Course 4: FundMe Contract

1. FundMe.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

error NotOwner();

contract FundMe {


    uint256 public constant MINIMUM_USD = 50 * 10 ** 18;

    function fund() public payable {
        require(msg.value.getConversionRate() >= MINIMUM_USD, "You need to
spend more ETH!");
```

Pada tahap pertama ini kita akan menambahkan contract di atas ini. Contract ini berfungsi mengirim fund/dana ke suatu akun, MINIMUM_USD adalah jumlah minimum yang harus di kirim oleh contract owner.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract FundMe {

    address public /* immutable */ i_owner;
    uint256 public constant MINIMUM_USD = 50 * 10 ** 18;

    function fund() public payable {
        require(msg.value.getConversionRate() >= MINIMUM_USD, "You need to
spend more ETH!");

    constructor() {
        i_owner = msg.sender;
    }
```

Tambahan address i_owner dan constructor untuk i_owner = msg.sender untuk mendata address yang menggunakan contract tersebut.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

    mapping(address => uint256) public addressToAmountFunded;
    address[] public funders;

contract FundMe {

    address public /* immutable */ i_owner;
```

```
    uint256 public constant MINIMUM_USD = 50 * 10 ** 18;

    function fund() public payable {
        require(msg.value.getConversionRate() >= MINIMUM_USD, "You need to
spend more ETH!");
        addressToAmountFunded[msg.sender] += msg.value;
        funders.push(msg.sender);
    }
    constructor() {
        i_owner = msg.sender;
    }
    modifier onlyOwner {
        // require(msg.sender == owner);
        if (msg.sender != i_owner) revert NotOwner();
        _;
    }

    function withdraw() payable onlyOwner public {
        for (uint256 funderIndex=0; funderIndex < funders.length;
funderIndex++){
            address funder = funders[funderIndex];
            addressToAmountFunded[funder] = 0;
        }
        funders = new address[](0);
        (bool callSuccess, ) = payable(msg.sender).call{value:
address(this).balance}("");
        require(callSuccess, "Call failed");
    }
```

Kita akan menambahkan beberapa fungsi, pertama ada yang diluar kontrak adalah mapping untuk menghitung berapa banyaknya akun yang harus didanakan, dan membuat array address funders yang akan digunakan dalam contract. Dalam function fund kita akan menambahkan jumlah akun yang haurs didanakan setiap kali function fund dipanggil dan address akan dimasukkan ke dalam array funders.

Kemudian kita membuat function baru yaitu function withdraw. Fungsi ini berfungsi untuk menarik semua dana yang telah diisi dalam fungsi fund. Kita ingin hanya contract owner yang dapa menggunakan function ini, cara yang akan kita pakai untuk itu adalah membuat modifier onlyOwner yang akan mengecek apabila yang menggunakan fungsi adalah owner, apabila bukan owner, maka akan di tolak.

2. PriceConverter.sol

Kita membutuhkan cara untuk membaca nilai mata uang usd tanpa merusak desentralized system blockchain, salah satu solusinya adalah menggunakan AggregatorV3Interface sebagai library yang akan mengeluarkan output harga mata uang usd dalam bentuk wei. Source codenya adalah sebagai berikut:

```
pragma solidity ^0.8.8;

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

// Why is this a library and not abstract?
// Why not an interface?
library PriceConverter {
    // We could make this public, but then we'd have to deploy it
    function getPrice() internal view returns (uint256) {
        // Rinkeby ETH / USD Address
        // https://docs.chain.link/docs/ethereum-addresses/
        AggregatorV3Interface priceFeed = AggregatorV3Interface(
            0x8A753747A1Fa494EC906cE90E9f37563A8AF630e
        );
        (, int256 answer, , , ) = priceFeed.latestRoundData();
        // ETH/USD rate in 18 digit
        return uint256(answer * 10000000000);
    }

    // 1000000000
    function getConversionRate(uint256 ethAmount)
        internal
        view
        returns (uint256)
    {
        uint256 ethPrice = getPrice();
        uint256 ethAmountInUsd = (ethPrice * ethAmount) / 1000000000000000000;
        // the actual ETH/USD conversion rate, after adjusting the extra 0s.
        return ethAmountInUsd;
    }
}
```

Kita akan mengimport library AggregatorV3Interface dari github, kemudian kita menambahkan fungsi getPrice yang berfungsi untuk menarik harga uang mata usd pada pasar, kemudian dengan fungsi getConversionRate kita akan mengkonversi nilai mata uang usd ke nilai mata uang crypto wei.

3. Import PriceConverter.sol ke FundMe.sol

Sekarang kita akan mengimport library yang telah kita buat kedalam kontrak kita dengan menambahkan dua baris kode ini:

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

dan menambahkan satu baris kode ini di dalam kontrak FundMe pada baris pertama:

```
using PriceConverter for uint256;
```

serta menambahkan fungsi getVersion untuk AggregatorV3Interface di dalam kontrak FundMe:

```
function getVersion() public view returns (uint256){
        AggregatorV3Interface priceFeed =
AggregatorV3Interface(0x8A753747A1Fa494EC906cE90E9f37563A8AF630e);
        return priceFeed.version();
    }
```

Terakhir kita akan menambahkan fallback dan receive untuk penyelesai kontrak di paling bawah kontrak:

```
fallback() external payable {
        fund();
    }
receive() external payable {
        fund();
    }
```