec laquelle les utilisateurs interagissent directement. Il est responsable de l'affichage des données et de la gestion des interactions utilisateur. Les principales caractéristiques du front end incluent :

- **Interface utilisateur (UI)** : Conçue pour être intuitive et facile à utiliser, elle permet aux électeurs de naviguer dans le système, de soumettre leurs votes et de consulter les résultats.
- **Expérience utilisateur (UX)** : L'accent est mis sur la création d'une expérience fluide et agréable pour les utilisateurs, en minimisant les frictions et en facilitant l'accès aux fonctionnalités.
- **Technologies utilisées** : Le front end est généralement développé avec des bibliothèques et des Framework modernes, tels que React, qui permettent de créer des interfaces dynamiques et réactives.

- **Back end**

Le back end est la partie du système qui gère le logique métier, l'authentification, et les interactions avec la base de données. Il est responsable du traitement des requêtes des utilisateurs, de la gestion des données et de la communication avec le front end. Les principales caractéristiques du back end incluent :

- **Logique métier** : Le back end contient la logique qui détermine comment les données sont traitées et comment les différentes fonctionnalités de l'application interagissent.
- **Gestion des utilisateurs** : Il gère l'authentification des électeurs, la création de comptes, et la gestion des rôles et des permissions.
- **Technologies utilisées** : Le back-end est souvent développé avec des Framework robustes comme Django, qui facilite le développement rapide d'applications web et offre des fonctionnalités intégrées pour la gestion des bases de données et des API.

- **Blockchain**

La blockchain est une technologie décentralisée qui permet d'enregistrer des transactions de manière sécurisée et immuable. Dans le contexte d'un système de vote électronique, la blockchain joue un rôle crucial en garantissant l'intégrité et la transparence des votes. Les principales caractéristiques de la blockchain incluent :

- **Immutabilité** : Une fois qu'une transaction est enregistrée sur la blockchain, elle ne peut pas être modifiée ou supprimée, ce qui garantit que les votes ne peuvent pas être altérés.
- **Transparence** : Les transactions sur la blockchain sont visibles par tous les participants, ce qui renforce la confiance dans le système.
- **Contrats intelligents** : Des règles et des logiques peuvent être codées dans des contrats intelligents, qui s'exécutent automatiquement lorsque certaines conditions sont remplies. Cela permet d'automatiser des processus tels que l'enregistrement des votes et le calcul des résultats.

IV.2.3 Conception détaillée

IV.2.3.1 **Diagramme des classes**

Le diagramme des classes décrit les différentes entités du système et leurs relations. Les principales classes incluent :

- **Utilisateur** : Gère les informations relatives aux utilisateurs du système (administrateurs, électeurs). Elle inclut des attributs comme le nom, l’adresse email, et

un identifiant unique. Les méthodes associées permettent de gérer l'authentification et les permissions.

- **Vote** : Représente un vote enregistré dans la blockchain. Les attributs incluent l'identifiant du vote, l'identifiant de l'électeur, et une référence au candidat choisi. Les méthodes associées permettent de valider et d'enregistrer les votes.
- **Candidat** : Contient les détails sur les candidats participants à une élection. Les attributs incluent le nom, l'affiliation politique, et une description du programme.
- **Election** : Modélise une élection et ses paramètres (dates, candidats, type de scrutin). Les méthodes permettent de créer, modifier, ou clôturer une élection.
- **SessionVerification** : Gère les étapes d'authentification et de validation des électeurs. Cette classe inclut des méthodes pour gérer les tokens d'authentification et les vérifications d'identité.

Chaque classe est associée à des méthodes et des attributs définissant leurs comportements et propriétés respectifs. Le diagramme des classes permet ainsi de structurer et de formaliser la logique du système tout en identifiant les dépendances entre les entités.

IV.2.3.1.1 **Description des packages et de leur organisation**

L'organisation du code source est structurée en plusieurs packages pour assurer une bonne lisibilité et une maintenabilité optimale :

- **Front end** : Contient les fichiers de l'interface utilisateur, incluant les composants, les feuilles de style, et les scripts. Ce package est divisé en modules tels que « components » (pour les éléments réutilisables), « pages » (pour les différentes vues de l'application), et « services » (pour les appels API).
- **Back end** : Inclut les modules Django pour la gestion des vues, des modèles, et des contrôleurs. Les packages suivants y sont présents
    - **models** : Définition des entités et leur mappage avec la base de données.
    - **views** : Logique métier et gestion des réponses aux requêtes.
    - **controllers** : Contrôle des flux entre les vues et les modèles.
    - **serializers** : Gestion de la sérialisation et de la désérialisation des données pour les API.
- **Blockchain** : Comprend les fichiers Solidity pour les smart contracts, les scripts de déploiement, et les intégrations avec Web3.py pour la communication avec la

blockchain. Ce package inclut également des tests unitaires pour vérifier la validité des contrats intelligents.

IV.3 Mise en œuvre technique

IV.3.1 Outils et technologies du front end

Le front end est la partie visible de l'application, celle avec laquelle les utilisateurs interagissent directement. Il est responsable de l'affichage des données et de la gestion des interactions utilisateur. Les principales caractéristiques du front-end incluent :

IV.3.1.1.1 Interface utilisateur (UI)

- **Définition** : L'interface utilisateur (UI) est la couche de l'application qui permet aux utilisateurs d'interagir avec le système. Elle comprend tous les éléments visuels, tels que les boutons, les formulaires, les menus et les tableaux, qui composent l'application.
- **Conception intuitive** : L'UI est conçue pour être intuitive et facile à utiliser, permettant aux électeurs de naviguer dans le système sans confusion. Une bonne conception UI utilise des conventions de design reconnues, des couleurs cohérentes et une typographie lisible pour guider l'utilisateur.
- **Accessibilité** : L'UI doit également être accessible à tous les utilisateurs, y compris ceux ayant des handicaps. Cela implique l'utilisation de balises HTML appropriées, de contrastes de couleurs suffisants et de la navigation au clavier.

IV.3.1.1.2 Expérience utilisateur (UX)

- **Définition** : L'expérience utilisateur (UX) englobe tous les aspects de l'interaction d'un utilisateur avec l'application. Cela inclut la facilité d'utilisation, la satisfaction de l'utilisateur et l'efficacité de l'application.
- **Création d'une expérience fluide** : L'accent est mis sur la création d'une expérience fluide et agréable pour les utilisateurs, en minimisant les frictions et en facilitant l'accès aux fonctionnalités. Cela peut inclure des animations douces, des transitions fluides entre les pages et des messages d'erreur clairs.

- **Tests utilisateurs** : Pour améliorer l'UX, des tests utilisateurs peuvent être réalisés pour recueillir des retours sur la navigation et l'interaction avec l'application. Ces retours peuvent ensuite être utilisés pour apporter des améliorations.

IV.3.2 Technologies utilisées

IV.3.2.1.1 Outils et technologies du front end

Le front end est généralement développé avec des bibliothèques et des Framework modernes, tels que **React**, qui permettent de créer des interfaces dynamiques et réactives.

- **React :**
  - **Définition** : React est une bibliothèque JavaScript développée par Facebook pour construire des interfaces utilisateur. Elle permet de créer des applications web dynamiques en utilisant des composants réutilisables.
  - **Composants** : Les composants React encapsulent leur propre logique et état, ce qui facilite la gestion de l'interface utilisateur. Cela permet également de diviser l'application en parties plus petites et plus gérables.
  - **Virtual DOM** : React utilise un DOM virtuel pour optimiser les mises à jour de l'interface utilisateur, ce qui améliore les performances de l'application en réduisant le nombre de manipulations directes du DOM.

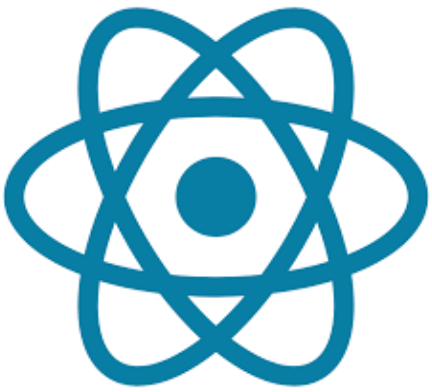


Figure 13 : Logo React

- **HTML/CSS**

- **HTML (HyperText Markup Language)** : Utilisé pour structurer le contenu de l'interface utilisateur. HTML fournit la structure de base de l'application, y compris les éléments tels que les titres, les paragraphes, les images et les formulaires.

Figure 14:Logo HTML5

- **CSS (Cascading Style Sheets)** : Utilisé pour styliser l'interface utilisateur. CSS permet de contrôler l'apparence visuelle de l'application, y compris les couleurs, les polices, les marges et les mises en page. L'utilisation de préprocesseurs CSS comme SASS ou LESS peut également faciliter la gestion des styles.

Figure 15 : Logo CSS3

- **JavaScript**

- **Définition** : JavaScript est un langage de programmation dynamique et orienté objet qui permet d'ajouter de l'interactivité et de la logique à l'interface utilisateur. Il est essentiel pour le développement web moderne et est largement utilisé pour créer des applications web réactives.
- **Fonctionnalités** : JavaScript permet de manipuler le DOM, de gérer les événements utilisateur, et d'effectuer des opérations asynchrones, ce qui est crucial pour une expérience utilisateur fluide. Il est également utilisé pour intégrer des bibliothèques et des Framework, comme React.
- **Axios** :
  - **Définition** : Axios est une bibliothèque JavaScript pour effectuer des requêtes HTTP. Elle est utilisée pour communiquer avec le back-end via des API REST.
  - **Fonctionnalités** : Axios simplifie l'envoi de requêtes asynchrones, la gestion des réponses et le traitement des erreurs. Il prend également en charge les promesses, ce qui permet d'écrire un code plus propre et plus lisible.
  - **Interception des requêtes et des réponses** : Axios permet d'intercepter les requêtes et les réponses, ce qui peut être utile pour ajouter des en-têtes d'authentification ou pour gérer des erreurs globalement.

Figure 16 : Logo JavaScript

IV.3.2.1.2 Outils et technologies du back end

- **Django**

- **Définition** : Django est un Framework web open-source écrit en Python, conçu pour faciliter le développement d'applications web robustes et sécurisées.
- **Caractéristiques**
    - **Architecture MVC** : Django suit le modèle Modèle-Vue-Contrôleur, ce qui permet de séparer le logique métier, l'interface utilisateur et le contrôle des flux de données.
    - **ORM intégré** : Django inclut un Object-Relational Mapping (ORM) qui simplifie l'interaction avec les bases de données, permettant de manipuler les données sous forme d'objets Python.
    - **Sécurité** : Django offre des fonctionnalités de sécurité intégrées pour protéger les applications contre les menaces courantes. Parmi ces fonctionnalités, on trouve :
        1. **Protection contre les attaques CSRF (Cross-Site Request Forgery)** : Django inclut un middleware qui vérifie les requêtes pour s'assurer qu'elles proviennent d'utilisateurs authentifiés.
        2. **Protection contre les attaques XSS (Cross-Site Scripting)** : Les templates Django échappent automatiquement les données, empêchant l'injection de scripts malveillants.
        3. **Protection contre les injections SQL** : L'ORM de Django utilise des requêtes paramétrées, ce qui réduit le risque d'injections SQL.

- **Django REST Framework (DRF)**

- **Définition**

Django REST Framework (DRF) est une puissante extension de Django, un Framework web populaire en Python, qui facilite la création d'API RESTful. REST (Representational State Transfer) est un style architectural qui permet de

concevoir des services web qui sont simples, évolutifs et interopérables. DRF fournit des outils et des abstractions qui simplifient le développement d'API, rendant le processus plus rapide et plus efficace.

- **Caractéristiques**
    - **Sérialisation :** DRF utilise des sérialiser pour définir comment les données doivent être converties. La sérialisation est un processus crucial dans le développement d'API, car elle permet de convertir des objets complexes, tels que des instances de modèles Django, en formats de données plus simples comme JSON ou XML. Cela facilite l'échange de données entre le front-end (par exemple, une application web ou mobile) et le back-end (le serveur).
    - **Authentification** : Il propose des mécanismes d'authentification et de permissions pour sécuriser les End points de l'API.
    - **Documentation automatique** : DRF peut générer automatiquement une documentation interactive pour les API, facilitant leur utilisation par d'autres développeurs

- **Base de données**
    - **Définition** : La base de données est un système de stockage qui permet de conserver et de gérer les données de l'application. Elle est essentielle pour la persistance des données, permettant de stocker des informations sur les utilisateurs, les votes, les candidats, et d'autres entités pertinentes.
    - **Technologie utilisée : MySQL**
        - **Définition** : MySQL est un système de gestion de base de données relationnelle (SGBDR) open-source qui utilise le langage SQL (Structured Query Language) pour gérer les données. Il est largement utilisé pour les applications web en raison de sa robustesse, de sa fiabilité et de sa performance.

Figure 17 : Logo MySQL

- **Caractéristiques** : MySQL permet de créer, lire, mettre à jour et supprimer (CRUD) des données. Il prend en charge les transactions, les jointures, et les index, ce qui permet d'optimiser les requêtes et d'assurer l'intégrité des données.
- **Utilisation avec phpMyAdmin** : phpMyAdmin est une interface web qui facilite la gestion de MySQL. Elle permet aux développeurs de créer et de gérer des bases de données, des tables, et des utilisateurs via une interface graphique conviviale, sans avoir à écrire des requêtes SQL manuellement. Cela simplifie les tâches d'administration de la base de données.

Figure 18:Logo phpMyAdmin

- **Serveur**

- **Technologie utilisée : WampServer**
  - **Définition** : WampServer est un environnement de développement web sous Windows qui permet de développer des applications web en utilisant Apache, MySQL et PHP.

Figure 19 : Logo Wampserver

- **Rôle de Wamp dans l'architecture** : Grâce à WampServer, j'ai pu utiliser simultanément les services de MySQL et phpMyAdmin. WampServer fournit une plateforme intégrée qui facilite la configuration et la gestion de ces services, permettant ainsi un développement efficace et rapide. Cela m'a permis de gérer la base de données tout en ayant accès à une interface graphique conviviale pour effectuer des opérations sur les données sans avoir à écrire de requêtes SQL manuellement.
- **Caractéristiques** : WampServer inclut un serveur web Apache, un serveur de base de données MySQL, et un interpréteur PHP. Cela permet aux développeurs de tester leurs applications localement avant de les déployer sur un serveur de production. WampServer offre également une interface graphique pour gérer les services, ce qui simplifie la configuration et le démarrage des serveurs.
- **Utilisation avec Django** : Bien que WampServer soit principalement utilisé pour des applications PHP, il peut également être configuré pour exécuter des applications Django en utilisant un serveur WSGI comme Gunicorn ou uWSGI. Cela permet aux développeurs de tester leurs applications Django dans un environnement similaire à celui de la production.

IV.3.2.1.3 Outils et technologies de la blockchain

- **Ethereum**

- **Définition :** Ethereum est une plateforme blockchain décentralisée qui permet de créer des contrats intelligents et des applications décentralisées (d'Apps). Lancée en 2015 par Vitalik Buterin et une équipe de développeurs, Ethereum a été conçue pour étendre les capacités de la blockchain au-delà des simples transactions monétaires, permettant ainsi le développement d'applications autonomes qui fonctionnent sans intermédiaire.
- **Caractéristiques**
    - **Contrats intelligents** : Les contrats intelligents sont des programmes autonomes qui s'exécutent sur la blockchain Ethereum. Ils permettent d'automatiser des processus en codant des règles dans des contrats qui s'exécutent automatiquement lorsque les conditions prédéfinies sont remplies. Par exemple, un contrat intelligent peut être utilisé pour gérer des transactions financières, des accords de location, ou même des systèmes de vote, garantissant que les termes de l'accord sont respectés sans nécessiter d'intermédiaire.
    - **Transactions sécurisées** : Les transactions sur Ethereum sont sécurisées par cryptographie, garantissant l'intégrité des données. Chaque transaction est enregistrée dans un bloc, qui est ensuite ajouté à la chaîne de blocs (blockchain). Cela rend les transactions transparentes et immuables, car une fois qu'un bloc est ajouté à la blockchain, il ne peut pas être modifié ou supprimé. De plus, Ethereum utilise un mécanisme de consensus (Proof of Work ou Proof of Stake) pour valider les transactions, ce qui renforce la sécurité du réseau.

Figure 20 : Logo Ethereum

- **Solidity**

- **Définition :** Solidity est un langage de programmation orienté objet utilisé pour écrire des contrats intelligents sur la blockchain Ethereum. Développé spécifiquement pour Ethereum, il permet aux développeurs de créer des applications décentralisées en définissant la logique métier et les règles de fonctionnement des contrats.

Figure 21 : Logo Solidity

- **Caractéristiques**

    - **Syntaxe similaire à JavaScript** : Solidity a une syntaxe qui rappelle celle de JavaScript, ce qui facilite l'apprentissage pour les développeurs déjà familiers avec ce langage. Cela permet une adoption plus rapide et une courbe d'apprentissage moins abrupte pour les nouveaux développeurs souhaitant créer des contrats intelligents.
    - **Types de données complexes** : Solidity permet de définir des structures de données complexes, essentielles pour la gestion des votes et des électeurs. Par exemple, les développeurs peuvent créer des structures personnalisées pour représenter des entités comme des électeurs, des candidats, ou des résultats d'élections. Cela permet de gérer efficacement les données et d'assurer que les contrats intelligents peuvent exécuter des logiques complexes.

- **Web3.py**

- **Définition :** Web3.py est une bibliothèque Python qui facilite l'interaction avec la blockchain Ethereum. Elle permet aux développeurs de créer des applications Python qui peuvent interagir avec des contrats intelligents, envoyer des transactions, et interroger des données sur la blockchain.

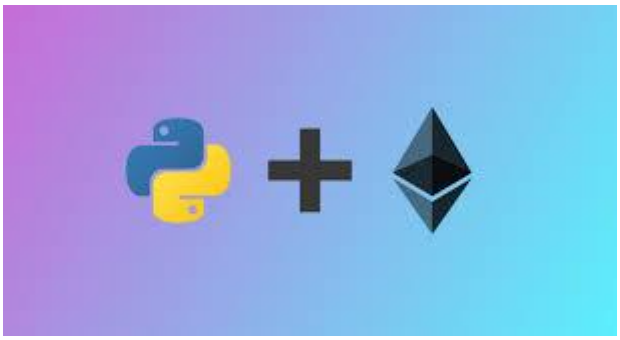


Figure 22:Illustration de Web3.py

- **Caractéristiques**

    - **Envoi de transactions** : Web3.py permet d'envoyer des transactions à la blockchain, comme l'enregistrement d'un vote ou le transfert d'Ether (la crypto monnaie d'Ethereum). Les développeurs peuvent facilement construire des transactions, les signer et les envoyer au réseau Ethereum, tout en gérant les frais de transaction (gas).
    - **Interrogation des données** : La bibliothèque facilite la récupération des données stockées sur la blockchain, comme l'historique des transactions, l'état des contrats intelligents, et les événements émis par ces contrats. Cela permet aux développeurs de créer des interfaces utilisateur dynamiques qui affichent des informations en temps réel sur l'état de la blockchain.

- **Truffle**

- **Définition :** Truffle est un Framework de développement pour Ethereum qui simplifie la gestion des contrats intelligents. Il fournit un ensemble d'outils pour écrire, tester et déployer des contrats intelligents de manière efficace, ce qui en fait un choix populaire parmi les développeurs d'applications décentralisées.

Figure 23 : Logo Truffle

- **Caractéristiques**
    - **Écriture et test de contrats** : Truffle permet aux développeurs d'écrire des contrats intelligents en Solidity et de les tester dans un environnement de développement local. Il inclut des outils pour exécuter des tests automatisés, garantissant que les contrats fonctionnent comme prévu avant leur déploiement sur le réseau principal. Cela réduit le risque d'erreurs et de vulnérabilités dans les contrats déployés.
    - **Gestion des migrations** : Truffle facilite la gestion des migrations, c'est-à-dire le processus de déploiement des contrats intelligents sur la blockchain. Les développeurs peuvent définir des scripts de migration qui décrivent comment et quand déployer les contrats, ce qui permet de gérer les mises à jour et les modifications des contrats au fil du temps. Cela est particulièrement utile dans un environnement de développement où les contrats peuvent nécessiter des ajustements fréquents. Truffle offre également une intégration avec Ganache, un simulateur de blockchain, permettant aux développeurs de tester leurs contrats dans un environnement contrôlé avant de les déployer sur le réseau principal.

- **Alchemy**

- **Introduction à Alchemy :** Alchemy est une plateforme de développement blockchain puissante qui fournit des outils et des services pour interagir avec les réseaux

décentralisés. Elle est reconnue pour sa facilité d'utilisation, ses performances optimisées et ses capacités avancées, telles que la gestion des clés et des signatures. Dans le cadre de votre projet, Alchemy joue un rôle crucial pour sécuriser les opérations cryptographiques.

Figure 24 : Logo Alchemy

- **Rôle d'Alchemy dans le projet**

- **Gestion des clés cryptographiques**
  - Création et stockage des clés : Alchemy permet de générer des paires de clés publiques et privées de manière sécurisée. Ces clés sont essentielles pour garantir l'intégrité et la confidentialité des transactions.
  - Sécurisation des clés : Alchemy offre des mécanismes avancés pour protéger les clés privées contre les accès non autorisés, ce qui est vital pour prévenir les attaques potentielles.
- **Signatures numériques**
  - Validation des transactions : Les signatures numériques assurent que les données envoyées (comme les votes dans votre système) proviennent bien d'une source authentique.
  - Utilisation des clés privées : Alchemy permet de signer des messages ou des transactions en utilisant la clé privée, sans exposer celle-ci à des tiers, ce qui renforce la sécurité.
- **Interaction avec la blockchain**
  - Soumission des transactions : Une fois signées, les transactions sont soumises à la blockchain via les API Alchemy.

- Vérification des signatures : Les clés publiques associées permettent de vérifier que les signatures correspondent bien aux données soumises.

- **Avantages d'Alchemy dans votre projet**

- **Fiabilité et performances**

  Les API d'Alchemy offrent des temps de réponse rapides et une haute disponibilité, garantissant un fonctionnement fluide du système, même sous une charge élevée.

- **Simplicité d'intégration**

  Les SDK d'Alchemy simplifient les interactions avec les réseaux blockchain, rendant le développement plus rapide et plus accessible.

- **Sécurité renforcée**

  Les clés privées ne quittent jamais l'environnement sécurisé d'Alchemy, minimisant ainsi les risques de compromission.

- **Suivi et analyse**

  Alchemy fournit des outils de monitoring et de diagnostic, permettant de suivre les performances des transactions et de résoudre rapidement les éventuels problèmes.

- **Fonctionnalités clés utilisées dans le projet**

- **Gestion des clés**
  - Génération des clés avec des algorithmes robustes comme ECDSA.
  - Stockage sécurisé des clés privées pour empêcher les accès non autorisés.

- **Signatures**
  - Signature des transactions avant soumission sur la blockchain.
  - Vérification des signatures pour valider l'authenticité des données.

- **APIs Blockchain**
  - Récupération des données nécessaires à partir de la blockchain.
  - Interaction avec les contrats intelligents pour enregistrer ou valider des informations.

IV.4 Mise en place du ZKP

Dans le cadre de ce projet, nous avons mis en place un système de preuve à divulgation nulle de connaissance (Zero-Knowledge Proof ou ZKP) en utilisant **Circom** et **SnarkJS**. L'objectif principal est de garantir la confidentialité des informations sensibles tout en permettant de prouver la validité d'une action, comme un vote, sans révéler les détails. Ce document détaille chaque étape de l'implémentation, ainsi que le rôle de chaque fichier généré.

IV.4.1 Architecture et fonctionnement

Les ZKP reposent sur trois propriétés fondamentales :

1. **Complétude** : Si l'assertion est vraie, un vérificateur honnête sera convaincu par le prouveur. Cela signifie que si le prouveur possède une information valide, il peut prouver cette validité au vérificateur sans révéler l'information elle-même.
2. **Soundness** : Si l'assertion est fausse, aucun prouveur malveillant ne pourra convaincre le vérificateur de sa véracité, sauf avec une probabilité négligeable. Cela garantit que les preuves fournies par le prouveur sont fiables et qu'un prouveur malveillant ne peut pas tromper le vérificateur.
3. **Zero-knowledge** : Si l'assertion est vraie, le vérificateur n'apprend rien d'autre que le fait que l'assertion est vraie. Cela signifie que le vérificateur ne peut pas déduire d'informations supplémentaires sur les données privées du prouveur.

IV.4.2 Circom

Circom (Circuit Compiler) est un langage et un compilateur spécialisé dans la création de circuits arithmétiques. Ces circuits définissent les relations mathématiques qui seront prouvées dans le cadre des ZKP.

- **Langage** : Circom est proche de JavaScript, mais conçu spécifiquement pour créer des circuits. Il permet d'écrire des contraintes arithmétiques de manière intuitive, facilitant la modélisation des problèmes complexes.

- **Rôle** : Écrire des circuits pour modéliser les contraintes de vérification. Par exemple, dans le cadre d'un vote, le circuit peut vérifier que le vote d'un électeur est valide sans révéler le contenu du vote.
- **Fichiers clés** : Les fichiers `.circom` contiennent la définition des circuits. Chaque fichier peut inclure des entrées, des sorties et des contraintes qui doivent être respectées.

IV.4.3 SnarkJS

SnarkJS est une bibliothèque JavaScript utilisée pour :

- **Générer des preuves** basées sur des circuits créés avec Circom. SnarkJS prend en charge la création de preuves cryptographiques qui peuvent être utilisées pour prouver des assertions sans divulguer d'informations sensibles.
- **Vérifier ces preuves**. Une fois qu'une preuve est générée, SnarkJS permet de la valider en utilisant les données publiques et les clés de vérification.
- **Effectuer des opérations cryptographiques** associées aux ZKP, telles que la génération de clés et la manipulation de données.

IV.5 La génération de preuves ZKP implique plusieurs étapes

1. **Définition du circuit** : Création d'un circuit arithmétique pour modéliser la logique à prouver. Cela inclut la définition des entrées, des sorties et des contraintes qui doivent être respectées.
2. **Compilation** : Conversion du circuit en formats exploitables (fichiers R1CS, WASM, etc.). Cette étape transforme le circuit en une représentation qui peut être utilisée pour générer des preuves.
3. **Génération des témoins** : Calcul des valeurs privées nécessaires pour générer une preuve. Les témoins sont des valeurs intermédiaires qui sont calculées en fonction des entrées fournies au circuit.
4. **Setup** : Génération des clés cryptographiques pour créer et vérifier des preuves. Cette étape est cruciale pour garantir la sécurité et l'intégrité du système.
5. **Génération de la preuve** : Production d'un fichier contenant la preuve. Cette preuve peut être utilisée pour prouver qu'une assertion est vraie sans révéler les données sous-jacentes.

6. **Vérification** : Validation de la preuve à l'aide des données publiques et de la clé de vérification. Cela permet de s'assurer que la preuve est valide et que les données privées n'ont pas été compromises.

Chaque étape produit des fichiers spécifiques. Voici une description détaillée de chaque fichier et son utilité.

IV.6 Détails des fichiers produits

IV.6.1 Circuit (vote.circom)

Le fichier `vote.circom` contient la définition du circuit arithmétique. Ce fichier décrit les relations mathématiques entre les variables d'entrée et de sortie. Par exemple :

- **Entrées :**

- **`secret` : Une valeur privée représentant un élément confidentiel (comme un vote). Cette valeur est essentielle pour garantir que le vote reste anonyme.**
- **`voter_id` : Identifiant unique de l'électeur. Cela permet d'assurer que chaque vote est associé à un électeur spécifique sans révéler son identité.**
- **`candidate_id` : Identifiant du candidat choisi. Ce champ est utilisé pour vérifier que le vote est dirigé vers un candidat valide.**

- **Contraintes** : Le circuit vérifie que l'ID du candidat est valide et que le vote correspond à une logique arithmétique élégante. Par exemple, il peut inclure des vérifications pour s'assurer que le `candidate_id` est dans une liste prédéfinie d'IDs valides, garantissant ainsi que seuls les candidats autorisés peuvent recevoir des votes.

Le circuit peut également inclure des contraintes supplémentaires pour gérer des scénarios spécifiques, comme la vérification que chaque électeur ne vote qu'une seule fois. Cela peut être réalisé en intégrant des mécanismes de contrôle d'unicité dans le circuit.

```
// Définition du circuit VoteCircuit
template VoteCircuit() {
    // Entrées
    signal input voter_id;          // ID du vote (valeur binaire)
    signal input secret;            // Secret (par exemple, un identifiant unique de l'électeur)
    signal input candidate_id;      // ID du candidat choisi
    signal output vote_valid;       // Indicateur de validité du vote (0 ou 1)
    signal output vote_hash;        // Hash du vote

    // Calcul du hash du vote (pour l'exemple, combinaison de l'électeur, du secret et du candi
    // Le hash est simplement une somme des valeurs, mais dans un cas réel, cela serait un hash
    vote_hash <== voter_id * secret + candidate_id;
```

Figure 25 : Fichier du circuit .circom

Apres compilation du fichier.circom on a ces fichiers : .wasm, .wtns, .sym

```
# Étape 1: Compiler le circuit Circom
compile_command = [
    "C:/Users/lyche/Downloads/circom-windows-amd64.exe",
    os.path.join(current_directory, "vote.circom"),
    "--r1cs",
    "--wasm",
    "--sym"
]
result = subprocess.run(compile_command, check=True, stdout=subprocess.PIPE, stderr=subproc
print("Compilation Circom réussie:", result.stdout.decode())
```

Figure 26 : Compilation du circuit

IV.6.2 Fichier R1CS (vote.r1cs)

Ce fichier est généré à partir du circuit Circom. Il contient une représentation des contraintes arithmétiques sous forme de système d'équations linéaires. Le fichier R1CS est utilisé pour vérifier que les entrées respectent les contraintes définies.

- **Rôle** : Le R1CS est une représentation intermédiaire utilisée pour la génération de témoins et de preuves. Il permet de formaliser les relations entre les variables d'entrée et de sortie, facilitant ainsi la vérification des assertions.

IV.6.3 Fichier WASM (vote.wasm)

Ce fichier WebAssembly contient le code exécutable pour calculer les témoins à partir des entrées fournies.

- **Rôle** : Permet d'effectuer des calculs efficaces des contraintes définies dans le circuit. Le fichier WASM est optimisé pour être exécuté dans un environnement JavaScript, ce qui le rend compatible avec les applications web.

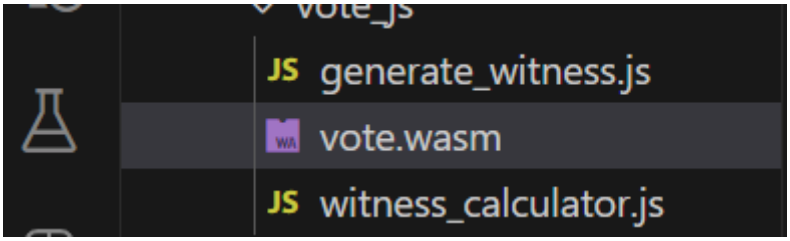


Figure 27 : Fichier .wasm

IV.6.4 Témoins (witness_<vote_id>.wtns)

Le fichier de témoins contient les valeurs intermédiaires calculées lors de l'exécution du circuit. Ces valeurs servent de preuve privée pour générer une preuve cryptographique.

- **Rôle** : Fournit les preuves mathématiques que les contraintes du circuit sont respectées sans révéler les entrées privées. Les témoins sont essentiels pour la génération de la preuve, car ils contiennent les informations nécessaires pour prouver la validité de l'assertion.

```
witness_command = [
    'cmd', '/c', "snarkjs", "wtns", "calculate",
    os.path.join(vote_js, "vote.wasm"),
    input_file_path,
    witness_path
]
result = subprocess.run(witness_command, check=True, stdout=subprocess.PIPE, stderr=su
print("Témoins générés:", result.stdout.decode())
```

Figure 28:Fichier du Témoin

IV.6.5 Clé Powers of Tau (powersoftau_phase2.ptau)

Ce fichier précalculé est utilisé pour générer les clés cryptographiques. Il garantit la sécurité du système en générant des valeurs aléatoires.

- **Rôle** : Assure que les clés produites sont uniques et inviolables. La clé Powers of Tau est cruciale pour le setup du système, car elle contribue à la sécurité des preuves générées.

IV.6.6 Clé de configuration (vote_final.zkey)

```
# Étape 2: Générer la clé de vérification (.zkey)
setup_command = [
    'cmd', '/c', "snarkjs",
    "groth16",
    "setup",
    os.path.join(current_directory, "vote.r1cs"),
    powersoftau_path,
    zkey_path
]
result = subprocess.run(setup_command, check=True, stdout=subprocess.PIPE, stderr=subproces
print("Clé de vérification générée :", result.stdout.decode())
```

Figure 29 : Génération de la clé prive

Le fichier ZKEY contient la clé privée utilisée pour générer les preuves. Cette clé est générée lors de la phase de setup.

- **Rôle** : Permet de créer des preuves à partir des témoins. La clé ZKEY est essentielle pour garantir que seules les preuves valides peuvent être générées.

IV.6.7 Clé de vérification (verification_key_<vote_id>.json)

```
#Exportation de la cle dans un fichier Json
cle_json = [
    'cmd', '/c', 'snarkjs', 'zkey',
    'export', 'verificationkey',
    zkey_path,
    verification_key
]
key_json = subprocess.run(cle_json, check=True, stdout=subprocess.PIPE, stderr=subprocess.P
print("Clé de vérification exportée :", key_json.stdout.decode())
```

Figure 30 : Exportation de la clé de vérification

Cette clé publique est utilisée pour vérifier la validité des preuves sans accès aux données privées.

- **Rôle** : Valide que la preuve générée correspond aux contraintes du circuit. La clé de vérification permet aux parties externes de s'assurer que la preuve est authentique sans avoir besoin de connaître les données privées.

IV.6.8 Preuve (proof_<vote_id>.json)

```
# Étape 4: Générer la preuve (.proof.json)
proof_command = [
    'cmd', '/c', "snarkjs", "groth16", "prove",
    zkey_path,
    witness_path,
    proof_path,
    public_path
]
result = subprocess.run(proof_command, check=True, stdout=subprocess.PIPE, stderr
print("Preuve générée avec succès:", result.stdout.decode())
```

Figure 31 : Génération des Preuves

Ce fichier contient la preuve cryptographique générée à partir des témoins et des clés. Il peut être partagé avec des tiers pour prouver la validité d'une action.

- **Rôle** : Fournit une preuve de conformité sans révéler les entrées privées. La preuve peut être utilisée pour démontrer qu'un vote a été effectué sans divulguer le contenu du vote.

IV.6.9 Données publiques (public_<vote_id>.json)

Ce fichier contient les données publiques associées à la preuve. Ces données sont utilisées lors de la vérification.

- **Rôle** : Permet aux vérificateurs de s'assurer que la preuve est valide sans compromettre la confidentialité. Les données publiques peuvent inclure des informations sur le vote, comme le `candidate_id`, qui sont nécessaires pour la vérification.

IV.7 Commandes principales et explications

IV.7.1 Compilation du circuit

Pour compiler le circuit, nous utilisons la commande suivante : `circom vote.circom --r1cs --wasm --sym`. Cette commande effectue plusieurs actions importantes :

- **`--r1cs`** : Cette option génère le fichier des contraintes R1CS (Rank-1 Constraint System). Ce fichier contient une représentation des contraintes arithmétiques sous forme de système d'équations linéaires. Il est essentiel pour la vérification des entrées et la génération des témoins.
- **`--wasm`** : Cette option produit le fichier WASM (WebAssembly) qui contient le code exécutable pour calculer les témoins. Le fichier WASM est optimisé pour être exécuté dans un environnement JavaScript, ce qui le rend compatible avec les applications web et permet des calculs rapides et efficaces.
- **`--sym`** : Cette option crée un fichier symbolique qui contient des informations sur les variables et les contraintes du circuit. Ce fichier est utile pour le débogage, car il permet de comprendre comment les variables sont liées et quelles contraintes sont appliquées.

```
compile_command = [
    "C:/Users/alkas/PycharmProjects/ProjetFinDAnnee/circom-windows-amd64.exe",
    os.path.join(current_directory, "vote.circom"),
    "--r1cs",
    "--wasm",
    "--sym"
]
```

Figure 32:Compilation du circuit

IV.7.2 Calcul des témoins

Une fois le circuit compilé, la prochaine étape consiste à calculer les témoins, qui sont des valeurs intermédiaires nécessaires pour générer une preuve. Pour cela, nous utilisons la commande `snarkjs wtns calculate vote.wasm input_<vote_id>.json witness_<vote_id>.wtns`.

- **`vote.wasm`** : Ce fichier exécutable contient le code qui effectue les calculs nécessaires pour générer les témoins. Il utilise les entrées fournies pour produire des valeurs intermédiaires.
- **`input_<vote_id>.json`** : Ce fichier contient les données d'entrée pour le calcul des témoins. Il inclut les valeurs privées, telles que le `secret`, `voter_id`, et `candidate_id`, qui sont nécessaires pour exécuter le circuit.
- **`witness_<vote_id>.wtns`** : Ce fichier généré contient les témoins calculés. Ces valeurs intermédiaires sont essentielles pour la génération de la preuve, car elles démontrent que les contraintes du circuit ont été respectées sans révéler les données privées.

```
witness_command = [
    'cmd', '/c', "snarkjs", "wtns", "calculate",
    os.path.join(vote_js, "vote.wasm"),
    input_file_path,
    witness_path
]
result = subprocess.run(witness_command, check=True, stdout=subprocess.PIPE, stderr=
print("Témoins générés:", result.stdout.decode())
```

Figure 33 : Génération des témoins

IV.7.3 Génération des clés

La génération des clés est une étape importante qui permet de créer les clés cryptographiques nécessaires pour générer et vérifier les preuves. La commande utilisée est `snarkjs groth16 setup vote.r1cs powersoftau_phase2.ptau vote_final.zkey`.

- **`vote.r1cs`** : Ce fichier contient les contraintes du circuit, qui sont utilisées pour générer les clés.
- **`powersoftau_phase2.ptau`** : Ce fichier pré-calculé est utilisé pour garantir la sécurité des clés générées. Il contient des valeurs aléatoires qui assurent que les clés produites sont uniques et inviolables.
- **`vote_final.zkey`** : Ce fichier contient la clé privée générée lors de la phase de setup. Cette clé est utilisée pour créer des preuves à partir des témoins.

```
setup_command = [
    'cmd', '/c', "snarkjs",
     "groth16",
     "setup",
     os.path.join(current_directory, "vote.r1cs"),
     powersoftau_path,
     zkey_path
]
result = subprocess.run(setup_command, check=True, stdout=subprocess.PIPE, stderr=
```

Figure 34: Génération des clés

IV.7.4 Exportation de la clé publique

Après avoir généré la clé privée, il est nécessaire d'exporter la clé publique pour permettre la vérification des preuves. La commande utilisée est `snarkjs zkey export verificationkey vote_final.zkey verification_key_<vote_id>.json`.

- **`vote_final.zkey`** : Ce fichier contient la clé privée qui a été générée précédemment.
- **`verification_key_<vote_id>.json`** : Ce fichier exporte la clé publique, qui sera utilisée pour vérifier la validité des preuves sans avoir accès aux données privées.

```
cle_json = [

    'cmd', '/c', 'snarkjs', 'zkey',
    'export', 'verificationkey',
    zkey_path,
    verification_key
]
key_json = subprocess.run(cle_json, check=True, stdout=subprocess.PIPE, stderr=subpr
print("Clé de vérification exportée :", key_json.stdout.decode())
```

Figure 35 : Exportation de la PubKey

IV.7.5 Génération de la preuve

Pour générer la preuve, nous utilisons la commande `snarkjs groth16 prove vote_final.zkey witness_<vote_id>.wtns proof_<vote_id>.json public_<vote_id>.json`.

- **`witness_<vote_id>.wtns`** : Ce fichier contient les témoins calculés précédemment.
- **`proof_<vote_id>.json`** : Ce fichier contient la preuve générée, qui peut être partagée avec des tiers pour prouver la validité d'une action sans révéler les données privées.
- **`public_<vote_id>.json`** : Ce fichier contient les données publiques associées à la preuve, qui seront utilisées lors de la vérification.

```
proof_command = [
    'cmd', '/c', "snarkjs", "groth16", "prove",
    zkey_path,
    witness_path,
    proof_path,
    public_path
]
result = subprocess.run(proof_command, check=True, stdout=subprocess.PIPE, stderr=sul
print("Preuve générée avec succès:", result.stdout.decode())
```

Figure 36 : Génération de la preuve

IV.7.6 Vérification de la preuve

Enfin, pour vérifier la preuve, nous utilisons la commande `snarkjs groth16 verify verification_key_<vote_id>.json public_<vote_id>.json proof_<vote_id >.json`.

- **`verification_key_<vote_id>.json`** : Ce fichier contient la clé publique qui a été exportée précédemment et qui est utilisée pour la vérification.
- **`public_<vote_id>.json`** : Ce fichier contient les données publiques associées à la preuve, qui sont nécessaires pour effectuer la vérification.
- **`proof_<vote_id>.json`** : Ce fichier contient la preuve générée, qui est comparée aux données publiques et à la clé de vérification pour s'assurer que la preuve est valide.

IV.8 Étapes d'intégration dans le projet

IV.8.1 Gestion des fichiers

Les fichiers générés au cours du processus sont organisés dans des répertoires spécifiques pour faciliter leur gestion :

- **Circuits** : Ce répertoire contient les fichiers `.circom` ainsi que leurs sorties compilées, ce qui permet de garder une trace des différentes versions du circuit.
- **Entrées** : Ce répertoire stocke les fichiers `input_<vote_id>.json` pour chaque vote, assurant que les données d'entrée sont facilement accessibles et bien organisées.
- **Témoins** : Ce répertoire conserve les fichiers `witness_<vote_id>.wtns`, qui contiennent les témoins calculés pour chaque vote.
- **Preuves** : Ce répertoire inclut les fichiers `proof_<vote_id>.json` et `public_<vote_id>.json`, qui sont essentiels pour la vérification des preuves.

IV.8.2 Automatisation

Pour garantir une exécution fluide et sans intervention manuelle, les commandes sont automatisées dans le projet à l'aide de la bibliothèque `subprocess` en Python. Cela permet d'exécuter les commandes shell directement depuis le code Python, facilitant ainsi l'intégration des différentes étapes du processus.

IV.8.3 Gestion des erreurs

Chaque étape du processus inclut des vérifications pour s'assurer que les fichiers nécessaires existent et que les commandes s'exécutent correctement. Les exceptions Python sont utilisées pour gérer les erreurs liées à l'exécution ou à l'absence de fichiers, ce qui permet de fournir des messages d'erreur clairs et de prendre des mesures appropriées en cas de problème.

IV.9 Sécurité et confidentialité

IV.9.1 Protection des clés

Il est crucial de protéger les fichiers `.zkey`, car ils contiennent des informations sensibles. Ces fichiers doivent être générés de manière sécurisée et ne doivent pas être exposés publiquement.

Les clés publiques peuvent être partagées librement, mais les clés privées doivent rester confidentielles pour garantir la sécurité du système.

IV.9.2 Isolation des témoins

Les fichiers de témoins contiennent des données sensibles et doivent être supprimés après usage pour éviter toute fuite d'information. Cela garantit que les informations privées ne sont pas accessibles après que les preuves ont été générées.

IV.9.3 Validation des preuves

La vérification repose sur des clés publiques et des données publiques, garantissant ainsi que les données privées ne sont jamais exposées. Ce mécanisme de validation permet de s'assurer que les preuves sont valides sans compromettre la confidentialité des informations sensibles.

PRESENTATION DU FONCTIONNEMENT DU PROTOTYPE :

V.1 GESTION DES CLES AVEC UTILISATION DE PyCryptodome

V.1.1 Définition de PyCryptodome

PyCryptodome est une bibliothèque Python qui fournit des primitives cryptographiques robustes et faciles à utiliser. Elle est conçue pour remplacer l'ancienne bibliothèque PyCrypto, offrant des fonctionnalités améliorées et une meilleure sécurité. PyCryptodome permet de gérer des clés cryptographiques, de chiffrer et de déchiffrer des données, ainsi que de signer et de vérifier des messages

V.1.2 Processus de Génération de Clés

1. **Création de la Clé** : La génération d'une paire de clés RSA se fait en utilisant la méthode `RSA.generate()`. Cette méthode prend en paramètre la taille de la clé, généralement de 2048 bits, qui est considérée comme un bon compromis entre sécurité et performance.
2. **Récupération de la Clé Publique** : Une fois la clé privée générée, la clé publique peut être extraite à l'aide de la méthode `publickey()`. La clé publique est utilisée pour chiffrer les données ou vérifier les signatures, tandis que la clé privée est utilisée pour déchiffrer les données ou signer les messages.
3. **Sérialisation des Clés** : Les clés peuvent être sérialisées au format PEM, un format standard pour le stockage des clés cryptographiques. La méthode `export_key(format='PEM')` permet de convertir la clé en une chaîne de caractères au format PEM, facilitant ainsi son stockage dans une base de données ou son transfert.
4. **Encodage en Base64** : Pour garantir que la clé publique peut être stockée de manière sécurisée dans une base de données, elle est souvent encodée en base64. Cela permet de représenter les données binaires sous forme de chaîne de caractères, ce qui est plus facile à manipuler dans des systèmes de gestion de bases de données.

V.2 GESTION DES CLES AVEC UTILISATION DE AWS

Amazon Web Services (AWS) est une plateforme de cloud computing proposée par Amazon, offrant plus de 200 services variés, tels que le stockage, le calcul et l'analyse de données. Elle permet aux entreprises et aux particuliers de déployer des solutions flexibles et évolutives sur Internet.

- **Définition** : AWS est la plateforme de cloud computing la plus complète et la plus adoptée au monde, permettant aux utilisateurs d'accéder à des services variés via Internet.
- **Services Offerts** : AWS propose plus de 200 services, incluant :
    - **Calcul** : Services comme Amazon EC2 pour le déploiement de serveurs virtuels.
    - **Stockage** : Solutions telles qu'Amazon S3 pour le stockage d'objets.
    - **Bases de données** : Options comme Amazon RDS pour la gestion de bases de données relationnelles.
    - **Analyse** : Outils pour le traitement et l'analyse de données massives.

V.2.1 Avantages d'AWS

- **Évolutivité** : Les utilisateurs peuvent facilement ajuster leurs ressources en fonction de la demande, ce qui permet de gérer efficacement les pics d'activité.
- **Coût** : AWS fonctionne sur un modèle de tarification à l'utilisation, permettant aux utilisateurs de ne payer que pour les ressources qu'ils consomment.
- **Sécurité** : AWS offre des outils et des services robustes pour garantir la sécurité des données, y compris des options de cryptage et des certifications de conformité.

V.2.2 Utilisation d'AWS

- **Pour les entreprises** : AWS aide les entreprises à réduire leurs coûts d'infrastructure, à améliorer leur agilité et à innover plus rapidement.
- **Pour les développeurs** : Les développeurs peuvent utiliser AWS pour créer, tester et déployer des applications sans avoir à gérer l'infrastructure sous-jacente.
- **Pour les gouvernements et les organisations à but non lucratif** : AWS permet de moderniser les systèmes et d'améliorer les services offerts aux citoyens.

V.2.3 Gestion des clés avec AWS KMS

AWS propose des solutions robustes pour la gestion des clés des électeurs dans un système de vote électronique, notamment via AWS Key Management Service (KMS). Ce service permet de créer, gérer et contrôler l'accès aux clés de cryptage, garantissant ainsi la sécurité et la confidentialité des données des électeurs.

- **Contrôle Centralisé** : AWS KMS offre un contrôle centralisé sur le cycle de vie des clés, permettant de créer et de gérer des clés de chiffrement de manière efficace.
- **Intégration avec d'autres Services AWS** : KMS s'intègre facilement avec d'autres services AWS, facilitant le chiffrement des données stockées et le contrôle d'accès.
- **Audit et Conformité** : Grâce à l'intégration avec AWS CloudTrail, chaque utilisation des clés est enregistrée, permettant un suivi et une vérification des accès.
- **Durabilité et Disponibilité** : AWS KMS assure la durabilité des clés avec plusieurs copies stockées dans des systèmes conçus pour garantir une disponibilité élevée.

V.2.4 Avantages de l'Utilisation d'AWS KMS

- **Sécurité Renforcée** : Les clés ne sont jamais exposées en texte clair, et toutes les opérations cryptographiques se déroulent dans des modules de sécurité matériels (HSM) validés.
- **Scalabilité** : KMS peut gérer des milliers de clés, s'adaptant à l'évolution des besoins de notre projet de vote électronique.
- **Clés Multi-Régions** : Possibilité de créer des clés interopérables dans plusieurs régions, ce qui est utile pour des architectures à haute disponibilité.

V.2.5 Cas d'Utilisation

- **Chiffrement des Données des Électeurs** : Les informations sensibles des électeurs peuvent être chiffrées à l'aide de clés gérées par AWS KMS, garantissant leur protection.
- **Signature Numérique** : Les développeurs peuvent intégrer facilement des fonctionnalités de signature numérique dans leurs applications, renforçant l'intégrité des données.
- **Gestion des Accès** : KMS permet de définir des politiques d'accès précises, assurant que seules les personnes autorisées peuvent utiliser ou gérer les clés.

En utilisant AWS KMS pour la gestion des clés des électeurs, nous pouvons garantir un niveau élevé de sécurité et de conformité, essentiel pour un système de vote électronique fiable.

Figure 37 : Logo Amazon Web Services

Dans notre projet, on a utilisé le AWS pour la génération et la gestion des clés des électeurs.

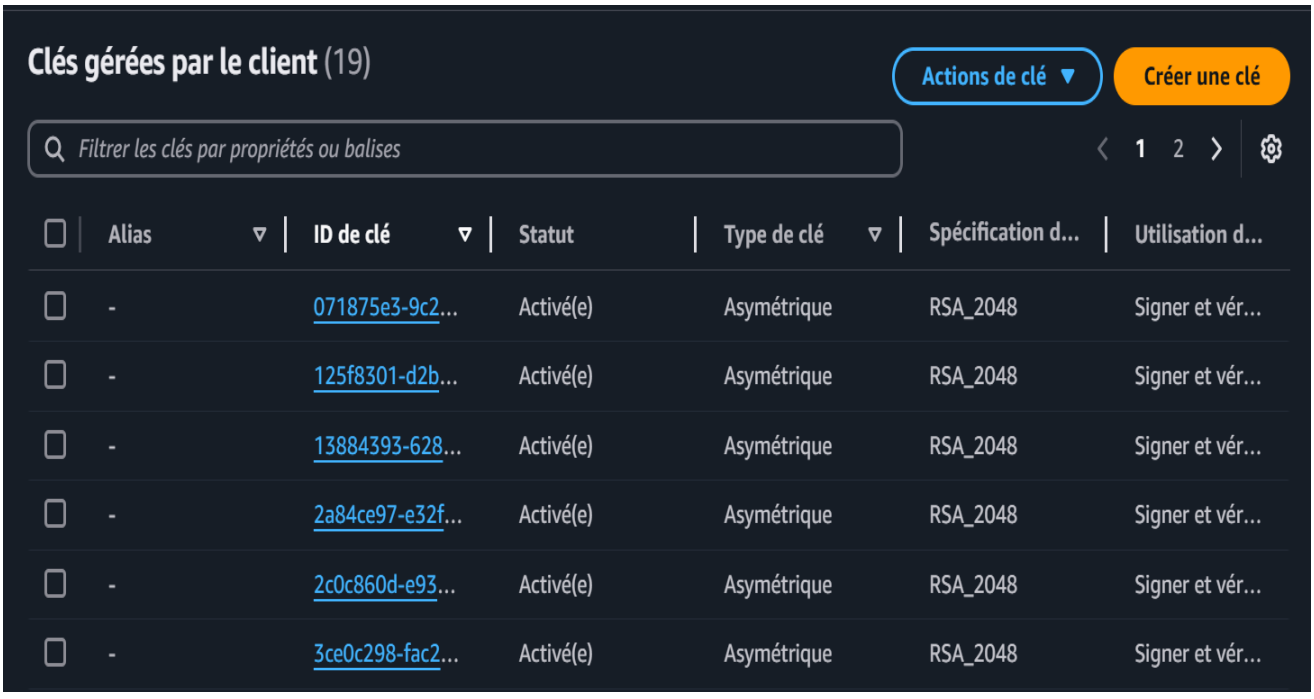


Figure 38 : Interface de gestion de clés AWS

Partie plus détaillée où on peut voir les clés publiques mais pas les clés privées :

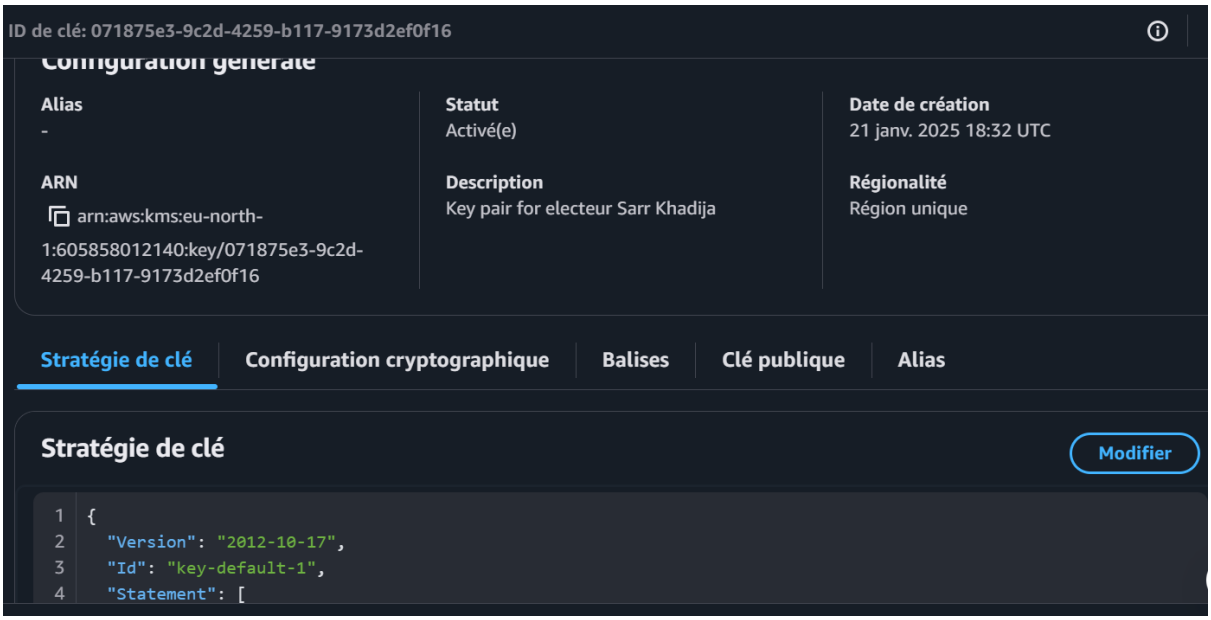


Figure 39 : Interface détaillée pour les clés

CONCLUSION

Le développement d'un système de vote électronique sécurisé basé sur l'intégration des preuves à divulgation nulle de connaissance (ZKP) représente une avancée significative dans la quête d'un processus électoral moderne, transparent et digne de confiance. À travers ce mémoire, nous avons exploré les défis techniques, les exigences fonctionnelles et les solutions innovantes nécessaires pour concevoir une plateforme capable de répondre aux besoins cruciaux de confidentialité, de vérifiabilité et de résilience face aux manipulations potentielles.

Dans un premier temps, nous avons étudié l'état de l'art des systèmes de vote électronique, tout en mettant en lumière leurs forces et leurs limites, notamment dans des environnements contrôlés et non contrôlés. Ce diagnostic a permis de poser les bases d'une problématique pertinente : comment garantir simultanément l'anonymat des électeurs et la transparence du processus électoral, sans compromettre l'intégrité des votes ?

La spécification et l'analyse des besoins ont permis d'identifier les acteurs clés du système, leurs rôles ainsi que les fonctionnalités fondamentales nécessaires à la mise en œuvre d'une solution pratique. L'utilisation des ZKP s'est avérée être un levier crucial pour assurer la confidentialité des votes tout en offrant une vérifiabilité indépendante, consolidant ainsi la confiance des électeurs et des observateurs.

Au cours de la conception et du développement du prototype, nous avons démontré comment l'architecture 3-Tiers, couplée à une blockchain publique et à des outils comme Circom et Arkworks, permet d'atteindre les objectifs visés. Les preuves ZKP ont été intégrées pour garantir un scrutin sécurisé, avec une immutabilité des données assurée par la blockchain. Les choix technologiques, notamment Django pour le back end et Ethereum pour la gestion décentralisée des votes, ont été motivés par leur robustesse et leur adaptabilité aux exigences du système.

Cependant, bien que le prototype conçu atteigne les objectifs fixés en termes de confidentialité et de vérifiabilité, certaines limites subsistent. La scalabilité sur des élections à grande échelle, les coûts liés aux transactions sur la blockchain et la complexité technique pour les utilisateurs novices sont autant de défis à relever.

Ces éléments ouvrent la voie à plusieurs perspectives d'amélioration, parmi lesquelles :

- **Optimisation de la partie front end** : Améliorer l'interface utilisateur pour garantir une expérience fluide et intuitive, facilitant ainsi l'interaction des électeurs avec le système.

- **Liaison avec la blockchain** : Renforcer l'intégration entre le front-end et la blockchain pour assurer une communication efficace lors des transactions, garantissant que les votes sont enregistrés de manière sécurisée et transparente.
- **Transactions dans la blockchain** : Explorer des solutions pour réduire les coûts des transactions sur la blockchain, notamment en optimisant les frais de gas et en utilisant des mécanismes de batch processing.
- **Utilisation de ZoKrates** : Intégrer ZoKrates pour générer des preuves ZKP de manière plus efficace, ce qui pourrait améliorer la performance et réduire les coûts de calcul associés à la vérification des votes.
- **Déploiement de l'application** : Travailler sur des stratégies de déploiement qui permettent une mise en production fluide et sécurisée, tout en garantissant que l'application est facilement accessible aux utilisateurs finaux.

En conclusion, ce mémoire illustre la pertinence et le potentiel des ZKP comme outil innovant pour transformer les systèmes de vote électronique. En répondant aux enjeux de transparence, de confidentialité et de sécurité, cette approche s’inscrit dans une vision d’avenir pour des élections numériques fiables et accessibles. La poursuite des travaux sur ce prototype pourrait avoir un impact majeur sur la modernisation des processus démocratiques, notamment dans des contextes spécifiques tels que celui du Sénégal, tout en servant de modèle pour d'autres pays confrontés à des problématiques similaires.