MyToken (MTK) | ERC-20 Smart Contract

1. Project Overview

MyToken (MTK) is a fully functional ERC-20 cryptocurrency token built on the Ethereum blockchain using Solidity. This project demonstrates a foundational understanding of smart contract development, token economics, and the Ethereum Virtual Machine (EVM).

The contract implements the standard ERC-20 interface, allowing for wallet compatibility, token transfers, and spending allowances, while also including robust error handling and event logging.

2. Technical Stack

- **Language:** Solidity (Version 0.8.x)
- **IDE:** Remix Ethereum IDE
- **Environment:** Remix VM (Cancun)
- **Standard:** ERC-20

3. Token Specifications

Based on the deployment configuration:

- **Token Name:** MyToken
- **Token Symbol:** MTK
- **Decimals:** 18
- **Total Supply:** 1,000,000 MTK (1,000,000 * 10^18 wei)

4. Key Features Implemented

- **Core ERC-20 Functions:** transfer, approve, transferFrom, balanceOf, allowance.
- **Helper Functions:** getTokenInfo() for quick frontend data retrieval.
- **Event Logging:** Emits Transfer and Approval events to the blockchain logs.
- **Security Validation:**
    - Prevents transfers to the zero address (burn prevention).
    - Checks for sufficient balances before transfers.
    - Checks for sufficient allowance during delegated transfers.

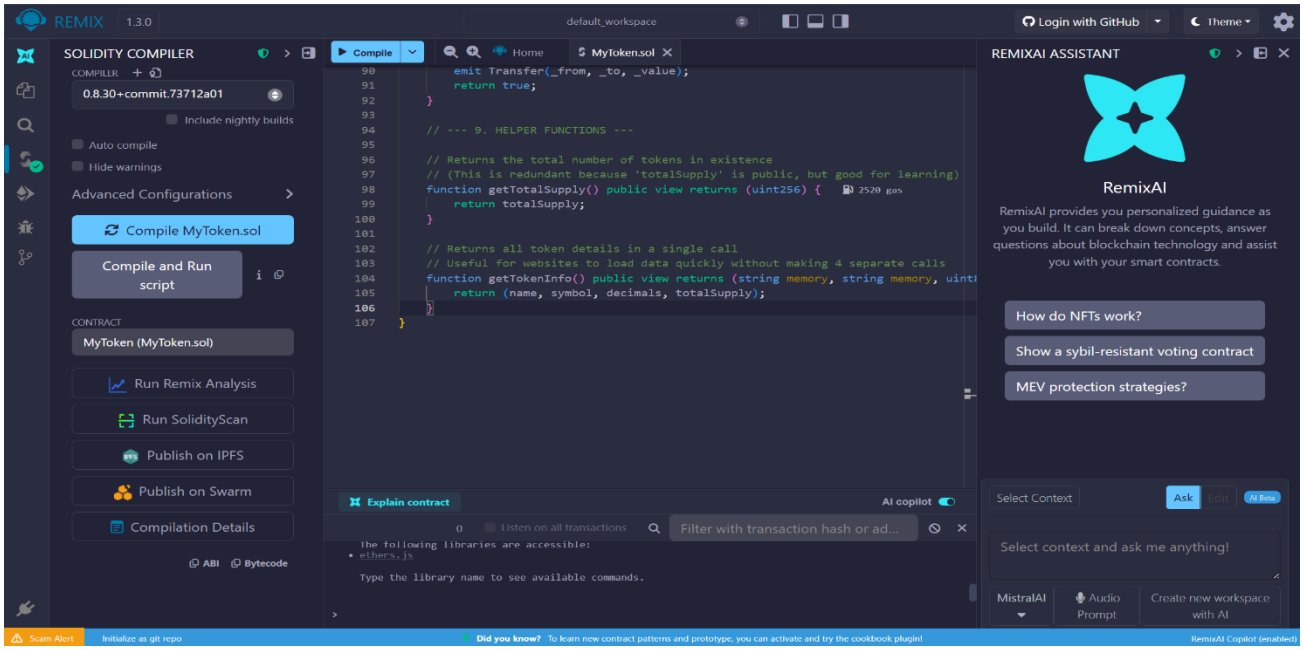
---

5. Development & Testing Log

This section documents the successful compilation, deployment, and stress-testing of the smart contract.

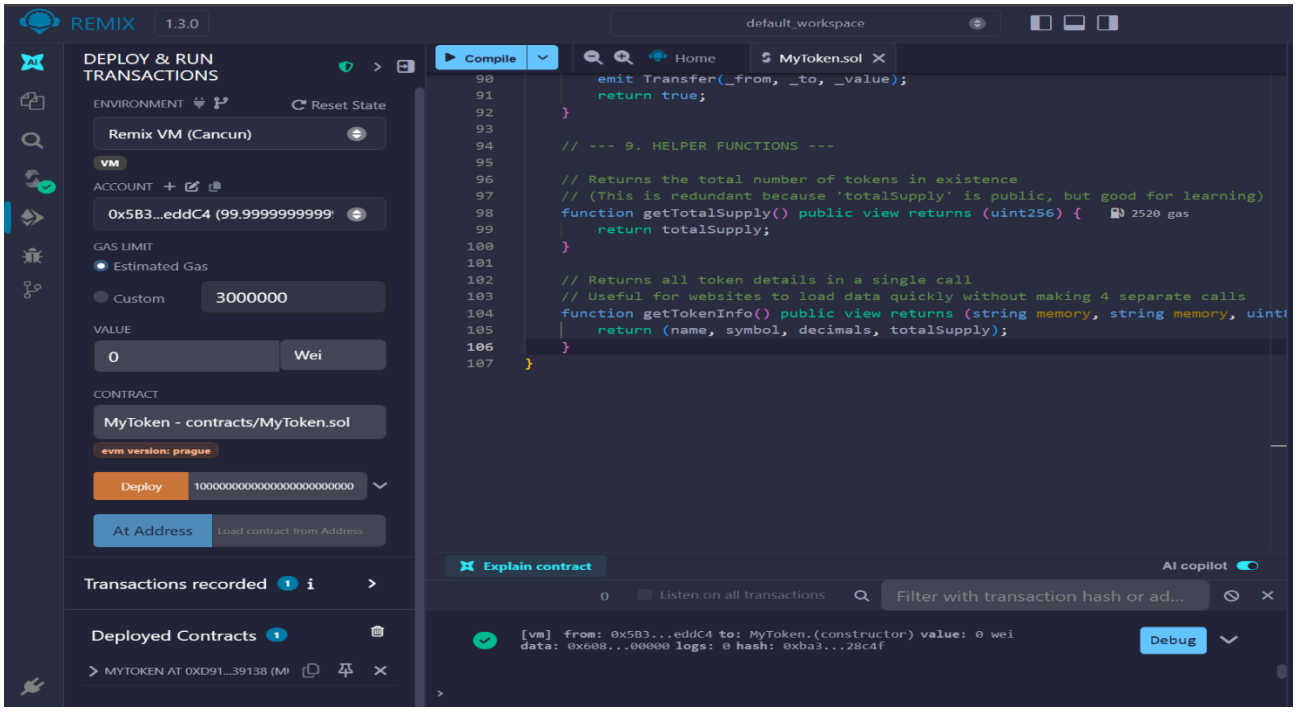
A. Compilation

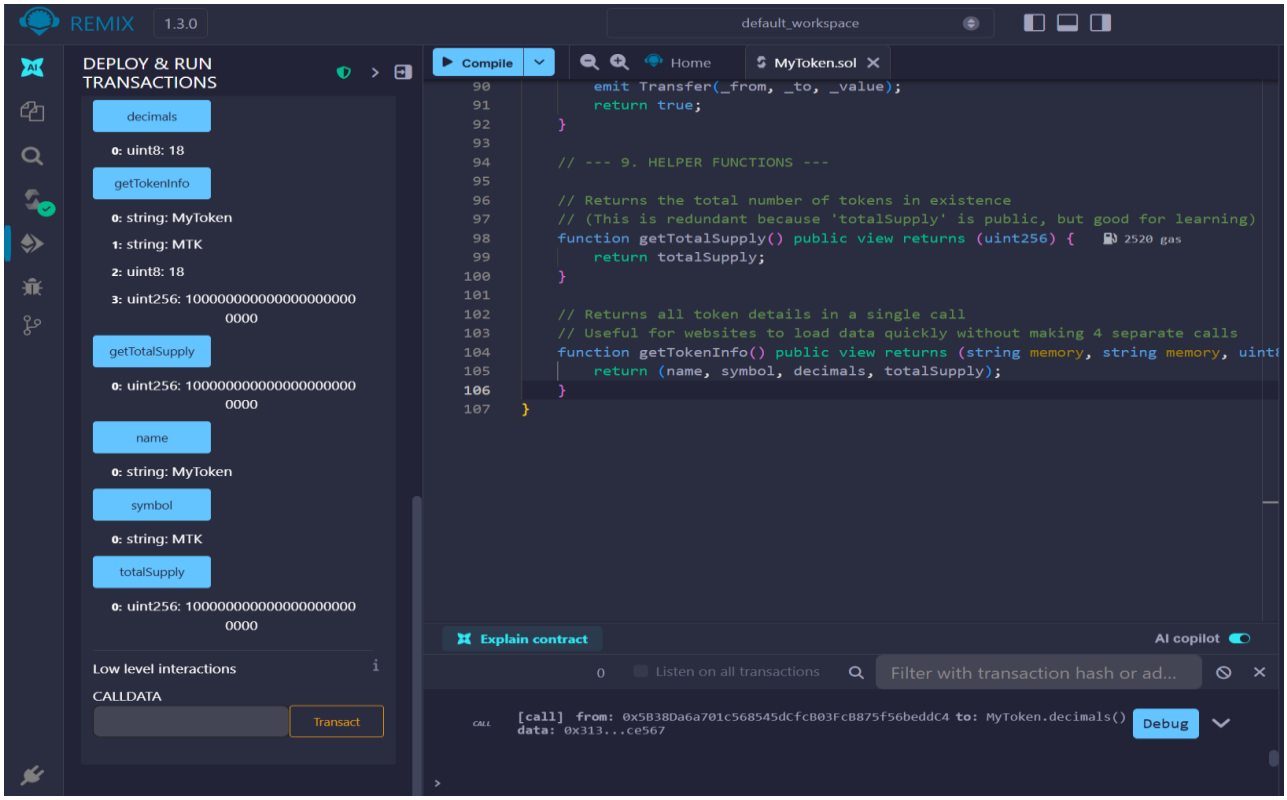
The contract was successfully compiled using the Solidity 0.8.30 compiler with no errors or warnings.



B. Deployment

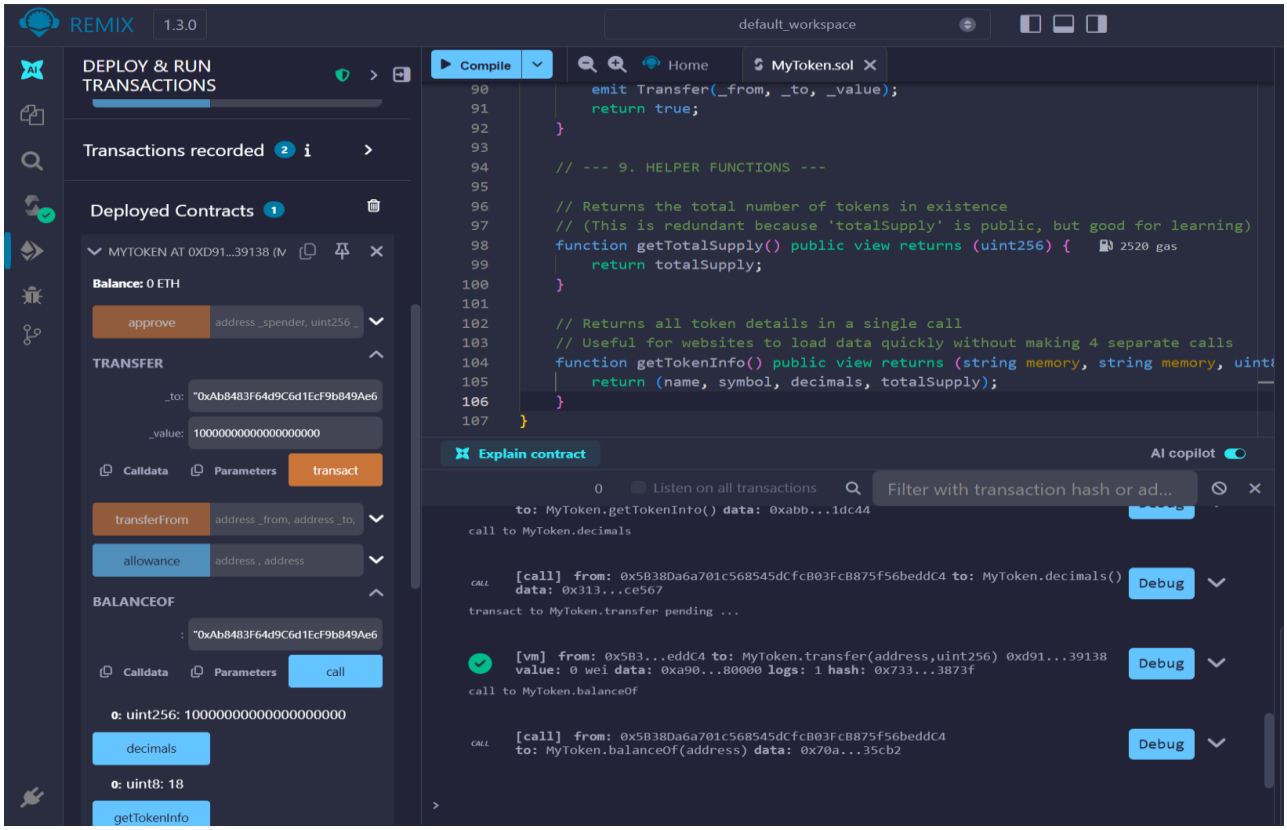
The contract was deployed to the local Remix VM environment. The state variables were initialized correctly upon deployment (Name, Symbol, Decimals, and Supply).

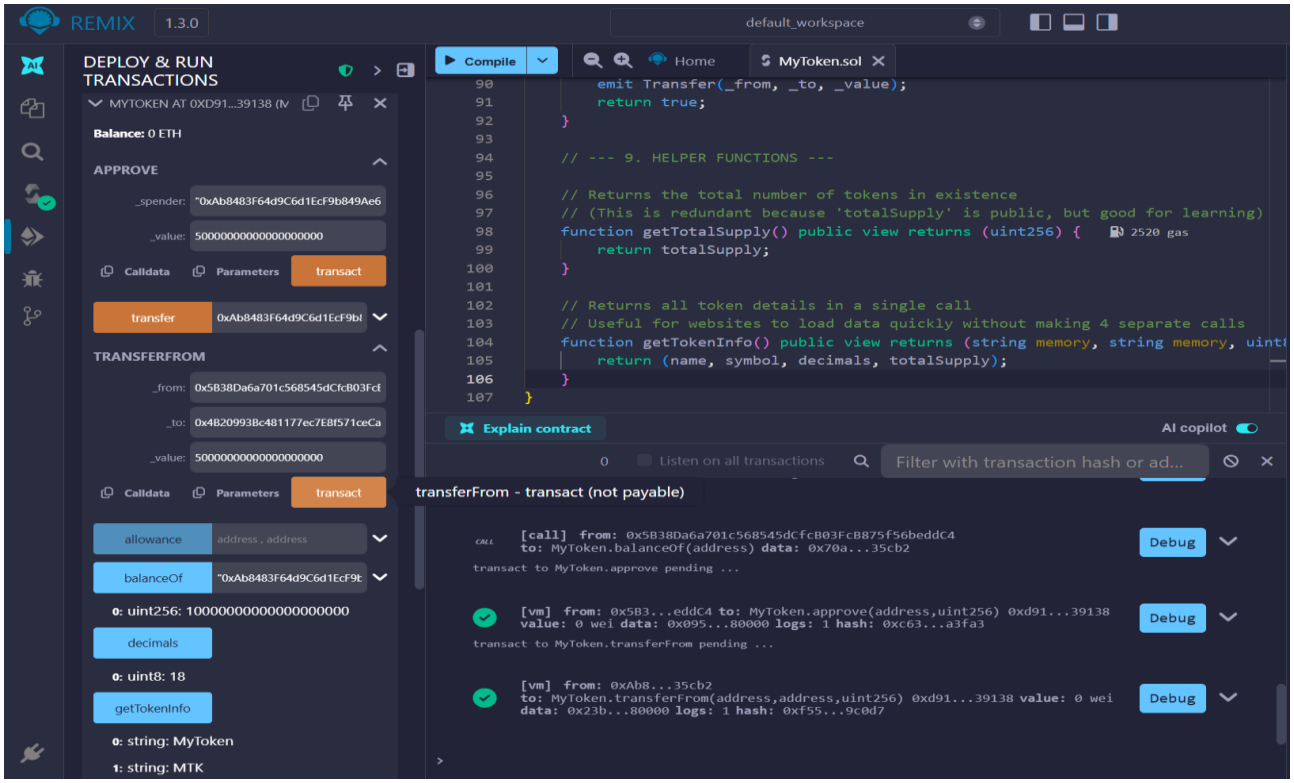




C. Core Functionality: Transfers & Events

The standard transfer function was tested. The transaction succeeded, balances were updated, and the blockchain emitted the correct Transfer event logs.

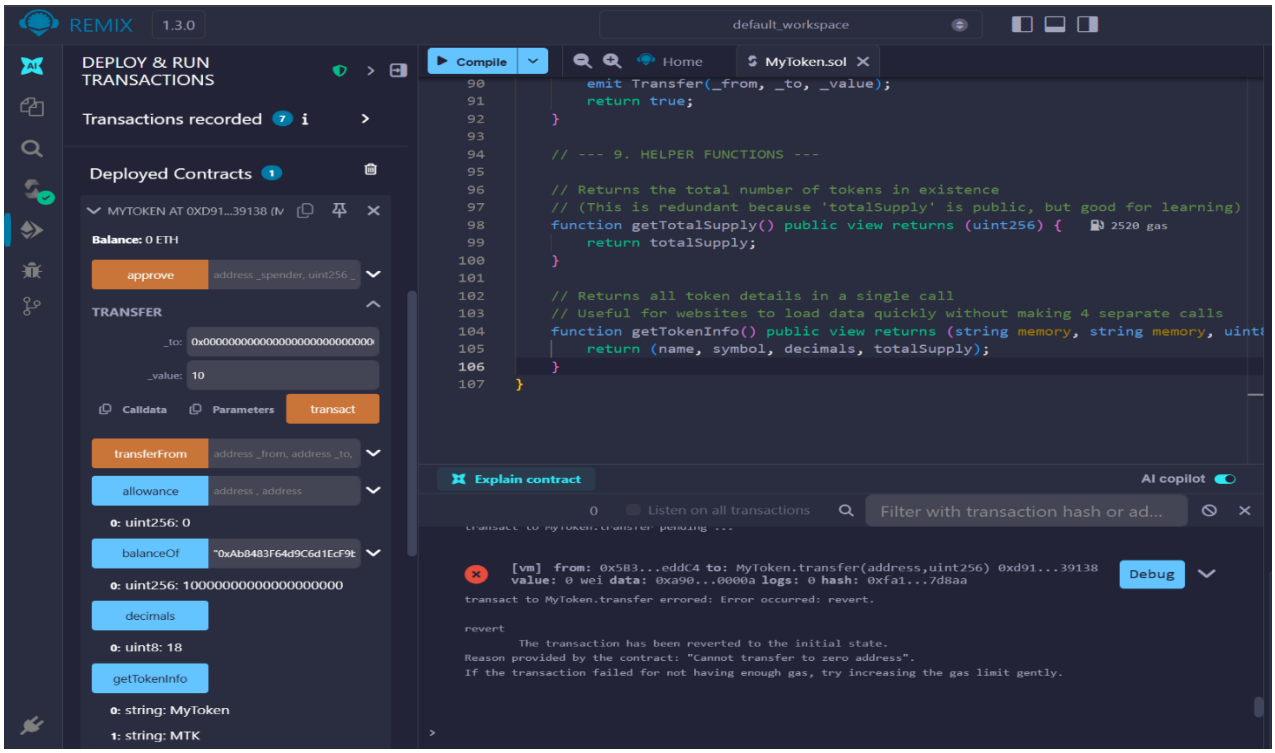




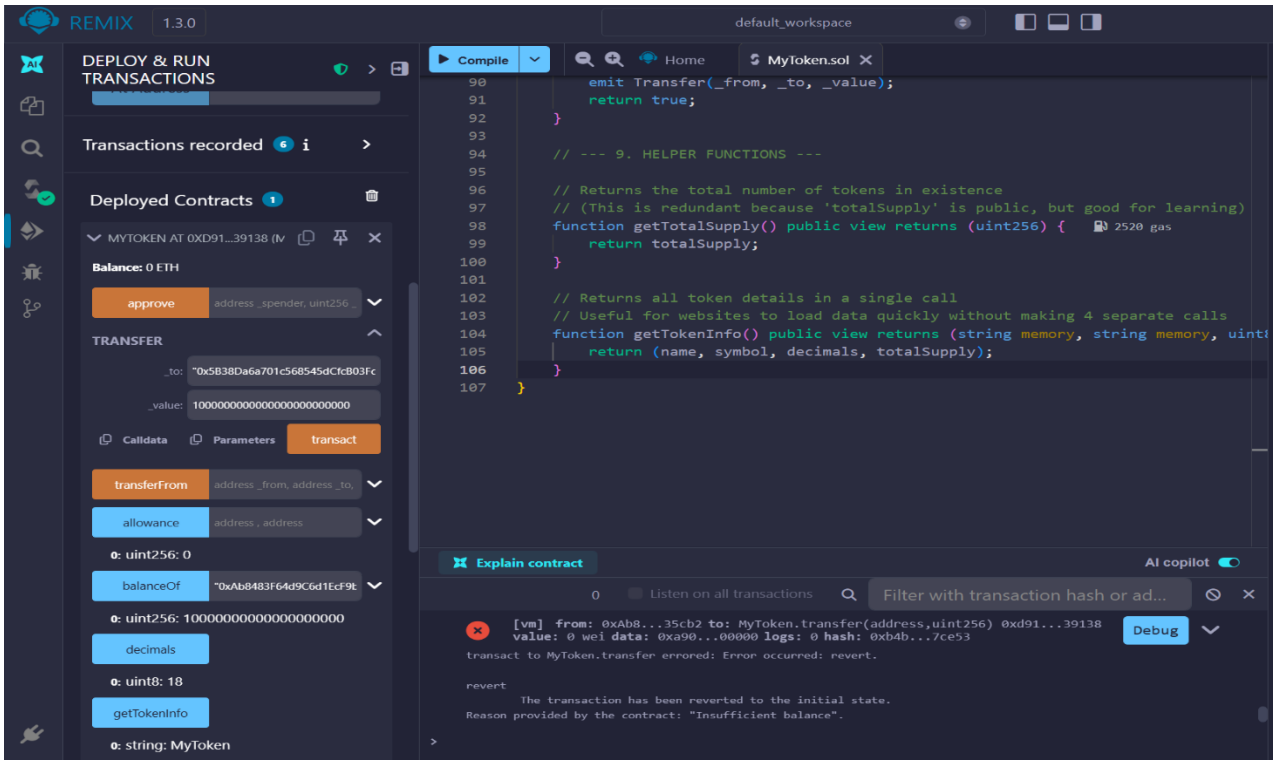
D. Security & Error Handling

To ensure contract integrity, specific edge cases were tested to verify that transactions revert (fail) under invalid conditions.

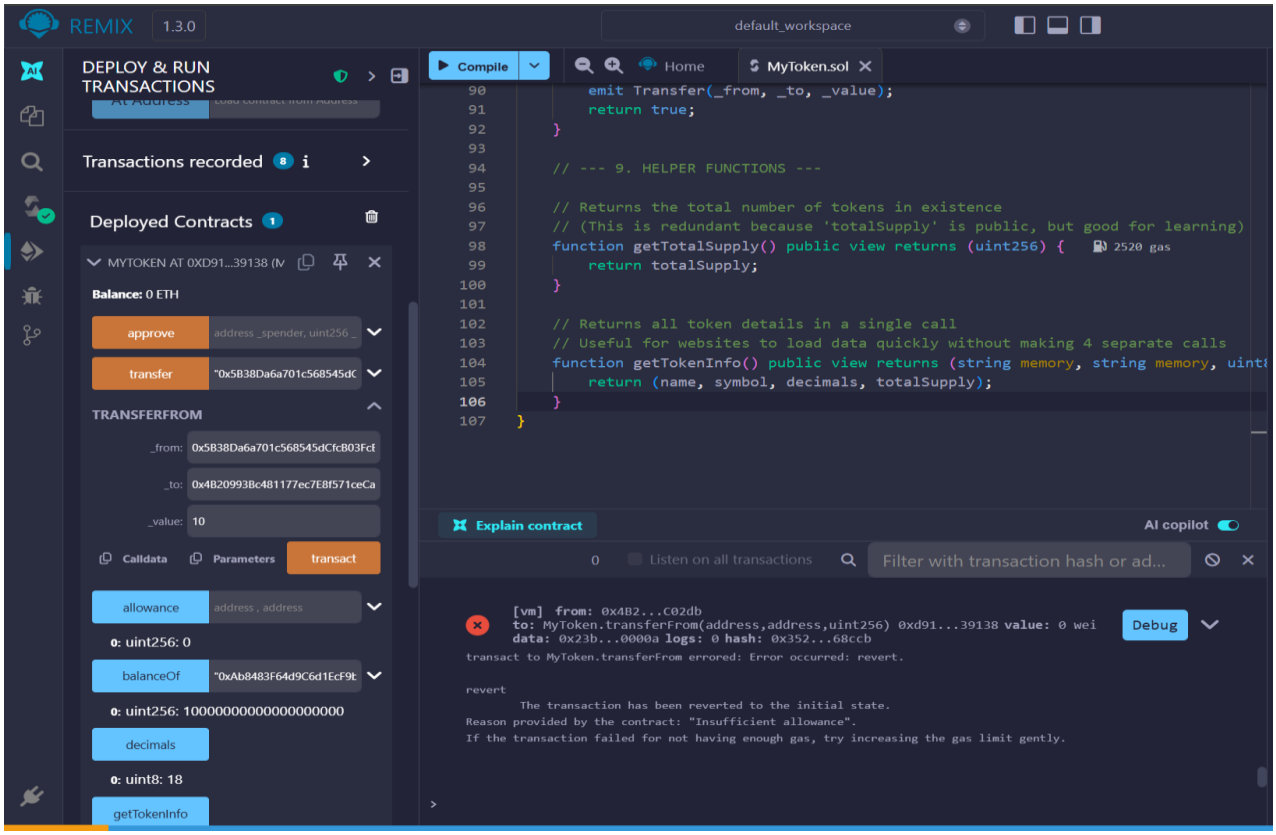
1. Zero Address Validation: Attempting to transfer tokens to the 0x000... address correctly reverts the transaction to prevent accidental token burning.



2. Balance Validation: Attempting to transfer more tokens than the sender possesses correctly reverts the transaction.



3. Allowance Validation: Attempting to spend tokens on behalf of another user without sufficient approval (transferFrom) correctly reverts.



6. How to Run This Project

1. **Open Remix:** Navigate to Remix IDE.
2. **Create File:** Create MyToken.sol in the contracts folder.
3. **Compile:** Use the Solidity Compiler tab (ensure version 0.8.x is selected).
4. **Deploy:**
    - Go to the "Deploy & Run Transactions" tab.
    - Select "Remix VM" as the environment.
    - Deploy the contract (Input initial supply if required by constructor).
5. **Interact:** Use the deployed contract buttons to execute transfers and check balances.

---

7. What I've Learned

Building this ERC-20 token provided practical insight into the mechanics of the Ethereum blockchain. Beyond just writing code, I gained a deeper understanding of:

- **The Power of Standards:** I learned why the ERC-20 interface is critical. By strictly following function names like transfer and approve, my token is automatically compatible with crypto wallets and exchanges.
- **State Management in Solidity:** I mastered using mapping to track user balances and allowances (mapping(address => uint256)), which is far more efficient than array loops used in traditional programming.
- **Security & Validation:** My testing highlighted the importance of require() statements. Without explicitly checking for the **Zero Address** or **Insufficient Balance**, the contract would be vulnerable to burning tokens or invalid arithmetic.
- **The "Decimals" Concept:** I learned that Solidity does not handle floating-point numbers. Managing 18 decimals means interacting with huge integers (1 Token = 1,000,000,000,000,000,000 wei), which requires careful calculation.
- **Event Logging:** I discovered that smart contracts are often "blind" to the outside world, but by emitting Events (like Transfer), I can create a trail of data that frontend applications can read and display to users.

---

8. Conclusion

This project successfully transformed a theoretical concept into a live, functional digital asset. By deploying **MyToken (MTK)**, I have not only built a cryptocurrency from scratch but also validated its security and functionality through rigorous testing in the Remix IDE.

This project serves as the "Hello World" of my blockchain journey, proving that I can write, deploy, and debug smart contracts. It lays the groundwork for more advanced development, such as building Decentralized Applications (DApps), integrating with frontend frameworks, or exploring complex DeFi protocols.