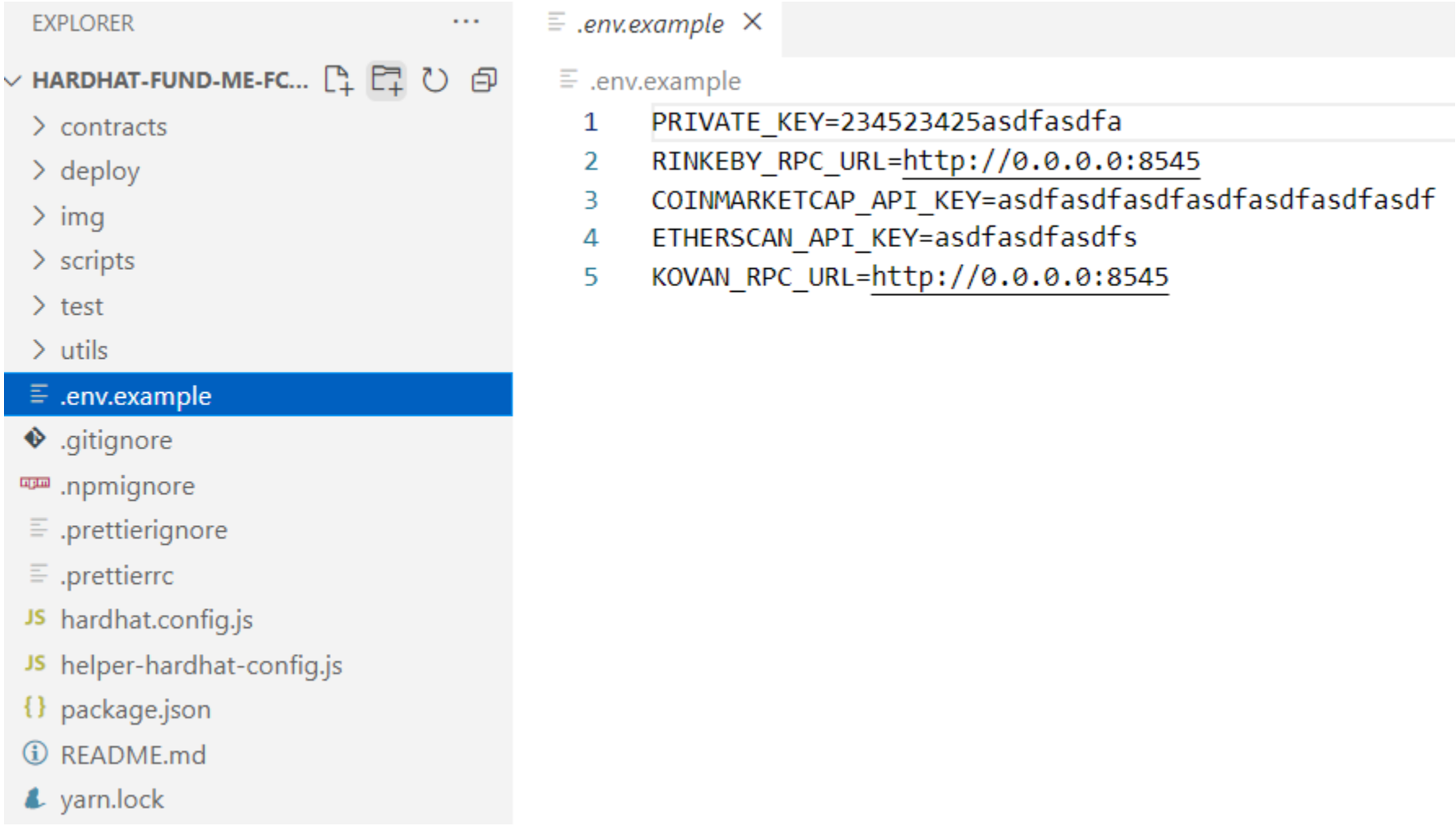
Hardhat Fund me



package.json

package.json > ...

```
{
  "name": "hardhat-project",
  "devDependencies": {
    "@nomiclabs/hardhat-ethers": "npm:hardhat-deploy-ethers@^0.3.0-beta.13",
    "@nomiclabs/hardhat-etherscan": "^3.0.0",
    "@nomiclabs/hardhat-waffle": "^2.0.2",
    "chai": "^4.3.4",
    "ethereum-waffle": "^3.4.0",
    "ethers": "^5.5.3",
    "hardhat": "^2.8.3",
    "hardhat-deploy": "^0.9.29",
    "hardhat-gas-reporter": "^1.0.7",
    "solidity-coverage": "^0.7.18",
    "@chainlink/contracts": "^0.3.1",
    "dotenv": "^14.2.0",
    "prettier-plugin-solidity": "^1.0.0-beta.19"
  },
  ▷ Debug
  "scripts": {
    "test": "hardhat test",
    "test:staging": "hardhat test --network rinkeby",
    "lint": "solhint 'contracts/*.sol'",
    "lint:fix": "solhint 'contracts/**/*.sol' --fix",
    "format": "prettier --write .",
    "coverage": "hardhat coverage"
  }
}
```

```solidity
// SPDX-License-Identifier: MIT
// 1. Pragma
pragma solidity ^0.8.0;
// 2. Imports
import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";
import "./PriceConverter.sol";

// 3. Interfaces, Libraries, Contracts
error FundMe__NotOwner();

/**@title A sample Funding Contract
 * @author Patrick Collins
 * @notice This contract is for creating a sample funding contract
 * @dev This implements price feeds as our library
 */
contract FundMe {
    // Type Declarations
    using PriceConverter for uint256;

    // State variables
    uint256 public constant MINIMUM_USD = 50 * 10**18;
    address private immutable i_owner;
    address[] private s_funders;
    mapping(address => uint256) private s_addressToAmountFunded;
    AggregatorV3Interface private s_priceFeed;

    // Events (we have none!)

    // Modifiers
    modifier onlyOwner() {
        // require(msg.sender == i_owner);
        if (msg.sender != i_owner) revert FundMe__NotOwner();
        _;
    }

    // Functions Order:
    //// 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 }
```

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

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

library PriceConverter {
  function getPrice(AggregatorV3Interface priceFeed)
    internal
    view
    returns (uint256)
  {
    (, int256 answer, , , ) = priceFeed.latestRoundData();
    // ETH/USD rate in 18 digit
    return uint256(answer * 10000000000);
  }

  // 1000000000
  // call it get fiatConversionRate, since it assumes something about decimals
  // It wouldn't work for every aggregator
  function getConversionRate(uint256 ethAmount, AggregatorV3Interface priceFeed)
    internal
    view
    returns (uint256)
  {
    uint256 ethPrice = getPrice(priceFeed);
    uint256 ethAmountInUsd = (ethPrice * ethAmount) / 1000000000000000000;
    // the actual ETH/USD conversation rate, after adjusting the extra 0s.
    return ethAmountInUsd;
  }
}
```

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

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

```

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

contract FunWithStorage {
    uint256 favoriteNumber; // Stored at slot 0
    bool someBool; // Stored at slot 1
    uint256[] myArray; /* Array Length Stored at slot 2,
    but the objects will be the keccak256(2), since 2 is the storage slot of the array */
    mapping(uint256 => bool) myMap; /* An empty slot is held at slot 3
    and the elements will be stored at keccak256(h(k) . p)

    p: The storage slot (aka, 3)
    k: The key in hex
    h: Some function based on the type. For uint256, it just pads the hex
    */
    uint256 constant NOT_IN_STORAGE = 123;
    uint256 immutable i_not_in_storage;

    constructor() {
        favoriteNumber = 25; // See stored spot above // SSTORE
        someBool = true; // See stored spot above // SSTORE
        myArray.push(222); // SSTORE
        myMap[0] = true; // SSTORE
        i_not_in_storage = 123;
    }

    function doStuff() public {
        uint256 newVar = favoriteNumber + 1; // SLOAD
        bool otherVar = someBool; // SLOAD
        // ^^ memory variables
    }
}
```