Test Case 1

You need to build python api with FastAPI framework to get current balance (token amount and usd amount) of CRV mainnet token for specific wallet:
https://etherscan.io/address/0xD533a949740bb3306d119CC777fa900bA034cd52

1. First endpoint should return current wallet balances for given param: wallet=0x…After fetching balance, app should save next data to mongoDB:wallet, last update time, current balance, current balance usd, history of balances (isotimestamp-value) for token and usd balances.
2. Second endpoint should return existing history from mongodb for given param: wallet=0x…

Please DO NOT use etherscan api to get token balance, you need to get it from blockchain node with web3py (or other methods you would like).

Final result must have a docker-compose file and be ready for testing with only one command: docker-compose up.

Wallet for testing: 0x7a16ff8270133f063aab6c9977183d9e72835428

Sources:

- Web3py – https://web3py.readthedocs.io/en/stable/
- CoinGecko Pricing API – https://www.coingecko.com/ru/api/documentation

Test Case 2

```
Unset
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

// Hi dear candidate!
// Please review the following contract to find the 2 vulnerbilities that
results in loss of funds.(High/Critical Severity)
// Please write a short description for each vulnerbillity you found alongside
with a PoC in hardhat/foundry.
// Your PoC submission should be ready to be run without any modification
// Feel free to add additional notes regarding informational/low severity
findings
```

```
import "openzeppelin-contracts/contracts/access/Ownable.sol";
import "openzeppelin-contracts/contracts/security/ReentrancyGuard.sol";

contract PonziContract is ReentrancyGuard, Ownable {
    event RegistrationDeadline(uint256 registrationDeadline);
    event Withdraw(uint256 amount);

    uint256 private registrationDeadline;

    address[] public affiliates_;
    mapping(address => bool) public affiliates;

    uint256 public affiliatesCount;

    modifier onlyAfilliates() {
        bool affiliate;
        for (uint256 i = 0; i < affiliatesCount; i++) {
            if (affiliates_[i] == msg.sender) {
                affiliate = true;
            }
        }

        require(affiliate == true, "Not an Affiliate!");
        _;
    }

    function setDeadline(uint256 _regDeadline) external onlyOwner {
        registrationDeadline = _regDeadline;
        emit RegistrationDeadline(registrationDeadline);
    }

    function joinPonzi(
        address[] calldata _afilliates
    ) external payable nonReentrant {
        require(
            block.timestamp < registrationDeadline,
            "Registration not Active!"
        );
        require(_afilliates.length == affiliatesCount, "Invalid length");
        require(msg.value == affiliatesCount * 1 ether, "Insufficient Ether");
        for (uint256 i = 0; i < _afilliates.length; i++) {
            _afilliates[i].call{value: 1 ether}("");
        }
        affiliatesCount += 1;
```

```
        affiliates[msg.sender] = true;
        affiliates_.push(msg.sender);
    }

    function buyOwnerRole(address newAdmin) external payable onlyAfilliates {
        require(msg.value == 10 ether, "Invalid Ether amount");
        _transferOwnership(newAdmin);
    }

    function ownerWithdraw(address to, uint256 amount) external onlyOwner {
        payable(to).call{value: amount}("");
        emit Withdraw(amount);
    }

    function addNewAffilliate(address newAfilliate) external onlyOwner {
        affiliatesCount += 1;
        affiliates[newAfilliate] = true;
        affiliates_.push(newAfilliate);
    }

    receive() external payable {}
}
```