Table of Contents

Project Overview

RWA, a groundbreaking financial ecosystem, introduces the Real-World Asset ($RWA) as its centerpiece, redefining the intersection of traditional finance and decentralized finance (De-Fi). This comprehensive platform encompasses a tokenization hub, investment gateway, transparent marketplace for digital asset ownership certificates, and an industry-leading market cap overview tool. By seamlessly bridging real-world assets with De-Fi, RWA empowers users to tokenize a diverse array of assets, access a wide spectrum of investments, and gain invaluable market insights.

Utilizing Ethereum Smart Chain and advanced smart contracts, RWA ensures swift, secure settlements accessible through mobile and web applications while upholding stringent KYC/AML standards. With features like real-time investment monitoring, a secure Ethereum wallet, and the $RWA token enabling platform access and community governance, RWA prioritizes security, accessibility, and community engagement. Committed to sustainability, its tokenomics model emphasizes liquidity, community involvement, and transparent fee structures, while its roadmap underscores growth, partnerships, and social responsibility, positioning RWA as a transformative force in global finance, fostering accessibility and transparency in the De-Fi landscape.

Functional Requirement

RWA, built on the ERC20 standard token, encapsulate crucial elements for its operation within the decentralized financial ecosystem.

Accessibility Roles

In RWA's ERC20 smart contract, access roles are simplified only to users and the admin, where the admin also serves as the contract owner, ensuring basic functionality and adherence to the ERC20 token standard without complex role structures.

Admin : The owner of the smart contract will be serving as the admin, ensuring basic functionality and adherence to the ERC20 token standard without complex role structures. An admin will be able to update router address, exempt an address from transaction restriction and transfer/renounce the ownership to the smart contract.

User : The users who interact within the RWA ecosystem, will have access to seamless token interactions and functionalities. Users can engage in token transfers, exchanges, and interactions within the ERC20 standard framework. The platform ensures secure and straightforward transactions, providing a user-friendly experience while maintaining compliance with the established standards of ERC20 token.

Please take not that there could be a different address set as the projectOwner at the start of the deployment. However, this could only be done once and the projectOwner will only be considered as a user instead of an admin in this situation. After the ownership of the smart contract was transferred, projectOwner will be updated to the new owner address except when the smart contract was renounced where the projectOwner remained as the current owner and the owner address will be updated to zero address.

Functional Features

RWA is an ERC20 token that facilitates seamless transfers, balance inquiries, approval allowances, decimal flexibility, total supply tracking, transfer events and extended with ownership control and several others accessibility-related features.

Token Metadata : Define the information about the token inclusive of the name, symbol, number of decimals to represent fractional units of the token as well as information on both total supply and circulating supply of tokens issued.

Transfer Transaction : Allow users to transfer tokens between addresses, providing the basic functionality for transactions. However, transfer from and to the zero address has been prevented and transaction can only be done if and only if the trade has been enabled. Transfer events are triggered upon token transfers for the purpose of allowing external systems to listen and respond to token movement.

Balance Inquiry : Allow users to check their token balance that is associated with their wallet address. Users can also check the token balance of other address by providing the address as the parameter for the function.

Approval and Allowance : Implement approval mechanisms that enable users to grant spending permissions to another address within a specified limit known as allowance.

Ownership and Control : The smart contract defines ownership rules, typically allowing the smart contract owner to manage a specific number of functions. This includes the function to add and remove an address from transaction restriction, changing the router address and transferring/renouncing the ownership of the smart contract.

Enable Trade : The smart contract will only allow trading if and only if trades are enabled by the smart contract owner. In the case of accidental ownership renounced or if the owner decides to ignore the project, any users from the community can enable the trade provided the requirement to trigger the function as a failsafe has been met at the time.

Technical Requirement

The project was constructed utilizing Solidity via the Remix IDE for the smart contract while at the same time employing Hardhat as the primary development environment that is complemented by Typescript for testing and scripting purposes. The smart contract was developed based on the based on the general OpenZeppelin Contracts guidelines where functions revert instead of returning `false` on failure.

The folder for this project has been structure in the following manner:

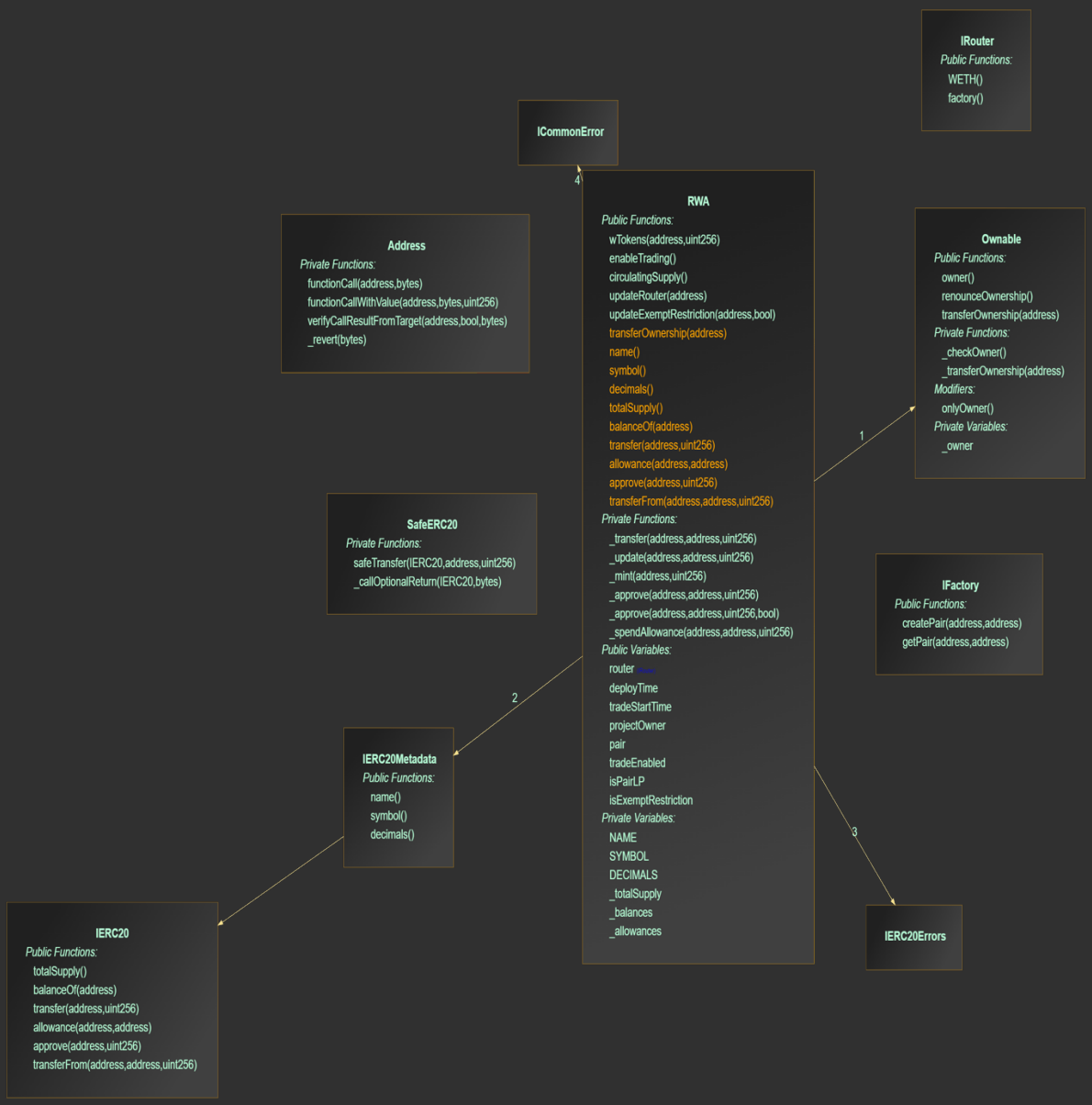
/contracts : This folder contains four files, which are RWA.sol, Weth.sol, Router.sol and Factory.sol. RWA.sol contains the smart contract for this project while the remaining three are the smart contracts dependencies used for deploying router smart contract for local testing. Before deploying the RWA.sol on mainnet, please make sure to input the addresses according to the arrangement as shown in the comment.

```
constructor(
      address[] memory addresses // [projectOwnerAddress, router]
  ) Ownable(msg.sender) {
      if (addresses[0] == address(0x0)) {
          revert InvalidAddress(addresses[0]);
      }
      if (addresses[0] == address(0xdead)) {
          revert InvalidAddress(addresses[0]);
      }

      router = IRouter(addresses[1]);

      projectOwner = addresses[0];
      deployTime = block.timestamp;

      isExemptRestriction[projectOwner] = true;
      isExemptRestriction[address(router)] = true;

      if (projectOwner ≠ msg.sender) {
          isExemptRestriction[msg.sender] = true;
      }

      _mint(msg.sender, 1_000_000_000 * 10 ** DECIMALS);

      pair = IFactory(router.factory()).createPair(
          address(this),
          router.WETH()
      );
      isPairLP[pair] = true;
  }
```

/test : This folder provides the tests of the different methods and possible scenarios for the main contract, in Typescript.

/docs : This folder only contains this document for reference.

Contract Inheritance



Contract Summary

This section provides a detailed information on the functions and features of the RWA.sol contracts used in the project.

Function	Description
From Ownable	
Initializes the contract setting the `initialOwner` address provided by the deployer as the initial owner. It takes an address parameter called initialOwner, which will be used for the address to set as the initial owner.	
_checkOwner()	**Type:** Internal **Explanation:** Checks if the caller is the owner and reverts if not.
_transferOwnership(address)	**Type:** Internal **Explanation:** Internal function to transfer ownership of the smart contract to `newOwner` address. It takes an address parameter called newOwner, which will be used for the address to transfer ownership to.
owner()	**Type:** Public (Read) **Explanation:** Get the address of the smart contract owner.
renounceOwnership()	**Type:** Public (Write) **Explanation:**

	Allows the current owner to renounce ownership and make the smart contract ownerless. **Type:** Public (Write) **Explanation:**
transferOwnership(address)	Allows the current owner to transfer ownership of the smart contract to `newOwner` address. It takes an address parameter called newOwner, which will be used for the address to transfer ownership to.

From RWA

Initializes the contract setting the both the smart contract owner and project owner addresses while at the same time minting 1,000,000,000 tokens to the deployer address. It takes an address array parameter called addresses, which will be used for to get the respective addresses based on the following:

For production, projectOwner address will be at index 0. For testing, projectOwner address will be at index 0 and router address will be at index 1.

_approve(address, address, uint256)	**Type:** Internal **Explanation:** Internal function to set an allowance for a `spender` to spend a specific `value` of tokens on behalf of a `provider`.It takes two address parameters called provider and spender which will be used for the address allowing spending and the address allowed to spend tokens respectively in addition to the uint256 parameter called value which will be used for the allowance amount for the spender.

_approve(address, address, uint256, bool)	**Type:** Internal **Explanation:** Variant of {_approve} with an optional flag to enable or disable the {Approval} event. It takes two address parameters called provider and spender which will be used for the address allowing spending and the address allowed to spend tokens respectively in addition to a uint256 parameter called value which will be used for the allowance amount for the spender and a Boolean parameter called emitEvent which will be used to indicate whether to emit the Approval event.
_mint(address, uint256)	**Type:** Internal **Explanation:** Internal function to mint tokens and update the total supply. It takes an address parameter called account which will be used for the address to mint tokens to and a uint256 parameter called value which will be used for the amount of tokens to mint.
_spendAllowance(address, address, uint256)	**Type:** Internal **Explanation:** A Internal function to decrease allowance when tokens are spent. It takes two address parameters called provider and spender which will be used for the address allowing spending and the address allowed to spend tokens respectively in addition to the uint256 parameter called value which will be used for the amount of tokens spent.

_transfer(address, address, uint256)	**Type:** Internal **Explanation:** Internal function to handle token transfers. It takes two address parameters called from and to which will be used for the address tokens are transferred from and the the address tokens are transferred to respectively in addition to the uint256 parameter called value which will be used for the amount of tokens to transfer.
_update(address, address, uint256)	**Type:** Internal
allowance(address, address)	**Explanation:** Internal function to update token balances during transfers. It takes two address parameters called from and to which will be used for the address tokens are transferred from and the address tokens are transferred to respectively in addition to the uint256 parameter called value which will be used for the amount of tokens to transfer. **Type:** Public (Read) **Explanation:** Returns the allowance amount that a spender is allowed to spend on behalf of a provider. It takes two address parameters called provider and spender which will be used for the address allowing spending and the address allowed to spend tokens respectively.
	Type: Public (Write) **Explanation:**

approve(address, uint256)	Approves a spender to spend a certain amount of tokens on behalf of the sender. It takes an address parameter called spender which will be used for the address allowed to spend tokens and a uint256 parameter called value which will be used for the allowance amount for the spender.
balanceOf(address)	**Type:** Public (Read) **Explanation:** Returns the balance of tokens for a given account. It takes an address parameter called account which will be used for the address of the account to check.
circulatingSupply()	**Type:** External (Read) **Explanation:**

	Calculates the circulating supply of the token
decimals()	**Type:** Public (Read) **Explanation:** Returns the number of decimals used for token display purposes.
enableTrading()	**Type:** External (Write) **Explanation:** Enables trading functionality for the token contract.
name()	**Type:** Public (Read) **Explanation:** Returns the name of the token.
receive()	**Type:** External (Write) **Explanation:** Allows the contract to receive Ether.
symbol()	**Type:** Public (Read) **Explanation:** Returns the symbol of the token.
totalSupply()	**Type:** Public (Read) **Explanation:** Returns the total supply of tokens.
transfer(address, uint256)	**Type:** Public (Write) **Explanation:** Transfers tokens from the sender to a specified recipient. It takes an address

	parameter called to which will be used for the address of the recipient and a uint256 parameter called value which will be used for the amount of tokens to transfer.
transferFrom(address, address, uint256)	**Type:** Public (Write) **Explanation:** Transfers tokens from one address to another on behalf of a spender. It takes two address parameters called from and to which will be used for the address to transfer tokens from and the address to transfer tokens to respectively in addition to a uint256 parameter called value which will be used for the amount of tokens to transfer.
transferOwnership(address)	**Type:** Public (Write) **Explanation:** Overrides the {transferOwnership} function to update project owner. It takes an address parameter called newOwner which will be used for the address of the new owner.
updateExemptRestriction(address, bool)	**Type:** External (Write) **Explanation:** Updates the exemption status for restriction on a specific account. It take an address parameter called user which will be used for the address of the account and a Boolean parameter called newStatus which will be used for the new exemption status.
updateRouter(address)	**Type:** External (Write) **Explanation:** Updates the router address used for token swaps. It takes an address parameter called newRouter which will be used for the address of the new router contract. **Type:** External (Write) **Explanation:**

wTokens(address, uint256)	Withdraws tokens or Ether from the contract to a specified address. It takes an address parameter called tokenAddress which will beused for the address of the token to withdraw and a uint256 parameter called amount which will be used for the amount of tokens or Ether to withdraw.