Upgradable contract:-

By default, smart contracts deployed on the Ethereum blockchain are immutable. While this helps achieve decentralization and security, it can reduce the functionality of a smart contract. Solving this problem requires using upgradeable smart contract patterns during development. Upgradeable smart contracts, created using proxy patterns, enable developers to modify contract functionality after deployment—without harming security or decentralization.

Please refer this site for more information
https://www.quicknode.com/guides/ethereum-development/smart-contracts/how-to-create-and-deploy-an-upgradeable-smart-contract-using-openzeppelin-and-hardhat

1. Create a directory name : mkdir UpgradeableContracts
2. Go to inside Upgradeable Contracts directory use. cd Upgradeable Contracts
3.Open this folder on VS code and go to vs code terminal and run this command
npm init -y
4.after that run this command **npm install --save-dev hardhat** for download hardhat
And run this command for initialize the hardhat npx hardhat init and press enter enter and enter and you can see you have installed one package name is **npm install --save-dev @nomicfoundation/hardhat-toolbox**

5.Go into the contracts folder, and delete the pre-existing Lock.sol file. That is a default smart contract template provided by Hardhat and we don't need it. Now create a new file in the contracts folder, named contractV1.sol, and paste the following code in the file:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract V1 {
  uint public number;

  function initialValue(uint _num) external {
     number=_num;
  }

  function increase() external {
     number += 1;
  }
}
```

Create another file in the contracts folder, and name it contractV2.sol. Paste the following code into the file:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract V2 {
  uint public number;

  function initialValue(uint _num) external {
      number=_num;
  }

  function increase() external {
      number += 1;
  }

  function decrease() external {
      number -= 1;
  }
}
```

And now you can also installed some dependencies like

```
npm  i @nomicfoundation/hardhat-ethers @nomiclabs/hardhat-etherscan
@openzeppelin/hardhat-upgrades ethers
```

After that go to hardhat.config.js file paste this code for setup

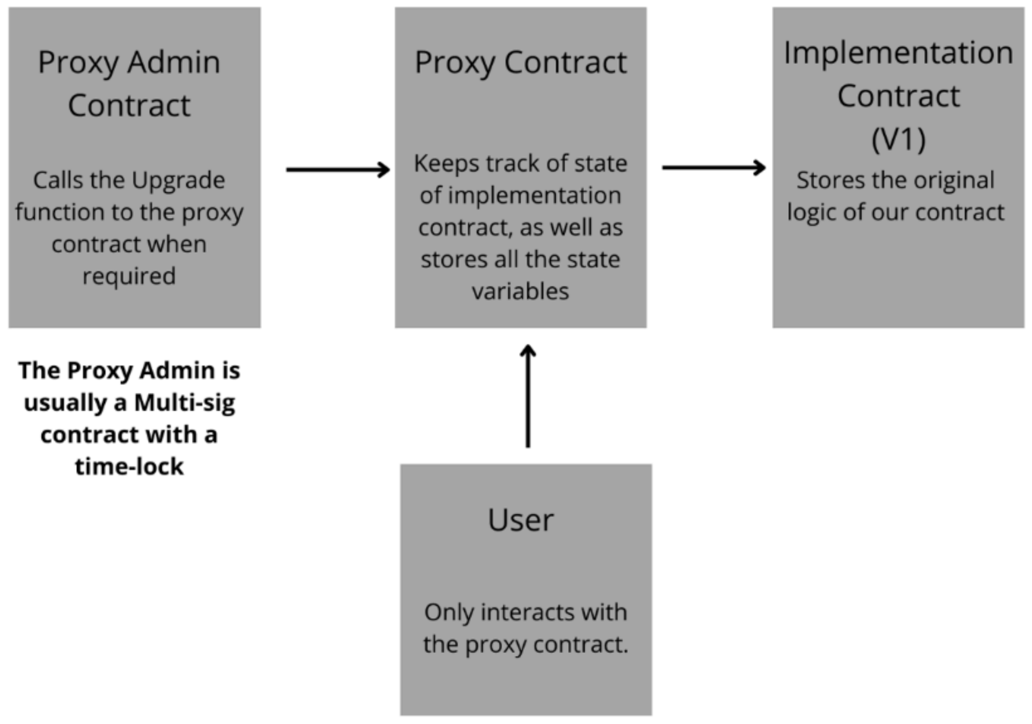
```
require('@nomicfoundation/hardhat-toolbox')
require('@openzeppelin/hardhat-upgrades');
// require("@nomiclabs/hardhat-etherscan");

const ALCHEMY_API_KEY = "jsRnIZJzjJL-fRXXos5zkrXFQ_J__Kfn";
const SEPOLIA_PRIVATE_KEY =
"03236289f13f5492e559360b5fdedbe260d72acdcb9e3c96e454e2c5c82ed116";
const API_KEY = "8GZXEF3T25M43R5F5W8ZR53RE6IKQPYJAM";

module.exports = {
  solidity: "0.8.23",
  networks: {
    sepolia: {
      url: `https://eth-sepolia.g.alchemy.com/v2/${ALCHEMY_API_KEY}`,
      accounts: [SEPOLIA_PRIVATE_KEY]
    }
  },
  etherscan: {
    apiKey: API_KEY,
  },
};
```

Now go to scripts section and delete existing file and create one new file contractV1.js
And paste this code

```
const { ethers, upgrades } = require("hardhat");

async function main() {
  const V1contract = await ethers.getContractFactory("V1");
  console.log("Deploying V1contract...");

  const deploymentOptions = {
    initializer: "initialValue",
    overrides: {
      gasLimit: 12400000,
      gasPrice: 1110000000,
    },
  };

  const v1contract = await upgrades.deployProxy(V1contract, [10], deploymentOptions);

  console.log("Deploying V1contract...Deployed");

  const a = await v1contract.waitForDeployment();
  // console.log("V1 Contract:", a);

  const address = a.target;
  console.log("V1 Contract deployed to:", address);
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

After that run this command **npx hardhat run --network sepolia scripts/contractV1.js**
For deploying in network

And its time to verify the code through hardhat so run this command
npx hardhat verify <Enter Your proxy Contract> **--network sepolia**


Proxy Admin Contract
Calls the Upgrade function to the proxy contract when required
The Proxy Admin is usually a Multi-sig contract with a time-lock
Proxy Contract
Keeps track of state of implementation contract, as well as stores all the state variables
Implementation Contract (V1)
Stores the original logic of our contract
User
Only interacts with the proxy contract.

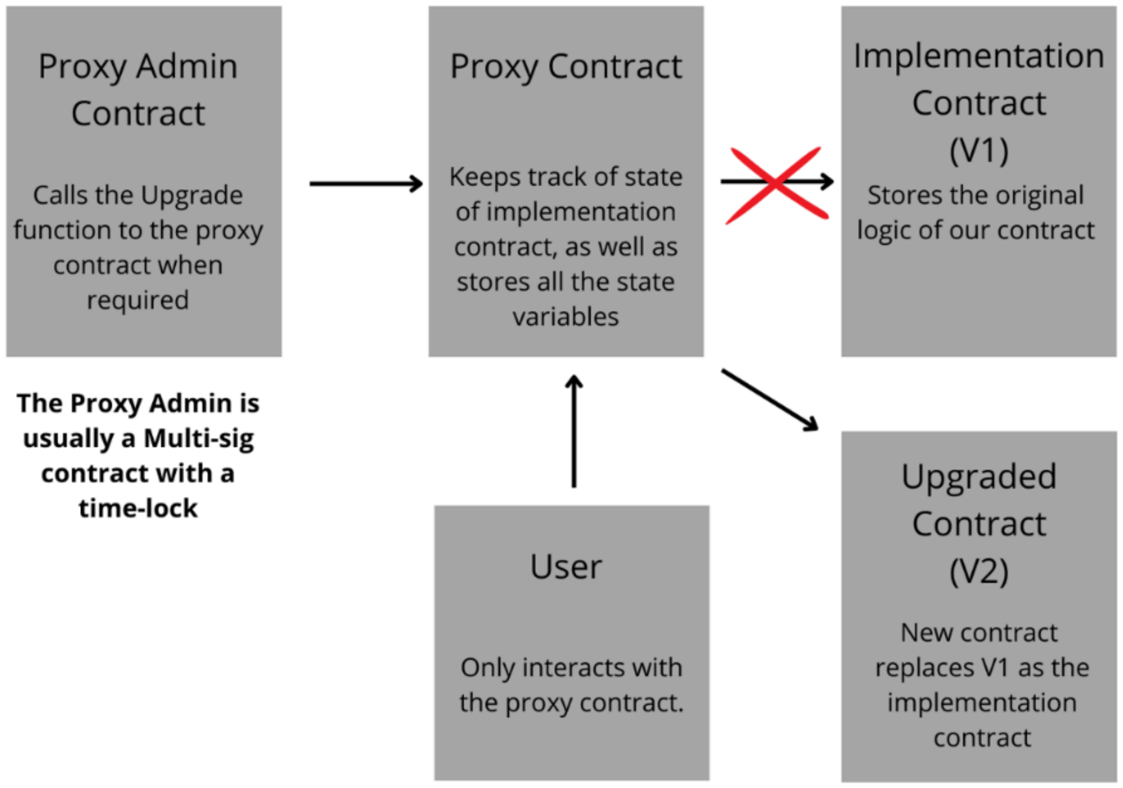
Upgradeable Smart Contracts Flowchart

Now you can create second file in scripts section for deploying upgradable contract so crearte file the file name is contractV2.js anad paste this code

```
const { ethers, upgrades } = require('hardhat')

const UPGRADEABLE_PROXY = '0x2C8D51B47bdfbd06A255E6d8726C18bF7D0f68f0'

async function main() {
   const V2Contract = await ethers.getContractFactory('V2')
   console.log('Upgrading V1Contract...')
```

```
    let upgrade = await upgrades.upgradeProxy(UPGRADEABLE_PROXY, V2Contract);

    // console.log('upgrade', upgrade);
    console.log('V1 Upgraded to V2');

    const address = upgrade.target;
    console.log('V2 Contract Deployed To:', address);
}

main().catch(error => {
    console.error(error)
    process.exitCode = 1
})
```

After that run this command
npx hardhat run --network sepolia scripts/contractV2.js
And get the address the address is same as the previous address but you again need to verify thats why you can verify the proxy address using this command
npx hardhat verify <Enter Your proxy Contract> --network sepolia


Proxy Admin Contract
Calls the Upgrade function to the proxy contract when required
The Proxy Admin is usually a Multi-sig contract with a time-lock
Proxy Contract
Keeps track of state of implementation contract, as well as stores all the state variables
Implementation Contract (V1)
Stores the original logic of our contract
User
Only interacts with the proxy contract.
Upgraded Contract (V2)
New contract replaces V1 as the implementation contract

Upgradeable Smart Contracts Flowchart