Nama : Raihan Puspa Anggita Putri
NIM : 1103184065
Kelas : TK-42-PIL

Lesson 11 : Hardhat Starter Kit

```
pragma solidity ^0.8.7;

import "@chainlink/contracts/src/v0.8/ChainlinkClient.sol";

/**
 * @title The APIConsumer contract
 * @notice An API Consumer contract that makes GET requests to obtain 24h trading volume of ETH in USD
 */
contract APIConsumer is ChainlinkClient {
  using Chainlink for Chainlink.Request;

  uint256 public volume;
  address private immutable oracle;
  bytes32 private immutable jobId;
  uint256 private immutable fee;

  event DataFullfilled(uint256 volume);

  /**
   * @notice Executes once when a contract is created to initialize state variables
   *
   * @param _oracle - address of the specific Chainlink node that a contract makes an API call from
   * @param _jobId - specific job for :_oracle: to run; each job is unique and returns different types of data
   * @param _fee - node operator price per API call / data request
   * @param _link - LINK token address on the corresponding network
   *
   * Network: Rinkeby
   * Oracle: 0xc57b33452b4f7bb189bb5afae9cc4aba1f7a4fd8
   * Job ID: 6b88e0402e5d415eb946e528b8e0c7ba
   * Fee: 0.1 LINK
   */
  constructor(
    address _oracle,
    bytes32 _jobId,
    uint256 _fee,
    address _link
  ) {
    if (_link == address(0)) {
```

```
import "@chainlink/contracts/src/v0.8/interfaces/KeeperCompatibleInterface.sol";

/**
 * @title The Counter contract
 * @notice  A keeper-compatible contract that increments counter variable at fixed time intervals
 */
contract KeepersCounter is KeeperCompatibleInterface {
  /**
   * Public counter variable
   */
  uint256 public counter;

  /**
   * Use an interval in seconds and a timestamp to slow execution of Upkeep
   */
  uint256 public immutable interval;
  uint256 public lastTimeStamp;

  /**
   * @notice Executes once when a contract is created to initialize state variables
   *
   * @param updateInterval - Period of time between two counter increments expressed as UNIX timestamp value
   */
  constructor(uint256 updateInterval) {
    interval = updateInterval;
    lastTimeStamp = block.timestamp;

    counter = 0;
  }

  /**
   * @notice Checks if the contract requires work to be done
   */
  function checkUpkeep(
    bytes memory /* checkData */
  )
```

```sol
import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

/**
 * @title The PriceConsumerV3 contract
 * @notice Acontract that returns latest price from Chainlink Price Feeds
 */
contract PriceConsumerV3 {
  AggregatorV3Interface internal immutable priceFeed;

  /**
   * @notice Executes once when a contract is created to initialize state variables
   *
   * @param _priceFeed - Price Feed Address
   *
   * Network: Rinkeby
   * Aggregator: ETH/USD
   * Address: 0x8A753747A1Fa494EC906cE90E9f37563A8AF630e
   */
  constructor(address _priceFeed) {
    priceFeed = AggregatorV3Interface(_priceFeed);
  }

  /**
   * @notice Returns the latest price
   *
   * @return latest price
   */
  function getLatestPrice() public view returns (int256) {
    (
      uint80 roundID,
      int256 price,
      uint256 startedAt,
      uint256 timeStamp,
      uint80 answeredInRound
    ) = priceFeed.latestRoundData();
    return price;
```

```
import "@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol";
import "@chainlink/contracts/src/v0.8/VRFConsumerBaseV2.sol";

/**
 * @title The RandomNumberConsumerV2 contract
 * @notice A contract that gets random values from Chainlink VRF V2
 */
contract RandomNumberConsumerV2 is VRFConsumerBaseV2 {
  VRFCoordinatorV2Interface immutable COORDINATOR;

  // Your subscription ID.
  uint64 immutable s_subscriptionId;

  // The gas lane to use, which specifies the maximum gas price to bump to.
  // For a list of available gas lanes on each network,
  // see https://docs.chain.link/docs/vrf-contracts/#configurations
  bytes32 immutable s_keyHash;

  // Depends on the number of requested values that you want sent to the
  // fulfillRandomWords() function. Storing each word costs about 20,000 gas,
  // so 100,000 is a safe default for this example contract. Test and adjust
  // this limit based on the network that you select, the size of the request,
  // and the processing of the callback request in the fulfillRandomWords()
  // function.
  uint32 constant CALLBACK_GAS_LIMIT = 100000;

  // The default is 3, but you can set this higher.
  uint16 constant REQUEST_CONFIRMATIONS = 3;

  // For this example, retrieve 2 random values in one request.
  // Cannot exceed VRFCoordinatorV2.MAX_NUM_WORDS.
  uint32 constant NUM_WORDS = 2;

  uint256[] public s_randomWords;
  uint256 public s_requestId;
```

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.4.24;

import "@chainlink/token/contracts/v0.4/LinkToken.sol";
```

```sol
import "@chainlink/contracts/src/v0.6/LinkTokenReceiver.sol";
import "@chainlink/contracts/src/v0.6/interfaces/ChainlinkRequestInterface.sol";
import "@chainlink/contracts/src/v0.6/interfaces/LinkTokenInterface.sol";
import "@chainlink/contracts/src/v0.6/vendor/SafeMathChainlink.sol";

/**
 * @title The Chainlink Mock Oracle contract
 * @notice Chainlink smart contract developers can use this to test their contracts
 */
contract MockOracle is ChainlinkRequestInterface, LinkTokenReceiver {
  using SafeMathChainlink for uint256;

  uint256 public constant EXPIRY_TIME = 5 minutes;
  uint256 private constant MINIMUM_CONSUMER_GAS_LIMIT = 400000;

  struct Request {
    address callbackAddr;
    bytes4 callbackFunctionId;
  }

  LinkTokenInterface internal LinkToken;
  mapping(bytes32 => Request) private commitments;

  event OracleRequest(
    bytes32 indexed specId,
    address requester,
    bytes32 requestId,
    uint256 payment,
    address callbackAddr,
    bytes4 callbackFunctionId,
    uint256 cancelExpiration,
    uint256 dataVersion,
    bytes data
  );

  event CancelOracleRequest(bytes32 indexed requestId);
```

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@chainlink/contracts/src/v0.6/tests/MockV3Aggregator.sol";
```

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

import "@chainlink/contracts/src/v0.8/mocks/VRFCoordinatorV2Mock.sol";
```

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "../../KeepersCounter.sol";

contract KeepersCounterEchidnaTest is KeepersCounter {
  constructor() KeepersCounter(8 days) {}

  function echidna_test_perform_upkeep_gate() public view returns (bool) {
    return counter == 0;
  }
}
```