




Hardhat Fund Me

 .env.example ✕

 .env.example

```
1 PRIVATE_KEY=234523425asdfasdfa
2 RINKEBY_RPC_URL=http://0.0.0.0:8545
3 COINMARKETCAP_API_KEY=asdfasdfasdfasdfasdf
4 ETHERSCAN_API_KEY=asdfasdfasdfs
5 KOVAN_RPC_URL=http://0.0.0.0:8545
```

 package.json ✕

 package.json > ...

```
1  {
2    "name": "hardhat-project",
3    "devDependencies": {
4      "@nomiclabs/hardhat-ethers": "npm:hardhat-deploy-ethers@^0.3.0-beta.13",
5      "@nomiclabs/hardhat-etherscan": "^3.0.0",
6      "@nomiclabs/hardhat-waffle": "^2.0.2",
7      "chai": "^4.3.4",
8      "ethereum-waffle": "^3.4.0",
9      "ethers": "^5.5.3",
10     "hardhat": "^2.8.3",
11     "hardhat-deploy": "^0.9.29",
12     "hardhat-gas-reporter": "^1.0.7",
13     "solidity-coverage": "^0.7.18",
14     "@chainlink/contracts": "^0.3.1",
15     "dotenv": "^14.2.0",
16     "prettier-plugin-solidity": "^1.0.0-beta.19"
17   },
18   "scripts": {
19     "test": "hardhat test",
20     "test:staging": "hardhat test --network rinkeby",
21     "lint": "solhint 'contracts/*.sol'",
22     "lint:fix": "solhint 'contracts/**/*.sol' --fix",
23     "format": "prettier --write .",
24     "coverage": "hardhat coverage"
25   }
26 }
27
```

```

1 // SPDX-License-Identifier: MIT
2 // 1. Pragma
3 pragma solidity ^0.8.0;
4 // 2. Imports
5 import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";
6 import "../PriceConverter.sol";
7
8 // 3. Interfaces, Libraries, Contracts
9 error FundMe__NotOwner();
10
11 /**@title A sample Funding Contract
12  * @author Patrick Collins
13  * @notice This contract is for creating a sample funding contract
14  * @dev This implements price feeds as our library
15  */
16 contract FundMe {
17     // Type Declarations
18     using PriceConverter for uint256;
19
20     // State variables
21     uint256 public constant MINIMUM_USD = 50 * 10**18;
22     address private immutable i_owner;
23     address[] private s_funders;
24     mapping(address => uint256) private s_addressToAmountFunded;
25     AggregatorV3Interface private s_priceFeed;
26
27     // Events (we have none!)
28
29     // Modifiers
30     modifier onlyOwner() {
31         // require(msg.sender == i_owner);
32         if (msg.sender != i_owner) revert FundMe__NotOwner();
33         _;
34     }
35
36     // Functions Order:
37     //// constructor

```

```

86         address funder = funders[funderIndex];
87         s_addressToAmountFunded[funder] = 0;
88     }
89     s_funders = new address[](0);
90     // payable(msg.sender).transfer(address(this).balance);
91     (bool success, ) = i_owner.call{value: address(this).balance}("");
92     require(success);
93 }
94
95 /** @notice Gets the amount that an address has funded
96  * @param fundingAddress the address of the funder
97  * @return the amount funded
98  */
99 function getAddressToAmountFunded(address fundingAddress)
100     public
101     view
102     returns (uint256)
103 {
104     return s_addressToAmountFunded[fundingAddress];
105 }
106
107 function getVersion() public view returns (uint256) {
108     return s_priceFeed.version();
109 }
110
111 function getFunder(uint256 index) public view returns (address) {
112     return s_funders[index];
113 }
114
115 function getOwner() public view returns (address) {
116     return i_owner;
117 }
118
119 function getPriceFeed() public view returns (AggregatorV3Interface) {
120     return s_priceFeed;
121 }
122 }

```

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";
5
6 library PriceConverter {
7     function getPrice(AggregatorV3Interface priceFeed)
8         internal
9         view
10        returns (uint256)
11    {
12        (, int256 answer, , , ) = priceFeed.latestRoundData();
13        // ETH/USD rate in 18 digit
14        return uint256(answer * 1000000000);
15    }
16
17    // 1000000000
18    // call it get fiatConversionRate, since it assumes something about decimals
19    // It wouldn't work for every aggregator
20    function getConversionRate(uint256 ethAmount, AggregatorV3Interface priceFeed)
21        internal
22        view
23        returns (uint256)
24    {
25        uint256 ethPrice = getPrice(priceFeed);
26        uint256 ethAmountInUsd = (ethPrice * ethAmount) / 1000000000000000000;
27        // the actual ETH/USD conversation rate, after adjusting the extra 0s.
28        return ethAmountInUsd;
29    }
30 }
31

```

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract FunWithStorage {
5     uint256 favoriteNumber; // Stored at slot 0
6     bool someBool; // Stored at slot 1
7     uint256[] myArray; /* Array Length Stored at slot 2,
8     but the objects will be the keccak256(2), since 2 is the storage slot of the array */
9     mapping(uint256 => bool) myMap; /* An empty slot is held at slot 3
10    and the elements will be stored at keccak256(h(k) . p)
11
12    p: The storage slot (aka, 3)
13    k: The key in hex
14    h: Some function based on the type. For uint256, it just pads the hex
15    */
16    uint256 constant NOT_IN_STORAGE = 123;
17    uint256 immutable i_not_in_storage;
18
19    constructor() {
20        favoriteNumber = 25; // See stored spot above // SSTORE
21        someBool = true; // See stored spot above // SSTORE
22        myArray.push(222); // SSTORE
23        myMap[0] = true; // SSTORE
24        i_not_in_storage = 123;
25    }
26
27    function doStuff() public {
28        uint256 newVar = favoriteNumber + 1; // SLOAD
29        bool otherVar = someBool; // SLOAD
30        // ^^ memory variables
31    }
32 }
33

```