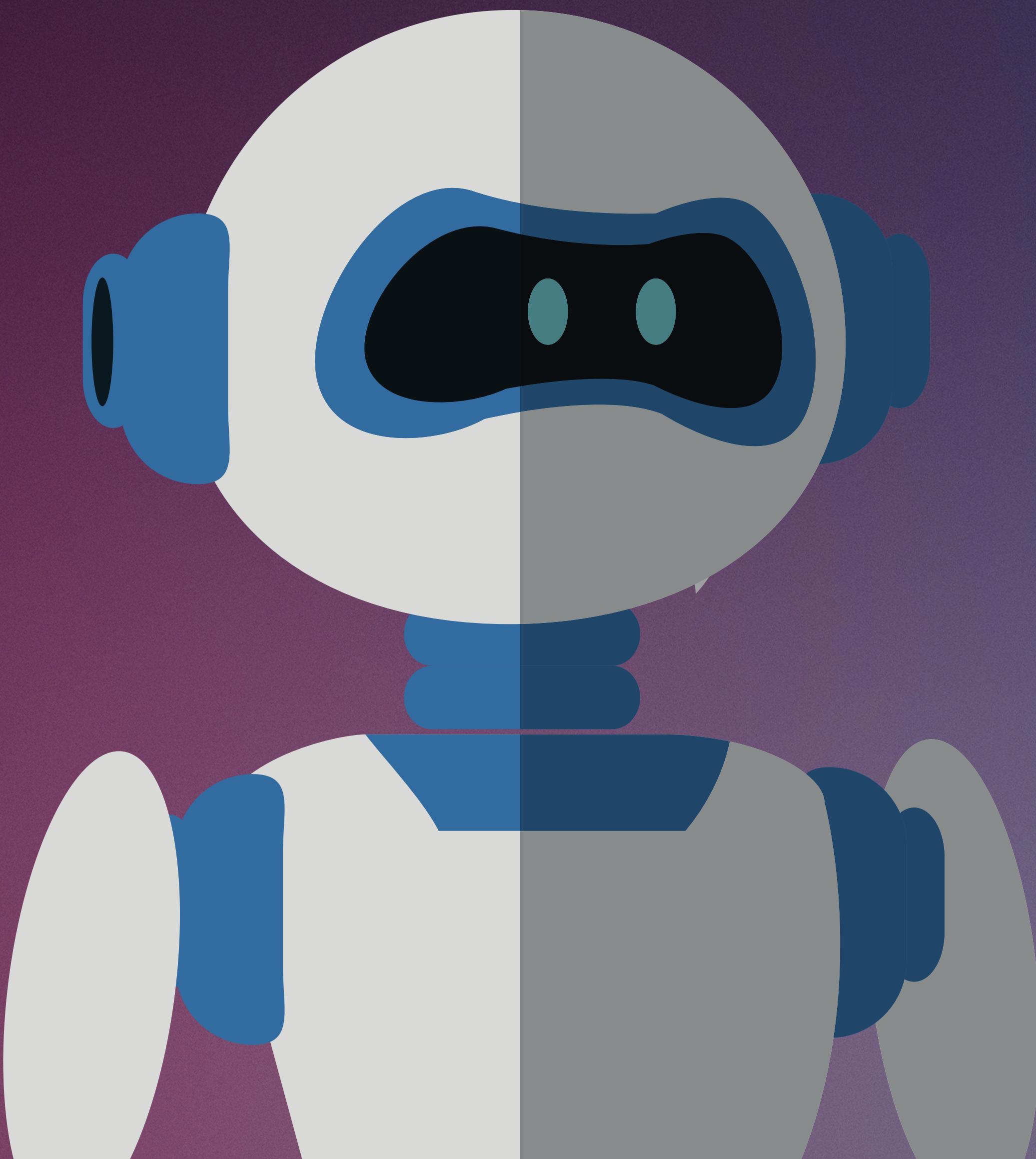


ILYAS SACHA

# GESTION DE RISQUE

BLOCKCHAIN POWERED



## Contents

1.	Introduction .....	2
1.1.	Objectif du projet .....	2
1.2.	Technologies Utilisées .....	2
1.3.	Résumé des Fonctionnalités.....	2
2.	Configuration et Préparation .....	3
2.1.	Environnement de Développement .....	3
2.2.	Script de Déploiement (deploy.js).....	4
2.3.	Étapes à Suivre .....	4
3.	Smart Contract (Structure Principale du Contrat).....	5
3.1.	Variables Clés .....	5
3.2.	Fonctions Clés .....	6
4.	Déploiement .....	7
4.1.	Préparation au Déploiement.....	7
4.2.	Résultats du Déploiement .....	7
5.	Test de DApp .....	7
5.1.	Lancement de l'Application .....	8
5.2.	Ajouter une Contrepartie .....	8
5.3.	Mettre à Jour une Exposition .....	10
5.4.	Calculer le Risque .....	11
5.5.	Calculer le Ration de couverture.....	12
5.6.	Visualiser les Informations d'une Contrepartie .....	13
5.7.	Effectuer une Transaction .....	14
6.	Questions d'Évaluation (Compréhension Technique).....	14
6.1.	Gestion des Limites d'Exposition .....	14
6.2.	Calculs de Risque .....	15
6.3.	Efficacité en termes de Gas.....	15
6.4.	Gestion des Cas Limites .....	15
6.5.	Stratégie de Test .....	15
7.	Compréhension de la Gestion des Risques .....	16
7.1.	Comparaison avec la Gestion Traditionnelle .....	16
7.2.	Gestion des Scénarios de Risque .....	16
8.	Implémentation et Innovation .....	16
8.1.	Fonctionnalités Supplémentaires .....	16
8.2.	Applications Réelles .....	16
8.3.	Réflexion sur le Processus de Développement.....	17

# 1. Introduction

## 1.1. Objectif du projet

Ce projet vise à développer un système Blockchain pour la gestion des risques de contrepartie, combinant des principes de finance et de technologie pour une solution transparente et efficace.

## 1.2. Technologies Utilisées

Blockchain	Langage	Outils	Environnement	User Interface
	 SOLIDITY	 Hardhat		
		 METAMASK		
				

## 1.3. Résumé des Fonctionnalités

**Ajout de contreparties :** Permet de configurer des profils financiers incluant le score de crédit et la limite d'exposition.

- **Mise à jour :**
- **Exposition :** Calcul dynamique des montants de risque courants.

**Collatéral :** Ajustement des garanties pour refléter les évolutions.

**Calcul des Risques :** Basé sur les expositions et les scores de crédit.

**Alertes :** Notifications en cas de dépassement de limites ou de ratios critiques.

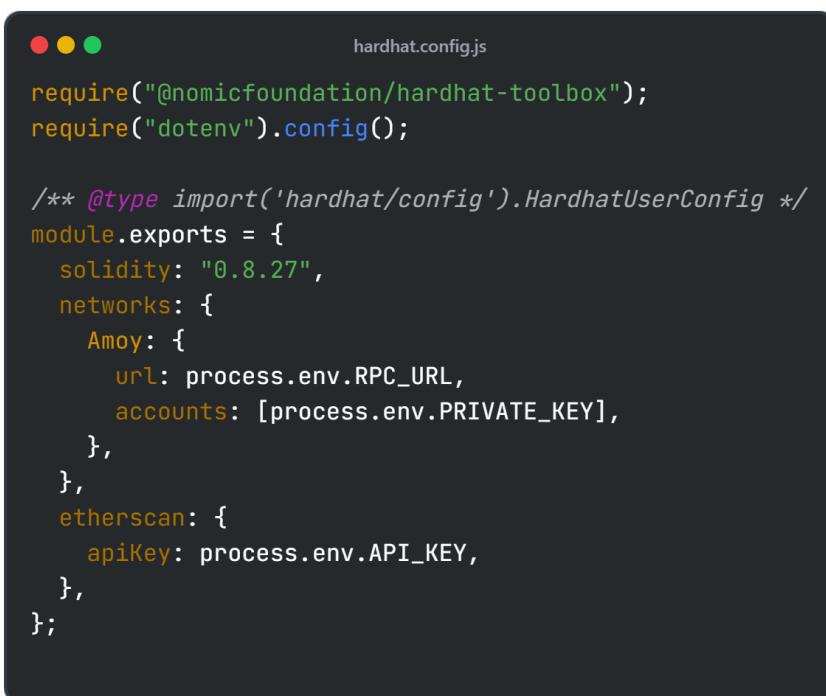
## 2. Configuration et Préparation

### 2.1. Environnement de Développement

Pour commencer le projet, plusieurs étapes de configuration sont nécessaires pour créer un environnement de développement adapté.

#### 💡 Hardhat

- ❖ **Hardhat** est un outil essentiel pour créer et gérer des contrats intelligents. Il aide à compiler le code, tester les fonctionnalités et déployer les contrats sur une blockchain.
- ❖ Le fichier **hardhat.config.js** configure les réseaux, comme le réseau Polygon, utilise des informations sensibles stockées dans un fichier **.env**

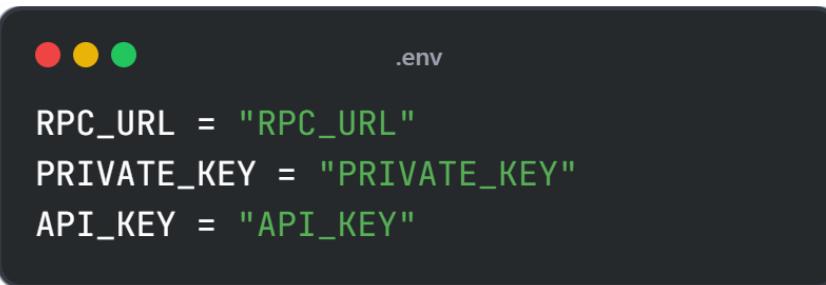


```
hardhat.config.js

require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config();

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.27",
  networks: {
    Amoy: {
      url: process.env.RPC_URL,
      accounts: [process.env.PRIVATE_KEY],
    },
    etherscan: {
      apiKey: process.env.API_KEY,
    },
  };
};
```

- ❖ Les informations sensibles comme les clés privées, l'URL du réseau, et la **clé API** pour l'explorateur blockchain sont stockées dans un fichier **.env**. Cela empêche que ces données soient visibles publiquement dans le code.



```
.env

RPC_URL = "RPC_URL"
PRIVATE_KEY = "PRIVATE_KEY"
API_KEY = "API_KEY"
```

## 2.2. Script de Déploiement (deploy.js)

- ❖ Ce script automatise le déploiement du contrat intelligent sur la blockchain. Une fois lancé, il renvoie l'adresse à laquelle le contrat a été déployé.
- ❖ Le script utilise **hre.ethers** pour interagir avec **Ethereum** et affiche l'adresse déployée dans la console.
- ❖ Riskma est le nom du contrat.



```
● ● ● deploy.js
const hre = require("hardhat");

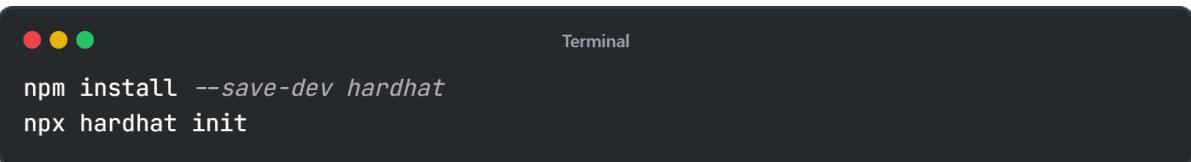
async function main() {
  const riksma_contract = await hre.ethers.getContractFactory("riskma");
  const deployed_riksma_contract = await riksma_contract.deploy()

  console.log(`Contract Address Deployed: ${deployed_riksma_contract.target}`);
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

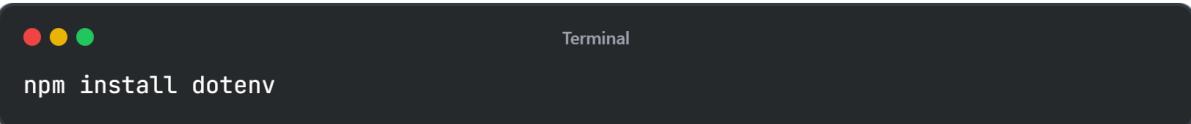
## 2.3. Étapes à Suivre

- ❖ Installez et Initialisez Node.js et les dépendances nécessaires en exécutant la commande (terminal)



```
● ● ● Terminal
npm install --save-dev hardhat
npx hardhat init
```

- ❖ Installez Configurez le fichier **.env** avec vos informations (**RPC URL**, **clés privées**, **Private Key**)



```
● ● ● Terminal
npm install dotenv
```

- ❖ **Compilez** le contrat pour vérifier qu'il est exempt d'erreurs

```
Terminal  
npx hardhat compile
```

- ❖ **Déployez** le contrat avec le script **deploy.js**

```
Terminal  
npx hardhat run scripts/deploy.js --network Amoy
```

- ❖ **Vérifiez** que le contrat est déployé avec succès.

```
Terminal  
npx hardhat verify --network Amoy <><Contract Address>>
```

## 3. Smart Contract (Structure Principale du Contrat)

Le contrat intelligent est le cœur du projet. Il définit toutes les règles et les interactions possibles pour gérer les contreparties et les expositions financières. Voici une présentation simplifiée pour tout comprendre.

### 3.1. Variables Clés

- ❖ **Contreparties:** Une structure de données stockant des informations sur chaque contrepartie, telles que leur score de crédit, limite d'exposition et montant actuel exposé.

```
riskma.sol  
  
struct Contrepartie {  
    address portefeuille;  
    uint256 scoreCredit;  
    uint256 limiteExposition;  
    uint256 expositionCourante;  
    uint256 collateral;  
    bool estActif;  
}
```

- ❖ **Expositions Nettes :** Une table de mappage entre deux adresses pour suivre l'exposition financière entre deux contreparties.

```
riskma.sol  
  
mapping(address => mapping(address => int256)) public netExpositions;
```

- ❖ **Propriétaire** : Adresse de l'utilisateur ayant des priviléges d'administration, comme ajouter ou gérer les contreparties.

```
● ● ● riskma.sol
address public owner;
```

## 3.2. Fonctions Clés

- ❖ **ajouterContrepartie** : Permet au propriétaire d'ajouter une nouvelle contrepartie avec un score de crédit et une limite d'exposition.

```
● ● ● riskma.sol
function ajouterContrepartie(
    address _portefeuille,
    uint256 _scoreCredit,
    uint256 _limiteExposition
) public onlyOwner { ... }
```

- ❖ **mettreAJourExposition** : Met à jour l'exposition courante d'une contrepartie après une transaction ou un changement financier.

```
● ● ● riskma.sol
function mettreAJourExposition(address _portefeuille, uint256
_nouvelleExposition) public { ... }
```

- ❖ **effectuerTransaction** : Permet d'enregistrer une transaction entre deux contreparties, en ajustant leurs expositions respectives.

```
● ● ● riskma.sol
function effectuerTransaction(address _to, int256 _montant) public { ... }
```

- ❖ **calculerRisque** : Calcule le risque associé à une contrepartie en fonction de son exposition actuelle et de son score de crédit.

```
● ● ● riskma.sol
function calculerRisque(address _portefeuille) public view returns (uint256) {
... }
```

## 4. Déploiement

Le déploiement consiste à placer le contrat intelligent sur une blockchain afin qu'il puisse être utilisé par des utilisateurs ou des applications. Voici une explication simplifiée et détaillée pour que n'importe qui puisse comprendre et reproduire le processus.

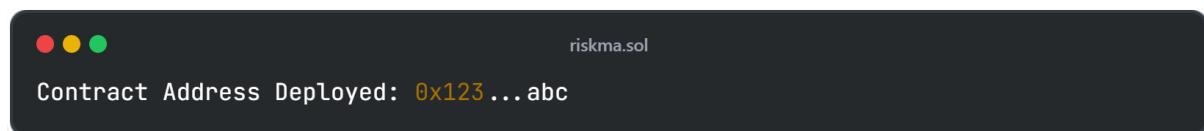
### 4.1. Préparation au Déploiement

Avant de déployer, assurez-vous que :

- ❖ Votre fichier `.env` contient les bonnes informations comme déjà expliqué.
- ❖ Vous avez configuré **Hardhat** avec les paramètres réseau dans **hardhat.config.js**.
- ❖ Le fichier **deploy.js** contient les instructions pour déployer automatiquement le contrat.
- ❖ **Compilez** le contrat avec **Hardhat**. Cela vérifie que le contrat est prêt à être déployé.
- ❖ **Déployez** le contrat

### 4.2. Résultats du Déploiement

- ❖ Une fois le contrat déployé, l'adresse sera affichée dans le terminal



A screenshot of a terminal window. At the top left are three colored dots (red, yellow, green). At the top right is the file name `riskma.sol`. Below that, the text `Contract Address Deployed: 0x123...abc` is displayed in white on a black background.

**NB** : Notez bien cette adresse, car elle est essentielle pour interagir avec le contrat.

- ❖ Vérifiez le contrat sur un explorateur blockchain (comme [Polygonscan](#)) pour confirmer qu'il est actif

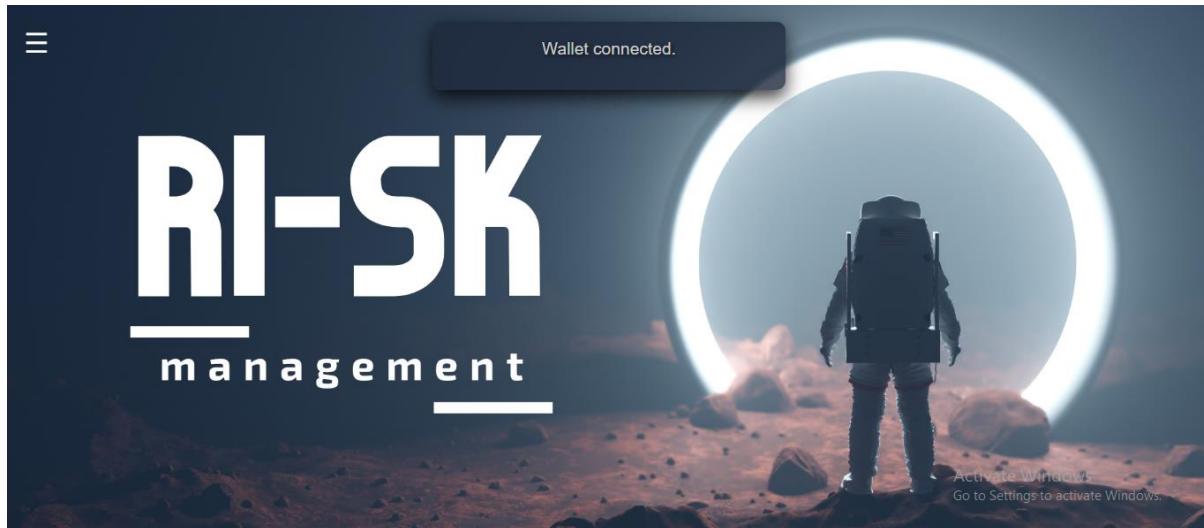
**NB** : Une fois vérifié, vous verrez un message comme "Contract verified successfully."

## 5. Test de DApp

La phase de test est essentielle pour valider les fonctionnalités de la DApp et s'assurer qu'elle répond aux exigences définies. Ce test pratique couvre l'ensemble des fonctionnalités, de l'ajout de contreparties à la gestion des transactions et au calcul des risques. Grâce à une interface intuitive, chaque action peut être effectuée directement depuis le navigateur avec l'intégration de **MetaMask**.

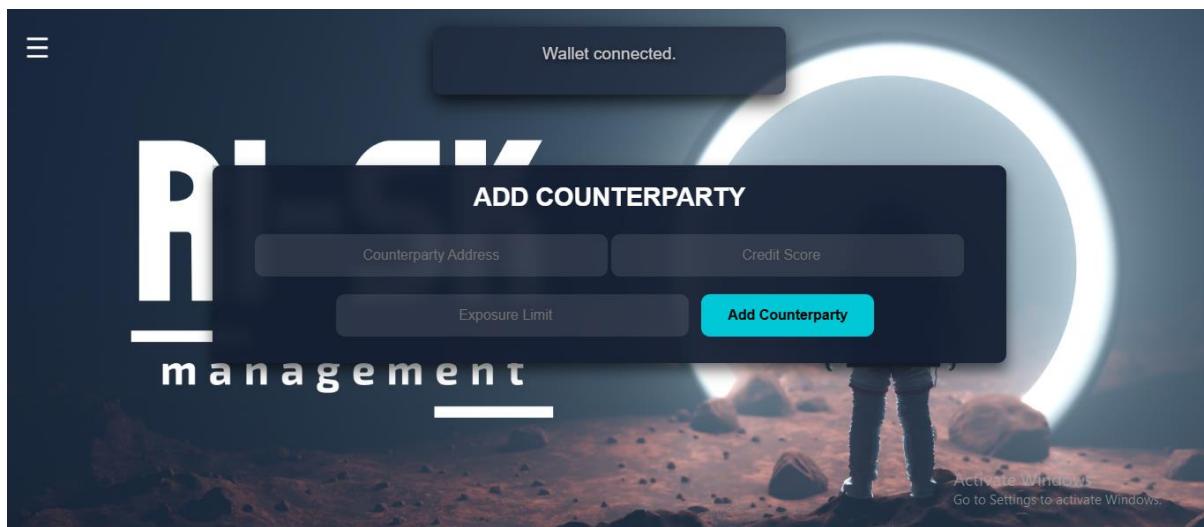
## 5.1. Lancement de l'Application

- ❖ **Démarrage :** Ouvrez le fichier index.html dans un navigateur pris en charge (Chrome ou Firefox). Assurez-vous que MetaMask est installé et configuré.
- ❖ **Connexion au Portefeuille :** Cliquez sur le bouton Connect Wallet et Autorisez la connexion dans MetaMask.

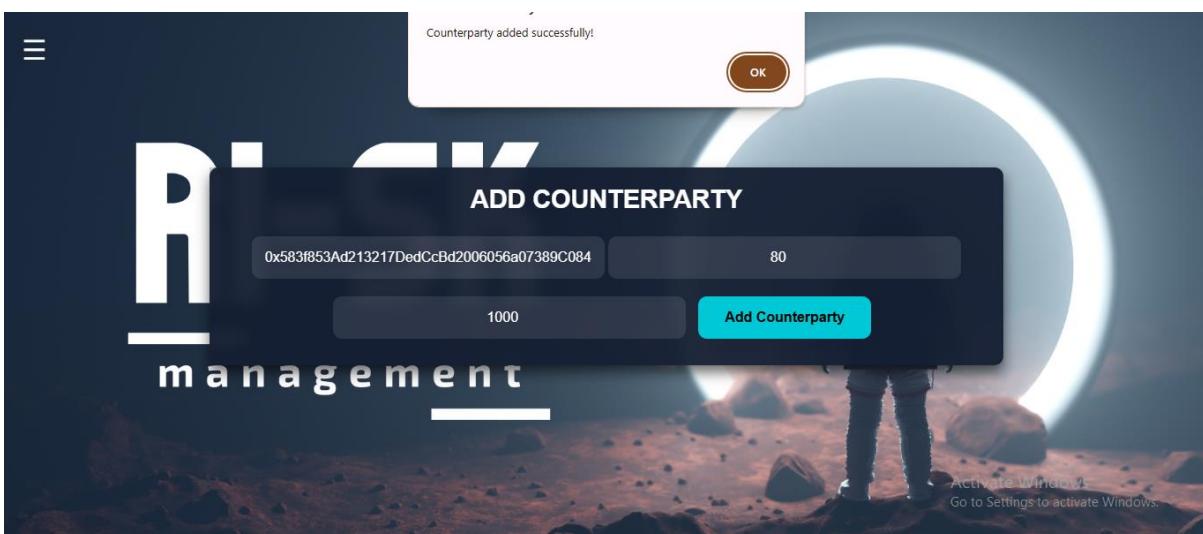
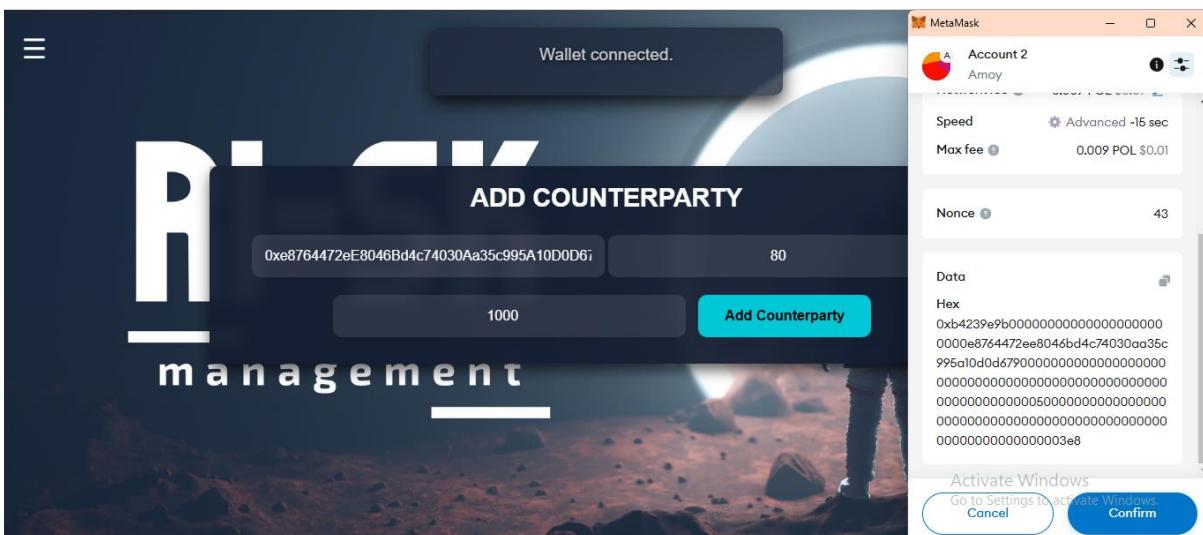


## 5.2. Ajouter une Contrepartie

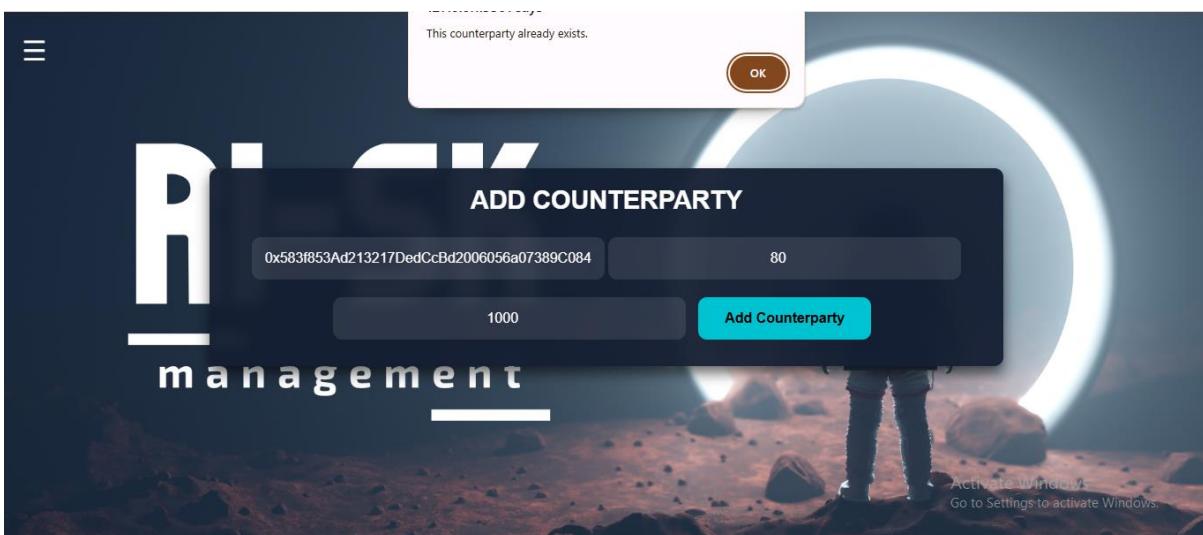
- ❖ Accédez à la section **Add Counterparty** via le menu latéral.
- ❖ Remplissez les champs



- ❖ Cliquez sur **Add Counterparty** pour soumettre.

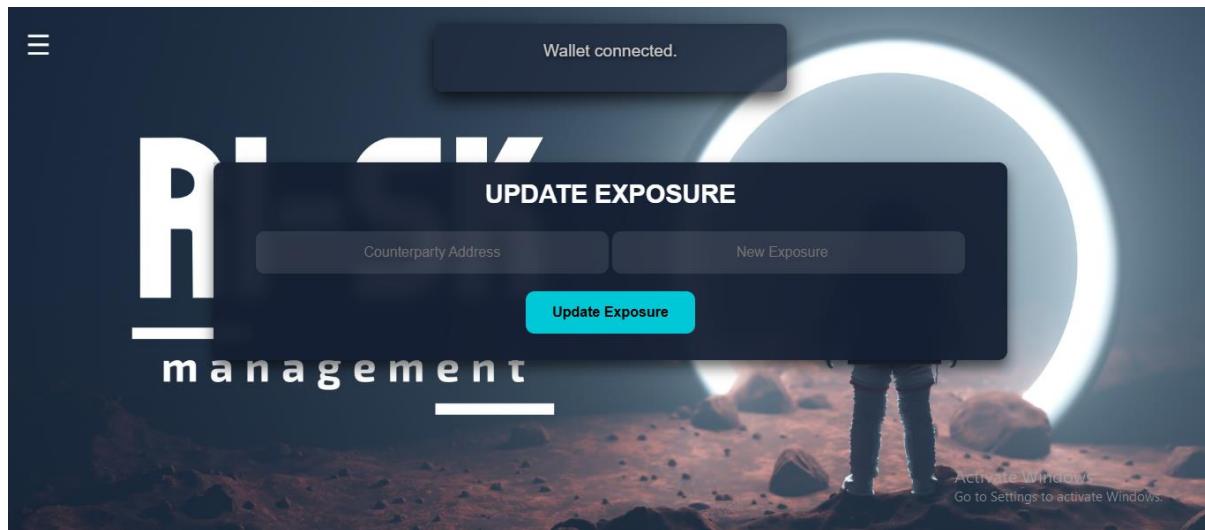


- ❖ Si vous essayez d'ajouter une contrepartie existante, ce message apparaîtra.

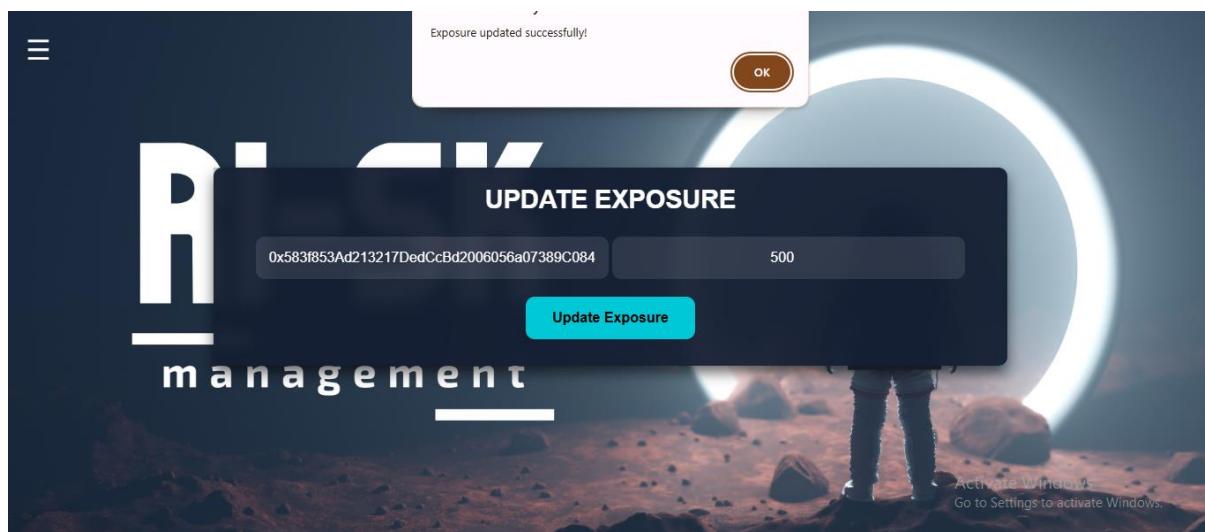


### 5.3. Mettre à Jour une Exposition

- ❖ Naviguez vers **Update Exposure**



- ❖ Entrez l'adresse de la contrepartie et la nouvelle exposition

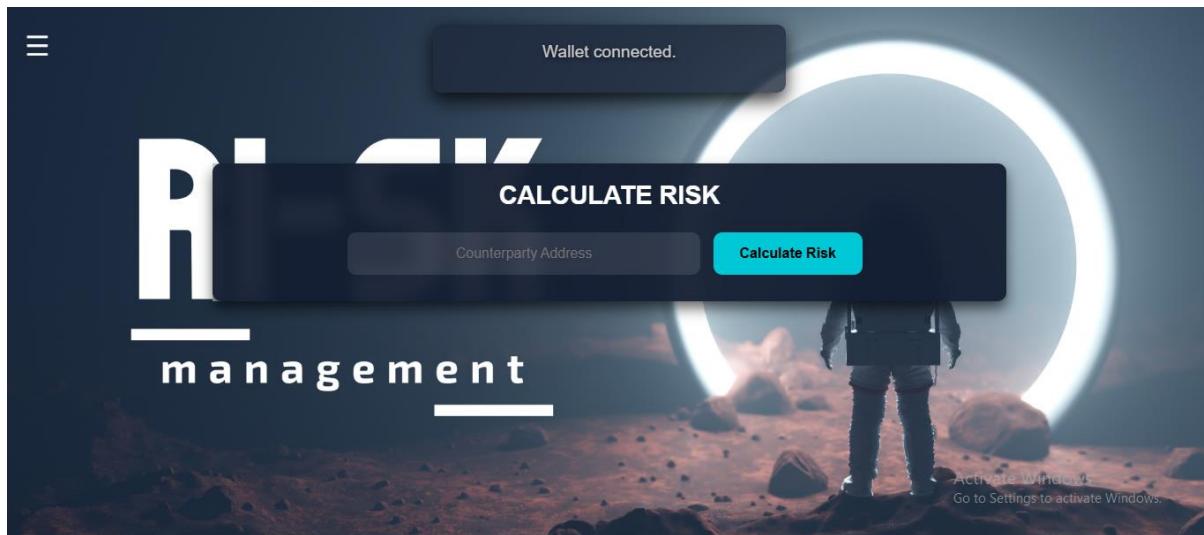


- ❖ Résultats attendus :

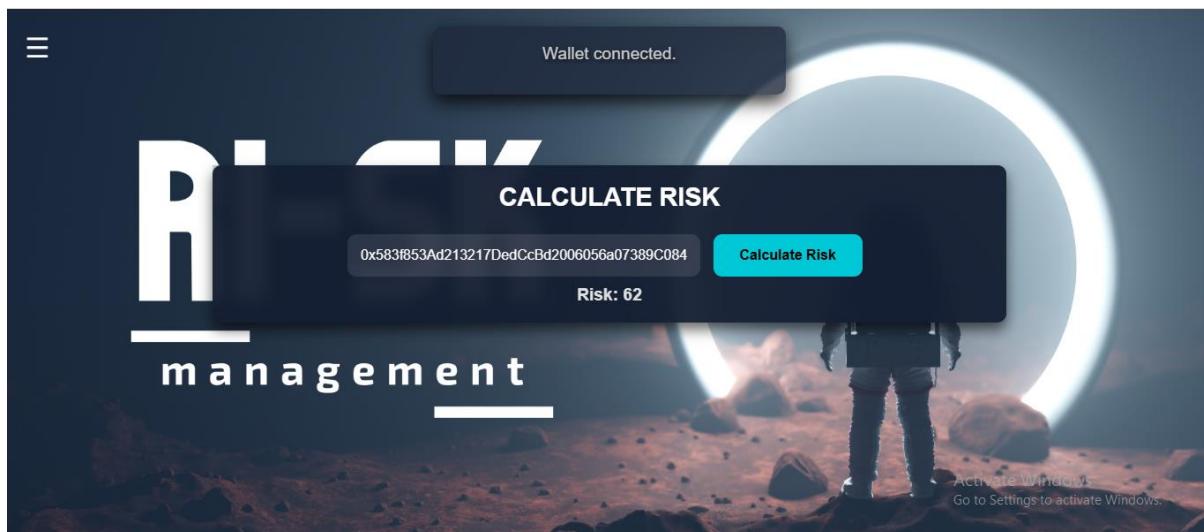
- La mise à jour est effectuée avec un message de confirmation.
- Si la limite d'exposition est dépassée, un message d'alerte est affiché.

## 5.4. Calculer le Risque

- ❖ Rendez-vous dans la section **Calculate Risk**.



- ❖ Entrez l'adresse Ethereum de la contrepartie.



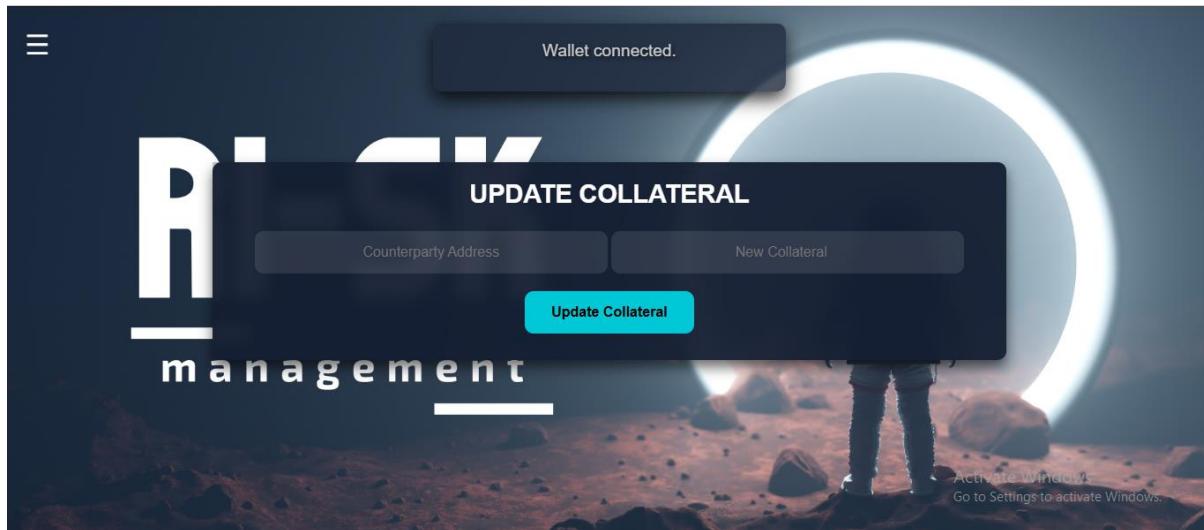
- ❖ **Résultats attendus :**

- Le risque calculé s'affiche sous forme de pourcentage.
- Si la contrepartie n'existe pas ou est inactive, un message d'erreur est retourné.

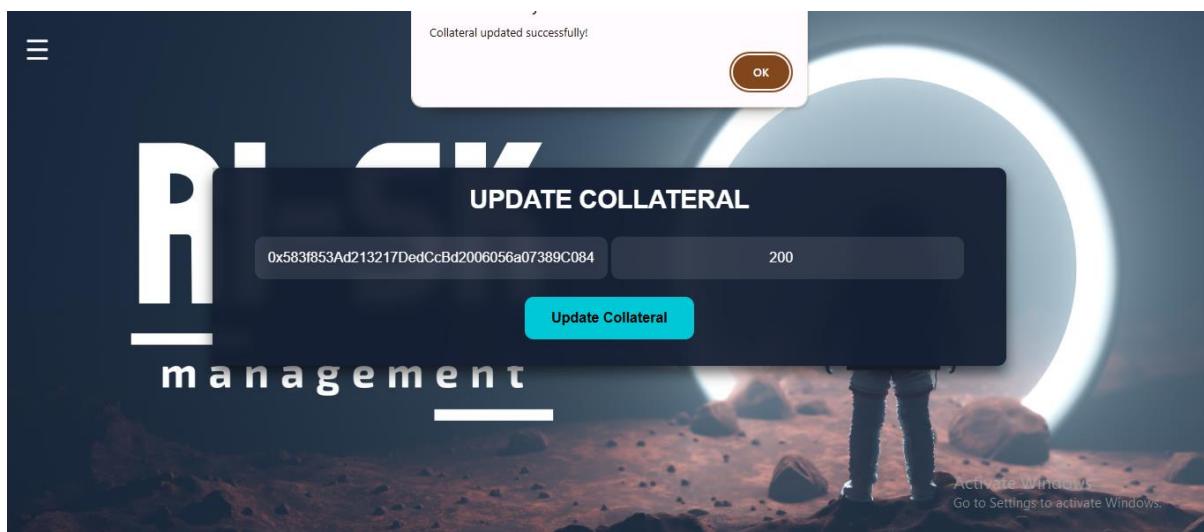
$$\text{Score de Risque} = \frac{\text{Exposition Courante}}{\text{Limite d'Exposition}} \times \frac{100}{\text{Score de Crédit}}$$

## 5.5. Calculer le Ration de couverture

- ❖ Rendez-vous dans la section **Update Collateral**.



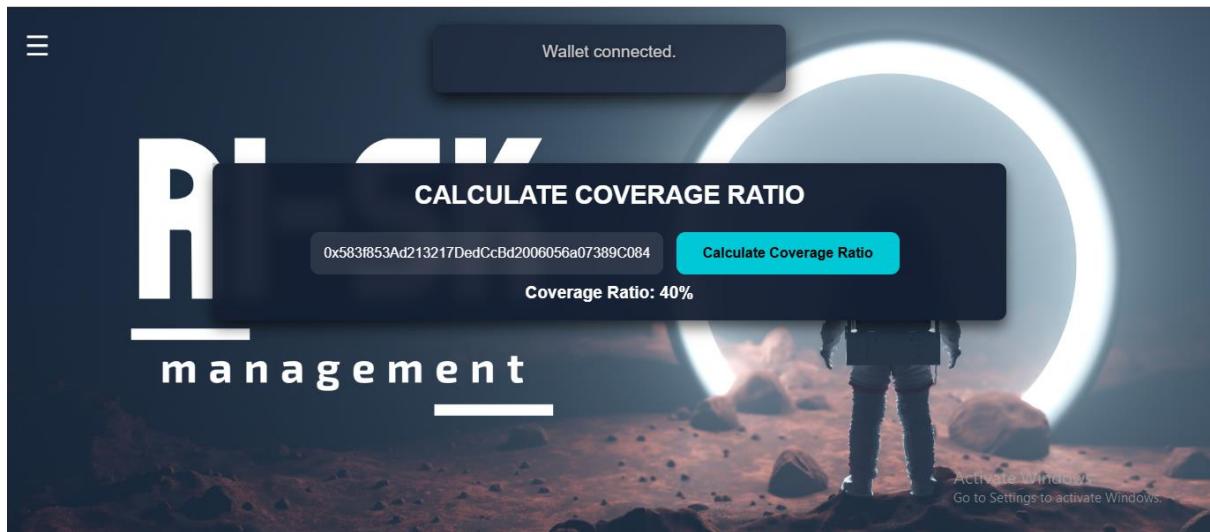
- ❖ Entrez l'adresse de la contrepartie et la nouvelle collateral



- ❖ Cette étape était nécessaire car le **ratio de couverture** est calculé comme suit.

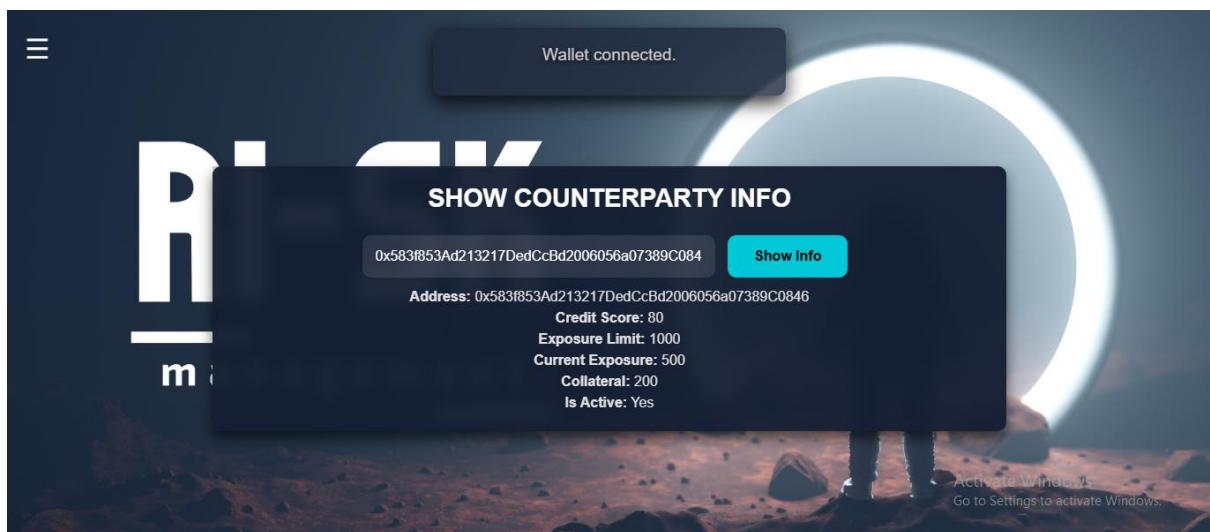
$$\text{Ratio de Couverture} = \frac{\text{Collatéral}}{\text{Exposition Totale}}$$

- ❖ Rendez-vous dans la section **Calculate Coverage Ratio**.
- ❖ Entrez l'adresse Ethereum de la contrepartie
- ❖ **Résultats attendus :**
  - le ratio de couverture calculé s'affiche sous forme de pourcentage.



## 5.6. Visualiser les Informations d'une Contrepartie

- ❖ Accédez à la section **Show Counterparty Info**.
- ❖ Entrez une adresse Ethereum existante.



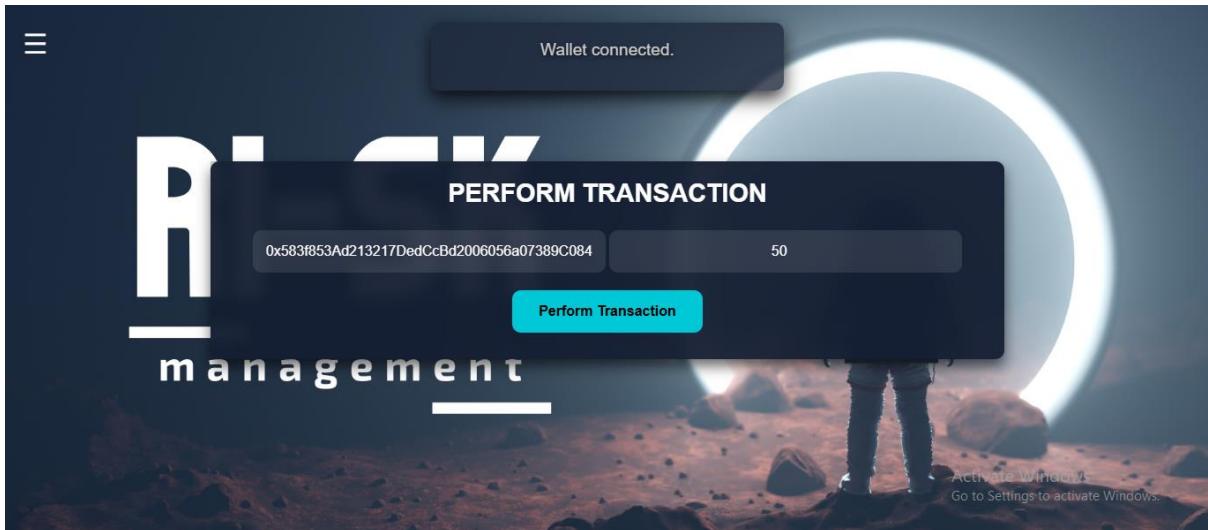
- ❖ **Résultats attendus :**
  - Les informations de la contrepartie (score de crédit, limite d'exposition, exposition courante, etc.) s'affichent.
  - Si l'adresse est invalide ou non enregistrée, un message d'erreur apparaît.
  - Si l'exposition courante de la contrepartie dépasse la limite d'exposition, son statut "Is Actif" deviendra "No" au lieu de "Yes".

## 5.7. Effectuer une Transaction

❖ Naviguez vers **Perform Transaction**.

❖ Entrez :

- L'adresse du destinataire.
- Le montant de la transaction.



❖ **Résultats attendus :**

- Si la transaction est valide : Message de succès avec des détails.
- Si une erreur survient (par exemple, manque de fonds ou contrepartie inactive) : Message d'échec.
- Le montant de la transaction est déduit de l'exposition courante de l'expéditeur et ajouté à l'exposition courante du destinataire.

## 6. Questions d'Évaluation (Compréhension Technique)

### 6.1. Gestion des Limites d'Exposition

❖ **Fonctionnement** : Le contrat intelligent suit l'exposition courante de chaque contrepartie via la variable `expositionCourante`. Si cette valeur **dépasse** la `limiteExposition`, une alerte est déclenchée et la contrepartie est marquée comme **inactive** (`estActif = false`).

❖ **Mesures de Sécurité** :

- Vérifications intégrées avec des modificateurs (`require`) pour empêcher l'exécution des fonctions sensibles si les limites sont dépassées.
- Événements tels que `LimiteDepassee` pour notifier les dépassements en temps réel.

## 6.2. Calculs de Risque

Le risque est calculé en prenant en compte :

- ❖ **Scores de Crédit** : Une pondération est appliquée en fonction de la fiabilité de la contrepartie. Exposition
- ❖ **Courante** : Le montant actuel est comparé à la limite d'exposition. Historique des
- ❖ **Transactions** : Les expositions nettes sont mises à jour après chaque transaction pour refléter l'état actuel des risques.

```
● ● ● riskma.sol  
return (c.expositionCourante * 10000) / (c.limiteExposition * c.scoreCredit);
```

## 6.3. Efficacité en termes de Gas

Optimisations :

- ❖ Réduction des écritures sur la blockchain en regroupant les mises à jour de données.
- ❖ Vérifications minimales pour limiter les calculs inutiles.
- ❖ Usage de fonctions internes comme `_verifierAlertesEtLimites` pour éviter des appels externes coûteux.

```
● ● ● riskma.sol  
_verifierAlertesEtLimites(_portefeuille);
```

## 6.4. Gestion des Cas Limites

**Congestion du Réseau** : Les utilisateurs peuvent ajuster la limite de gas pour garantir l'exécution en période de congestion.

**Transactions Échouées** : Les écritures sur la blockchain sont encapsulées dans des transactions atomiques, donc aucune modification n'est appliquée si l'opération échoue.

**Entrées Invalides** : Vérifications via require empêchent les entrées non valides (ex. : montants négatifs ou adresses inexistantes).

## 6.5. Stratégie de Test

- ❖ **Scénarios Testés** :
  - Ajout d'une contrepartie avec des données valides et invalides.
  - Simulation de dépassements de limites pour déclencher des alertes.
  - Transactions entre contreparties pour valider les mises à jour des expositions.
- ❖ **Résultats** : Tous les scénarios testés ont confirmé le bon fonctionnement des mécanismes de sécurité et des calculs de risque.

## 7. Compréhension de la Gestion des Risques

### 7.1. Comparaison avec la Gestion Traditionnelle

#### **Avantages de la Blockchain :**

- ❖ Transparence grâce au registre distribué.
- ❖ Résilience via des mécanismes automatisés et sécurisés.

#### **Limitations :**

- ❖ Dépendance à la qualité des données entrées.
- ❖ Coûts en gas pour des transactions complexes.

### 7.2. Gestion des Scénarios de Risque

**Augmentation Soudaine de l'Exposition :** Le système bloque les contreparties lorsque les limites sont dépassées.

**Détérioration du Score de Crédit :** Une exposition excessive combinée à un score faible déclenche des alertes de risque élevé.

**Transactions Multiples Simultanées :** Les mises à jour d'exposition sont atomiques pour éviter des incohérences.

## 8. Implémentation et Innovation

### 8.1. Fonctionnalités Supplémentaires

- ❖ Notifications d'alertes en temps réel via des événements blockchain (AlerteRisque, LimiteDepassee).
- ❖ Gestion automatisée des états actifs/inactifs des contreparties.

### 8.2. Applications Réelles

**Banques et Institutions Financières :** Suivi des risques entre filiales ou partenaires commerciaux.

**Plateformes Décentralisées :** Gestion des risques pour les prêts peer-to-peer (DeFi).

**Modifications nécessaires pour la production :** Tests approfondis, intégration avec des systèmes externes, et audits de sécurité.

### 8.3. Réflexion sur le Processus de Développement

#### Défis :

- ❖ Complexité des calculs en temps réel.
- ❖ Limiter les coûts en gas sans compromettre la sécurité.

#### Solutions :

- Simplification des formules et regroupement des mises à jour.
- Tests exhaustifs pour valider chaque fonctionnalité.

**Améliorations Futures :** Ajouter des interfaces API pour faciliter l'intégration avec des outils tiers.