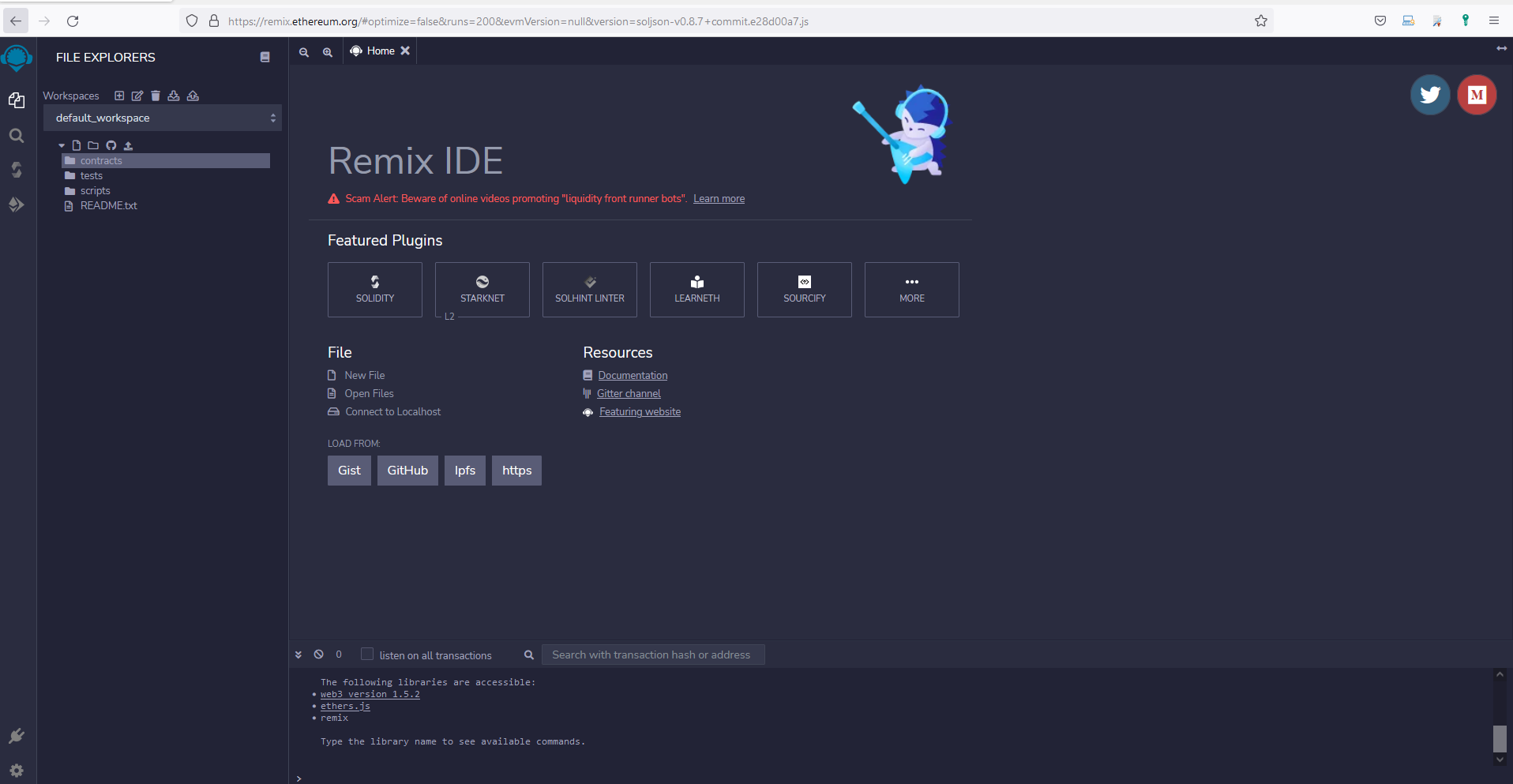
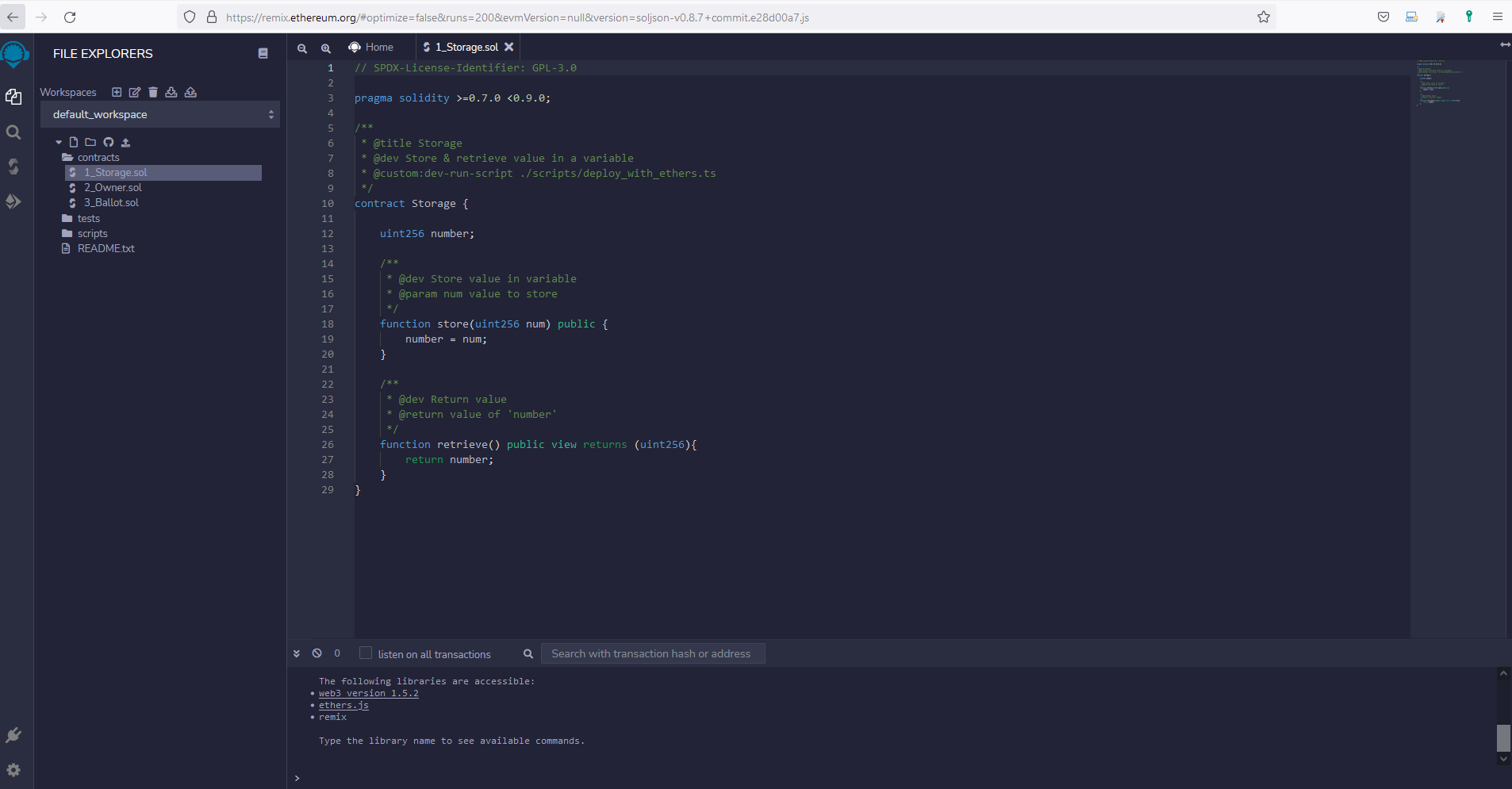
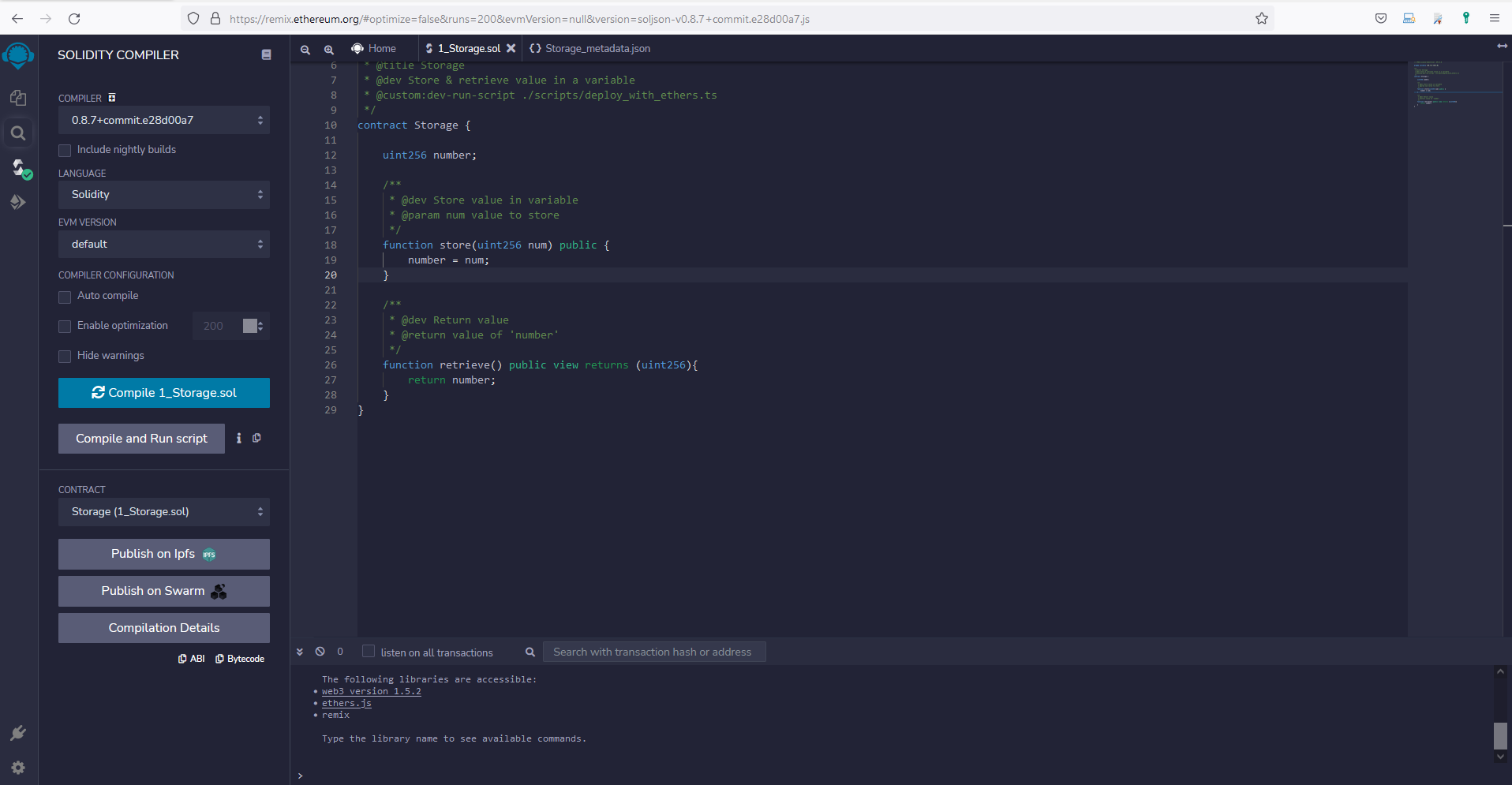
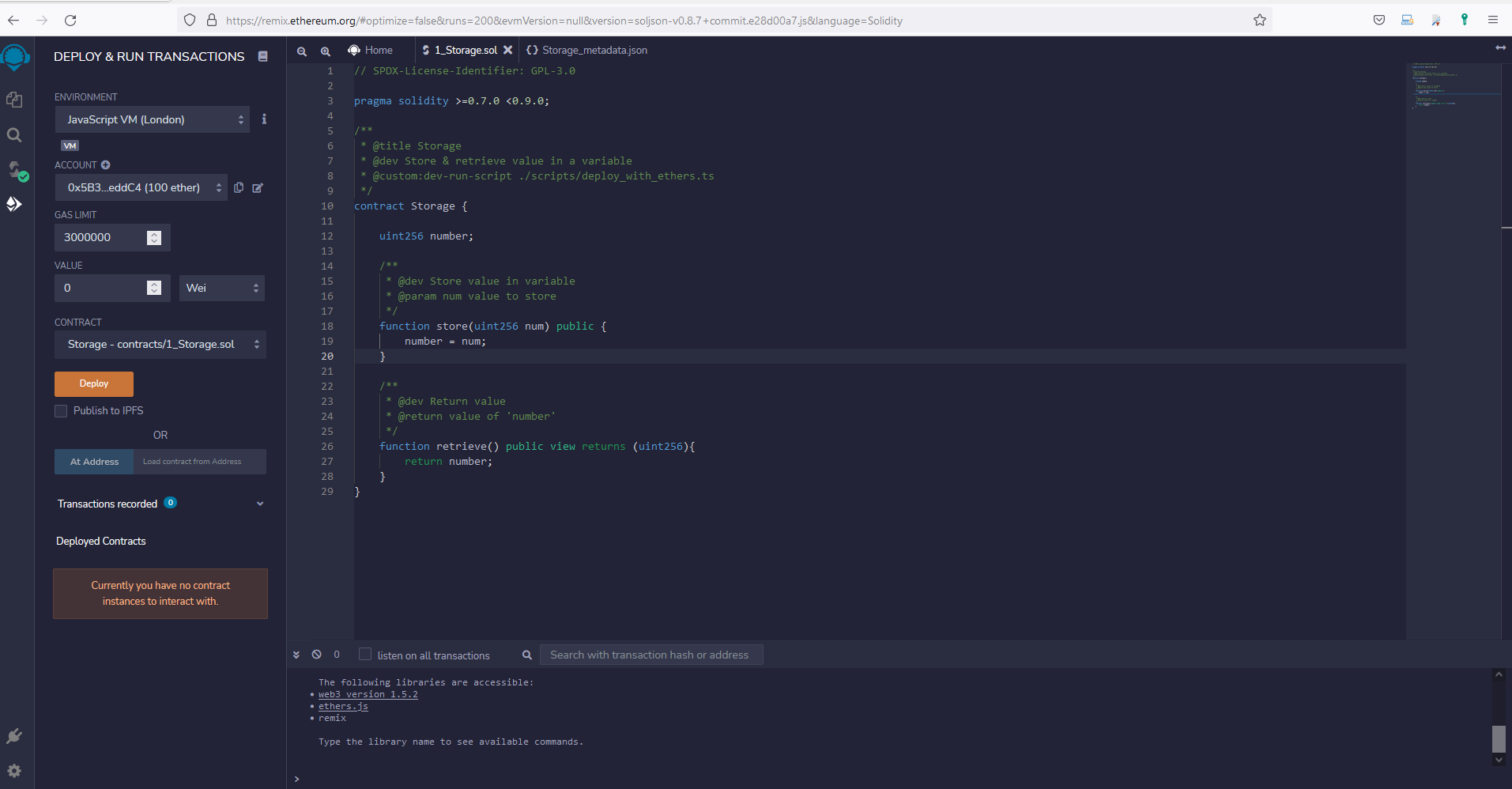
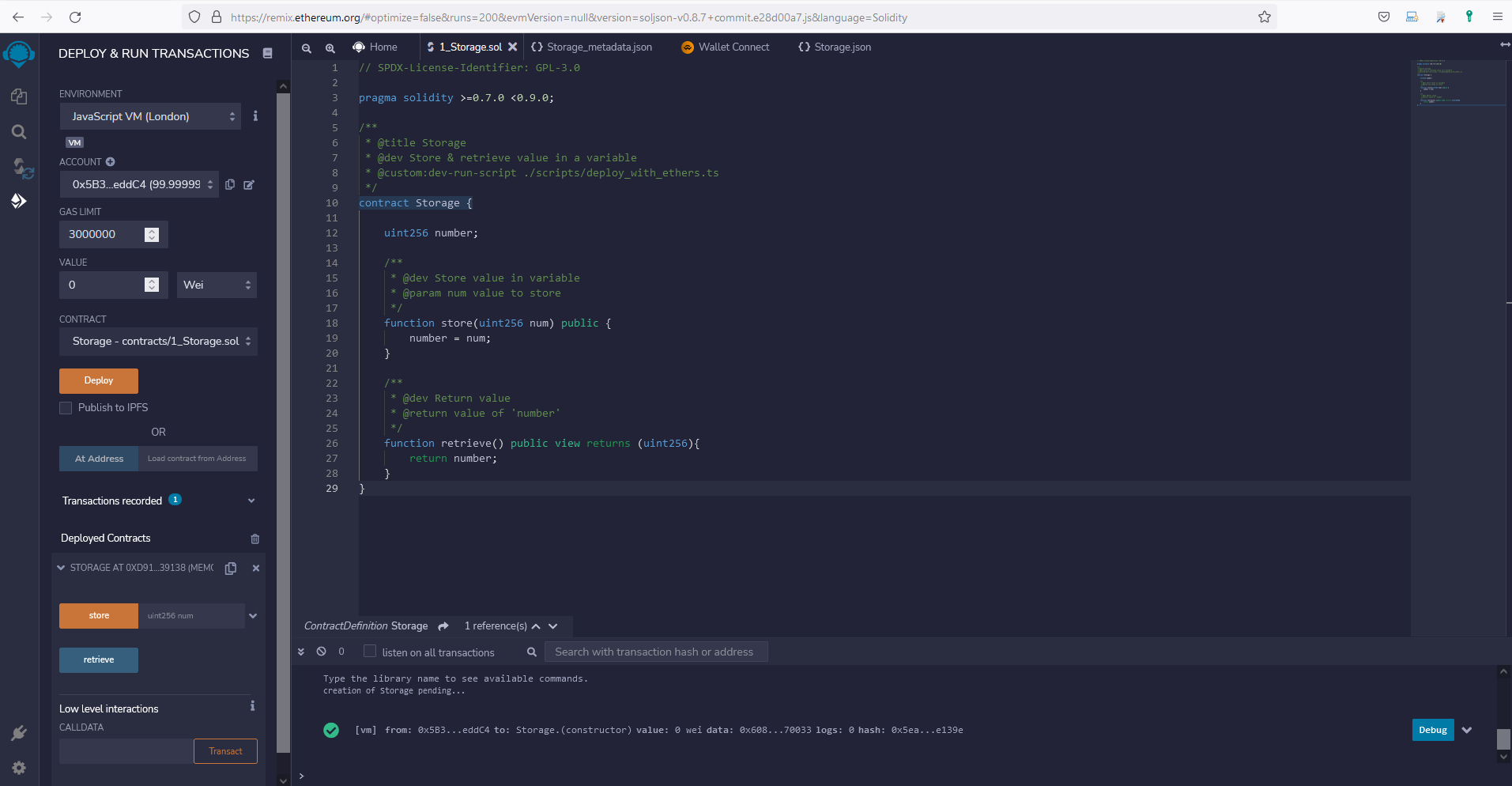
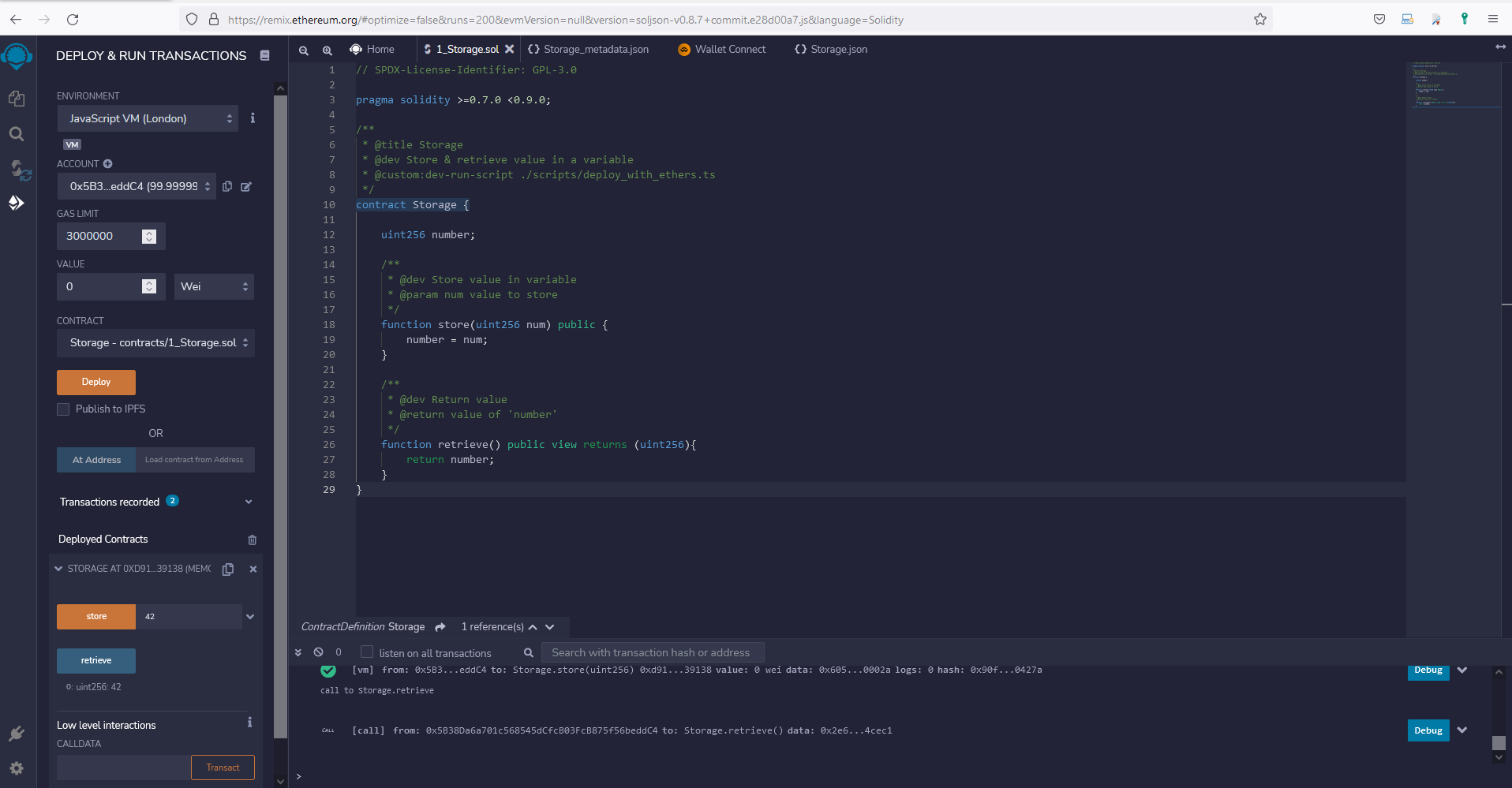
Лабораторная работа №3 – Работа в IDE Remix. Написание и тестирование простейших смарт-контрактов.

1. Запускаем браузер и открываем <https://remix.ethereum.org/>   
   Откроется окно IDE Remix  
     
   Окно содержит 4 основные рабочие зоны (слева направо – вертикальный тулбар с кнопками, панель выбранного режима (меняется в зависимости от того, какая кнопка нажата на тулбаре) и занимающий большую часть экрана редактор исходного текста, под которым находится панель с логом работы.
2. В режиме File Explorers можно выбирать файл, с которым мы хотим работать в редакторе, а также выполнять с файлами различные действия (добавлять, удалять, располагать в папках и т.п.). По умолчанию для файлов контрактов создается папка contracts, в которой изначально находится набор файлов примеров контрактов.
3. Откроем файл 1\_Storage.sol из папки contracts в редакторе и прочитаем его исходный текст.  
     
   Как нетрудно видеть, данный контракт представляет собой простейшее хранилище для единственного целочисленного значения, обладающее функциями чтения (retrieve) и записи (store) этого значения.
4. Перейдем к компиляции смарт-контракта. Для этого необходимо нажать третью сверху кнопку на тулбаре - . После этого на панели отобразится набор параметров компилятора и кнопка Compile. Нажмем на нее и увидим, что под кнопкой компиляции появилась секция Contract и в выпадающем списке появилось имя нашего контракта. При неуспешной компиляции в этом месте было бы отображено сообщение об ошибке и ее описание. Кроме того, на данной панели можно задать дополнительные параметры компиляции, такие как язык и версию его компилятора, версию EVM (виртуальной машины эфира) под которую проводится компиляция и другие настройки.  
     
   Компиляция контракта, по сути, представляет собой формирование байт-кода самого контракта и ABI-описания интерфейса этого контракта, которые в дальнейшем используются для установки контракта в блокчейн-сеть и вызов его, соответственно. После успешной компиляции вы можете получить эти артефакты в буфер обмена нажав соответствующие кнопки в нижней части панели. Это может быть полезно при ручной работе с контрактом вне IDE (например, вы можете сформировать транзакцию по установке контракта с помощью консоли web3 самого узла, кошельке или другом инструменте).
5. Когда контракт успешно скомпилирован, его можно установить в блокчейн-сеть для дальнейшего использования. Для того, чтобы сделать это средствами IDE Remix, нажмем на четвертую кнопку на тулбаре - . После этого на панели отобразится набор параметров установки контракта.   
     
     
     
   Основные важные параметры – это:
   1. Environment – выбор окружения, в котором будет установлен смарт-контракт. Для выбора доступны как варианты локальных виртуальных машин самой IDE (Javascript VM), так и реальных блокчейн-сетей, доступ к которым осуществляется с помощью соответствующего провайдера. Injected provider подразумевает подключение с использованием Metamask или аналогичного расширения браузера. Web3 provider подразумевает прямое подключение к реальному узлу Ethereum (geth, например), Hardhat и Ganache provider предназначены для подключения к отладочным узлам Hardhat и Ganache соответственно. Wallet connect служит для подключения через совместимые мобильные кошельки по QR-коду. Для целей отладки кода контракта наиболее удобно использовать локальную ВМ (Javascript VM).
   2. Account – выбор аккаунта, с которого будет производится транзакция по установке смарт-контракта. При использовании виртуальной машины заранее будет создан ряд аккаунтов, на каждом из которых будет иметься по 100 ETH, аккаунты можно переключать в выпадающем списке.
   3. Value – возможность передать средства в транзакцию установки контракта.
   4. Contract – выбор контракта для установки.
   5. Кнопка Deploy, выполняющая установку контракта, и, при наличии параметров конструктора – поле для ввода этих параметров рядом с кнопкой.

После настройки параметров и нажатия кнопки Deploy происходит формирование и исполнение транзакции установки смарт-контракта. При успешном выполнении транзакции в логе (ниже редактора исходного кода) видно сообщение об этом, а в панели внизу появляется установленный контракт с его адресом в списке Deployed Contracts. Кликом по значку “>” перед его именем можно открыть интерфейс использования контракта, содержащий вертикально расположенный набор кнопок, каждая из которых соответствует определенной функции контракта. Для функций, принимающих параметры, рядом с кнопкой представлено поле ввода этих параметров. 

1. Введем в поле параметров функции store значение и нажмем кнопку store, чтобы сохранить это значение в хранилище смарт-контракта. В логе появится новая транзакция с вызовом метода и данные будут сохранены. После этого нажмем кнопку retrieve и убедимся, что под ней появится корректное возвращаемое значение.  
   
2. Задание для самостоятельной работы – реализовать рассмотренный на лекции контракт Hello World, который получает через конструктор строку с приветствием и затем по вызову метода greet выдает ее вызывающему.
3. Модифицировать контракт так, чтобы он запоминал своего создателя и выдавал ему особое приветствие, отличающееся от обычного.
4. (\*) Реализовать и протестировать смарт-контракт токена, соответствующий стандарту ERC-20.  
   Стандарт ERC-20 предназначен для реализации взаимозаменяемых (fungible) токенов. Взаимозаменяемость подразумевает то, что токены могут объединяться в суммы, выделяться из них и, в целом, два экземпляра однотипного токена являются абсолютно эквивалентными (пример – деньги, рубль в одной купюре/монете и рубль в другой купюре/монете ничем не отличаются).

Ключевые идеи:

* Начальное количество токенов задается при конструировании и кладется на баланс создателя.
* Балансы аккаунтов хранятся в виде mapping (address => uint256);
* Перевод с баланса аккаунта разрешается только его владельцу с учетом текущего баланса
* Механизм allowance работает как разрешение владельца токенов какому-то другому аккаунту самостоятельно списать определенное количество токенов. Для хранения разрешенных объемов списания используется конструкция mapping(address=>mapping(address=>uint256)).
* Контракт должен соответствовать интерфейсу ERC20Interface

contract ERC20Interface {

function totalSupply() public view returns (uint);

function balanceOf(address tokenOwner) public view returns (uint balance);

function allowance(address tokenOwner, address spender) public view returns (uint remaining);

function transfer(address to, uint tokens) public returns (bool success);

function approve(address spender, uint tokens) public returns (bool success);

function transferFrom(address from, address to, uint tokens) public returns (bool success);

event Transfer(address indexed from, address indexed to, uint tokens);

event Approval(address indexed tokenOwner, address indexed spender, uint tokens);

}

Подсматривать реализации в интернете не рекомендуется, попробуйте сперва реализовать самостоятельно, а после сравните получившийся код со стандартными реализациями.

1. (\*) Подключить Remix к поднятому на прошлом занятии узлу geth, используя Web3 provider и установить смарт-контракт в тестовой сети Goerli. Попробовать обменяться токенами с одногруппниками в этой сети.