Курс “Блокчейн” | Д/З

№1 К занятию “Обзорная лекция - смарт-контракты”, 6 баллов

Задание:

Просмотреть код контрактов заданного преподавателем проекта, найти место, реализующее определенную логику и привести diff (результат работы git diff), изменяющий поведение смарт-контракта:

- https://github.com/gnosis/MultiSigWallet/blob/master/contracts/MultiSigWallet.sol - сделать, чтобы с баланса multisig-контракта за одну транзакцию не могло бы уйти больше, чем 66 ETH
- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/f2112be4d8e2b8798f789b948f2a7625b2350fe7/contracts/token/ERC20/ERC20.sol - сделать, чтобы токен не мог бы быть transferred по субботам
- https://github.com/mixbytes/solidity/blob/076551041c420b355ebab40c24442ccc7be7a14a/contracts/token/DividendToken.sol - сделать чтобы платеж в ETH принимался только специальной функцией, принимающей помимо ETH еще комментарий к платежу (bytes[32]). Простая отправка ETH в контракт запрещена

Как выполнить:

- github репозиторий с одним файлом README.md, в котором приведены результаты git diff всех заданий

№2 К занятию “Разработка смарт-контрактов”, 4 балла

Задание:

Выбрать идею курсового проекта:

- аукцион на покупку некоторого товара
- DAO с долевым токеном и голосованиями за proposals
- выдача цифрового сертификата (NFT) с процедурой offchain доказательства владения им
- продажа и offchain-offline предъявление билетов на мероприятия
- onchain регистрация и платная подписка на сервисы
- Verified Credentials - выдача и проверка некоторого документа удостоверяющим центром, его проверка
- сбор и аналитика данных в одном из блокчейнов: Ethereum, Avalanche, TON, Polkadot, Solana, etc,,,
- ваша идея, согласованная с преподавателем

Создать template проекта в Hardhat/Brownie (что больше понравится)

- пустые базовые контракты
- пустые тесты базового функционала
- (опционально) любые предварительные интерфейсы, mocks, описания ключевых функций

Как выполнить:

- github репозиторий с README.md, с базовым описанием проекта

№3 К занятию “Blockchain software ecosystem”, 8 баллов

Задание:

- подключиться к Ethereum ноде и замониторить отправку цен токенов: USDT/ETH, USDC/ETH и LINK/ETH из Chainlink
  - зарегистрироваться и получить бесплатный доступ к archive ноде Ethereum в сервисе Alchemy
  - найти oracle feeds для заданных пар токенов в Chainlink
    - пары: ETH/USD, LINK/ETH, USDT/ETH
  - написать монитор:
    - непрерывно мониторящий последние блоки
    - выводящий в лог все новые события изменения цен

Как выполнить:

- github репозиторий с файлами проекта, в README.md кроме инструкций должен быть образец вывода в лог

№4 К занятию “Solidity + EVM, low-level patterns”, 10 баллов

Задание:

- создать в форке mainnet собственный токен
- сделать swap-пару в Uniswap v2
- провести обмен одного токена на другой
- в Hardhat или Brownie

Как выполнить:

- репозиторий на Github, в README кроме инструкций по запуску - образец вывода теста, который в форке майннет проводит все операции

№5 К занятию “Solidity, typical patterns”, 15 баллов

Задание:

- Write simple voting contract and cover with tests
- Description
  - A pack of contracts that allows users to vote for proposals, using token balances. Users own an ERC20 token, representing “voting power” or

DAO ownership shares. Proposals are simply the keccak256 hashes and can be “accepted”, “rejected” or “discarded” (if TTL of proposal is expired). The fact of acceptance of a proposal is fixed in the event, nothing else is stored in contracts.

- User story
  - A,B,C have 25, 40 and 35 voting tokens of total 100. “A “creates a proposal (text document, having hash) and publishes this hash in contract, voting with her 25 tokens “for” it. Then B also votes “yes” with his 40 tokens. So, 25+40 > 50% of total votes (100), proposal is accepted: event is fired, proposal is removed from queue. Same situation with proposals when > 50% of “no” votes is gathered. If a proposal stays in an indefinite state (no threshold votes gathered) until TTL expires, it cannot be “accepted” or “declined” and will be thrown away with “discarded” status next time when a new proposal is created.
  - [NOTE] business logic can slightly differ in your implementation if needed
- Requirements
  - Business logic requirements:
    - During creation totalSupply = 100.000000 (decimals = 6) tokens are minted to contract owner
    - Any owner of voting tokens can create a proposal, time-to-live(TTL) of proposal is 3 days, after that time proposal becomes “discarded” if not enough votes are gathered
    - Votes can be “for” or ”against” the proposal. Proposal becomes “accepted” or “declined” completed if > 50% of votes for the same decision (“for” or “against”) is gathered
    - When votes threshold is reached, event is emitted and proposal is removed from queue
    - There are no more than N=3 current proposals, new proposals cannot be added until old ones will be “accepted”, “declined” or “discarded” by TTL
    - If > 1 old proposals are obsolete, then addition of a new proposal automatically “kicks out” the most obsolete proposal, making it “discarded”.

        - voting should not “freeze” tokens
        - but, voting should handle a situation, when voter transfers his tokens to another address and votes another time
- Contracts requirements
    - Contracts should be written in Solidity
    - Contracts should follow official Solidity style guide
    - All functions should contain good comments
    - Functions should optimally use gas
    - Functions should contain checks to disallow possibility of contract DoS
    - Contracts should contain “view” functions, useful for building DApp for this voting
    - Project requirements
        - Project should be built using Hardhat or Brownie framework
        - Tests should cover all normal workflows of voting
        - Tests should cover EACH condition, leading to revert
        - Project should contain README.md with instructions how to build and run tests on Ubuntu 20.04+

Как выполнить:

- репозиторий на Github, в README образец вывода теста, который проверяет все базовые функции.

№6 К занятию “DeFI, protocols code review”, 8 баллов

Задание:

- в форке mainnet и Hardhat/Brownie
- провести три swapa в Uniswap V2 с использованием flashloan
    - циклический маршрут, например:
        - wETH -> LINK, LINK -> USDT, USDT->wETH
    - взять flashloan
    - выполнить swaps

    - вернуть flashloan (с убытками)

Как выполнить:

- репозиторий на Github, в README.md - образец вывода теста, который проверяет все базовые функции.

№7 К занятию “Symmetric cryptography”, 6 баллов

Задание:

- Сделать программу - консольное приложение, используя языки: C/C++, Rust, Python, Go, Node.JS.
- Программа принимает три параметра командной строки, типа:
- ./program --file <filename> --numbilets 20 --parameter 42
    - Параметры: имя файла с ФИО студентов, число билетов, параметр, изменяющий распределение. Программа выдает в консоль строку из файла + номер билета. Номера билетов детерминировано связаны с ФИО и параметром, меняющим распределение. Распределение должно быть максимально равномерным
- Входные данные (параметры командной строки):
    - 1) файл, где каждая строка - это ФИО студента, типа:
        - Иванов Иван Иванович
        - Ярцев Ярослав Ярославович
        - …
        - Петров Петр Петрович
    - 2) число билетов N (билеты нумеруются с 1 до N (включая N))
    - 3) численный параметр, детерминированно меняющий распределение (при его изменении распределение номеров билетов максимально изменяется). При использовании одного и того же параметра, одна и та же строка Фамилия-Имя из файла всегда генерирует один и тот же номер билета
- Выходные данные: вывод в STDOUT строк вида:
    - Иванов Иван Иванович: 21

    - Ярцев Ярослав Ярославович: 12
    - …
    - Петров Петр Петрович: 11

Как выполнить:

- github репозиторий с файлами проекта
    - отдельный репозиторий на GitHub с образцом вывода в README.md

№8 К занятию “Peer-to-peer networks and consensus”, 13 баллов

Задание:

- Необходимо разработать простейший DApp который:
    - принимает от пользователя бинарный файл (например, PNG) и загружает его в IPFS (в свою ноду)
    - хеш загруженного файла отправляет на хранение в простой смарт-контракт, хранящий mapping (address => ipfs_hash)
    - по запросу, получает ipfs_hash из контракта (используя текущий адрес), достает его из IPFS, и демонстрирует в браузере
- Проект должен работать на локальных dev-версиях софта: Ethereum нода, IPFS нода, node.js запущенный локально
- желательно чтобы контракт эффективно использовал storage, не тратя лишние байты на хранение
- (большой плюс) продумать как разместить и разместить frontend проекта без сервера в ipfs

Как выполнить:

- github репозиторий с файлами проекта
    - В README должна быть рабочая инструкция как с нуля запустить проект на локальной машине (запуск локальной ноды, деплой контракта в нее, запуск локальной IPFS ноды, запуск и конфигурирование DApp (указание в нем адреса контракта и location IPFS ноды) и описание последовательности действий чтобы воспроизвести цепочку

“залить файл, записать в контракт, прочитать из контракта, скачать и показать файл”. Несмотря на простоту контракта, тесты должны быть представлены хотя бы минимально.