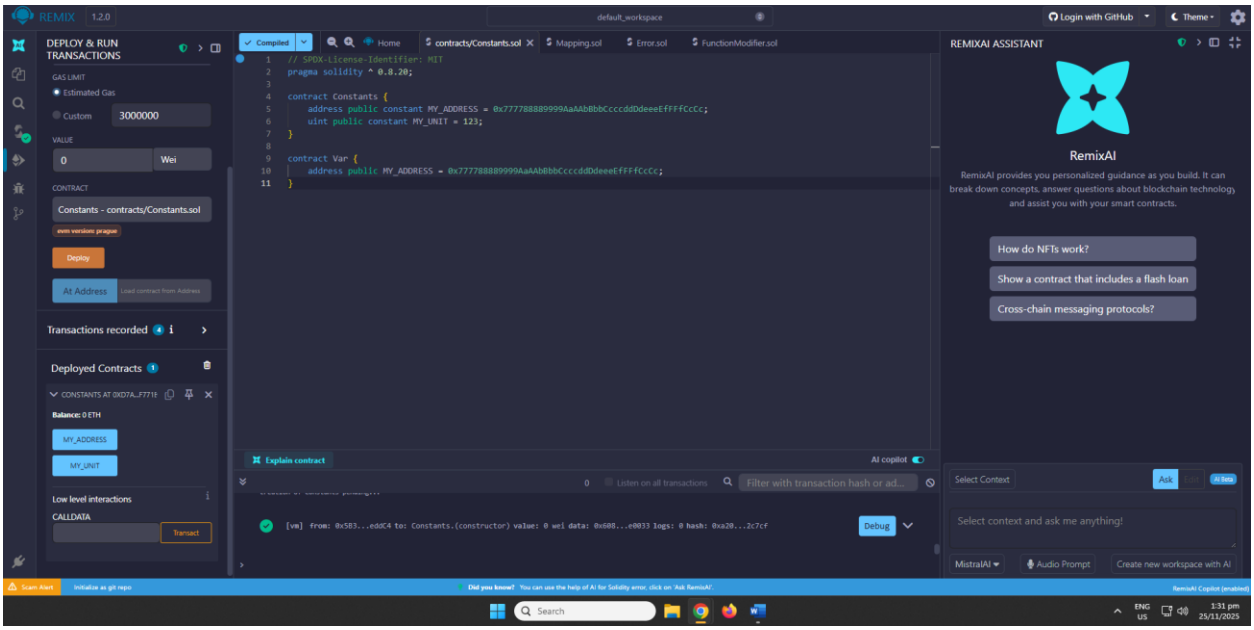
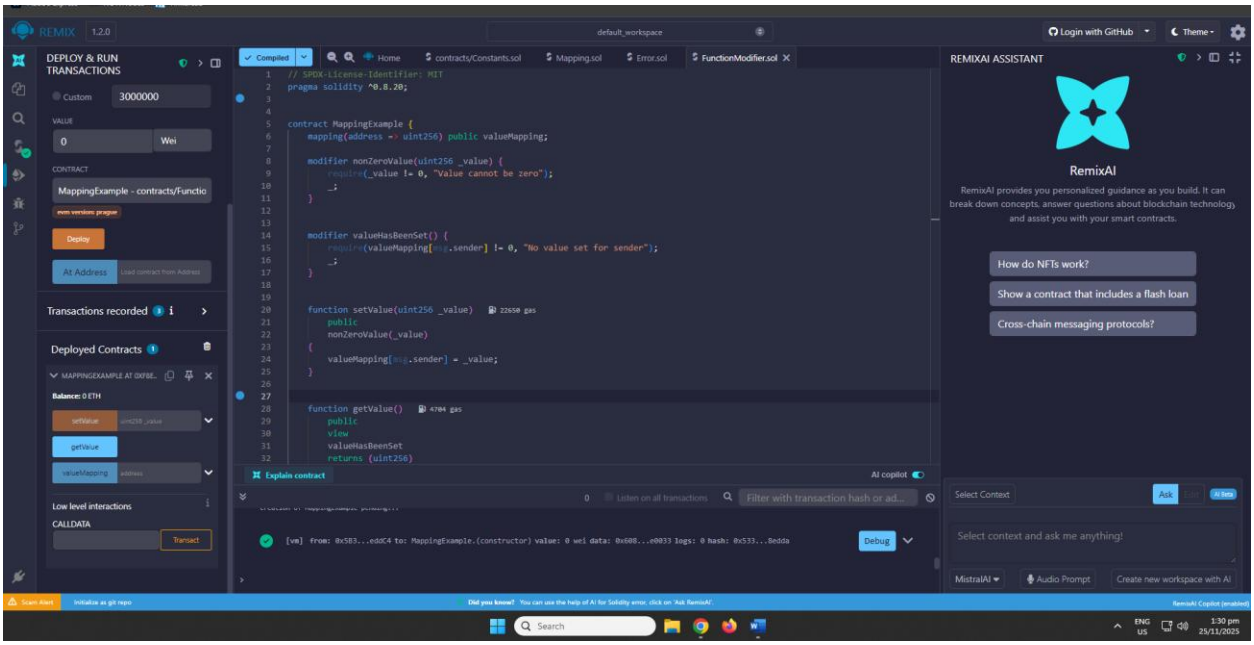
CORTEZ, LAWRENCE NEIL M.
NW-301

Constants



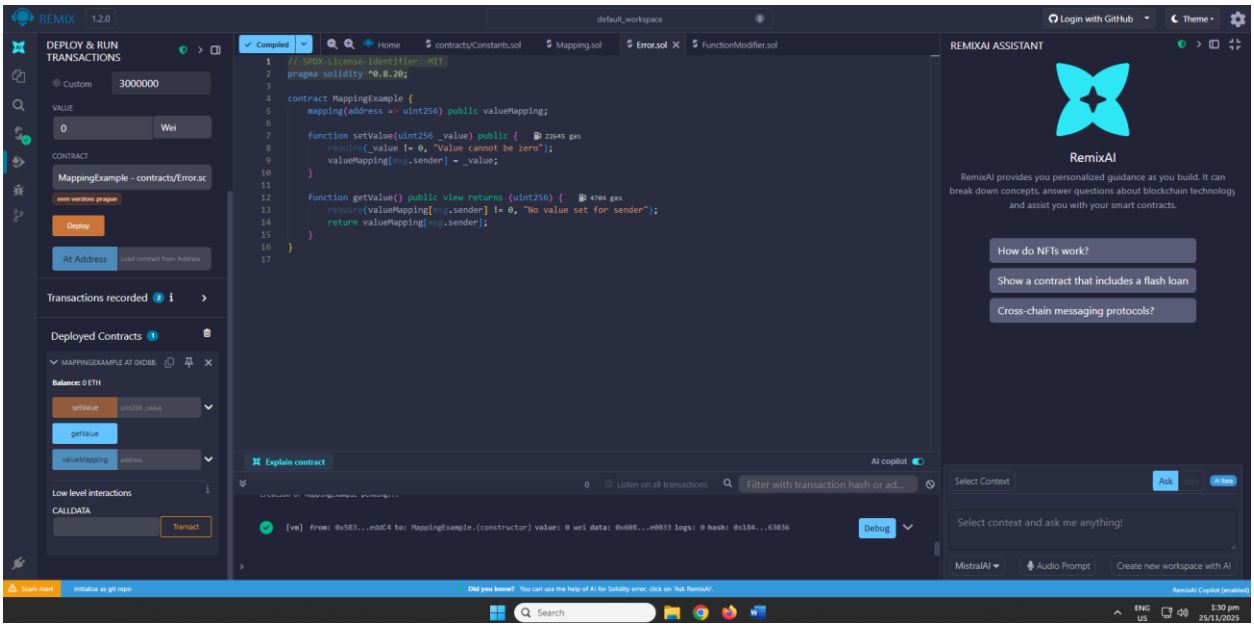
CORTEZ, LAWRENCE NEIL M.
NW-301

Mapping



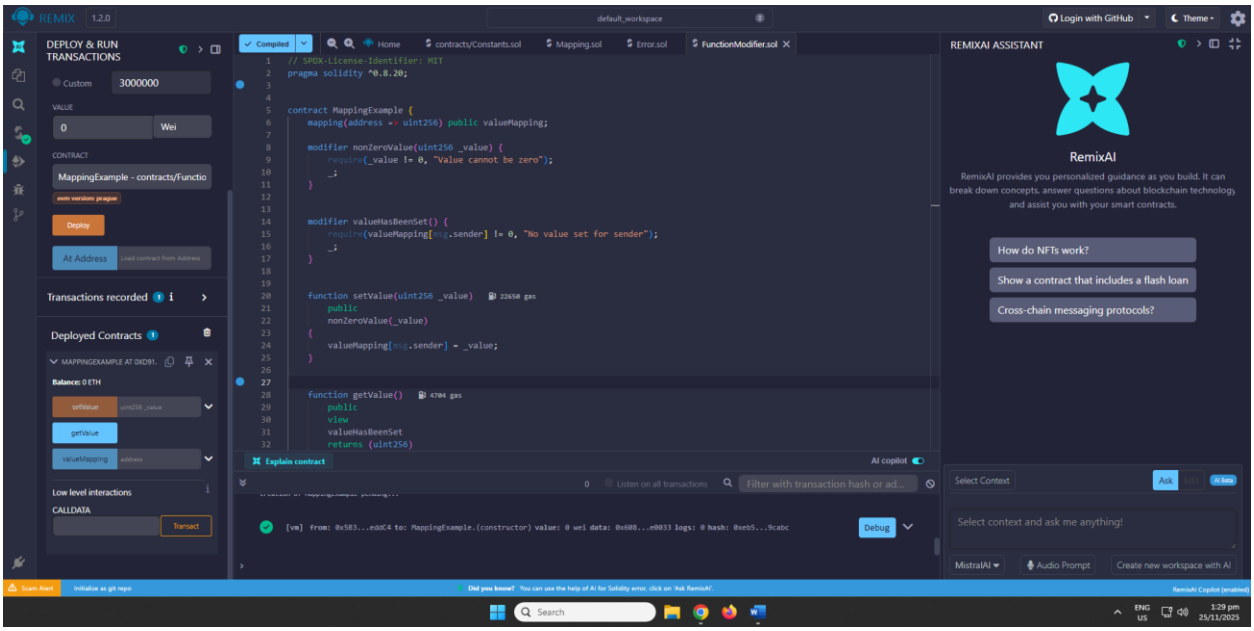
CORTEZ, LAWRENCE NEIL M.
NW-301

ErrorHandling



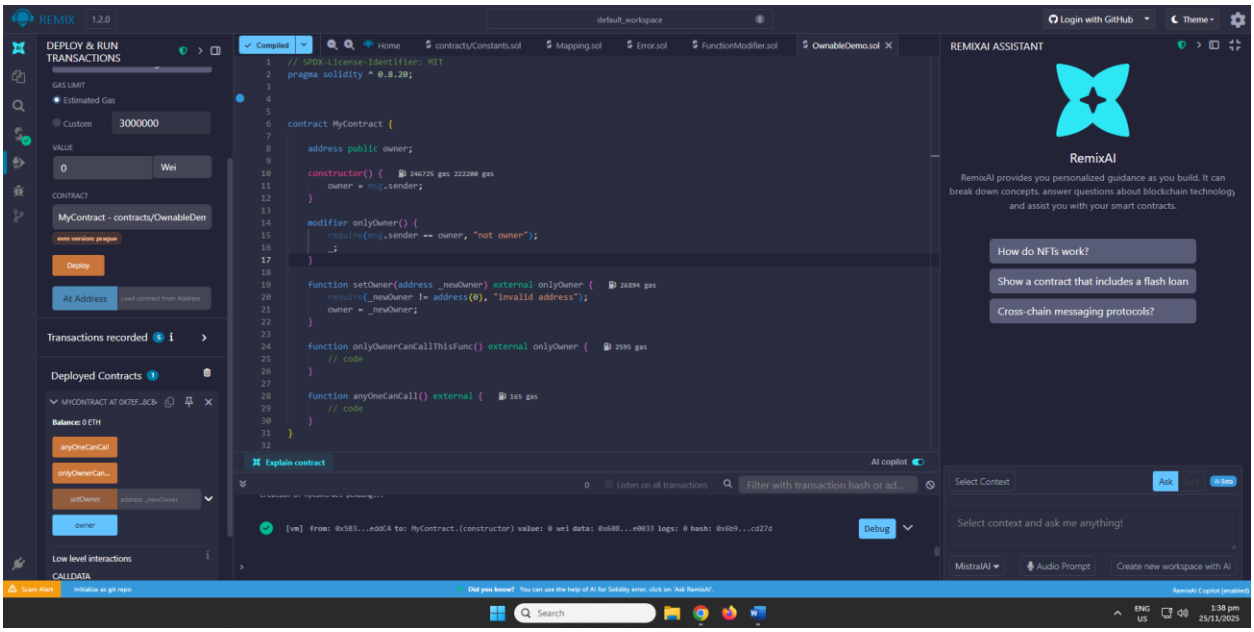
CORTEZ, LAWRENCE NEIL M.
NW-301

FunctionModifier



CORTEZ, LAWRENCE NEIL M.
NW-301

OwnableDemo



- **Function Modifiers & Ownable**
  When should you use a modifier like onlyOwner instead of inline checks, and what risks arise if ownership isn't managed properly?

  You should use a modifier like onlyOwner when the same access control logic is applied to multiple functions. Modifiers improve readability, reusability, and maintainability compared to repeating inline require checks in each function. If ownership isn't managed properly—such as allowing an invalid address to become owner, or failing to restrict critical functions—unauthorized users could gain control, leading to loss of funds, contract misuse, or security vulnerabilities.

CORTEZ, LAWRENCE NEIL M.
NW-301

- **Error Handling**
  How do you choose between require, revert, and assert, and why might custom errors be better than error strings?

  I choose between require, revert, and assert based on the type of condition I am checking. I use require for validating inputs or external conditions, because it stops execution and refunds remaining gas. Revert is used when I want to explicitly terminate execution in more complex scenarios. Assert is reserved for internal invariants that should never fail; failing an assert consumes all remaining gas. Custom errors are often better than string messages because they are more gas-efficient—they encode the error type instead of storing a string in memory, which saves cost on deployment and frequent function calls.

- **Constants & Variables**
  When should a value be constant, immutable, or mutable, and how does that choice affect gas cost and flexibility?

  I use constant for values that will never change, as they are stored in bytecode and are very gas-efficient. I use immutable for values that are set once during contract deployment and cannot change afterward, offering a balance between flexibility and gas savings. Regular mutable variables are used when the value needs to change over time, but they are more expensive in terms of gas because they are stored in contract storage. Choosing the right type ensures a balance between efficiency, flexibility, and contract functionality.

GITHUB LINK:

https://github.com/L-Cortez/BLOCKTECH/tree/main/Prelim%20Lab%20Activity%202%20Solidity%20Basics%20II

CORTEZ, LAWRENCE NEIL M.
NW-301