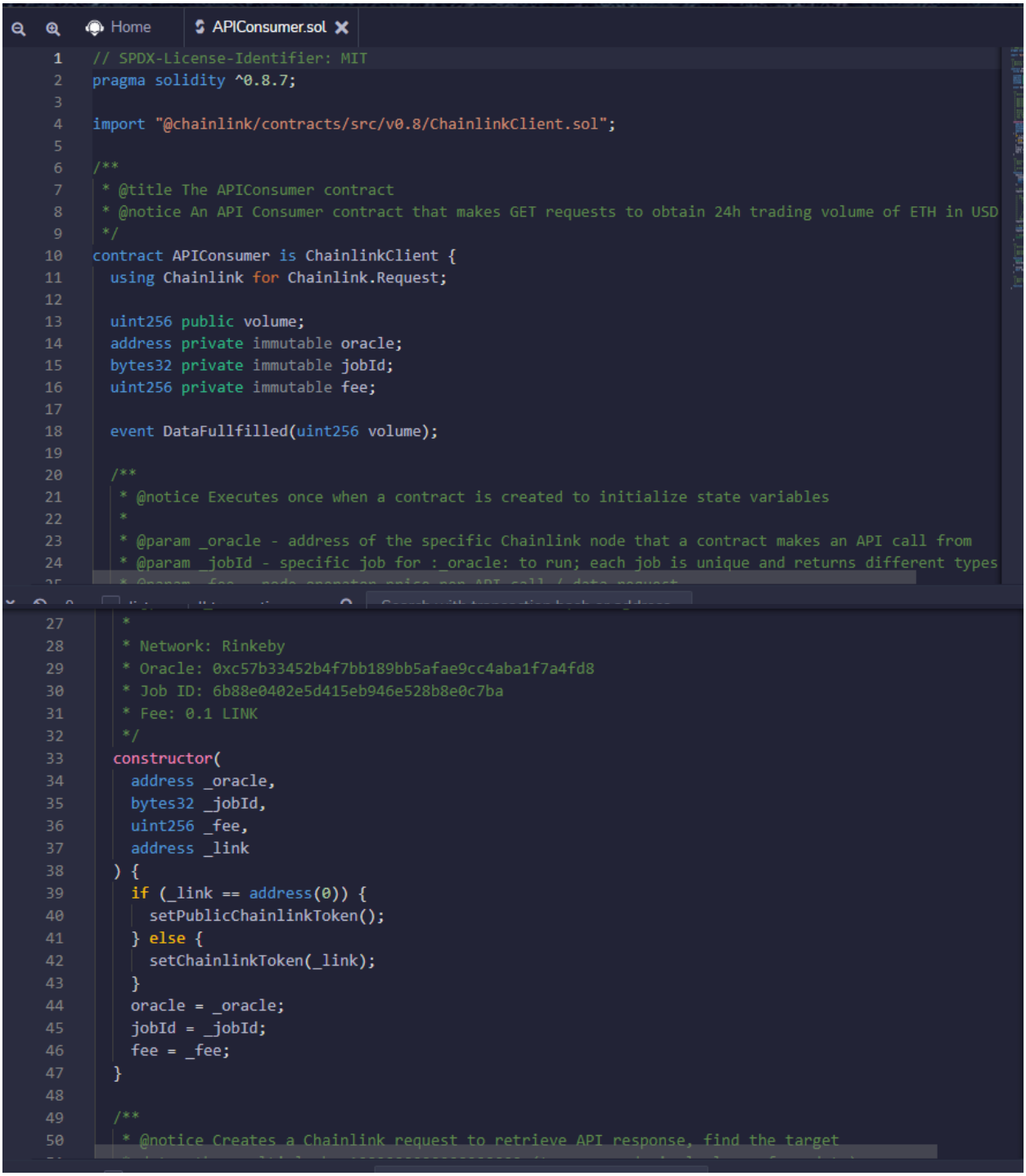
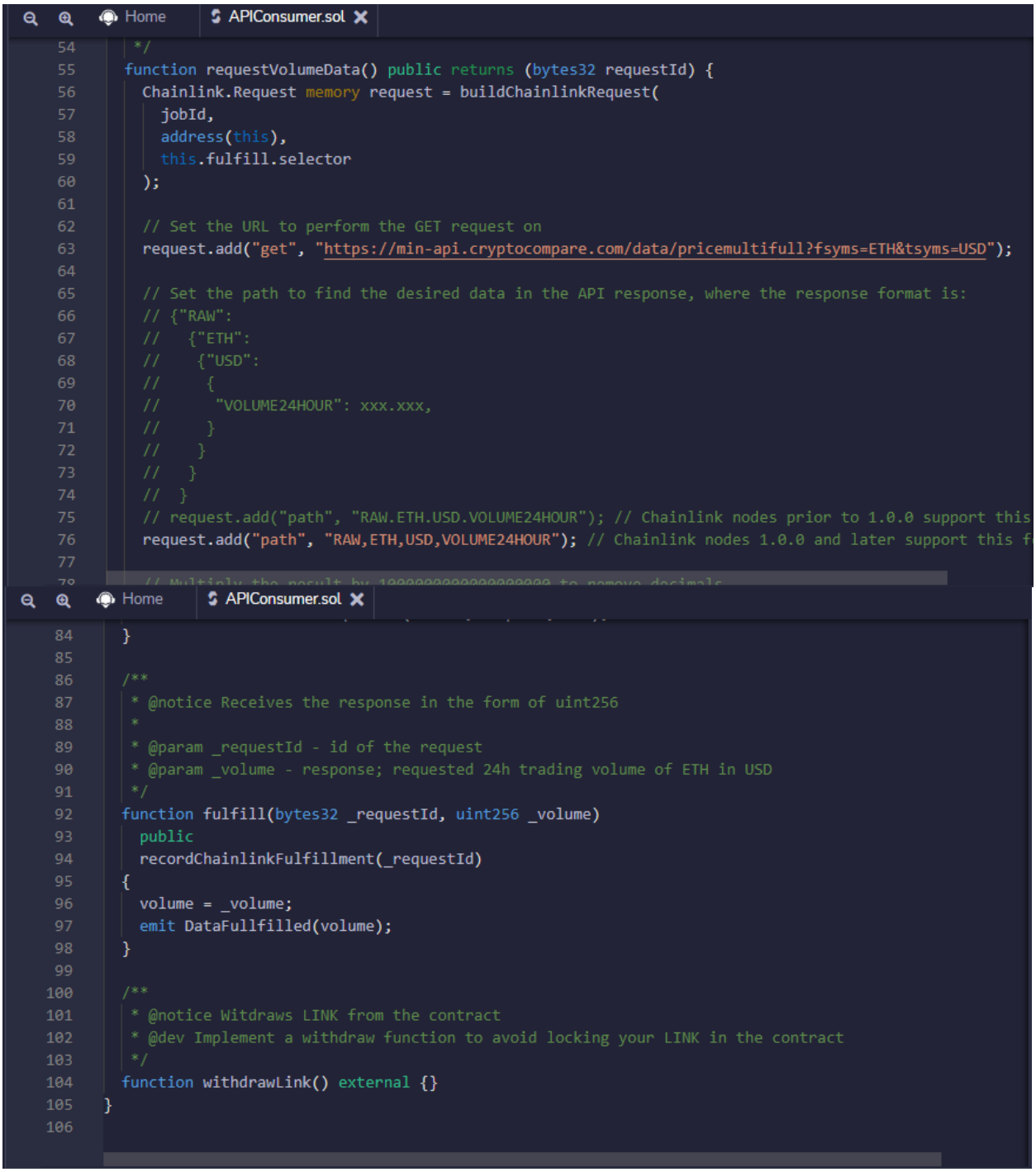
Hardhat Starter Kit



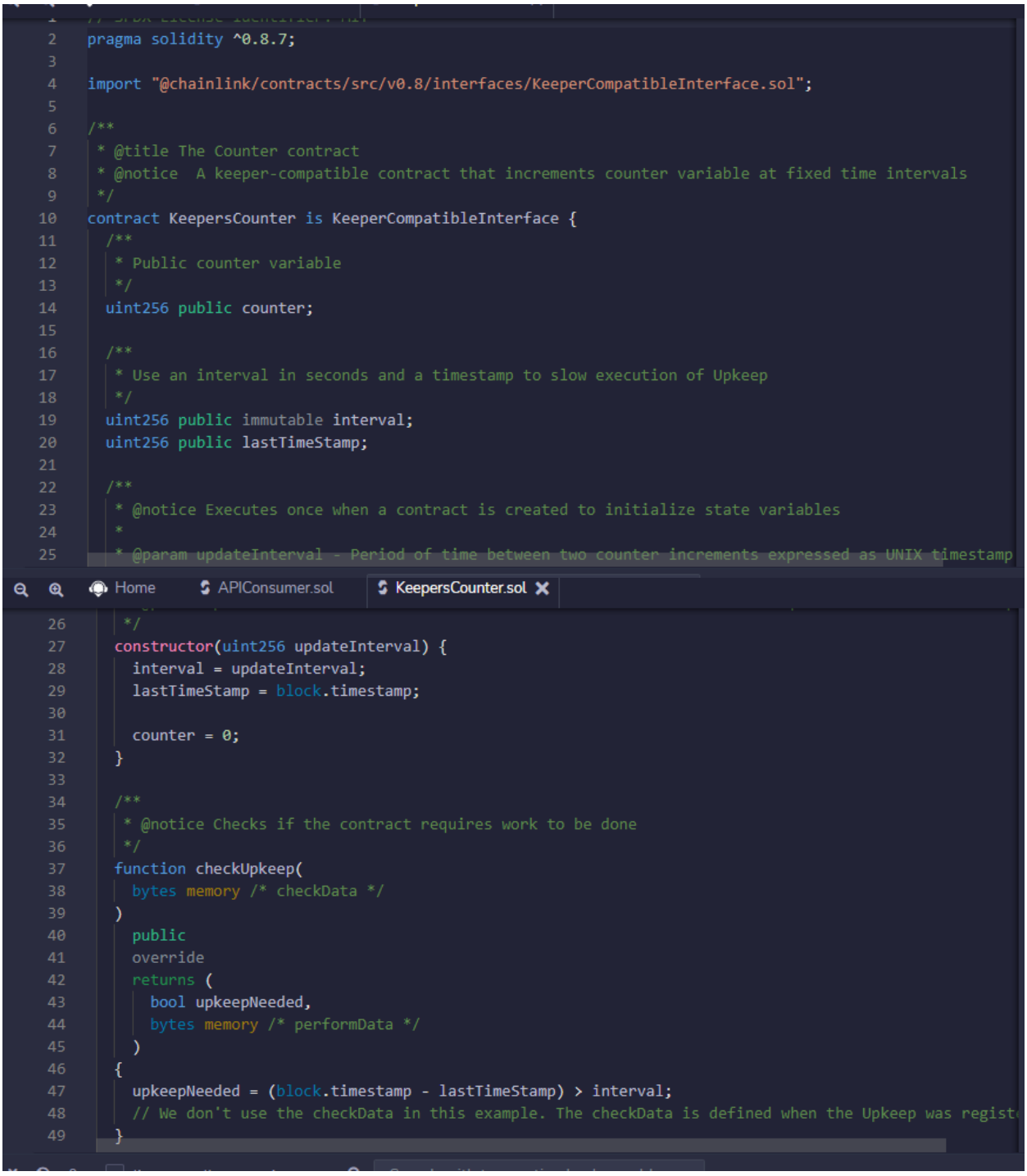


Home APIConsumer.sol

```
54    */
55   function requestVolumeData() public returns (bytes32 requestId) {
56     Chainlink.Request memory request = buildChainlinkRequest(
57       jobId,
58       address(this),
59       this.fulfill.selector
60     );
61
62     // Set the URL to perform the GET request on
63     request.add("get", "https://min-api.cryptocompare.com/data/pricemultifull?fsyms=ETH&tsyms=USD");
64
65     // Set the path to find the desired data in the API response, where the response format is:
66     // {"RAW":
67     //   {"ETH":
68     //    {"USD":
69     //     {
70     //      "VOLUME24HOUR": xxx.xxx,
71     //     }
72     //    }
73     //   }
74     //  }
75     // request.add("path", "RAW.ETH.USD.VOLUME24HOUR"); // Chainlink nodes prior to 1.0.0 support this
76     request.add("path", "RAW,ETH,USD,VOLUME24HOUR"); // Chainlink nodes 1.0.0 and later support this f
77
78     // Multiply the result by 1000000000000000000 to remove decimals
```

Home APIConsumer.sol

```
84   }
85
86   /**
87    * @notice Receives the response in the form of uint256
88    *
89    * @param _requestId - id of the request
90    * @param _volume - response; requested 24h trading volume of ETH in USD
91    */
92   function fulfill(bytes32 _requestId, uint256 _volume)
93     public
94     recordChainlinkFulfillment(_requestId)
95   {
96     volume = _volume;
97     emit DataFullfilled(volume);
98   }
99
100  /**
101   * @notice Witdraws LINK from the contract
102   * @dev Implement a withdraw function to avoid locking your LINK in the contract
103   */
104  function withdrawLink() external {}
105 }
106
```



```
pragma solidity ^0.8.7;

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
```

Home | APIConsumer.sol | KeepersCounter.sol

```
   */
  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 */
  )
    public
    override
    returns (
      bool upkeepNeeded,
      bytes memory /* performData */
    )
  {
    upkeepNeeded = (block.timestamp - lastTimeStamp) > interval;
    // We don't use the checkData in this example. The checkData is defined when the Upkeep was regist
  }
```

Home | APIConsumer.sol | KeepersCounter.sol | KeepersCounterEchidnaTest.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;
  }
}

```