1. Module d’enregistrement et d’attestation des nœuds

Objectif :

Permettre à chaque nœud de :

- Soumettre sa preuve d’attestation (rapport signé par IAS).
- Être reconnu comme un participant fiable du réseau.

Smart contract associé :

- `AttestationRegistry.sol`
    - Enregistrement du hash de la citation d’attestation.
    - Stockage des métadonnées (ID du nœud, domaine, date d'attestation, etc.).

Mise en place :

- Créer une structure `NodeInfo` dans le contrat.
- Ajouter une fonction `registerNode()`.
- Ajouter des mécanismes de vérification via hash du rapport.

2. Module de gestion des demandes de tranches

Objectif :

- Permettre aux **clients consommateurs** de soumettre des demandes de tranches.
- Stocker et suivre l’état de chaque demande.

Smart contract associé :

- `SliceRequestManager.sol`
    - Soumission des requêtes de tranches.
    - Suivi de l’état : en attente, acceptée, refusée.

Mise en place :

- Définir une struct `SliceRequest` (ressources, QoS, domaine, timestamp...).
- Ajouter `submitRequest()`, `updateStatus()`.

3. Module de coordination du consensus (leader, validation)

Objectif :

- Représenter le processus de **sélection du leader** et de **validation des blocs**.
- Historiser les blocs d’orchestration validés.

Smart contract associé :

- `ConsensusLog.sol`
    - Historique des blocs.
    - Vote et validation des propositions du leader.

Mise en place :

- Struct `BlockProposal`.
- Mapping des votes par nœuds.
- Fonction `submitProposal()`, `voteOnBlock()`.

4. Module d’évaluation bilatérale (Game Theory)

Objectif :

- Permettre aux consommateurs et aux nœuds de **s’évaluer mutuellement**.
- Stocker les scores de fiabilité, détecter les comportements malveillants.

Smart contract associé :

- `ReputationManager.sol`
    - Attribution de scores.
    - Historique des interactions.

Mise en place :

- Struct `Evaluation`.
- Fonction `evaluateNode()`, `evaluateConsumer()`.
- Score cumulé.

5. Module de gestion des ressources réseau

Objectif :

- Suivre l’état des ressources disponibles par nœud.
- Vérifier que les ressources allouées respectent les capacités.

Smart contract associé :

- `ResourceLedger.sol`
    - Mise à jour de l’état des ressources.
    - Historique d’allocation.

Mise en place :

- Struct `ResourceState`.
- Fonction `updateResources()`, `checkAvailability()`.

Déploiement des smart contracts

Étapes générales :

1. **Modéliser chaque module** dans un fichier `.sol` avec Remix
2. Compiler avec **Remix IDE**
3. Tester localement

   (pas encore fait)
4. Déployer sur un **testnet** (ex. Sepolia) via **MetaMask**.
5. Une fois fonctionnels, interconnecter les modules (contrats pouvant s’appeler entre eux).