BLOCKCHAIN TESTS 1 - BLUME TOKEN

Blockchain Developer Task Test

---

1. INTRODUCTION / OVERVIEW OF THE PROJECT

This test is designed to evaluate your proficiency in **blockchain development**, **smart contract security**, **DeFi protocols**, and overall **technical problem-solving skills**.

About BLUME TOKEN (BLX)

BLUME is a cryptocurrency company focused on **decentralized finance (DeFi)** solutions. Our project, **BLUME TOKEN (BLX)**, is designed to power an ecosystem that includes:

- **Liquidity Pools** – Enabling users to provide liquidity and earn transaction fees.
- **Vaults** – Secure smart contract-based storage for BLX tokens.
- **Staking** – Users stake BLX to earn rewards.
- **Liquid Staking** – Users receive derivative tokens for their staked BLX, allowing further participation in DeFi activities.

We are looking for developers who can **demonstrate expertise in blockchain development** by implementing smart contracts and integrating them into a front-end built with **React.js.**

You have **1 day** to complete this test from the moment you acknowledge it.

---

2. WHAT WE EXPECT

We expect you to:

✅ Implement **secure and optimized smart contracts** using Solidity.
✅ Integrate smart contracts into a **React.js** frontend.
✅ Ensure **security best practices**, including **audit-readiness** and **breach-proof testing**.
✅ Deploy and interact with smart contracts on a **testnet (Ethereum)**.
✅ Showcase **liquidity pools, staking, liquid staking, and vaults** functionality.
✅ Provide a **video walkthrough** explaining your implementation, **security measures**, and

audit process.
✅ Submit a **documentation file** justifying the design decisions and approach taken.

---

3. TASKS

This section provides a comprehensive breakdown of the tasks you are required to complete for the **BLUME TOKEN (BLX) Blockchain Developer Test**. Each task is critical in evaluating your ability to develop, secure, and integrate blockchain-based smart contracts into a **functional decentralized finance (DeFi) application**.

1. SMART CONTRACT DEVELOPMENT

You will develop and deploy multiple smart contracts that support BLUME TOKEN's ecosystem. These contracts must be written in **Solidity** and deployed on an **Ethereum (ERC-20) testnet**. You are expected to follow **best security practices**, optimize for **gas efficiency**, and ensure **audit-readiness**.

1.1 BLUME TOKEN (BLX) Smart Contract

- Develop a **BLX token** following the **ERC-20** token standard.
- Ensure that the contract includes **standard functionalities**:
    - `transfer()`, `approve()`, `transferFrom()`, `balanceOf()`, `totalSupply()`, etc.
    - Allow users to **mint** and **burn** tokens (if applicable).
    - Include **access control mechanisms** to prevent unauthorized actions.
    - Implement **SafeMath** to prevent overflows and underflows.
    - Consider **anti-bot and anti-whale mechanisms** (e.g., limit large transactions).
- Deploy the contract on a **testnet** (Ethereum Goerli, Sepolia).

1.2 Liquidity Pool Smart Contracts

- Develop smart contracts to **enable liquidity pools** where users can deposit **BLX and another token (e.g., BNB or ETH)** to provide liquidity.
- The contract should:
    - Implement **Automated Market Maker (AMM) logic** similar to Uniswap/PancakeSwap.
    - Allow users to **add liquidity** and **remove liquidity**.
    - Distribute **trading fees as rewards** to liquidity providers.
    - Support **BLX token swaps** with a simple **DEX interface**.

    - Prevent **slippage issues and price manipulation attacks**.
- Ensure that **users who provide liquidity are rewarded** with **LP (Liquidity Provider) tokens**.
- Implement **Oracle price feed integration** (e.g., Chainlink or other price oracles) to **reduce price manipulation risks**.

1.3 Vaults Smart Contracts

- Implement **vault contracts** where users can securely **store their BLX tokens**.
- Key functionalities must include:
    - **Deposit and withdraw mechanisms** with time-locking options.
    - A **yield-generating mechanism**, where deposited BLX tokens **earn 10% APY interest over time**.
    - **Auto-compounding** rewards to maximize returns.
    - Integration with **staking and liquidity pools** for automated yield farming.
- Ensure vault **security** with **role-based access control (RBAC)**, **pause mechanisms**, and **emergency withdrawal functions**.

1.4 Staking & Liquid Staking Smart Contracts

1.4.1 Staking Contracts

- Develop a **staking mechanism** where users can stake **BLX tokens** to earn rewards.
- The staking contract should:
    - Allow users to **stake and unstake** BLX tokens.
    - Provide **variable APRs** based on the **duration of staking**.
    - Implement **reward distribution** logic, ensuring fair and transparent calculations.
    - Offer **penalties for early withdrawals** to encourage long-term staking.

1.4.2 Liquid Staking Contracts

- Implement **liquid staking**, allowing users to stake BLX while receiving a derivative token, **stBLX**.
- Users should be able to **use stBLX in DeFi protocols** while still earning staking rewards.
- Security checks should prevent **double-staking exploits**.

---

2. SECURITY & AUDIT READINESS

Blockchain security is **critical**. Your smart contracts must undergo **security testing** and be **audit-ready**.

2.1 Conduct Audit-Breach Proof Tests

- Perform **static and dynamic analysis** using tools like:
  - **Slither** (for static analysis).
  - **MythX** (for security vulnerability detection).
  - **Certik Audit** (optional but recommended).
- Run **automated and manual penetration tests** to identify vulnerabilities.
- Ensure **audit-breach proofing** by simulating different attack vectors.

2.2 Implement Security Measures

Your contracts must be protected against common attacks, including:

✅ **Reentrancy Attacks** – Use the **Checks-Effects-Interactions pattern** or **ReentrancyGuard**.
✅ **Flash Loan Exploits** – Ensure that **only valid collateralized loans are allowed**.
✅ **Integer Overflows & Underflows** – Use **SafeMath** or Solidity's built-in overflow protection.
✅ **Oracle Manipulation** – Use **trusted decentralized oracles** (e.g., Chainlink).
✅ **Access Control Flaws** – Implement **RBAC (Role-Based Access Control)** using **Ownable or AccessControl.sol**.
✅ **Gas Optimization** – Reduce unnecessary storage variables, minimize on-chain computation, and use **events for state changes**.

2.3 Gas Fee Optimization

- Ensure **low gas fees** by:
  - Using **efficient storage variables** (e.g., `uint256` over `uint8`).
  - Minimizing **loops and expensive computations**.
  - **Batch processing** transactions instead of single calls.
  - Using **Layer-2 solutions** (e.g., Arbitrum, Optimism) if applicable.

---

3. FRONTEND INTEGRATION

Once your smart contracts are deployed, you need to build a simple **React.js front-end**.

3.1 Build a React.js & Laravel Application

- Develop a **simple mock UI** to interact with your smart contracts.
- The UI should have the key components to demonstrate the actions required to perform the tests

3.2 Implement Web3 Wallet Connection

- Integrate **MetaMask** for seamless user authentication.

By completing these tasks, you will **demonstrate your ability to build a secure, scalable, and fully functional DeFi application**. Ensure that all features are **properly implemented, tested, and documented** before submission.

---

4. WHAT YOU MUST DELIVER AND DEMONSTRATE

Smart Contract Deliverables

- Fully **functional, tested, and deployed** BLUME TOKEN (BLX) smart contracts.
- **Liquidity Pool**, **Vaults**, **Staking**, and **Liquid Staking** contracts.
- **Security measures** implemented and explained.

Frontend & Backend Integration

- A working mock UI that interacts with the smart contracts.
- Ability to perform actions like **staking, liquidity pools, and vaults management** via the UI.

Security & Testing

- Smart contract **security tests** and audit certification.
- **Demonstration of audit-breach proof tests**.

---

5. VIDEO SHOWCASE (WHAT YOUR VIDEO MUST SHOWCASE)

You must submit a detailed **video showcase** that clearly demonstrates the features and functionalities of your implementation. The video should provide a step-by-step explanation of the developed components, highlighting how they work and the approach taken.

Instead of creating one long video, you can submit multiple shorter videos, each focusing on a specific feature or functionality. This will make it easier to review and understand the different aspects of the project.

The video must include a clear voiceover explanation, where you walk through your implementation, justifying your decisions and the logic behind your work. All videos should be uploaded as **private or unlisted on YouTube or Vimeo**, with the **links shared** in the submission.

- Cover the **contract deployment process**.
- Show how you **integrated the contracts with React.js**.
- Explain your **security tests & audit procedures**.
- Demonstrate **staking, liquidity pools, and vaults functionality**.

---

6. DURATION

1 Day (from the moment you acknowledge the task).

---

7. OUR EVALUATION CRITERIA

We will evaluate your submission based on:

CRITERIA	DETAILS	WEIGHT (%)
Smart Contract Quality	Secure, optimized, and audit-ready contracts	30%
Frontend & Backend Integration	Functional UI with React.js	20%
Security & Testing	Audit-breach proof tests and best security practices	20%
Code Readability & Documentation	Well-structured and commented code, documentation clarity	15%
Video Showcase	Clear, detailed, and explanatory video submission	15%

8. SUBMISSION REQUIREMENTS

Your submission must include:

📌 **Smart contract source code** (GitHub).
📌 **Frontend & Backend code** (GitHub).
📌 **Security audit report** (documenting tests conducted and results).
📌 **Deployment details** (testnet address, transaction hashes).
📌 **Video showcase link** (YouTube/Vimeo).
📌 **Documentation file** (justifying decisions and implementation approach).

Task Submission Checklist

#	REQUIREMENT	COMPLETED	DEMONSTRATED	ATTACHED
1	Smart contract source code & proof of verification	☐	☐	☐
2	Liquidity Pool, Vault, Staking Contracts	☐	☐	☐
3	React.js Integration	☐	☐	☐
4	Security audit report	☐	☐	☐
5	Deployment details	☐	☐	☐
6	Video showcase link	☐	☐	☐
7	Documentation file	☐	☐	☐

Number of Required Submissions

ITEM	NUMBER REQUIRED
Smart Contract Code and verification proof	1 (Deployed & Secured)
Frontend & Backend Code	1 (Functional with UI)
Security Audit Report	1 (Detailed findings)

Deployment Details	1 (Contract Address & TX Hashes)
Video Showcase	1 (Private YouTube/Vimeo Link)
Documentation File	1 (Explanation & Justification)

FINAL NOTES

This test will evaluate your **technical skills, problem-solving ability, and security awareness**. Make sure to **adhere to the submission requirements** and **meet the deadline**. Good luck!