Blockchain for Content Creators and Video Streaming

Team RSA

Problem statement:

When you want to stream a video you usually go to applications using central services like Youtube, DailyMotion, Netflix, etc. We want to remove these services and replace them with a decentralised Video Streaming service that is run by basically the customers and the content creators.

Proposed Solution:

We explore a blockchain based decentralised approach. Smart con- tracts will be use to for access control as well as payments made from consumer to Content Creator. We will store files in a decen- tralised File System like IPFS. We will also store video metadata that has non-critical video data for viewing available videos. The Consumer will interact with a dApp for end-end usage. Content Creators can also upload videos using the dApp.

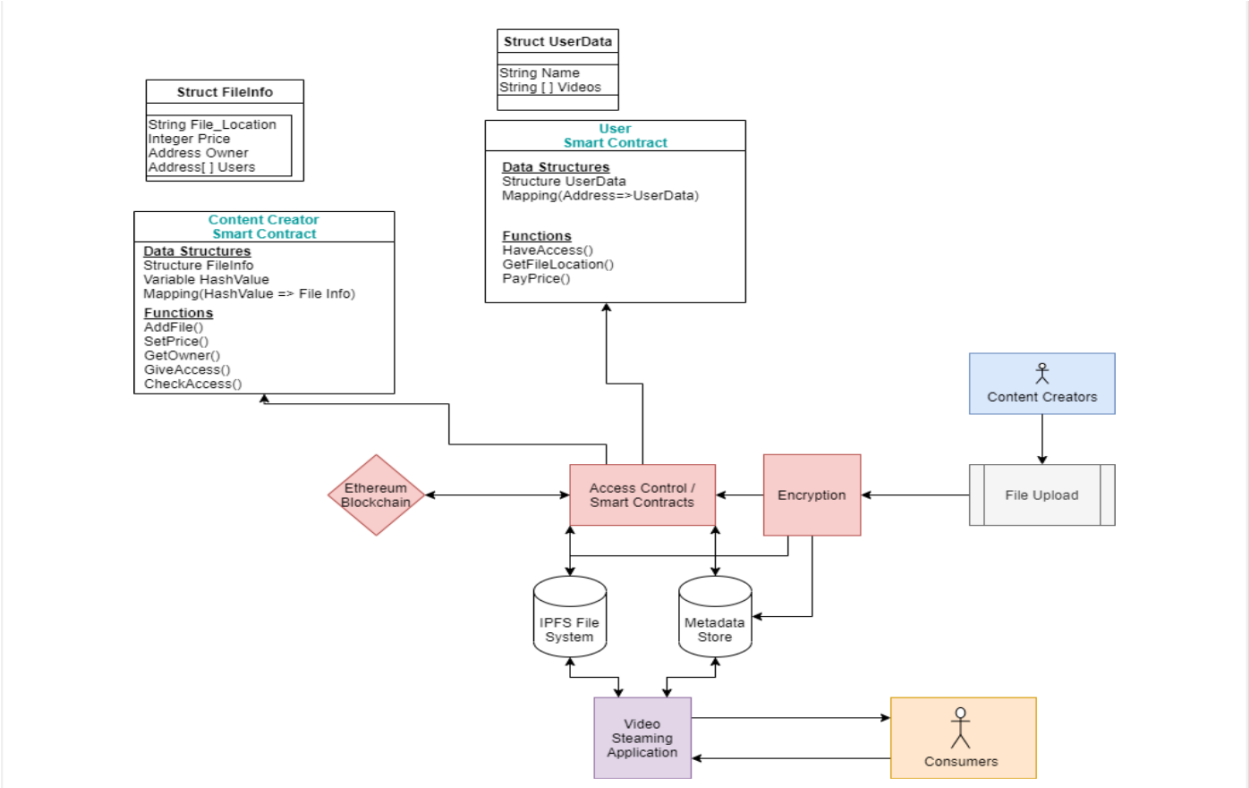
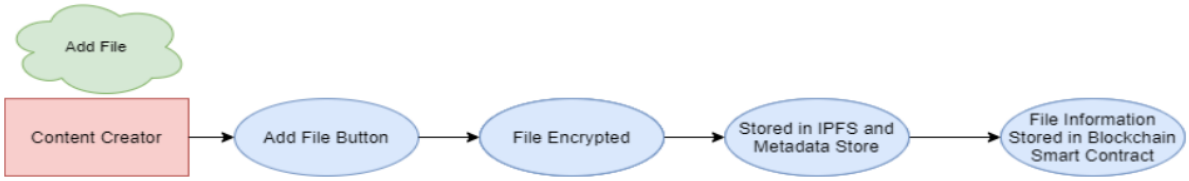
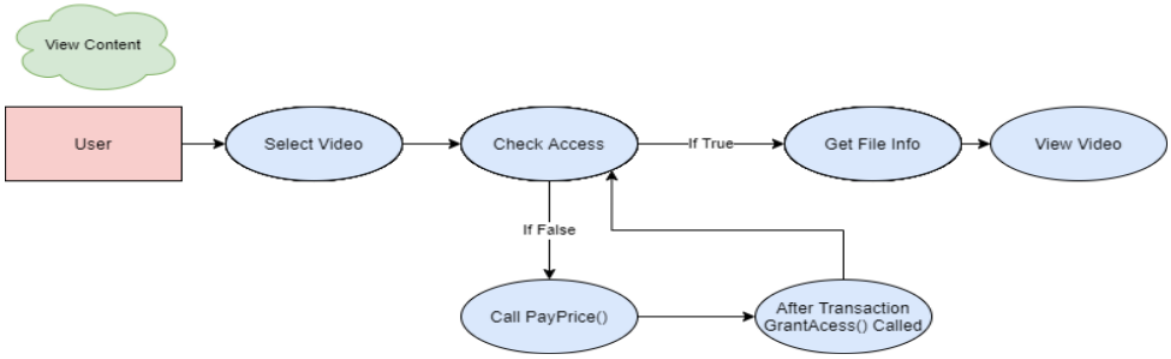


Figure 1: Proposed System Architecture





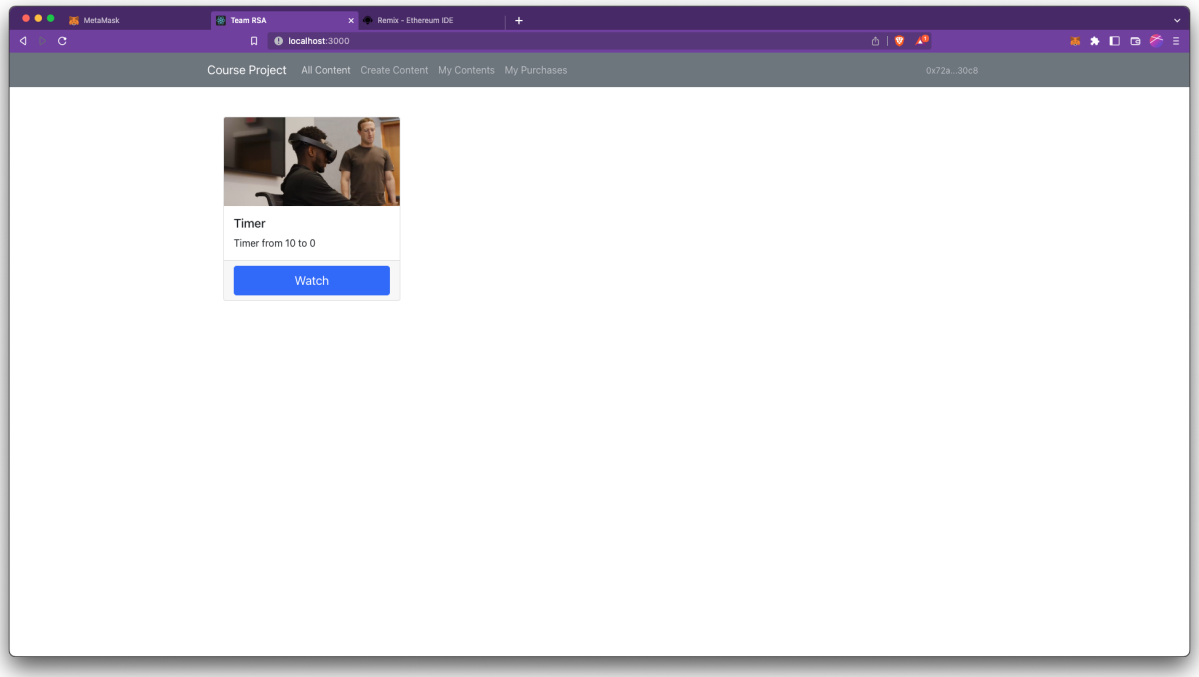
Contract:

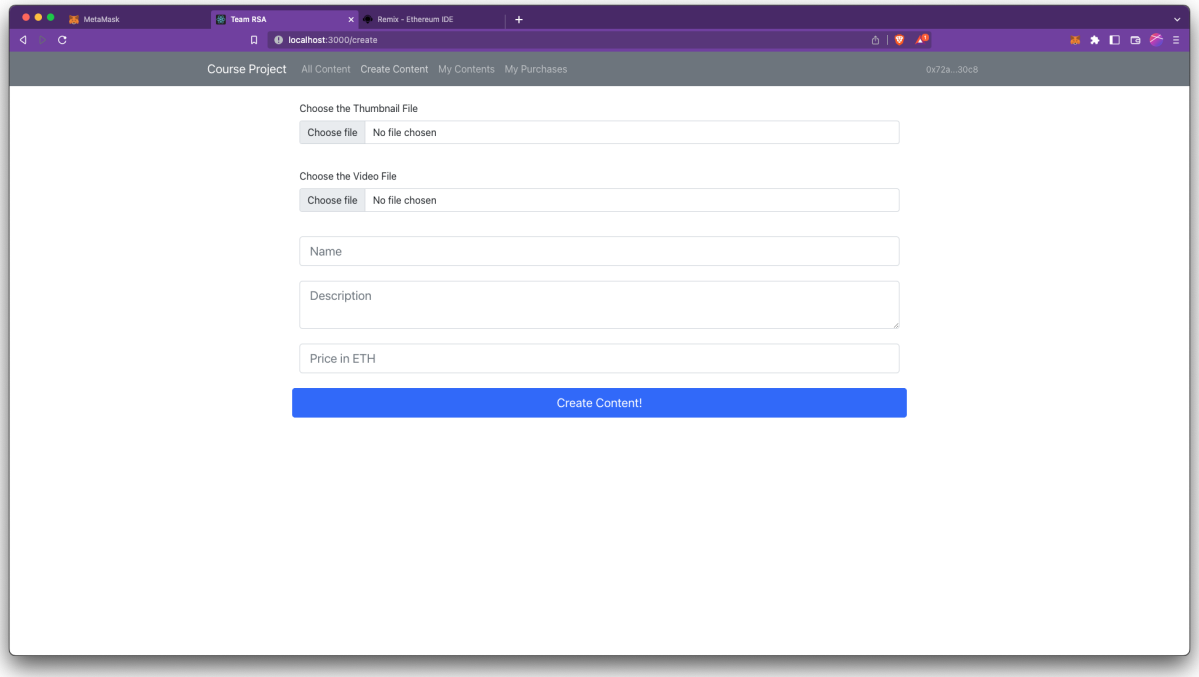
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract StreamContract {

  struct ContentInfo {
    string fileLocation;
    uint price;
    address payable owner;
    address[] usersWithAccess;
    // mapping(uint => address[]) usersWithAccess;
  }

  struct UserInfo {
    uint[] addedContentsId;
    uint[] boughtContentsId;
  }

  uint public contentId;
```

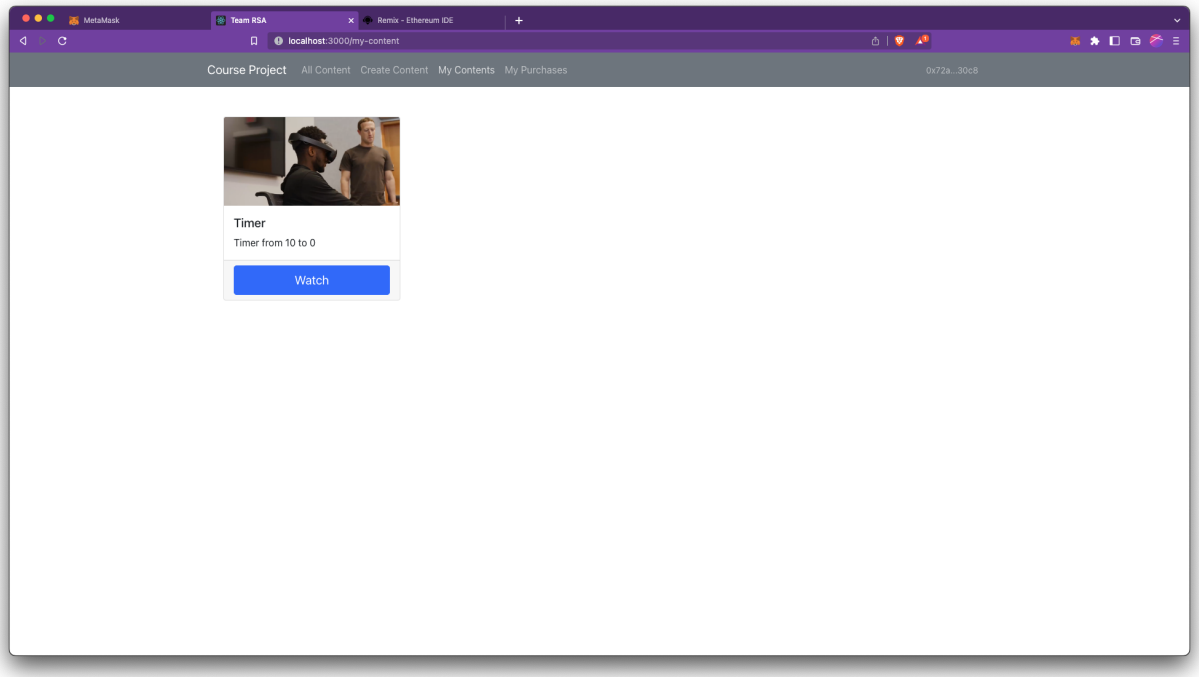
```
  address[] usersWithAccess;
  uint[] addedContentsId;
  uint[] boughtContentsId;

  mapping(uint => ContentInfo) public contents;
  mapping(address => UserInfo) users;

  constructor() {
    contentId = 0;
  }

  function addFile(string memory fileLocation, uint price) public returns
(uint) {

    contentId++;

    contents[contentId] = ContentInfo(
      fileLocation,
      price,
      payable(msg.sender),
      usersWithAccess
    );
    contents[contentId].usersWithAccess.push(msg.sender);

    users[msg.sender] = UserInfo(
      addedContentsId,
      boughtContentsId
    );
    users[msg.sender].addedContentsId.push(contentId);

    return contentId;
  }

  function getUserWithAccess(uint id) public view returns (address[]
memory) {
    return contents[id].usersWithAccess;
  }

  function getUsersPurchases() public view returns (uint[] memory) {
    return users[msg.sender].boughtContentsId;
  }

  function getUsersAddedContents() public view returns (uint[] memory) {
```

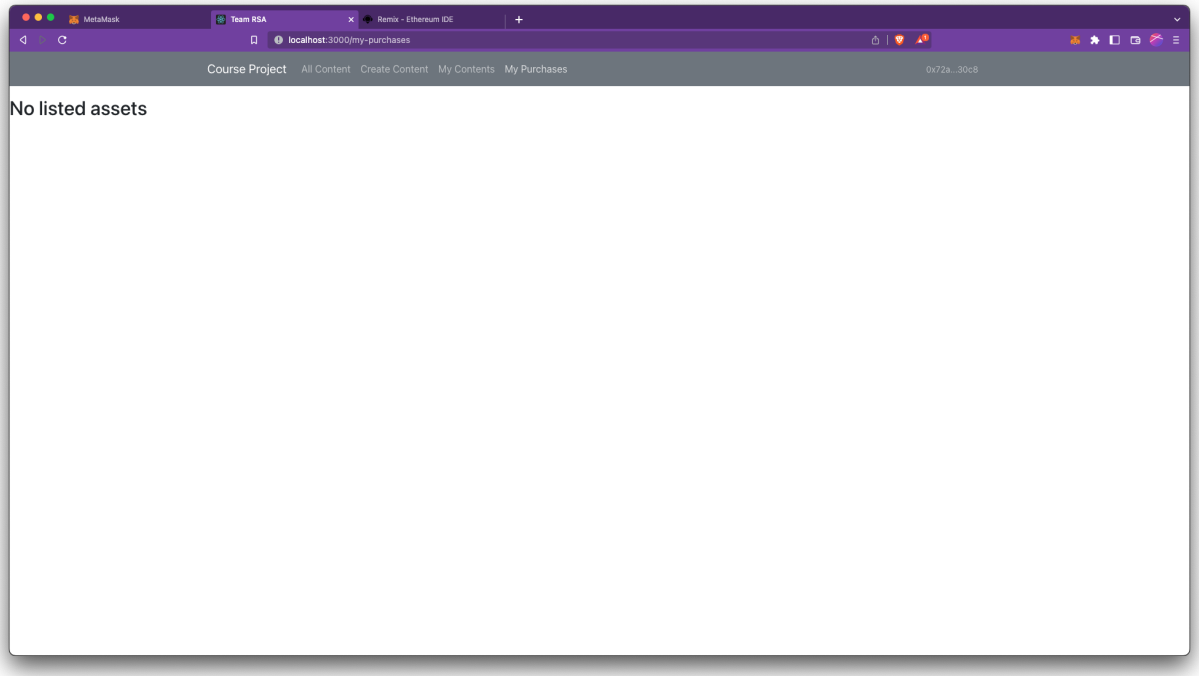
```
    return users[msg.sender].addedContentsId;
  }

  function getOwner(uint id) public view returns (address) {
    return contents[id].owner;
  }

  function giveAccess(uint id) public {
    contents[id].usersWithAccess.push(msg.sender);

    users[msg.sender].boughtContentsId.push(id);
  }

  function checkAccess(uint id, address userAddress) public view returns
(bool) {
    for (uint i = 0; i < contents[id].usersWithAccess.length; i++) {
      if (contents[id].usersWithAccess[i] == userAddress) {
        return true;
      }
    }
    return false;
  }

  function payPrice(uint id) public payable returns(bool) {
    require(msg.value >= contents[id].price, "Not enough Ether");
    address payable owner = contents[id].owner;
    owner.transfer(contents[id].price);

    giveAccess(id);

    return true;
  }
}
```

Frontend:


MetaMask
Team RSA
Remix - Ethereum IDE
localhost:3000
Course Project
All Content
Create Content
My Contents
My Purchases
0x72a...30c8
Timer
Timer from 10 to 0
Watch


MetaMask
Team RSA
Remix - Ethereum IDE
localhost:3000/create
Course Project
All Content
Create Content
My Contents
My Purchases
0x72a...30c8
Choose the Thumbnail File
Choose file
No file chosen
Choose the Video File
Choose file
No file chosen
Name
Description
Price in ETH
Create Content!


MetaMask
Team RSA
Remix - Ethereum IDE
localhost:3000/my-content
Course Project
All Content
Create Content
My Contents
My Purchases
0x72a...30c8
Timer
Timer from 10 to 0
Watch


MetaMask
Team RSA
Remix - Ethereum IDE
localhost:3000/my-purchases
Course Project
All Content
Create Content
My Contents
My Purchases
0x72a...30c8
No listed assets