Tâche 2 : Analyse / Spécifications

- Définir la liste de candidats, cas d'utilisation et exigences.
- Rédiger le cahier des charges minimal et le plan de test.

Liste des candidats :

- Nous avons 3 candidats : Amadou, Demba et Omar

Cas d'utilisation (User Stories)

- **En tant qu'électeur,** je vote une seule fois pour mon candidat.
- **En tant qu'utilisateur,** je consulat la liste des candidats et le nombre de votant pour chacun d'entre eux
- **En tant que développeur,** je veux un smart contract qui garde une trace des adresses qui ont voté pour empêcher les votes multiples.

Cahier des charges

1. Objectif du projet Créer une application web décentralisée (dApp) pour un système de vote transparent. Le smart contract, déployé sur une blockchain, doit gérer les votes de manière sécurisée et immuable.

2. Architecture de la solution

- **Wallet :** un interface d'interaction des utilisateurs via leur portefeuille.
- **Interface Utilisateur :** Un front-end développé en HTML/CSS/JavaScript permettant aux utilisateurs de sélectionner un candidat et de voter.
- **Smart Contract :** Le "cœur" de la dApp, écrit en Solidity. Il contient la logique de vote, la liste des candidats, la liste des participants (adresses ayant voté) et le compte des votes.
- **Blockchain :** Le smart contract sera déployé sur une blockchain compatible Ethereum (comme Ganache en local) pour la phase de développement.
- **Connectivité :** L'interface utilisateur communique avec le smart contract via une bibliothèque comme Web3.js ou Ethers.js.

4. Exigences fonctionnelles (EF)

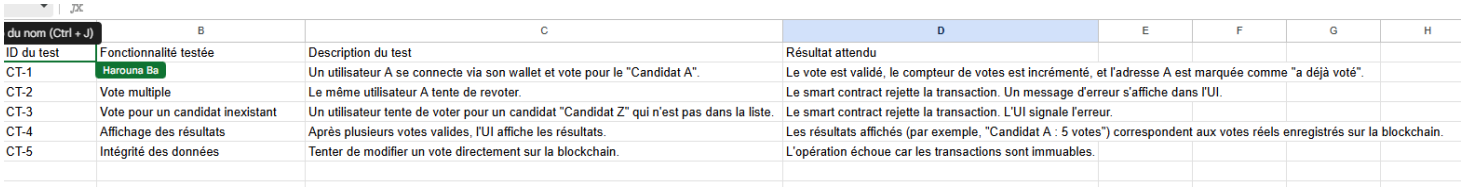
- **EF-1 :** Le système doit permettre à un utilisateur de se connecter via un wallet.
- **EF-2 :** L'UI doit afficher la liste des candidats et permettre à l'utilisateur de sélectionner l'un d'eux.
- **EF-3 :** Le smart contract doit vérifier que l'adresse de l'électeur n'a pas encore voté avant d'enregistrer le vote.
- **EF-4 :** Le smart contract doit incrémenter le nombre de votes du candidat choisi.
- **EF-5 :** Le smart contract doit mettre à jour l'état de l'électeur pour marquer qu'il a voté.
- **EF-6 :** Le système doit afficher les résultats du vote en temps réel.

5. Exigences non fonctionnelles (ENF)

- **ENF-1 (Sécurité)**
- **ENF-2 (Transparence)**
- **ENF-3 (Facilité d'utilisation)**

6. Plan de Test

https://docs.google.com/spreadsheets/d/1HmViN-sv9ZVcbFblSTmTsnQOLrUZs_nQE5niUUyWvRk/edit?usp=sharing



du nom (Ctrl + J)	B	C	D	E	F	G	H
ID du test	Fonctionnalité testée	Description du test	Résultat attendu				
CT-1	Harouna Ba	Un utilisateur A se connecte via son wallet et vote pour le "Candidat A".	Le vote est validé, le compteur de votes est incrémenté, et l'adresse A est marquée comme "a déjà voté".				
CT-2	Vote multiple	Le même utilisateur A tente de revoter.	Le smart contract rejette la transaction. Un message d'erreur s'affiche dans l'UI.				
CT-3	Vote pour un candidat inexistant	Un utilisateur tente de voter pour un candidat "Candidat Z" qui n'est pas dans la liste.	Le smart contract rejette la transaction. L'UI signale l'erreur.				
CT-4	Affichage des résultats	Après plusieurs votes valides, l'UI affiche les résultats.	Les résultats affichés (par exemple, "Candidat A : 5 votes") correspondent aux votes réels enregistrés sur la blockchain.				
CT-5	Intégrité des données	Tenter de modifier un vote directement sur la blockchain.	L'opération échoue car les transactions sont immuables.				