Comprehensive Deployment and Usage Guide

Prerequisites

Before you start, ensure you have the following:

- **Node.js** and **npm** installed
- **Truffle** or **Hardhat** development framework
- **MetaMask** or another Ethereum wallet
- **Infura** or another Ethereum node service provider
- **OpenZeppelin Contracts** library installed

Contract 1: AlignerNFT

Description

The `AlignerNFT` contract is an ERC721 token with additional functionalities such as minting, pausing, and URI management.

Deployment

1. **Install Dependencies**:

```
npm install @openzeppelin/contracts
```

2. **Create Deployment Script**:

   Create a new deployment script in your `migrations` folder, e.g., `2_deploy_alignernft.js`:

```
const AlignerNFT = artifacts.require("AlignerNFT");

module.exports = function (deployer, network, accounts) {
  const initialOwner = accounts[0]; // Replace with the desired owner
address
  deployer.deploy(AlignerNFT, initialOwner);
};
```

3. **Deploy Contract**:

```
truffle migrate --network <network-name>
```

Usage

For the Owner

1. **Minting Tokens**:

```
const instance = await AlignerNFT.deployed();
await instance.safeMint(recipientAddress, { from: ownerAddress });
```

2. **Minting Batch Tokens**:

```
await instance.safeMintBatch(recipientAddress, quantity, { from:
ownerAddress });
```

3. **Pausing and Unpausing**:

```
await instance.pause({ from: ownerAddress });
await instance.unpause({ from: ownerAddress });
```

4. **Setting Base URI**:

```
await instance.setBaseURI("https://example.com/api/token/", { from:
ownerAddress });
```

5. **Adding Minters**:

```
await instance.setMinter(minterAddress, true, { from: ownerAddress });
```

6. **Removing Minters**:

```
await instance.setMinter(minterAddress, false, { from: ownerAddress });
```

Contract 2: IWO

Description

The `IWO` contract is an ERC20 token with minting capabilities and supports EIP-2612 (permit).

Deployment

1. **Create Deployment Script**:

   Create a new deployment script in your `migrations` folder, e.g., `3_deploy_iwo.js`:

```
const IWO = artifacts.require("IWO");

module.exports = function (deployer, network, accounts) {
  const initialOwner = accounts[0]; // Replace with the desired owner
address
  deployer.deploy(IWO, initialOwner);
};
```

2. **Deploy Contract**:

```
truffle migrate --network <network-name>
```

Usage

For the Owner

1. **Minting Tokens**:

```
const instance = await IWO.deployed();
await instance.mint(recipientAddress, amount, { from: ownerAddress });
```

2. **Transferring Ownership**:

```
await instance.transferOwnership(newOwnerAddress, { from: ownerAddress });
```

Contract 3: MockUSDT

Description

The `MockUSDT` contract is a mock ERC20 token used for testing purposes.

Deployment

1. **Create Deployment Script**:

   Create a new deployment script in your `migrations` folder, e.g., `4_deploy_mockusdt.js`:

```
const MockUSDT = artifacts.require("MockUSDT");

module.exports = function (deployer) {
  deployer.deploy(MockUSDT);
};
```

2. **Deploy Contract**:

```
truffle migrate --network <network-name>
```

Usage

- **Initial Minting**: The initial mint is done during deployment. The deployer address receives all the minted tokens.

Additional Usage Examples

Checking Balances

For ERC20 tokens like `IWO` and `MockUSDT`:

```
const iwoInstance = await IWO.deployed();
const balance = await iwoInstance.balanceOf(userAddress);
console.log(balance.toString());

const usdtInstance = await MockUSDT.deployed();
const balance = await usdtInstance.balanceOf(userAddress);
console.log(balance.toString());
```

Comprehensive Guide for ProjectContract

Step-by-Step Guide

1. **Deploy Contracts**:

   Deploy the `IWO`, `MockUSDT`, `AlignerNFT`, and `ProjectContract` contracts as outlined in their respective sections.

2. **Create Project (Owner/Project Owner)**:

```
const projectInstance = await ProjectContract.deployed();
const biddingStartDate = Math.floor(Date.now() / 1000); // Current time in
seconds
const biddingDuration = 7 * 24 * 60 * 60; // 1 week

await projectInstance.createProject(
  "ProjectName",
  "ProjectDescription",
  "SocialInfo",
  biddingStartDate,
  biddingDuration,
  { from: ownerAddress } // or projectOwnerAddress
);
```

3. **Add Vesting Round (Owner/Project Owner)**:

```
const projectId = 1; // Replace with your project ID
const roundAmount = 1000000; // Replace with your desired round amount
const iwoPrice = 10; // Replace with your desired IWO price

await projectInstance.addVestingRound(projectId, roundAmount, iwoPrice, {
from: ownerAddress } // or projectOwnerAddress);
```

4. **Whitelist Addresses (Owner/Project Owner)**:

```
const userAddress = "0x123...abc"; // Replace with the actual user address
to whitelist

await projectInstance.addToWhitelist(projectId, userAddress, { from:
ownerAddress } // or projectOwnerAddress);
```

5. **Place Bids (Bidders)**:

```
const allocationSize = 1000; // USDT amount
const vestingLength = 6; // Vesting length in months

await projectInstance.placeBid(projectId, allocationSize, vestingLength, {
from: bidderAddress });
```

6. **End Bidding (Owner/Project Owner)**:

```
await projectInstance.endBidding(projectId, { from: ownerAddress } // or
projectOwnerAddress);
```

7. **Withdraw Tokens (Bidders)**:

   Ensure the bidding has ended and the tokens are vested.

```
await projectInstance.withdraw(projectId, { from: bidderAddress });
```

8. **Withdraw USDT (Owner)**:

```
await projectInstance.withdrawUSDT({ from: ownerAddress });
```

Additional Usage Examples for ProjectContract

Updating Project Owner (Project Owner)

```
const newOwnerAddress = "0xabc...123"; // Replace with new owner address

await projectInstance.updateProjectOwner(projectId, newOwnerAddress, { from:
currentProjectOwnerAddress });
```

Updating Operator Address (Owner/Project Owner)

```
const newOperatorAddress = "0xdef...456"; // Replace with new operator address

await projectInstance.updateOperatorAddress(projectId, newOperatorAddress, { from:
ownerAddress } // or projectOwnerAddress);
```

Updating Token Addresses (Owner)

```
const newIWOAddress = "0xghi...789"; // Replace with new IWO token address
const newUSDTAddress = "0xjkl...012"; // Replace with new USDT token address
const newNFTAddress = "0xabc...345"; // Replace with new NFT contract address

await projectInstance.updateTokenAddresses(newIWOAddress, newUSDTAddress,
newNFTAddress, { from: ownerAddress });
```

Notes

- Ensure you have the appropriate addresses and permissions set for each action.
- Replace placeholders like `<network-name>` with actual network configurations.
- These instructions assume basic familiarity with Truffle or Hardhat for Ethereum development. Adjust paths and commands based on the actual development environment used.

This comprehensive guide should help you effectively deploy and manage the provided smart contracts.

Comprehensive Deployment and Usage Guide for ProjectContract

Prerequisites

Before you start, ensure you have the following:

- **Node.js** and **npm** installed
- **Truffle** or **Hardhat** development framework

- **MetaMask** or another Ethereum wallet
- **Infura** or another Ethereum node service provider
- **OpenZeppelin Contracts** library installed

ProjectContract

Description

The `ProjectContract` manages project creation, bidding, and vesting processes, leveraging the `IWO`, `USDT`, and `AlignerNFT` contracts. It also includes functionalities for the contract owner to withdraw USDT.

Deployment

1. **Install Dependencies**:

```
npm install @openzeppelin/contracts
```

2. **Create Deployment Script**:

   Create a new deployment script in your `migrations` folder, e.g., `5_deploy_projectcontract.js`:

```
const ProjectContract = artifacts.require("ProjectContract");
const IWO = artifacts.require("IWO");
const MockUSDT = artifacts.require("MockUSDT");
const AlignerNFT = artifacts.require("AlignerNFT");

module.exports = async function (deployer, network, accounts) {
  const owner = accounts[0]; // Replace with the desired owner address
  const iwo = await IWO.deployed();
  const mockUSDT = await MockUSDT.deployed();
  const alignerNFT = await AlignerNFT.deployed();

  await deployer.deploy(ProjectContract, iwo.address, alignerNFT.address,
mockUSDT.address, owner);
};
```

3. **Deploy Contract**:

```
truffle migrate --network <network-name>
```

Usage

For the Owner

1. **Creating a Project**:

```
const projectInstance = await ProjectContract.deployed();
const biddingStartDate = Math.floor(Date.now() / 1000); // Current time in
seconds
const biddingDuration = 7 * 24 * 60 * 60; // 1 week

await projectInstance.createProject(
  "ProjectName",
  "ProjectDescription",
  "SocialInfo",
  biddingStartDate,
  biddingDuration,
  { from: ownerAddress }
);
```

2. **Adding a Vesting Round**:

```
const projectId = 1; // Replace with your project ID
const roundAmount = 1000000; // Replace with your desired round amount
const iwoPrice = 10; // Replace with your desired IWO price

await projectInstance.addVestingRound(projectId, roundAmount, iwoPrice, {
from: ownerAddress });
```

3. **Whitelisting Addresses**:

```
const userAddress = "0x123...abc"; // Replace with the actual user address
to whitelist

await projectInstance.addToWhitelist(projectId, userAddress, { from:
ownerAddress });
```

4. **Ending Bidding**:

```
await projectInstance.endBidding(projectId, { from: ownerAddress });
```

5. **Updating Project Details**:

```
await projectInstance.updateProjectDetails(
  projectId,
  "NewProjectName",
  "NewProjectDescription",
  "NewSocialInfo",
  { from: ownerAddress }
);
```

6. **Pausing and Unpausing Contract**:

```
await projectInstance.pause({ from: ownerAddress });
await projectInstance.unpause({ from: ownerAddress });
```

7. **Withdrawing USDT**:

```
await projectInstance.withdrawUSDT({ from: ownerAddress });
```

For the Project Owner

1. **Creating a Project**:

```
const projectInstance = await ProjectContract.deployed();
const biddingStartDate = Math.floor(Date.now() / 1000); // Current time in
seconds
const biddingDuration = 7 * 24 * 60 * 60; // 1 week

await projectInstance.createProject(
  "ProjectName",
  "ProjectDescription",
  "SocialInfo",
  biddingStartDate,
  biddingDuration,
  { from: projectOwnerAddress }
);
```

2. **Adding a Vesting Round**:

```
const projectId = 1; // Replace with your project ID
const roundAmount = 1000000; // Replace with your desired round amount
const iwoPrice = 10; // Replace with your desired IWO price

await projectInstance.addVestingRound(projectId, roundAmount, iwoPrice, {
from: projectOwnerAddress });
```

3. **Whitelisting Addresses**:

```
const userAddress = "0x123...abc"; // Replace with the actual user address
to whitelist
```

```
await projectInstance.addToWhitelist(projectId, userAddress, { from:
projectOwnerAddress });
```

4. **Ending Bidding**:

```
await projectInstance.endBidding(projectId, { from: projectOwnerAddress });
```

5. **Updating Project Details**:

```
await projectInstance.updateProjectDetails(
  projectId,
  "NewProjectName",
  "NewProjectDescription",
  "NewSocialInfo",
  { from: projectOwnerAddress }
);
```

For the Bidders

1. **Placing a Bid**:

   Ensure the contract is not paused and the bidding period is active.

```
const projectId = 1; // Replace with actual project ID
const allocationSize = 1000; // USDT amount
const vestingLength = 6; // Vesting length in months

await projectInstance.placeBid(projectId, allocationSize, vestingLength, {
from: bidderAddress });
```

2. **Checking Bidding Details**:

```
const bidDetails = await projectInstance.getBidDetails(projectId,
bidderAddress);
console.log(bidDetails);
```

3. **Checking Project Details**:

```
const projectDetails = await projectInstance.getProjectDetails(projectId);
console.log(projectDetails);
```

4. **Withdrawing Tokens**:

   Ensure the bidding has ended and the tokens are vested.

```
await projectInstance.withdraw(projectId, { from: bidderAddress });
```

Step-by-Step Guide

1. **Deploy Contracts**:

   Deploy the `IWO`, `MockUSDT`, `AlignerNFT`, and `ProjectContract` contracts as outlined in the respective sections.

2. **Create Project (Owner/Project Owner)**:

```
const projectInstance = await ProjectContract.deployed();
const biddingStartDate = Math.floor(Date.now() / 1000); // Current time in
seconds
const biddingDuration = 7 * 24 * 60 * 60; // 1 week

await projectInstance.createProject(
  "ProjectName",
  "ProjectDescription",
  "SocialInfo",
  biddingStartDate,
  biddingDuration,
  { from: ownerAddress } // or projectOwnerAddress
);
```

3. **Add Vesting Round (Owner/Project Owner)**:

```
const projectId = 1; // Replace with your project ID
const roundAmount = 1000000; // Replace with your desired round amount
const iwoPrice = 10; // Replace with your desired IWO price

await projectInstance.addVestingRound(projectId, roundAmount, iwoPrice, {
from: ownerAddress } // or projectOwnerAddress);
```

4. **Whitelist Addresses (Owner/Project Owner)**:

```
const userAddress = "0x123...abc"; // Replace with the actual user address
to whitelist

await projectInstance.addToWhitelist(projectId, userAddress, { from:
ownerAddress } // or projectOwnerAddress);
```

5. **Place Bids (Bidders)**:

```
const allocationSize = 1000; // USDT amount
const vestingLength = 6; // Vesting length in months

await projectInstance.placeBid(projectId, allocationSize, vestingLength, {
from: bidderAddress });
```

6. **End Bidding (Owner/Project Owner)**:

```
await projectInstance.endBidding(projectId, { from: ownerAddress } // or
projectOwnerAddress);
```

7. **Withdraw Tokens (Bidders)**:

   Ensure the bidding has ended and the tokens are vested.

```
await projectInstance.withdraw(projectId, { from: bidderAddress });
```

8. **Withdraw USDT (Owner)**:

```
await projectInstance.withdrawUSDT({ from: ownerAddress });
```

Additional Usage Examples

Updating Project Owner (Project Owner)

```
const newOwnerAddress = "0xabc...123"; // Replace with new owner address

await projectInstance.updateProjectOwner(projectId, newOwnerAddress, { from:
currentProjectOwnerAddress });
```

Updating Operator Address (Owner/Project Owner)

```
const newOperatorAddress = "0xdef...456"; // Replace with new operator address

await projectInstance.updateOperatorAddress(projectId, newOperatorAddress, { from:
ownerAddress } // or projectOwnerAddress);
```

Updating Token Addresses (Owner)

```
const newI

WOAddress = "0xghi...789"; // Replace with new IWO token address
const newUSDTAddress = "0xjkl...012"; // Replace with new USDT token address
const newNFTAddress = "0xabc...345"; // Replace with new NFT contract address

await projectInstance.updateTokenAddresses(newIWOAddress, newUSDTAddress,
newNFTAddress, { from: ownerAddress });
```

Notes

- Ensure you have the appropriate addresses and permissions set for each action.
- Replace placeholders like `<network-name>` with actual network configurations.
- These instructions assume basic familiarity with Truffle or Hardhat for Ethereum development. Adjust paths and commands based on the actual development environment used.

This comprehensive guide should help you effectively deploy and manage the provided smart contracts.