**How You Should Start This Project (As Developer)**

**Step 1: Set Up Your Development Environment**

🔹 Install important tools:

* **Node.js** (Backend and Web3 tools)
* **Hardhat** (for smart contract development)
* **MetaMask** (for wallet connection)
* **Ganache** or **Hardhat Network** (local blockchain for testing)

🔹 Install libraries:

bash

CopyEdit

npm install --save-dev hardhat @openzeppelin/contracts ethers dotenv

* Hardhat → Write and test smart contracts
* OpenZeppelin Contracts → Use secure, audited templates (ERC20, ERC721, ERC-3643 base)
* Ethers.js → Interact with blockchain

🔹 Set up a GitHub repo for version control.

**Step 2: Initialize Hardhat Project (Smart Contract Work)**

🔹 Inside your project folder:

bash

CopyEdit

npx hardhat

Choose: "**Create a basic sample project**"

🔹 Project structure will be:

lua

CopyEdit

contracts/ --> Your smart contracts (e.g., T-REX Token)

scripts/ --> Deployment scripts

test/ --> Unit tests

hardhat.config.js

**Step 3: Build the Basic Smart Contracts**

🔹 First smart contracts to create:

* IdentityRegistry.sol (registers verified investors)
* Compliance.sol (rules for transfers)
* T-REX.sol (ERC-3643 token based on OpenZeppelin's ERC20)

Example Basic T-REX Token Start:

solidity

CopyEdit

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract TREXToken is ERC20 {

constructor(uint256 initialSupply) ERC20("Asset Bond Token", "ABT") {

\_mint(msg.sender, initialSupply);

}

}

Later, you will extend this with compliance and identity checks.

**Step 4: Write Deployment Scripts**

🔹 Write deploy.js inside scripts/:

javascript

CopyEdit

async function main() {

const [deployer] = await ethers.getSigners();

console.log("Deploying contracts with the account:", deployer.address);

const Token = await ethers.getContractFactory("TREXToken");

const token = await Token.deploy(1000000); // 1 Million Tokens

console.log("Token deployed to:", token.address);

}

main().catch((error) => {

console.error(error);

process.exitCode = 1;

});

Run deployment:

bash

CopyEdit

npx hardhat run scripts/deploy.js --network localhost

**Step 5: Start Backend and Frontend Development**

🔹 Backend:

* Node.js + Express API
* APIs for:
  + Register User (with KYC info)
  + Mint new tokens
  + Transfer tokens
  + Recover lost wallets

🔹 Frontend:

* React.js + Ethers.js
* Pages:
  + Login/Register (KYC submission)
  + Token Dashboard (view owned tokens)
  + Admin Panel (mint new tokens)

🔹 Wallet Integration:

* MetaMask Connect Button
* Display wallet address and balance

**Step 6: Add Compliance and KYC Integration**

🔹 Add smart contract checks for:

* Only KYC-verified addresses can receive tokens
* Limit token transfers if regulations demand
* (Optional) Integration with KYC providers (like Sumsub) via backend

**Step 7: Test Smart Contracts**

🔹 Write unit tests for:

* Identity registration
* Token minting
* Token transferring
* Compliance rules

Example Test in Hardhat (Mocha):

javascript

CopyEdit

const { expect } = require("chai");

describe("TREXToken", function () {

it("Should deploy and mint initial supply", async function () {

const [owner] = await ethers.getSigners();

const Token = await ethers.getContractFactory("TREXToken");

const token = await Token.deploy(1000000);

expect(await token.totalSupply()).to.equal(1000000);

});

});

**Step 8: Deployment (Testnet First)**

🔹 Use Goerli or Mumbai Testnet. 🔹 Get free test ETH/MATIC. 🔹 Deploy smart contracts on Testnet.

Deploy on testnet:

bash

CopyEdit

npx hardhat run scripts/deploy.js --network goerli

**Step 9: Build Investor Frontend and Admin Portal**

🔹 Admin:

* Mint tokens
* Manage investors

🔹 Investor:

* Buy/Sell tokens
* View holdings
* Wallet recovery request

**Step 10: Security Audit + Launch**

🔹 Run **MythX**, **Slither** static analysis on smart contracts 🔹 Manual smart contract review 🔹 Launch on Mainnet!

**🎯 Your Starter Goal for the First Week**

| **Task** | **Description** |
| --- | --- |
| 1. Set up Hardhat | Initialize project, install libraries |
| 2. Create basic ERC-3643 token | Write basic smart contract |
| 3. Deploy locally | Deploy on localhost using Hardhat network |
| 4. Connect simple React frontend | Create MetaMask connect button and wallet display |
| 5. Write unit tests | Start basic contract testing |

**⚡ Key Tools You Will Be Using:**

* Solidity (Smart Contracts)
* Hardhat (Testing & Deployment)
* Node.js + Express (Backend APIs)
* React.js + Ethers.js (Frontend Web3)
* IPFS (for off-chain document storage)
* MetaMask (Wallet connection)
* KYC Provider API (optional integration)

**🔥 Bonus Tip:**

* Build a **very basic POC (Proof of Concept)** first — just mint 1 token, transfer it, and verify KYC manually.
* Once your POC works → scale it to a full production-grade system.