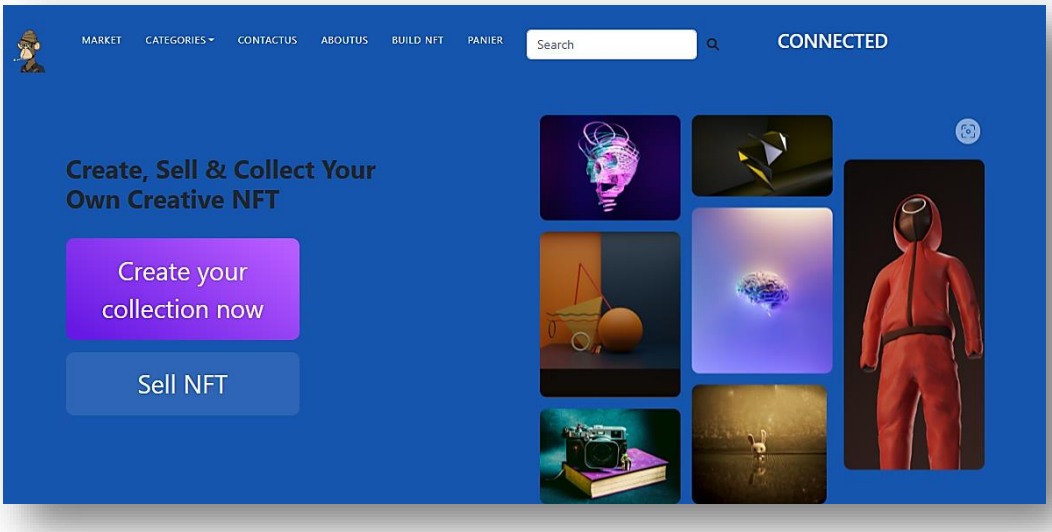
كلية العلوم والتقنيات بطنجة
Faculté des Sciences et Techniques de Tanger
جامعة عبد الملك السعدي
Université Abdelmalek Essaadi

Cycle d'ingénieur – LSI

Project : NFT Marketplace As Single page application



Encadré par :

Lotfi EL AACHAK

Réalisé par :

- Akarmass Basma
- Ouidad Oualhaj
- Tariq Kiro

Table des matières

Remerciement

Au terme de ce travail, nous tenons à exprimer notre très vive reconnaissance à notre cher professeur et encadrant M. Lotfi EL AACHAK pour son suivi et pour son énorme soutien qu'il n'a cessé de nous prodiguer au long de cette semestre.

L'Objectif :

L'objectif du projet est le développement min d'un Marketplace des NFTs « web based DApp », les transactions au niveau du Marketplace se fait à travers crypto monnaies « Ethereum, bitcoin, etc.. ».

- L'utilisateur s'authentifier à travers une Ethereum wallet.
- Il peut vendre acheter des NFTs
- Il peut procéder pour le Minting des NFTs.
- Le Marketplace doit être basé sur une architecture hybride avec une base de données NOSQL
- MongoDB pour stoker les informations d'ordre générale, par contre les transactions des NFTs doivent être gérées par des smart contrats.

Outils :

- Spring boot
- Solidity
- Micro Services
- Angular
- Ether.js
- MongoDB
- Meta Mask
- Hardhat
- Devops : CI/CD
- Jenkens
- Docker
- Github

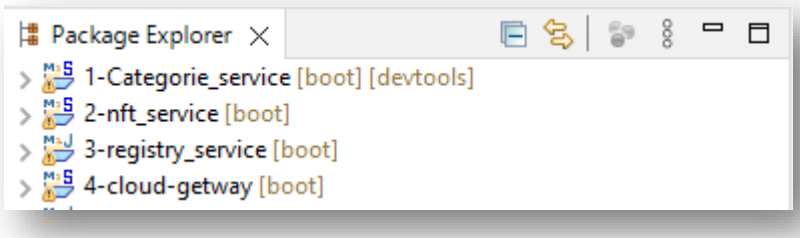
I. La démarche à suivre pour réaliser ce projet :

Avant de commencer voici la démarche à suivre durant ce projet :

A. *Réaliser les micro-services nécessaires*
B. *Ecrire a Smart Contrat avec Solidty*
C. *Tester le Contrat*
D. *Déployer le Contrat à la blockchain en utilisant Hardhat*
E. *Créer Front end avec Angular*
F. *Connecter Front end avec Smart Contrat en utilisant Ether.js*

II. La partie Backend :

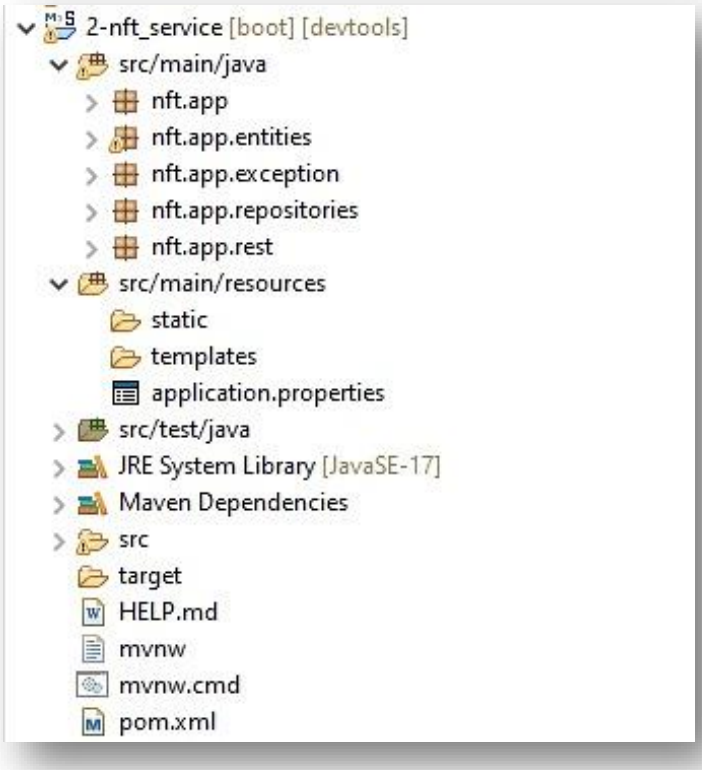
1) Architecture du Project:



Le projet se compose de 4 Microservices :

- NFT-service
- Categorie-service
- Registry-service
- Cloud-Gateway

1) Le micro service : “NFT-service".



2.1 Entité 'NFT' :

NFT.java

```java
package nft.app.entities;

import java.io.Serializable;

@Document("NFT")
public class NFT implements Serializable{

    @Id
    private long id_nft;
    private String nom_nft;
    private String description;
    private double prix;
    private String image;
    private String cat;
```

L'annotation *@Document de MongoDB* est utilisée pour indiquer que cette classe Java doit être mappée en tant que document MongoDB. Elle est similaire à l'annotation *@Entity* de l'API de persistance Java

2.2 NftRepository :

- L'interface doit extraire de Mongo Repository puisque notre SGBD est MongoDB :
- Utilisation d'annotation *@Repository*

NftRepository.java

```java
package nft.app.repositories;

import org.springframework.data.domain.Page;

@Repository
public interface NftRepository extends MongoRepository<NFT,Long>{
}
```

2.3 NFTRestController :

❖ *Create NFT :*

```
@PostMapping("/nfts")
public NFT saveNft(@RequestBody NFT nft) {
    return nftRepository.save(nft);
}
```

❖ *GetAllNfts :*

```
@GetMapping("/nfts")
public List<NFT> getAllNfts(){
    return nftRepository.findAll();
}
```

❖ *GetNFTById*

```
@GetMapping("/nfts/{id}")
public ResponseEntity<Optional<NFT>> getNftById(@PathVariable Long id) {
    Optional<NFT> nft = nftRepository.findById(id);
    return ResponseEntity.ok(nft);
}
```

❖ *Delete NFT*

```
@DeleteMapping("/nfts/{id}")
public ResponseEntity<Map<String, Boolean>> deleteNft(@PathVariable Long id){
    NFT nft = nftRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("nft not exist with id :" + id));

    nftRepository.delete(nft);
    Map<String, Boolean> response = new HashMap<>();
    response.put("deleted", Boolean.TRUE);
    return ResponseEntity.ok(response);
}
```

2.4 NFTApplication

NftApplication.java

```java
package nft.app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;

@EnableDiscoveryClient
@SpringBootApplication
public class NftApplication {

    @LoadBalanced
    public static void main(String[] args) {
        SpringApplication.run(NftApplication.class, args);
    }

}
```

2.5 Application.propereties :

application.properties

```
server.port=8082
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=demo


spring.application.name=NFT-SERVICE

#eureka client config

eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
eureka.client.service-url.default-zone=http://localhost:8761/eureka
eureka.instance.hostname=localhost
```

- La propriété **server.port** définit le numéro de port sur lequel l'application écoutera les requêtes entrantes.
- Les propriétés **spring.data.mongodb.host, spring.data.mongodb.port et spring.data.mongodb.database** définissent les paramètres de connexion à une base de données MongoDB.

- La propriété **spring.application.name** définit le nom de l'application.

- Les propriétés de configuration d'Eureka client définissent la configuration pour l'enregistrement de l'application auprès d'un serveur Eureka et la récupération de la liste des services enregistrés.
- eureka.instance.hostname définit le nom d'hôte de l'instance enregistrée.

Remarque :

Ce fichier est un fichier de configuration, Il peut être utilisé pour configurer des propriétés liées à l'application telles que :

- Les informations de connexion à la base de données (mongoDB dans notre cas)
- Les informations de connexion à un service distant (Eureka)
- Les paramètres de démarrage de l'application (port…)

On peut utiliser un autre fichier de configuration de même nom (Application) mais avec l'extension .yml !

2.6 Le fichier de configuration Web.xml :

Ce fichier est écrit par défaut avec la création de projet Spring Boot et il contient les dépendances qui permettent d'intégrer facilement des composants web tels que les contrôleurs, les services REST et les vues au sein d'une application Spring Boot.

nft_service/pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
```

Chaque dépendance à un rôle précise :

- *Spring-boot-starter-actuator* :

Cette dépendance inclut des outils pour surveiller et gérer l'application en production, tels que les indicateurs de performance et les endpoints de surveillance.

- *Spring-boot-starter-data-mongodb*:

Cette dépendance inclut les composants nécessaires pour utiliser MongoDB avec Spring Data pour stocker et récupérer des données.

- *spring-boot-starter-data-rest:*

Est utilisée pour créer des API REST exposant les données stockées dans une base de données de manière simple et rapide

- *Spring-boot-starter-web* :

Cette dépendance inclut les composants nécessaires pour développer des applications web, tels que Spring MVC et Tomcat.

- *Spring-boot-starter-test* :

Cette dépendance inclut les composants nécessaires pour tester l'application, tels que JUnit, Hamcrest et Spring Test.

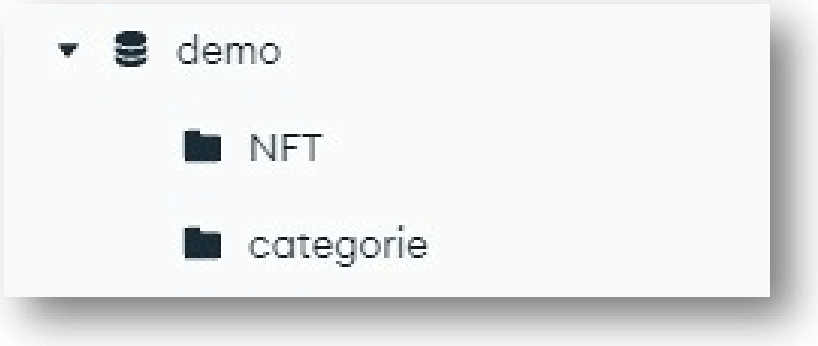
- *Spring-cloud-starter-netflix-eureka-client*:

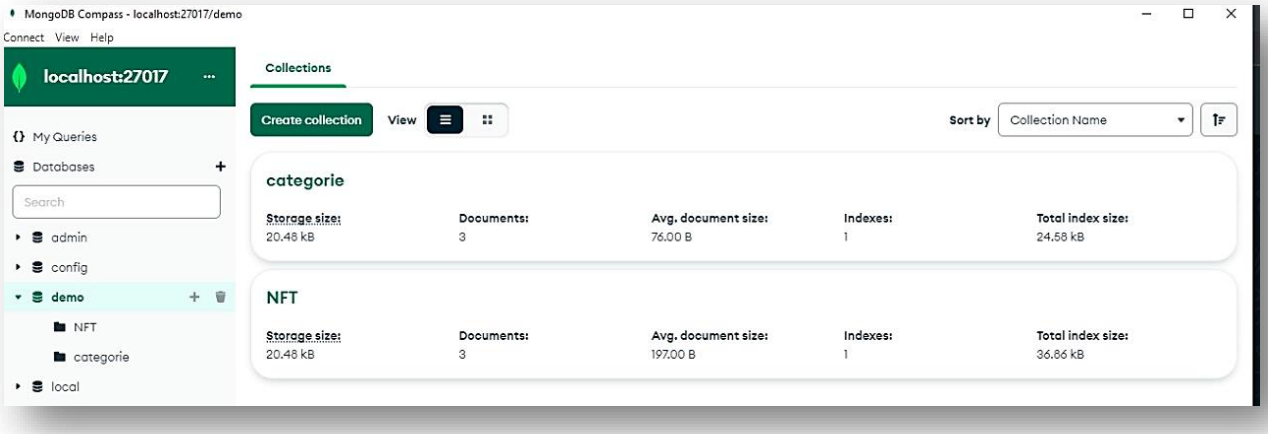
Cette dépendance inclut les composants nécessaires pour utiliser Eureka Client avec Spring Cloud pour enregistrer et découvrir des services dans un environnement Eureka.

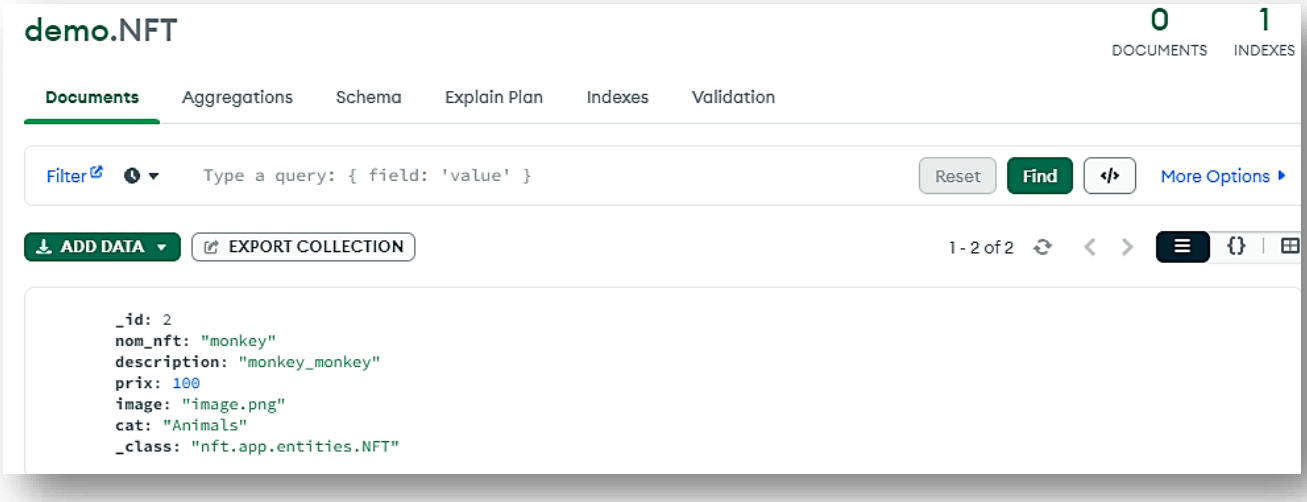
2.7 La base de données MongoDB :

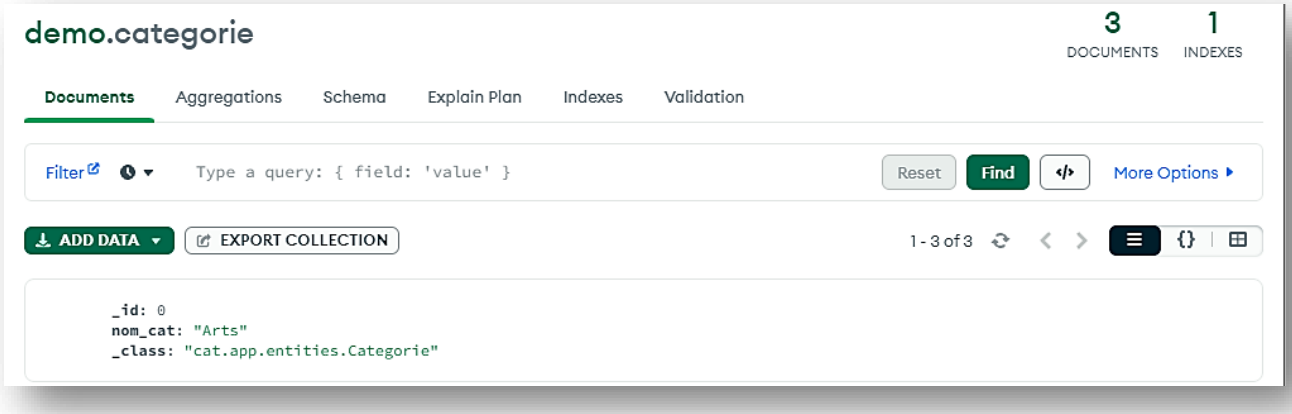
MongoDB est une base de données NoSQL, c'est-à-dire qu'elle ne suit pas la structure de données traditionnelle basée sur des tables, des lignes et des colonnes utilisées dans les systèmes de gestion de bases de données relationnelles (RDBMS). Au lieu de cela, MongoDB utilise une structure de données documentaire, où les données sont stockées sous forme de documents au format BSON (Binary JSON) qui peuvent être facilement interprétés par les applications. Cette structure de données est plus flexible et adaptée aux besoins des applications modernes.

Ici notre Base de données s'appelé « demo » :



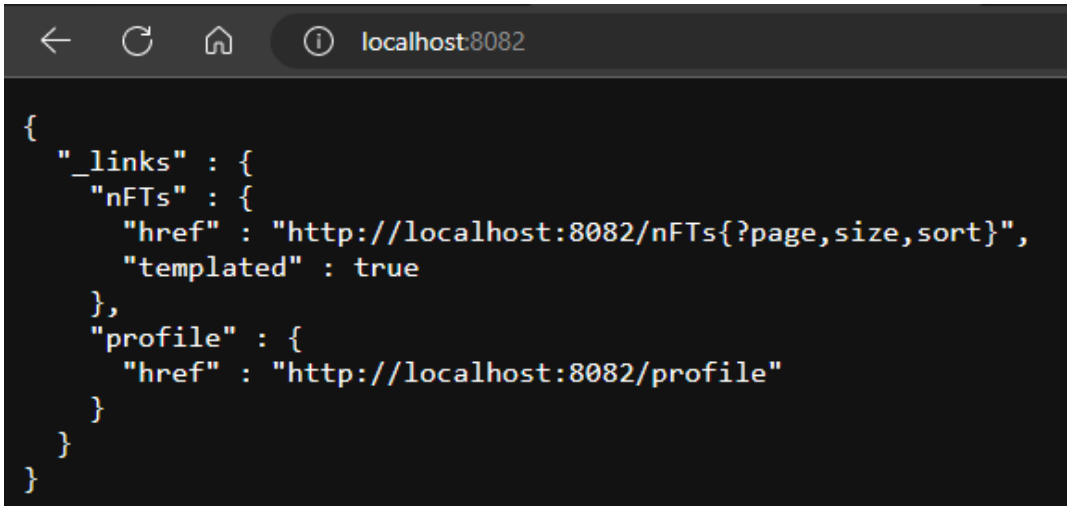

MongoDB Compass - localhost:27017/demo
Connect View Help
localhost:27017
My Queries
Databases
Search
admin
config
demo
NFT
categorie
local
Collections
Create collection
View
Sort by
Collection Name
categorie
Storage size: 20.48 kB
Documents: 3
Avg. document size: 76.00 B
Indexes: 1
Total index size: 24.58 kB
NFT
Storage size: 20.48 kB
Documents: 3
Avg. document size: 197.00 B
Indexes: 1
Total index size: 36.86 kB

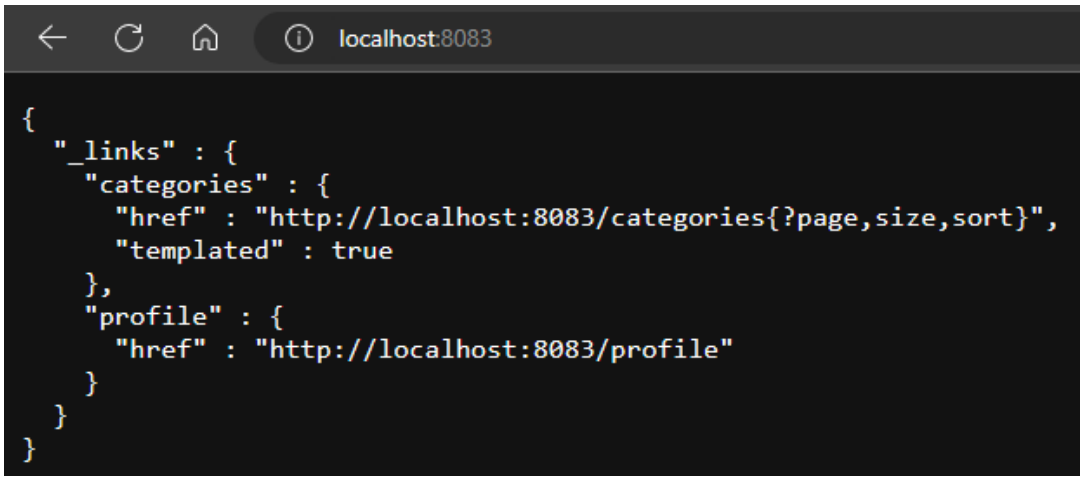

demo.NFT
0 DOCUMENTS
1 INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
Type a query: { field: 'value' }
Reset
Find
More Options
ADD DATA
EXPORT COLLECTION
1 - 2 of 2
_id: 2
nom_nft: "monkey"
description: "monkey_monkey"
prix: 100
image: "image.png"
cat: "Animals"
_class: "nft.app.entities.NFT"


demo.categorie
3 DOCUMENTS
1 INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
Type a query: { field: 'value' }
Reset
Find
More Options
ADD DATA
EXPORT COLLECTION
1 - 3 of 3
_id: 0
nom_cat: "Arts"
_class: "cat.app.entities.Categorie"

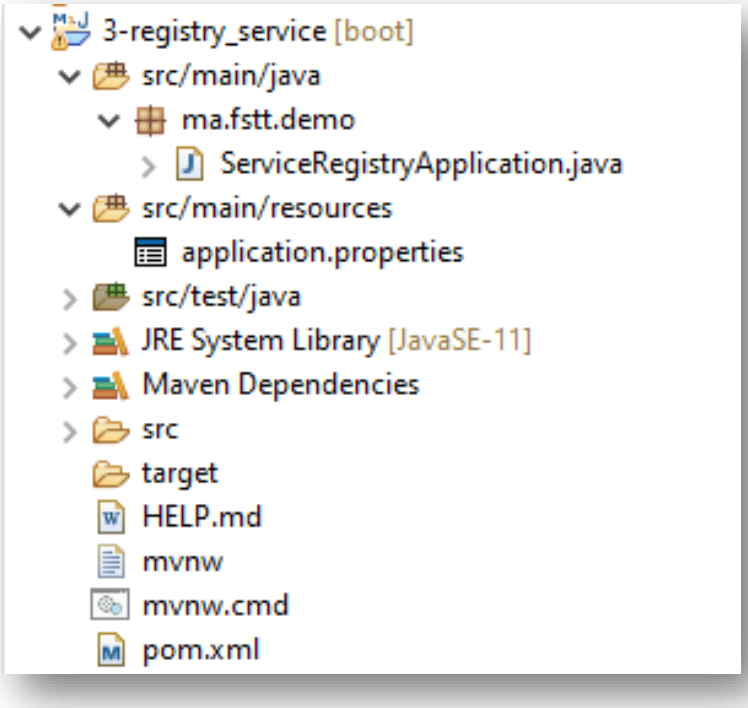
Remarque :

Pour implémenter le service NFT (de même pour les autres service) il faut l'exécuter et accéder en utilisant le port qu'on a choisir pour le service, ce qui fournit une mauvaise gestion des microservices, mais le service Registre résolue se problème !





2) Le microservice: “registry-service".



L'utilisation d'un registre de services (comme Eureka, qui est fourni par Spring Cloud Netflix) avec Spring Boot peut offrir plusieurs avantages pour les applications basées sur des microservices.

Un des avantages principaux est la découverte de service automatique. Les services enregistrés dans le registre peuvent être automatiquement découverts par d'autres services, ce qui facilite la communication entre les différents composants de l'application. Cela permet également de gérer les dépendances entre les services de manière centralisée.

3.1 ServiceRegistryApplication

*ServiceRegistryApplication.java

```java
package ma.fstt.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceRegistryApplication.class, args);
    }
}
```

@EnableEurekaServer : est utilisée pour que notre application Spring Boot (registry-server) agisse comme un serveur Eureka.

3.1 Application properties

application.properties

```
server.port = 8761
eureka.client.register-with-eureka= false
eureka.client.fetch-registry= false
```

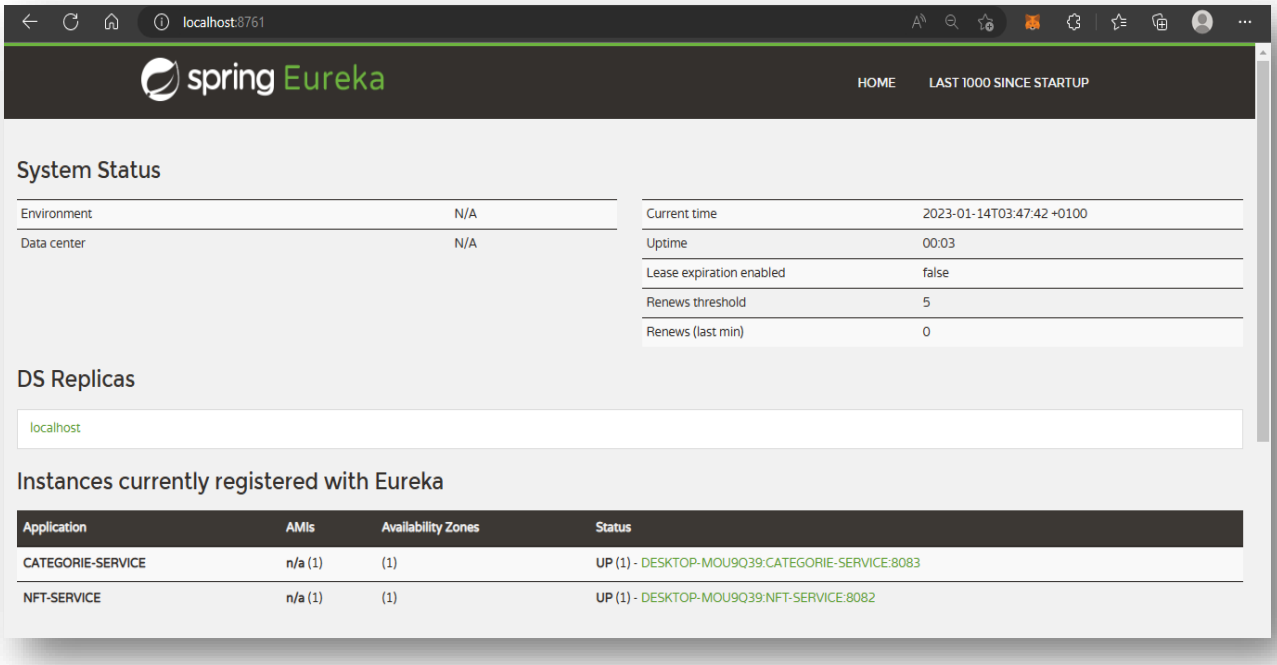
3.2 Pom.xml

3-registry_service/pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

La dépendance *eureka-server* permet aux différents services de l'application de s'enregistrer auprès du serveur et de s'y faire connaître. Les autres services peuvent ensuite utiliser Eureka pour découvrir les services enregistrés et les utiliser pour communiquer entre eux.

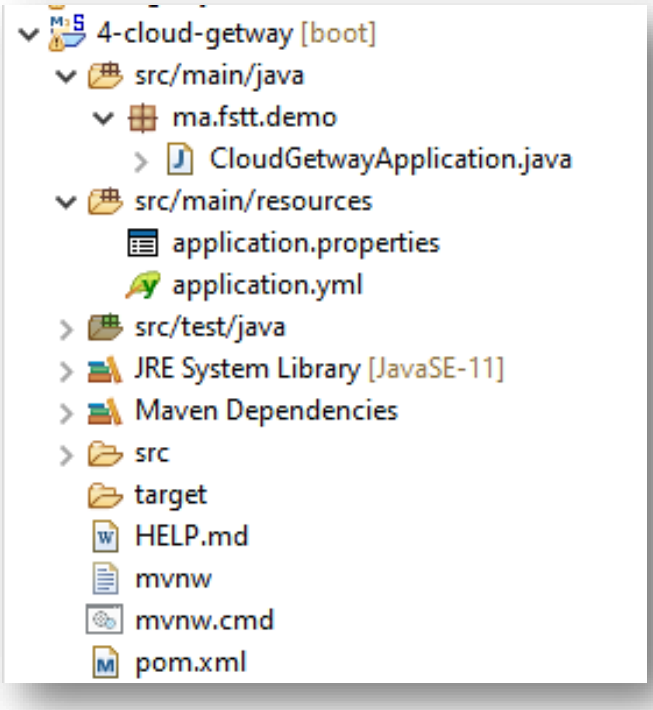
3.3 L'exécution de service Register :



On peut accéder à nos service (catégorie ou NFT) à partir de cette interface de Eureka

Application	AMIs	Availability Zones	Status
CATEGORIE-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-MOU9Q39:CATEGORIE-SERVICE:8083
NFT-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-MOU9Q39:NFT-SERVICE:8082

3) Le microservice: “getway-service".



Le service Gateway ou passerelle de service, est un composant logiciel qui sert de point d'entrée unique pour les services d'une application basée sur des microservices. Il permet de regrouper les fonctionnalités de différents services en une seule API.

4.1 CloudGetwayApplication

CloudGetwayApplication.java

```java
package ma.fstt.demo;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableEurekaClient
public class CloudGetwayApplication {

    public static void main(String[] args) {
        SpringApplication.run(CloudGetwayApplication.class, args);
    }

}
```

@EnableEurekaApplication : est utilisée pour que le service Getway agisse comme un client de serveur Eureka

4.2 Application.yml

application.yml

```yaml
server:
  port: 9191

spring:
  application:
    name: API-GATWAY
  cloud:
    gateway:
      routes:
      - id: NFT-SERVICE
        uri: lb://NFT-SERVICE
        predicates:
        - Path=/nft/**
      - id: CATEGORIE-SERVICE
        uri: lb://CATEGORIE-SERVICE
        predicates:
        - Path=/cat/**
      - id: PANIER-SERVICE
        uri: lb://PANIER-SERVICE
        predicates:
        - Path=/panier/**
```

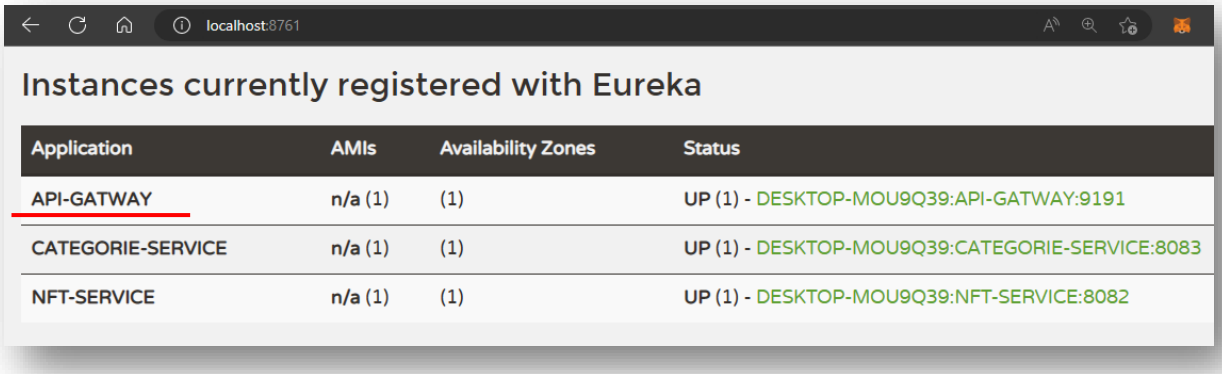
4.3 Pom.xml

-cloud-getway/pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

La dépendance *spring-cloud-starter-getway* est utilisé pour …

4.4 L'exécution du service Getway



I. Smart Contracts :

1) Les blockchaines :

- La blockchain est un registre numérique distribué qui stocke une série de transactions cryptographiquement sécurisées appelées blocs. Chacun de ces blocs est lié aux blocs précédents, créant ainsi une chaîne de blocs (blockchain) qui rend difficile la modification des données historiques. La technologie blockchain est souvent utilisée pour créer des systèmes de registre distribué tels que les crypto-monnaies, les systèmes de vote en ligne et les systèmes de stockage de données décentralisés.

1.1 Ethereum :

- Ethereum est une blockchain open-source qui permet la création de smart contracts et l'exécution de programmes décentralisés sur sa plateforme. Il a été lancé en 2015 par Vitalik Buterin et a été conçu pour être un système plus généraliste que Bitcoin, qui était principalement conçu pour les transactions financières.

- Ethereum utilise sa propre crypto-monnaie appelée Ether (ETH) pour payer les frais de transaction et les frais de calcul des contrats intelligents. Les utilisateurs peuvent également créer leurs propres tokens personnalisés sur la plateforme Ethereum, appelés tokens ERC-20, qui peuvent être utilisés pour représenter des actifs numériques tels que des jetons de jeux vidéo, des titres de propriété et des jetons de sécurité.

- Ethereum est une plateforme de développement puissante qui offre une grande flexibilité pour la création de contrats intelligents et d'applications décentralisées.

2) Les Smart Contracts :

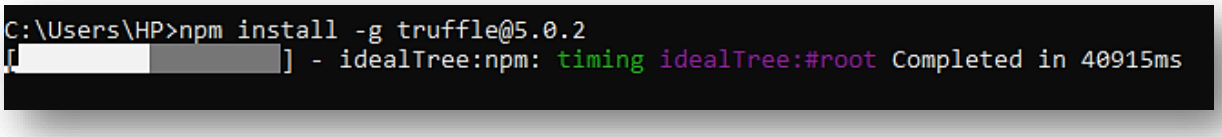
- Les smart contracts sont des programmes informatiques qui exécutent automatiquement les termes d'un contrat numérique lorsque certaines conditions prédéfinies sont remplies. Ils sont souvent utilisés pour automatiser les transactions financières, les transferts de propriété et d'autres types de contrats.
- Les smart contrats sont souvent exécutés sur une blockchain car cela offre des avantages tels que la transparence, l'immuabilité et la sécurité accrue. Les smart contrats peuvent utiliser les informations stockées dans la blockchain pour déclencher des actions automatisées lorsque certaines conditions sont remplies.

Pour créer le smart Contrat nous avons tester plusieurs outils afin de maitriser **SOLIDTY** :

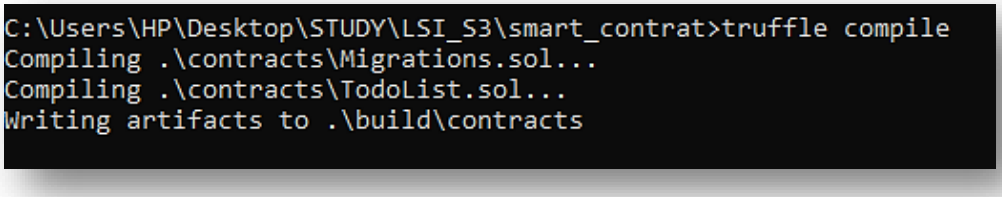
2.1 « Truffle »

Premièrement nous avons utilisé « Truffle » comme un framework open-source pour la création et le déploiement de smart contracts.

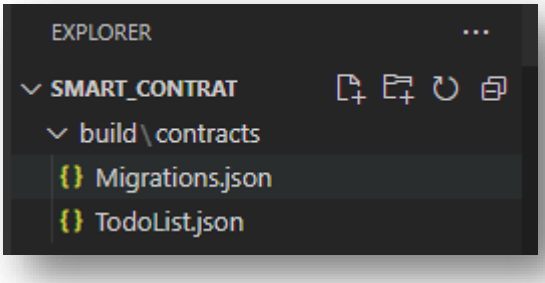
- Installation de Truffle :



- Compile Truffle via la commande :



❖ Résultat :



❖ Écrire le code de votre contrat intelligent dans le langage Solidity et le placer dans le dossier "contracts". Déployer les contrats sur la blockchain en utilisant la commande "truffle migrate" pour exécuter les migrations définies dans les fichiers de migration.

```
C:\Users\HP\Desktop\STUDY\LSI_S3\smart_contrat>truffle migrate
Compiling .\contracts\TodoList.sol...
Writing artifacts to .\build\contracts

⚠️  Important ⚠️
If you're using an HDWalletProvider, it must be Web3 1.0 enabled or your migration will hang.


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0xdad48a8f2ebfbfd6100de69af16aa8d55690004e767fe6069cdeb6377cc1dfdb
   > Blocks: 0            Seconds: 0
   > contract address:    0x54C3Bdd7Ec2EF533f5B118874277Bf4e58aEAc89
   > account:             0x11cD2939AdF456212947ABD7be2A4DD8B3eb19D9
   > balance:             99.99586798
   > gas used:            206601
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00413202 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00413202 ETH


2_deplay_contracts.js
=====================
ReferenceError: Migrations is not defined
    at module.exports (C:\Users\HP\Desktop\STUDY\LSI_S3\smart_contrat\migrations\2_deplay_contracts.js:4:19)
    at C:\Users\HP\AppData\Roaming\npm\node_modules\truffle\build\webpack:\packages\truffle-migrate\migration.js:59:1
    at C:\Users\HP\AppData\Roaming\npm\node_modules\truffle\build\webpack:\packages\truffle-require\require.js:101:1
    at FSReqCallback.readFileAfterClose [as oncomplete] (node:internal/fs/read_file_context:68:3)
Truffle v5.0.2 (core: 5.0.2)
```

2.2 « Ganache »

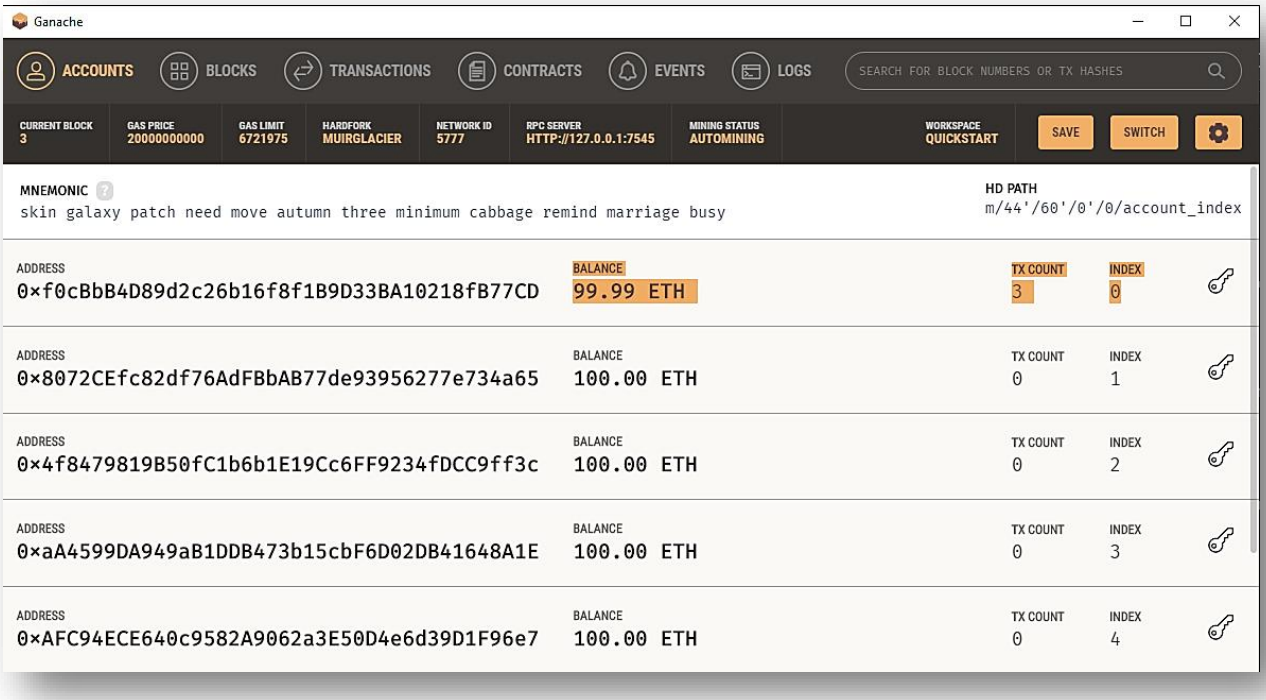
- Téléchargez et installez Ganache sur votre ordinateur.

Ganache est un client Ethereum qui permet de **simuler une blockchain locale pour les tests et les développements**. Il peut être utilisé avec Truffle pour faciliter la création et le déploiement de smart contracts.

- Lancez Ganache et créez un nouveau réseau en cliquant sur le bouton "Quickstart". Cela créera un ensemble de comptes de développement pré-remplis avec des Ethers pour les tests.

- Copiez l'URL de RPC (Remote Procedure Call) de Ganache dans le champ "Server" de Truffle. Cela permettra à Truffle de se connecter à la blockchain de Ganache pour les tests et les déploiements.

Voici ici, Comme la Transaction était bien passer du Truffle vers La blockchain Ganche on constate que la Balance d'Ethereum du premier utilisateur est diminué.



- Interagir avec les contrats déployés en utilisant les bibliothèques de Truffle pour envoyer des transactions et appeler les fonctions du contrat.

```
C:\Users\HP\Desktop\STUDY\LSI_S3\smart_contrat>truffle console
truffle(development)> todoList=await TodoList.deployed()
undefined
truffle(development)> todoList
TruffleContract {
  constructor: [Function: TruffleContract] {
    _constructorMethods: {
      setProvider: [Function: setProvider],
      new: [Function: new],
      at: [Function: at],
      deployed: [Function: deployed],
      defaults: [Function: defaults],
      hasNetwork: [Function: hasNetwork],
      isDeployed: [Function: isDeployed],
      detectNetwork: [Function: detectNetwork],
      setNetwork: [Function: setNetwork],
      setWallet: [Function: setWallet],
      resetAddress: [Function: resetAddress],
      link: [Function: link],
      clone: [Function: clone],
      addProp: [Function: addProp],
      toJSON: [Function: toJSON],
      decodeLogs: [Function: decodeLogs]
    },
    _properties: {
      contract_name: [Object],
      contractName: [Object],
truffle(development)>
      timeoutBlocks: [Object],
      autoGas: [Object],
      numberFormat: [Object],
      abi: [Object],
      network: [Function: network],
      networks: [Function: networks],
      address: [Object],
```

Voici l'adresse :

```
truffle(development)> todoList.address
'0x2aD3B277A31e559e89963D3ba9A6639f77E759aF'
truffle(development)>
```

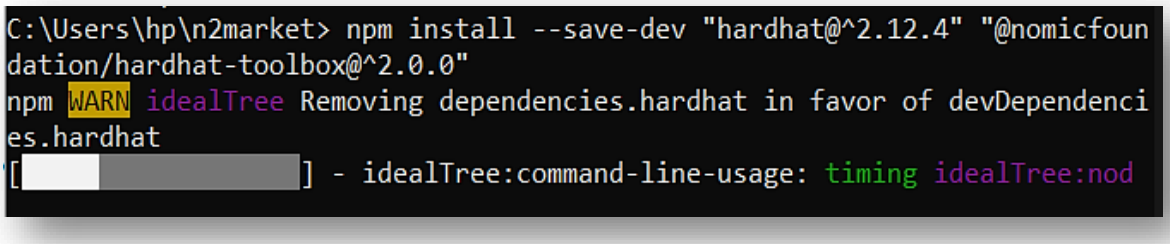
2.3 « Hardhat »

Or on a constaté que d'après les outils mentionnés dans le projet, On trouve **Hardhat** c'est pourquoi on a décidé d'inverser le choix et d'utiliser aussi Hardhat

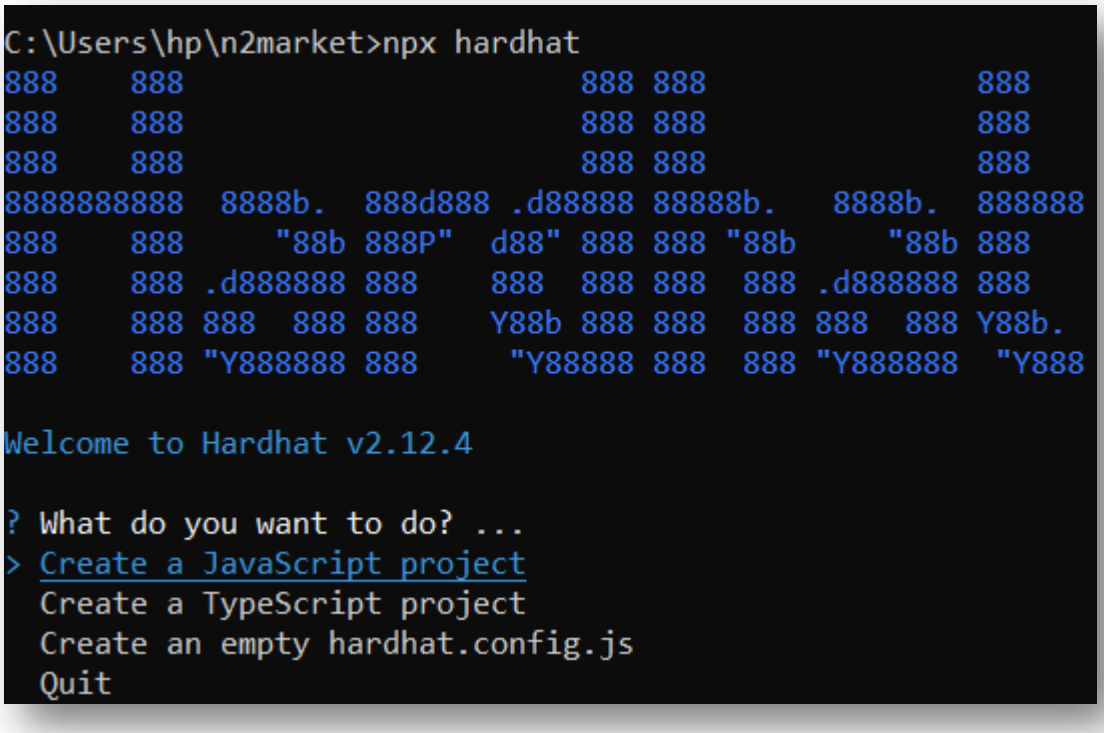
Hardhat offre des fonctionnalités similaires à Truffle, comme la compilation et le déploiement de contrats intelligents, ainsi que des outils pour les tests et la gestion des erreurs.

Hardhat nécessite des outils comme **Node.js** et **npm** pour fonctionner !

Installation de Hardhat en utilisant la commande :



On exécute la commande suivante pour vérifier si Hardhat est bien installé :



On installe les dépendances nécessaires :

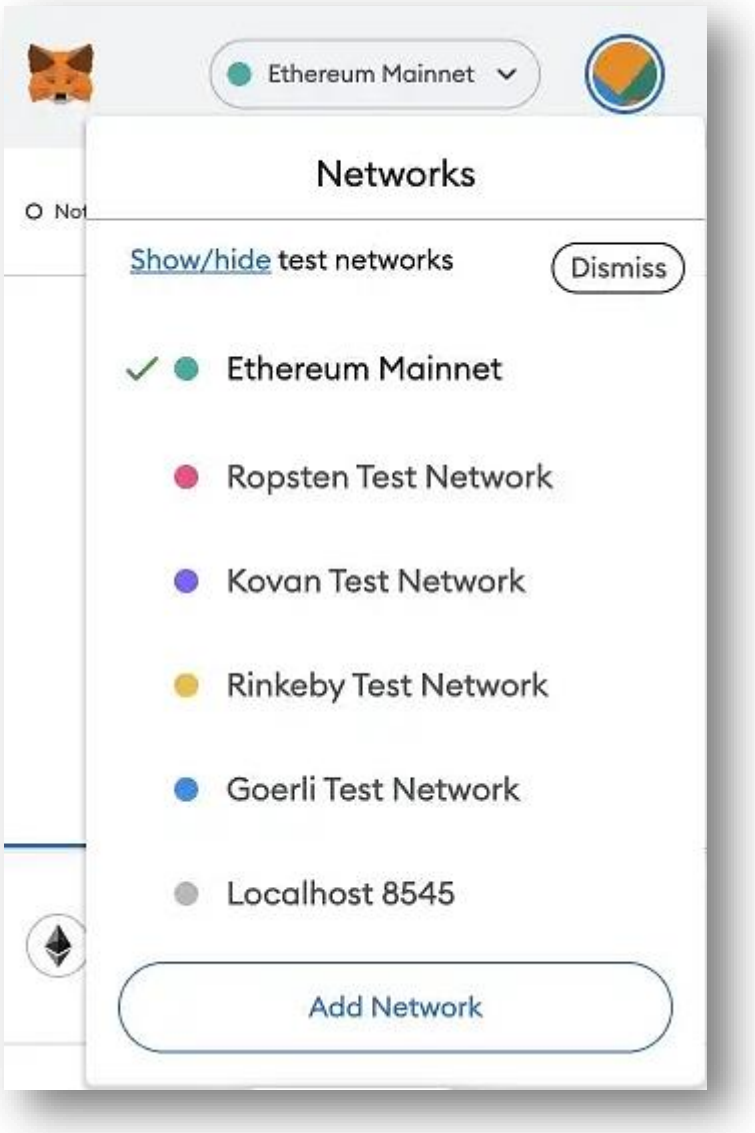
```
√ What do you want to do? · Create a JavaScript project
√ Hardhat project root: · F:\bureau\HARDHAT
√ Do you want to add a .gitignore? (Y/n) · y

You need to install these dependencies to run the sample project:
  npm install --save-dev "hardhat@^2.12.4" "@nomicfoundation/hardhat-toolbox@^2.0.0"

Project created

See the README.md file for some example tasks you can run
```

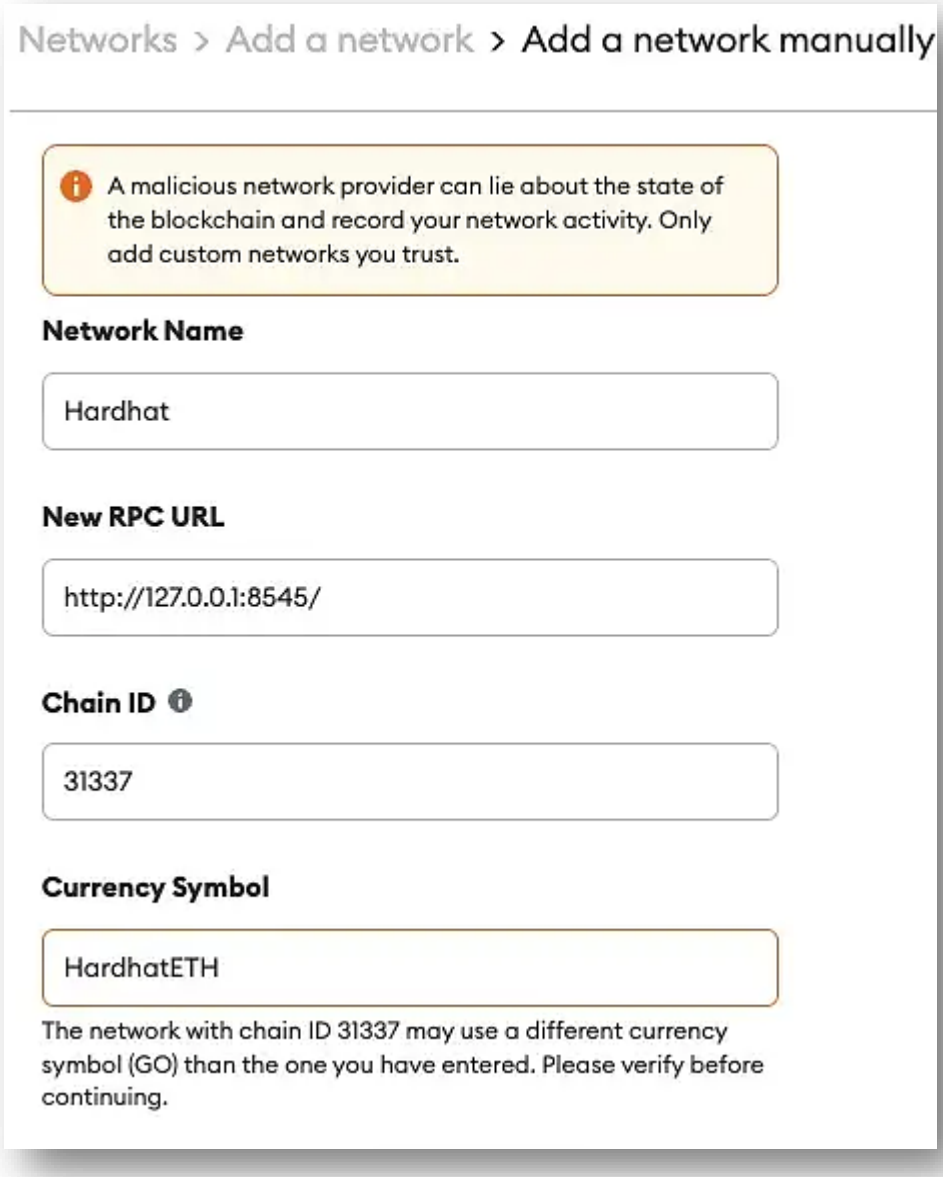
```
F:\bureau\HARDHAT>npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
========

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8 (10000 ETH)
Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC (10000 ETH)
Private Key: 0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a

Account #3: 0x90F79bf6EB2c4f870365E785982E1f101E93b906 (10000 ETH)
Private Key: 0x7c852118294e51e653712a81e05800f419141751be58f605c371e15141b007a6

Account #4: 0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65 (10000 ETH)
Private Key: 0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a

Account #5: 0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc (10000 ETH)
Private Key: 0x8b3a350cf5c34c9194ca85829a2df0ec3153be0318b5e2d3348e872092edffba
```

Ajouter un réseau local a MetaMask

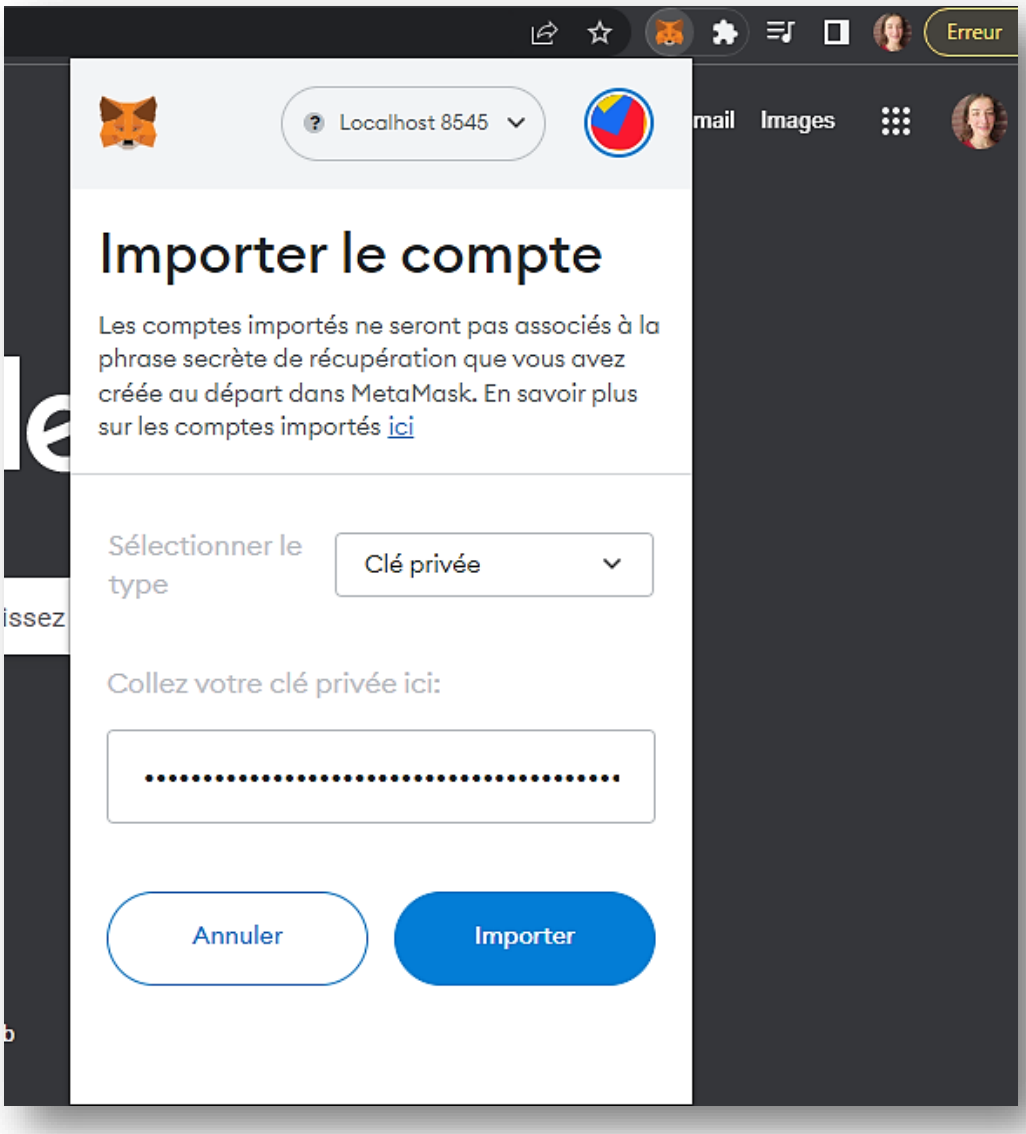
Avec notre réseau local Hardhat en cours d'exécution, nous pouvons ensuite configurer notre Metamask pour y connecter.

Dans un navigateur avec Metamask installé, sélectionnez le menu déroulant de réseau (ce sera probablement le menu déroulant avec "Ethereum Mainnet" listé). Notez que vous devrez avoir activé "Afficher les réseaux de test" pour afficher la liste complète, comme indiqué ci-dessous sur la capture d'écran."

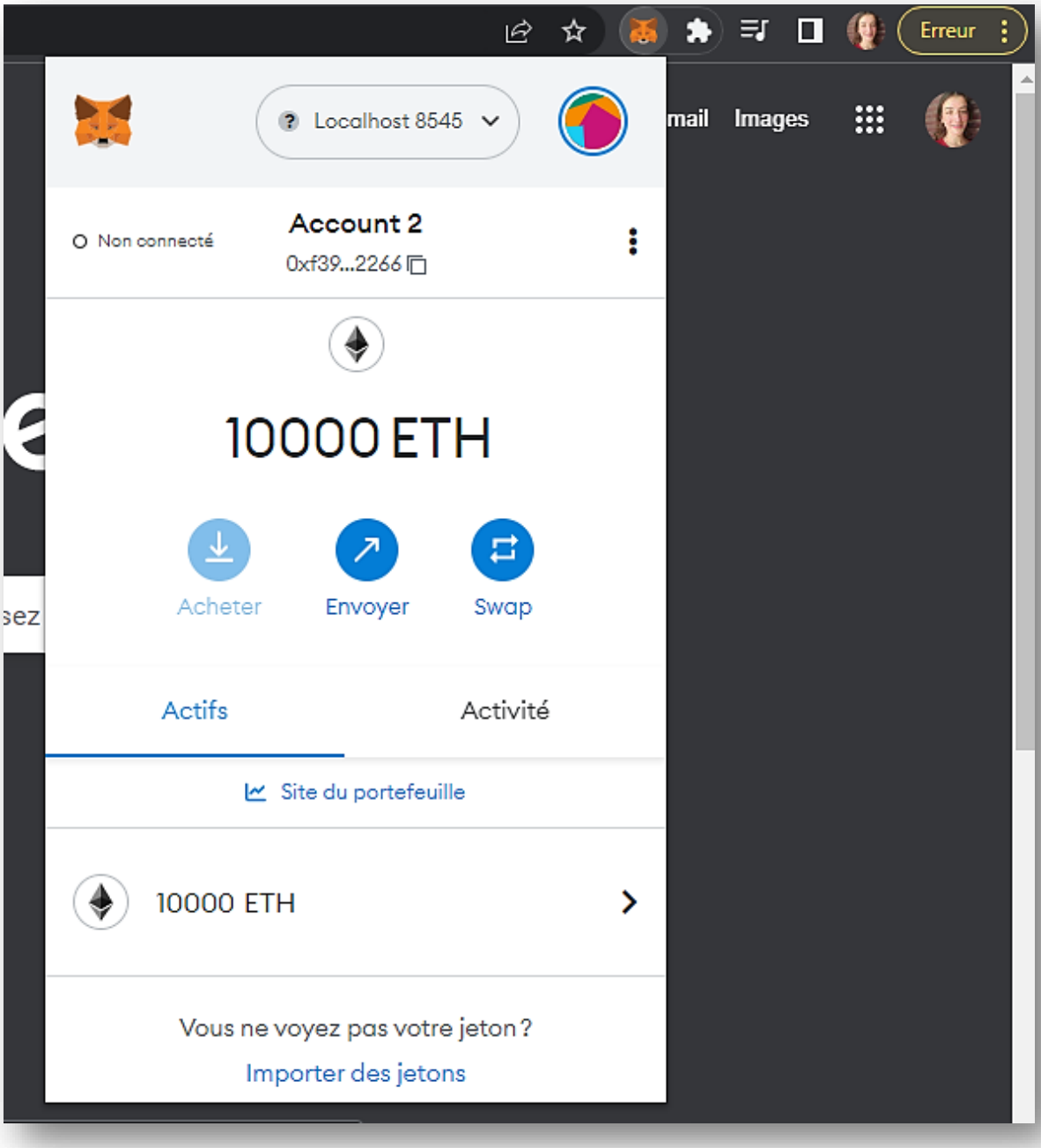



Networks > Add a network > Add a network manually
A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust.
Network Name
Hardhat
New RPC URL
http://127.0.0.1:8545/
Chain ID
31337
Currency Symbol
HardhatETH
The network with chain ID 31337 may use a different currency symbol (GO) than the one you have entered. Please verify before continuing.

ISélectionnez l'option "Importer un compte" et Metamask vous demandera la clé privée. Collez les clés privées précédemment obtenues



Une fois importé, un compte avec l'adresse correspondante sera ajouté à Metamask. Vous devriez également pouvoir voir 10000 HARDHATETH dans le compte importé




```
C:\Users\HP\Desktop\STUDY\LSI_S3\Tutoriel_1\nft>npx hardhat console
Downloading compiler 0.8.17
Compiled 1 Solidity file successfully
Welcome to Node.js v18.12.1.
```

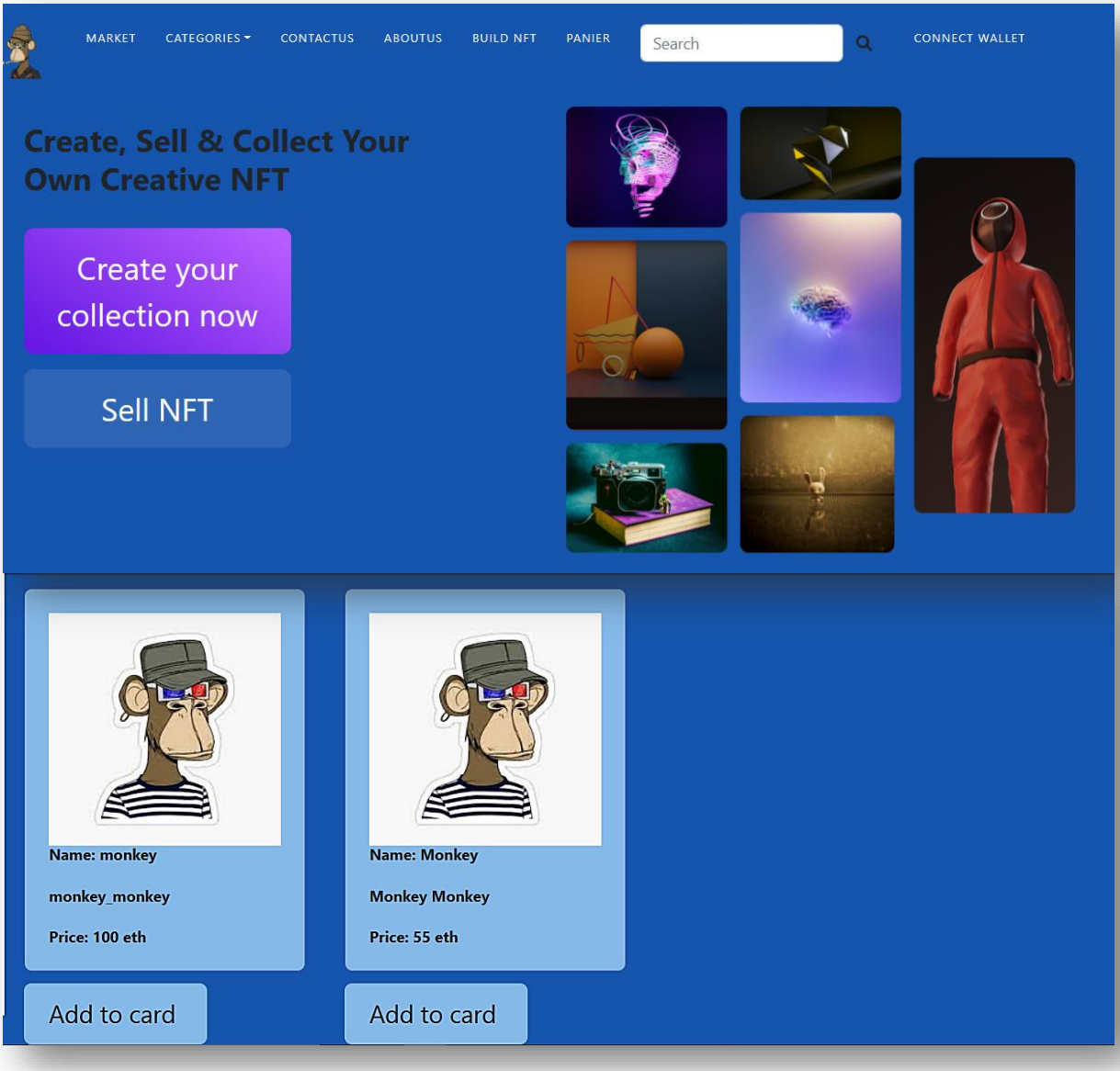
```
C:\Users\HP\Desktop\STUDY\LSI_S3\Hardhat-EthersJs>npx create-next-app n2dmarket
Need to install the following packages:
  create-next-app@13.1.1
Ok to proceed? (y) y
√ Would you like to use TypeScript with this project? ... No / Yes
√ Would you like to use ESLint with this project? ... No / Yes
Creating a new Next.js app in C:\Users\HP\Desktop\STUDY\LSI_S3\Hardhat-EthersJs\n2dmarket.

Using npm.

Installing dependencies:
- react
- react-dom
- next
- @next/font
- typescript
- @types/react
- @types/node
- @types/react-dom

[                    ] \ idealTree:n2dmarket: sill idealTree buildDeps
```

```
> const t4 = new ethers.providers.JsonRpcProvider( "HTTP://172.27.224.1:8545");
undefined
> t4
JsonRpcProvider {
  _isProvider: true,
  _events: [],
  _emitted: { block: -2 },
  disableCcipRead: false,
  formatter: Formatter {
    formats: {
      transaction: [Object],
      transactionRequest: [Object],
      receiptLog: [Object],
      receipt: [Object],
      block: [Object],
      blockWithTransactions: [Object],
      filter: [Object],
      filterLog: [Object]
    }
  },
  anyNetwork: false,
  _networkPromise: Promise {
    <pending>,
    [Symbol(async_id_symbol)]: 4686,
    [Symbol(trigger_async_id_symbol)]: 13
  },
  _maxInternalBlockNumber: -1024,
  _lastBlockNumber: -2,
  _maxFilterBlockRange: 10,
  _pollingInterval: 4000,
  _fastQueryDate: 0,
  connection: { url: 'HTTP://172.27.224.1:8545' },
  _nextId: 43,
  _eventLoopCache: { detectNetwork: null, eth_chainId: null }
}
```

II. Front end :

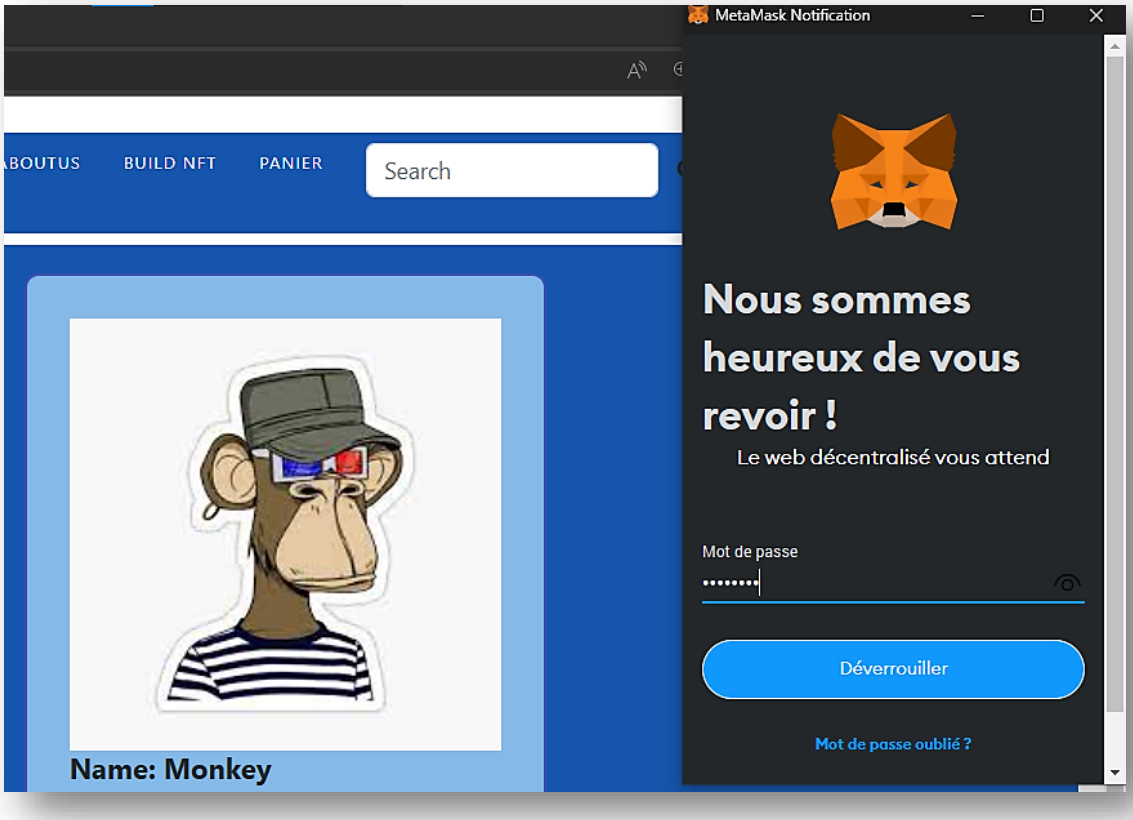
1. Page home :

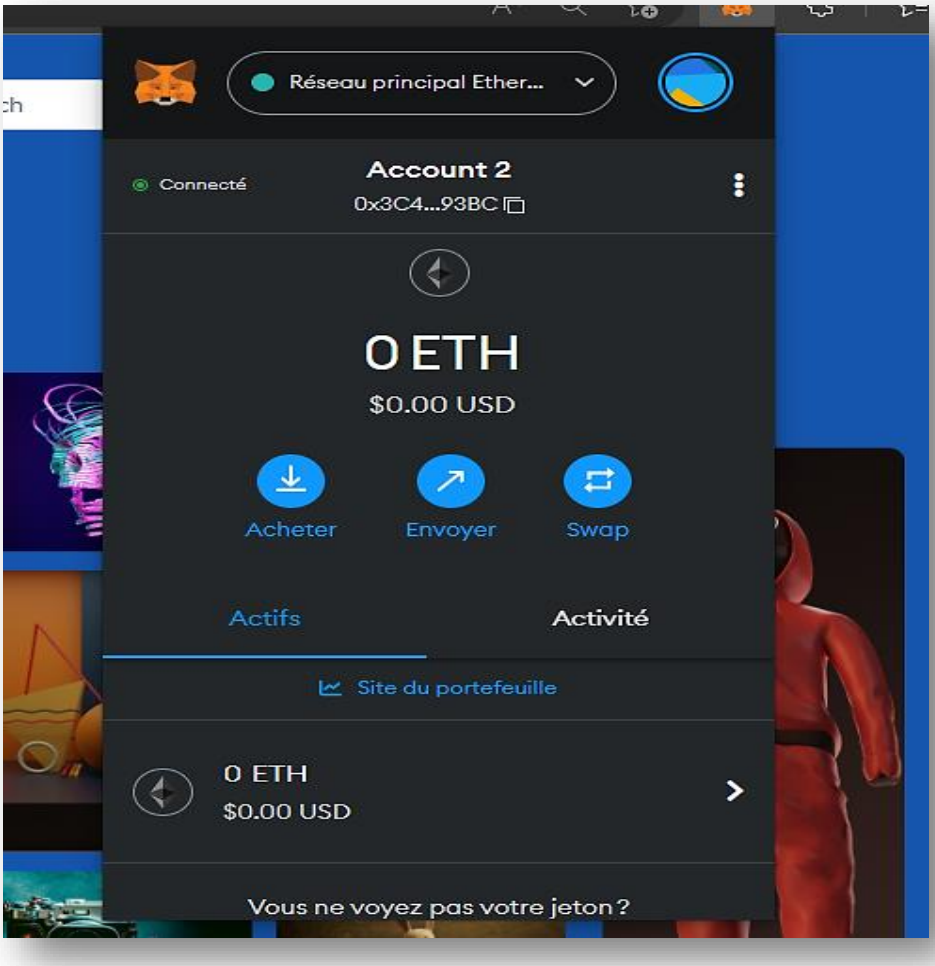
C’est la première page qui s’affiche après l’exécution de l’application, c’est une interface d’affichage des NFTs :



2. Authentification

Pour créer un NFT il faut d'abord s'authentifier !






```
<li>
  <a *ngIf="!walletConnected" aria-expanded="true" (click)="connectToWallet()">
    Connect Wallet
  </a>
</li>
```

Le service Wallet qui contient les fonctions de vérification et connexion à Metamask

```
PS F:\bureau\nft-front_tarik> ng g service services/wallet
CREATE src/app/services/wallet.service.spec.ts (357 bytes)
CREATE src/app/services/wallet.service.ts (135 bytes)
PS F:\bureau\nft-front_tarik>
```

wallet.service.ts

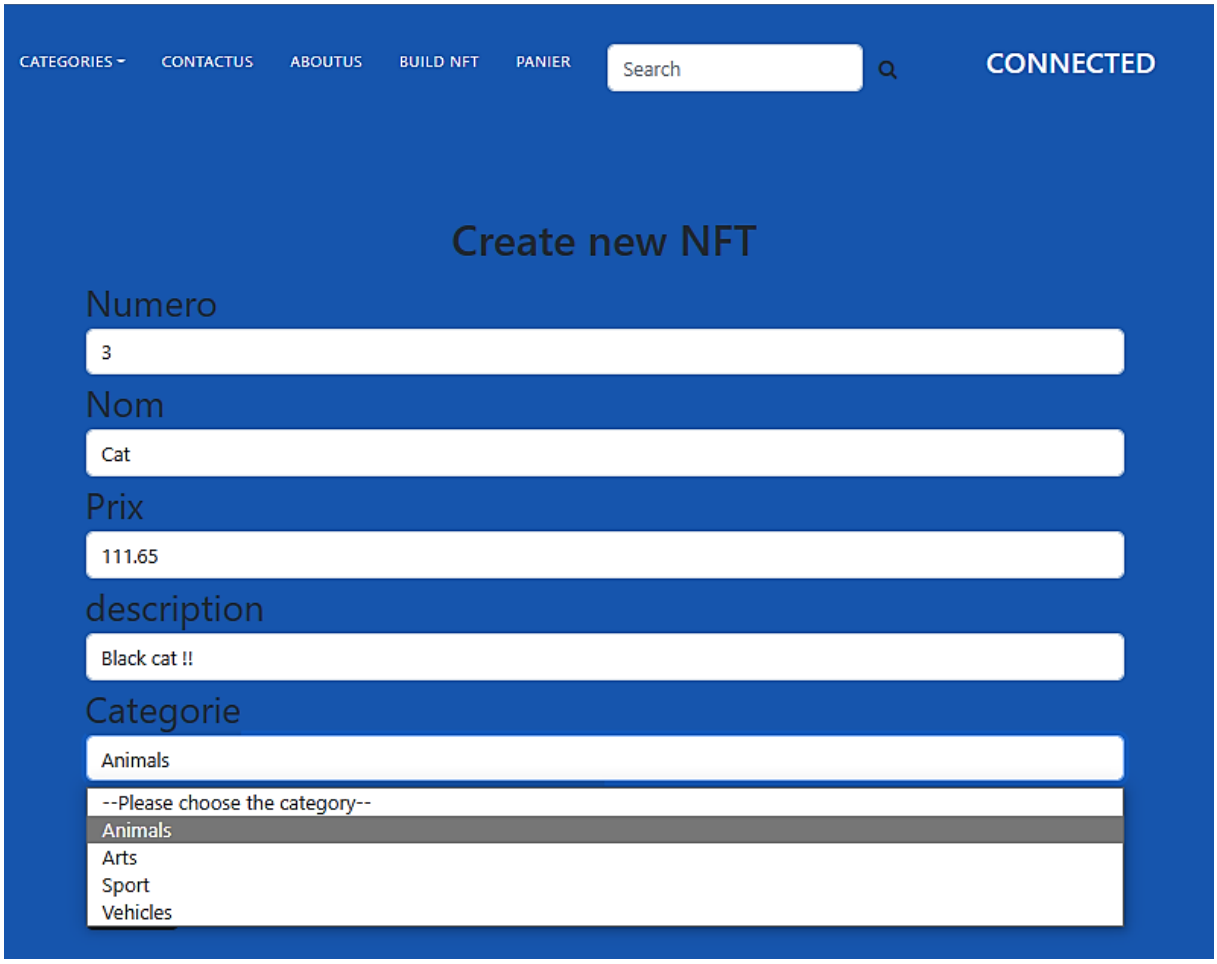
src > app > services > wallet.service.ts > WalletService > checkWalletConnected

```ts
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class WalletService {
  public ethereum;
  constructor() {
    const {ethereum} = <any>window
    this.ethereum = ethereum
   }

  public connectWallet = async () => {
    try{
      if(!this.ethereum) return alert("Please install meta mask");
      const accounts = await this.ethereum.request({method: 'eth_requestAccounts'});
    }
    catch(e){
      throw new Error("No thereum object found")
    }
  }
```

```ts
  public checkWalletConnected = async () => {
    try{
      if(!this.ethereum) return alert("Please install meta mask ")
      const accounts = await this.ethereum.request({method: 'eth_accounts'});
      return accounts;
    }
    catch(e){
      throw new Error("No ethereum object found");
    }
  }
}
```

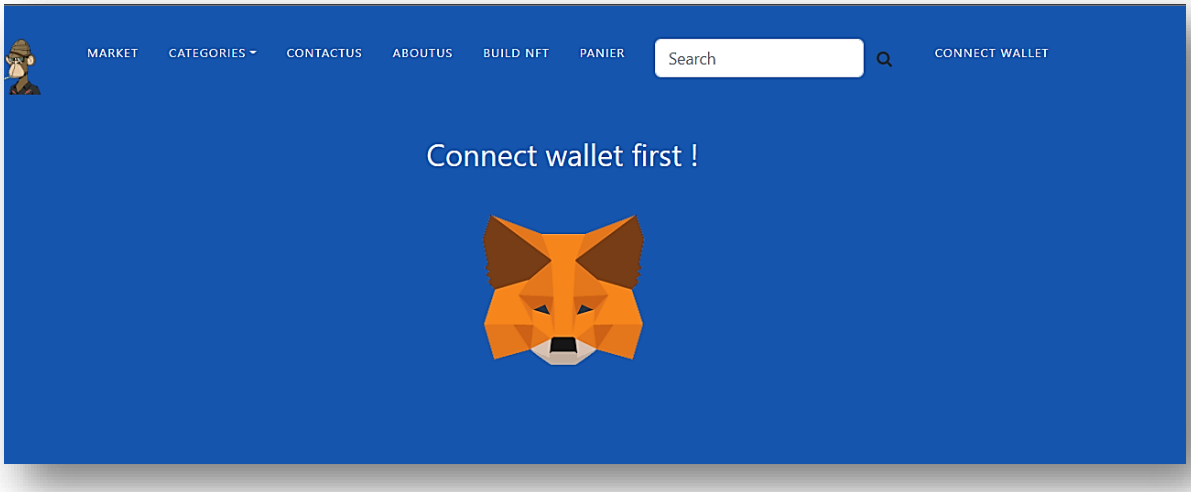
3. BuildNFT

Le formulaire de création d'un nouvel NFT :

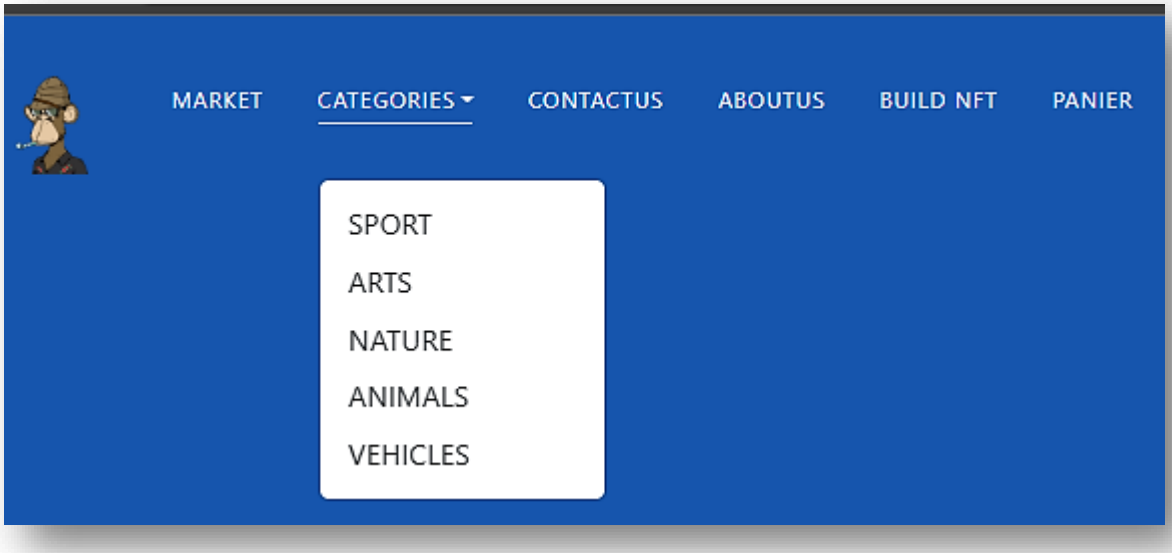



```
_id: 3
nom_nft: "Cat"
description: "Black cat !!"
prix: 111.65
image: "C:\fakepath\Capture d'écran 2023-01-11 232003.png"
cat: "dog"
_class: "nft.app.entities.NFT"
```

Si l'utilisateur n'a pas connecté à son Wallet, il n'a pas le droit de créer un NFT !

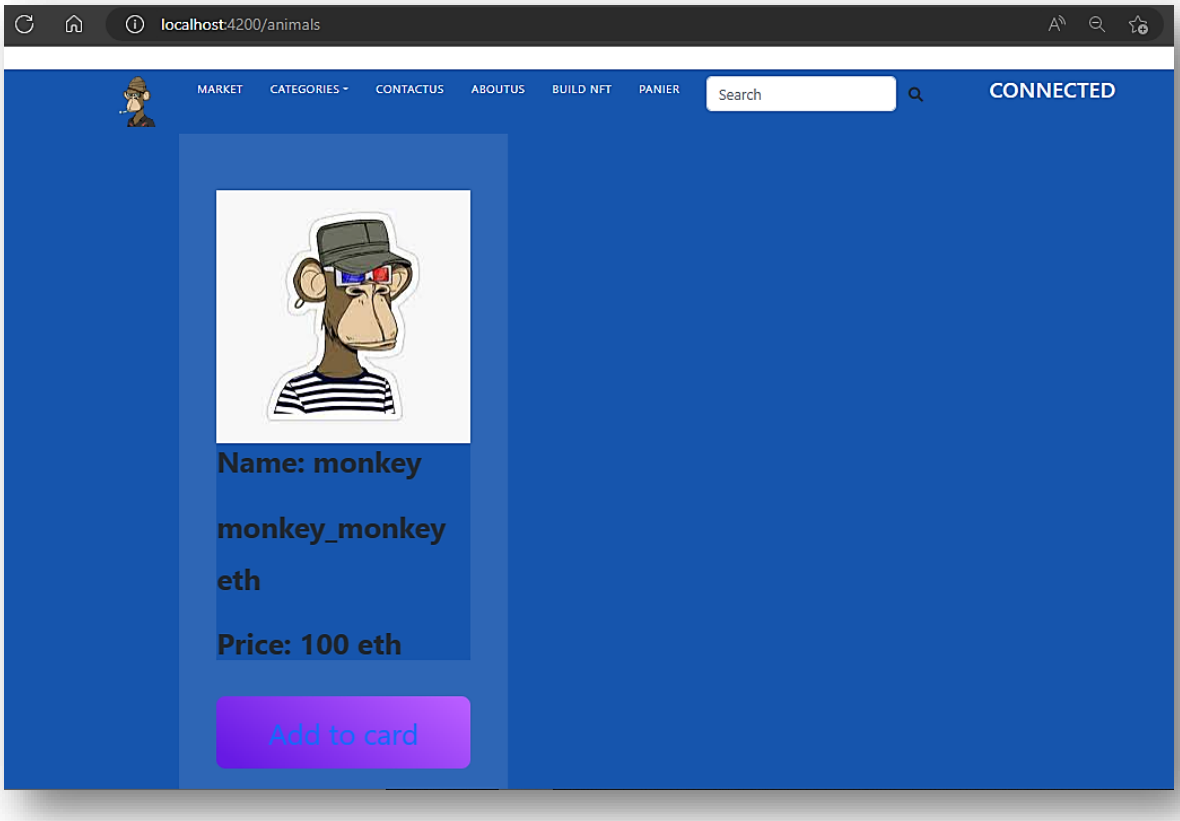


4. Catégories

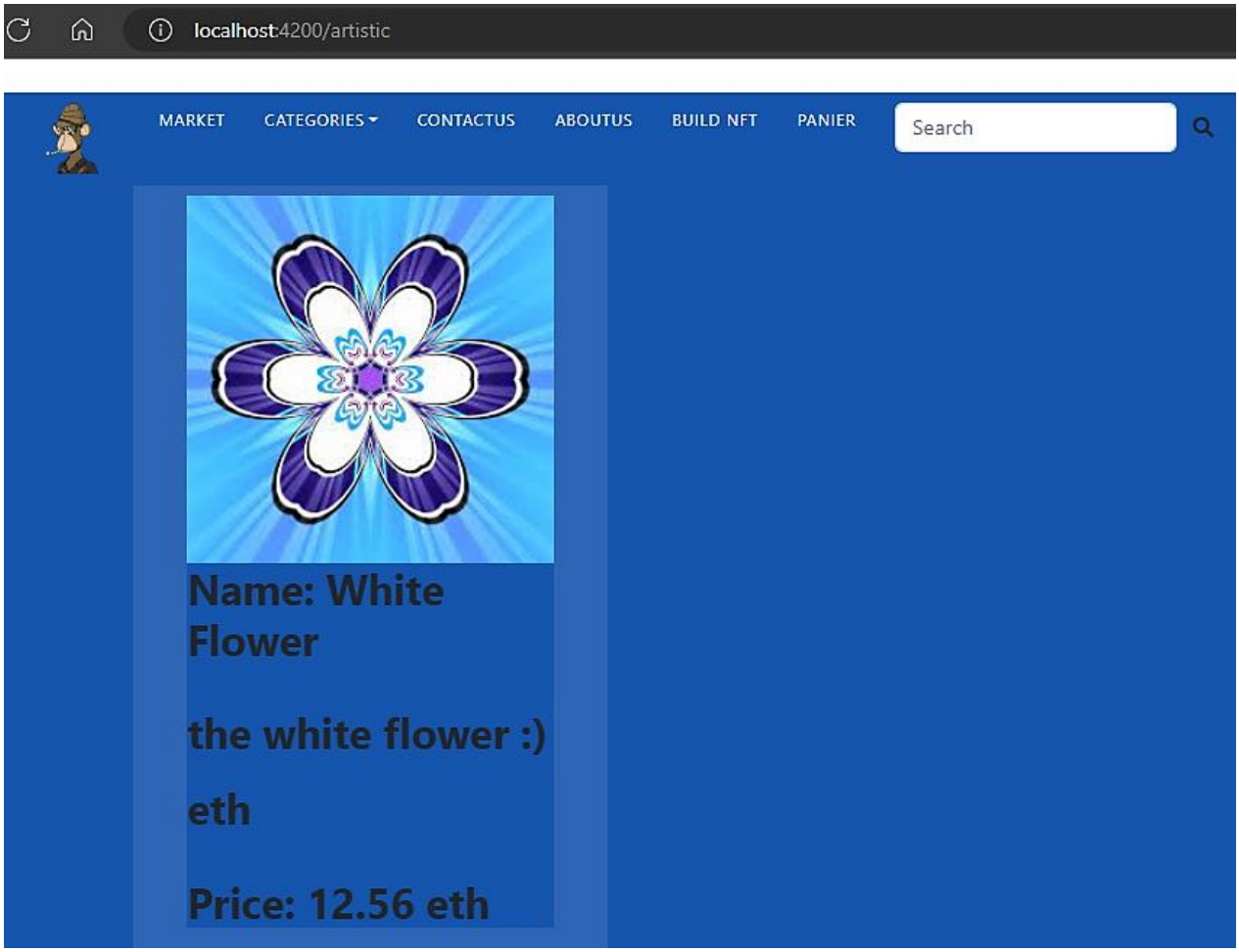


Filtrage des NFTs selon les catégories

- Animals



❖ Arts


localhost:4200/artistic
MARKET
CATEGORIES
CONTACTUS
ABOUTUS
BUILD NFT
PANIER
Search
Name: White Flower
the white flower :)
eth
Price: 12.56 eth

Conclusion

Développer une marketplace NFT (Non-Fungible Token) en utilisant les technologies de blockchain et de smart contract a été une opportunité passionnante pour notre équipe. Nous avons pu explorer les possibilités offertes par la blockchain pour créer une plateforme transparente, sécurisée et décentralisée pour l'échange de tokens NFT uniques. Nous avons également pu découvrir les possibilités des smart contracts pour automatiser les processus de vente et d'achat sur la plateforme, ainsi que pour garantir la propriété et la traçabilité des tokens NFT.

Ce projet a été une expérience enrichissante pour notre équipe, car il nous a permis de découvrir de nouvelles technologies et de les utiliser pour créer une solution innovante.