BSCS-S24-005

03-134211-054 MUHAMMAD ALI EHTISHAM

03-134211-049 HASEEB ALAM JAN

DATAMARK

(COPYRIGHT PROTECTION USING BLOCKCHAIN)

In partial fulfilment of the requirements for the degree of

Bachelor of Science in Computer Science

Supervisor: **Dawood Akram**

Department of Computer Sciences

Bahria University, Lahore Campus

January 2025

C e r t i f i c a t e

We accept the work contained in the report titled

DATAMARK (Copyright Protection using Blockchain)

written by

MUHAMMAD ALI EHTISHAM

HASEEB ALAM JAN

as a confirmation to the required standard for the partial fulfilment of the degree of

Bachelor of Science in Computer Science.

Approved by:

Supervisor: **Dawood Akram**

_____ (Signature)

January 05, 2025

DECLARATION

We hereby declare that this project report is based on our original work except for citations and quotations which have been duly acknowledged. We also declare that it has not been previously and concurrently submitted for any other degree or award at Bahria University or other institutions.

Enrolment	Name	Signature
03-134211-054	**MUHAMMAD ALI EHTISHAM**	
03-134211-049	**HASEEB ALAM JAN**	

Date : January 05, 2025

DATAMARK
(COPYRIGHT PROTECTION USING BLOCKCHAIN)

ABSTRACT

The objective of this project is to develop a copyright project platform for digital content regardless of the category. This report emphasizes the use of blockchain technology for protecting the rights to digital content. This allows us to leverage blockchain technology to provide an innovative decentralized application (DAPP) that enables users, to secure their intellectual property while maintaining public transparency and authenticity.

The system enables users to upload their digital content, which after uploading is protected with a hash that is generated from the content's unique fingerprint and the creator's identity. Such metadata, together with the hash of the content and other relevant, such as timestamps, is kept on the blockchain. This provides a consistent and immutable indicator of ownership that can be presented during a copyright infringement or dispute that involves ownership.

To further assist the ecosystem, the platform allows third parties to create a "public proof URL" to request usage rights. This capability makes it easy to exploit digital assets, into which third parties may pay for legal use, and royalties are managed in a transparent manner through smart contracts on the blockchain.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

There is no doubt that anyone can become a global content creator now which would have been quite difficult in the past. But along with these benefits also crawled the difficulties in safeguarding the mish mash of creative ideas as well as the practices of fair use. The most widespread form of protecting copyright these days depends on authorities and agents which are weak spots, costly and dangerous because of data corruption and losses.

All these points can be easily overcome by using blockchain technology as it is the only solution to billions of problems humble in itself. Upon using the blockchain, a system can be developed in which the ownership of the content is recorded, verified and maintained without other people managing it. Because of cryptographic hashing, each content is able to possess a distinctive feature and because of smart contracts, time spent on administrative clauses is now left only as an afterthought.

The project's "Copyright Protection Using Blockchain" focuses on these precisely issues, offering solutions built around a decentralized architecture that protects digital content ownership. Content can be created with the assurance of its validity and quality, and together with a highly effective and transparent royalty system, content can be monetized.

1.2 Problem Statements

The growth in digital content production has brought along massive complications in the enforcement of copyright and its equitable use. Normally, traditional copyright systems are built around decentralized governance. Such systems are often inefficient, have high operational costs, and are prone to data breaches and other vulnerabilities. The enforcement of these systems is also opaque, making the rights and ownership of creators ill-defined and very difficult to verify in instances of unconsented use or infringement.

Additionally, they do not assist creators with managing royalties or monetizing content effectively, which prevents creators from being able to earn a reasonable amount from their creation. Organizations and users that want to check ownership of the content or wish to use it also have a lot of problems as the existing methods are labor intensive, expensive, and very slow.

The absence of a trustworthy system for the administration of the legal rights of digital content not only makes creators vulnerable to pirated content but also slows the development of a just and viable digital market, economy for creators to flourish within.

This project seeks to address these challenges by leveraging blockchain technology to create a decentralized, immutable, and transparent platform that ensures secure content ownership, simplifies verification processes, and enables creators to monetize their work efficiently.

1.3 Aims and Objectives

The purpose of this study is to design a DAPP to avail secure copyright, registration and monetization of the digital content using blockchain technology while

enhancing transparency and fair dealing of the intellectual property rights at the same time.

i. **Content Ownership and Verification:**

Develop a system that creates and digitally signs hashes of content and creator identity which is permanently recorded on a distributed ledger as proof of ownership.

ii. **Decentralized Copyright Protection:**

Use smart contracts to create accounts, register, store and retrieve digital content ownership information on the blockchain.

iii. **User Authentication:**

Develop a secure user authentication system that uses email and Ethereum wallet addresses, creating a distinct identity for each user.

iv. **Content Monetization and Royalty Management:**

Make it possible for content owners to sell their assets and set up royalties and accept direct payments via smart contracts on the blockchain.

Automate the process of buying royalties for third parties.

v. **Public Proof Accessibility:**

Provide an open platform for third parties to verify ownership or obtain usage rights without requiring intermediaries, thereby promoting trust and transparency.

vi. **Scalability and Integration**:

Ensure the solution integrates seamlessly with existing digital ecosystems and supports scalability to accommodate a growing number of users and content types.

vii. **Real-World Deployment**:

Deploy the system on a blockchain network (e.g., VANARCHAIN) with a React-based frontend, ensuring user-friendly interaction and reliable performance.

1.4 Scope of Project

The goal of the "Copyright Protection Using Blockchain (DATAMARK)" project is to change the way people think about digital copyright management by allowing for the development of an innovative model for protection and monetization of intellectual

property in a decentralized, transparent, and secure manner. This project is useful to actual content authors like writers, artists, photographers, digital developers, etc who are able to claim ownership, control the uses of and earn income from their works.

This emphasizes the role of reunion loyalty and weather estimating in ensuring ownership of content on the platform is unchangeable and true, thereby in no need of centralized powers or go between. The use of Smart contracts ensures that royalties and ownership of properties are easily paid and verified within the system making it a fair without the possibility of cheating. Other users also within the Third parties can show evidence of ownership and alternatively apply for permission for use through this simplified process thus engendering credibility and simplification.

This project has no limitation, particularly when it comes to the scale of the undertaking, different types of digital content can be supported (text, image, video) and it will be combined with the innovative web 3.0. Now the initial stage was setup on Sepolia test network and now has been moved to VanarChain testnet as the main platform, however, this solution is suitable for future implementation on public blockchains as well.

CHAPTER 2

Software Requirement Specification

2.1 Purpose

This document aims at identifying requirements for the Copyright Protection Using Blockchain (DATAMARK). The goal of the project is to offer a decentralized platform for securing digital content & verifying ownership of digital content for copyright protection and enable selling of royalites of digital content rights for monetization leveraging blockchain technology.

2.2 Scope

The project caters to digital content creators and third-party users by:

- Providing immutable proof of ownership for digital content.
- Automating copyright verification and royalty payments via blockchain smart contract.
- To enable third parties to buy usage rights, or to check ownership without obscuring operations.

2.3 Definition, Abbreviations & Acronyms

DAPP: Decentralized Application

Blockchain: A distributed ledger technology providing immutability and transparency.

Smart Contract: A self-executing contract on the blockchain.

EVM: Ethereum Virtual Machine

HASH: A hash is a mathematical function that converts an input of arbitrary length into an encrypted output of a fixed length. Thus, regardless of the original amount of data or file size involved, its unique hash will always be the same size.

2.4 Overall Description

2.4.1 Product Perspective

The system operates as a standalone decentralized web application utilizing blockchain technology for core operations, React for frontend UI and MongoDB for managing user credentials.

2.4.2 Product Features

2.4.3 User Registration and Authentication

User signs up using an email, sets a password and links their EVM compatible wallet. Smart contract checks and verify unique user credentials on blockchain to make sure existing credentials weren’t used to register again and user can login using the same credentials while registering.

2.4.4 Content Patent

User uploads content the content is hash and stored on-chain including metadata (e.g. timestamp, ownership, content URL).

2.4.5 Ownership Verification

Each piece of digital content when stored on-chain is assigned a public URL which can be used to verify the ownership of that digital content and access information related to that digital content, same page which the URL leads can be used to buy royalty for that digital content.

2.4.6 Content Monetization

- Users can make contents available for sale and set royalty fees while uploading the content.
- Third party can acquire the rights for the utilization of the product with coming pre-programmed system of distributing royalties.

2.4.7 Royalty Management

User is allowed to manage royalties he/she has sold for his content and similarly user can also manage the royalites he/she has bought for content not owned by them.

2.4.8 Functional requirements

- **User Management**

Allow user registration with an email and wallet address.

Validate user uniqueness using on-chain checks.

- **Content Upload**

Hash the content and creator's identity.

Store hashes and metadata on the blockchain.

- **Ownership Proof**

Generate public proof URLs for content ownership.

- **Content Monetization**

Enable listing content for sale with royalty settings.

Process payments and distribute royalties via smart contracts.

- **Third-Party Verification**

Allow third-party users to request usage rights or verify ownership.

2.4.9 Non-Functional requirements

- **Security**

Implement secure hashing for content and user credentials.

Ensure data integrity and authenticity through blockchain immutability.

- **Performance**

Optimize smart contract gas usage for cost-efficiency.

Handle multiple user requests concurrently without delays.

- **Scalability**

Support a growing user base and various content types.

- **Usability**

Provide a user-friendly interface with clear workflows for content registration, verification, and monetization.

- **Availability**

Ensure high availability of the DAPP, accessible 24/7.

CHAPTER 3

Literature Review

The DATAMARK project operates at the intersection of blockchain technology and IP management. This literature review explores concepts, relevant studies, and gaps in the existing framework that have led to the project's initiation.

3.1 Blockchain Technology and Its Applications

Blockchain technology, first introduced by Satoshi Nakamoto through Bitcoin in 2008, has evolved into a highly versatile platform for secure and decentralized record-keeping. Ethereum furthered the utility of blockchain by introducing smart contracts, which are programmable agreements executed without intermediaries. The attributes of transparency, immutability, and decentralization make blockchain an ideal candidate for solving copyright management issues.

3.1.1 Smart Contracts

They are essential for the automation of registration, verification, and royalty transactions within DATAMARK. Research, such as "Smart Contracts for Digital Rights Management" (Li et al., 2019), examines the capabilities of blockchain technology in automating intellectual property transactions.

3.1.2 Intellectual Property Management Challenges

Conventional intellectual property management systems significantly depend on centralized authorities, a reliance that frequently results in inefficiencies:

- **Ownership Ambiguity**

Without proper verification mechanisms, disputes over ownership are common.

- **Unauthorized Use**

Creative work is frequently used without permission due to the lack of availability of ownership information.

- **Ineffective Royalties Administration**

The systems currently in place are not transparent, and creators face challenges in tracking royalty payments.

Studies like "Blockchain for Copyright Protection" by Gipp et al., (2017) highlighted how blockchain technology can deal with these inefficiencies as a way of providing an immutable ledger for content ownership and transactions.

3.1.3 Existing Blockchain-Based Solutions

Many blockchain-based systems aim to address copyright and intellectual property issues, including platforms like Ascribe and Po.et. While these systems focus on registering content on the blockchain, they lack complete functionality, such as user identity integration, public proof mechanisms, and royalty management, which DATAMARK addresses.

3.1.4 Gaps in Current Systems

There is a lack of integration between ownership verification and on-chain royalty management in most existing systems.

- **Accessibility Issues**

Many interfaces require sophisticated technical know-how or focus solely on digital art (such as NFTs), thereby limiting access to most content producers.

- **Costly Fees**

Gas fees incurred through blockchains that many use cause significant barriers to entry.

3.1.5 Pertinence of DATAMARK

DATAMARK furthers the building blocks provided but also takes those building blocks in meaningful new directions:

- **User Identity Integration**

The combination of content and user identity hashes guarantees full ownership verification by DATAMARK.

- **Public proof mechanism**

Allows third parties to verify ownership or start royalty flows without compromising privacy.

- **Comprehensive scope**

It supports any possible content types (text, media, documents) with built-in royalties, so wider usability is guaranteed.

3.1.6 Conclusion

This literature review stresses the need for a secure, transparent, and user-friendly blockchain-based copyright protection system. DATAMARK addresses the deficiencies of extant solutions by confronting basic challenges in ownership verification, royalties' management, and accessibility to render it a distinctive and important contribution to the field.

CHAPTER 4

DESIGN AND METHODOLOGY

4.1 System Architecture

The “Copyright Protection Using Blockchain (DATAMARK)” system is solely designed on decentralized architecture which leverages blockchain for security immutability and transparency. Its incorporation of a frontend, a backend, smart contracts on a blockchain, and a NoSQL database.

4.1.1 Architecture Diagram:

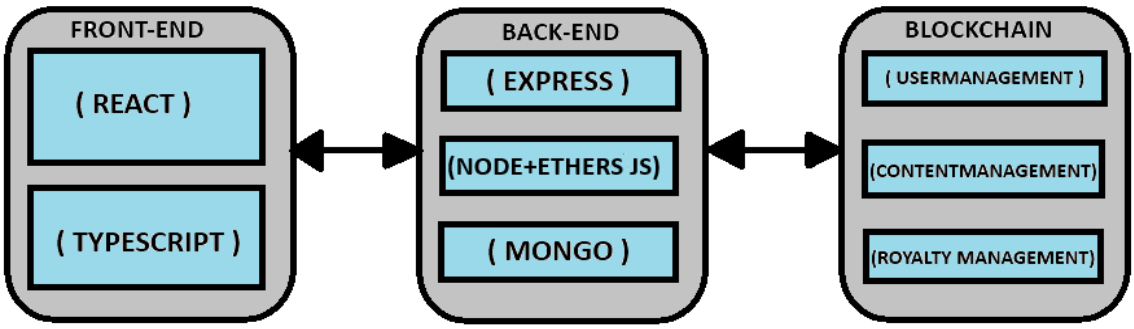


Figure 4.1: Architecture Diagram

4.1.2 Frontend

- React and TypeScript for the user interface.

- Users interact with the DAPP to upload content, view ownership details, and manage royalties.

4.1.3 Backend

- Node.js with Express handles API requests, user authentication, and off-chain data storage.
- Interfaces with the Ethereum blockchain through **Ethers.js**.

4.1.4 Blockchain

- Smart contracts on the Ethereum blockchain (deployed on VanarChain test-net) manage:
    - User registration/verification.
    - Content registration/verification.
    - Royality management.

4.1.5 Database

- MongoDB (Atlas) stores hashed user credentials and metadata for user management.

4.2 Workflow Methodology

4.2.1 User Registration:

- Users register with an email and connect their MetaMask wallet.
- A smart contract ensures email and wallet address uniqueness on-chain.
- Off-chain, hashed credentials are stored in MongoDB.

4.2.2 Content Upload:

- The content is hashed using a combination of content data and user identity.
- The hash related metadata like timestamp, ownership details are stored on the blockchain for immutability.

4.2.3 Ownership Verification:

- The platform generates a unique, public URL for ownership proof.
- Third parties can verify ownership using this URL without additional authentication.

4.2.4 Content Monetization:

- Creators list their content for sale and set royalties.
- Smart contracts automate the payment process, ensuring transparency.

4.2.5 Royalty Management:

- Buyers can purchase usage rights via the platform.
- Smart contracts handle royalty distribution to content creators.

4.2.6 Workflow sequence diagram

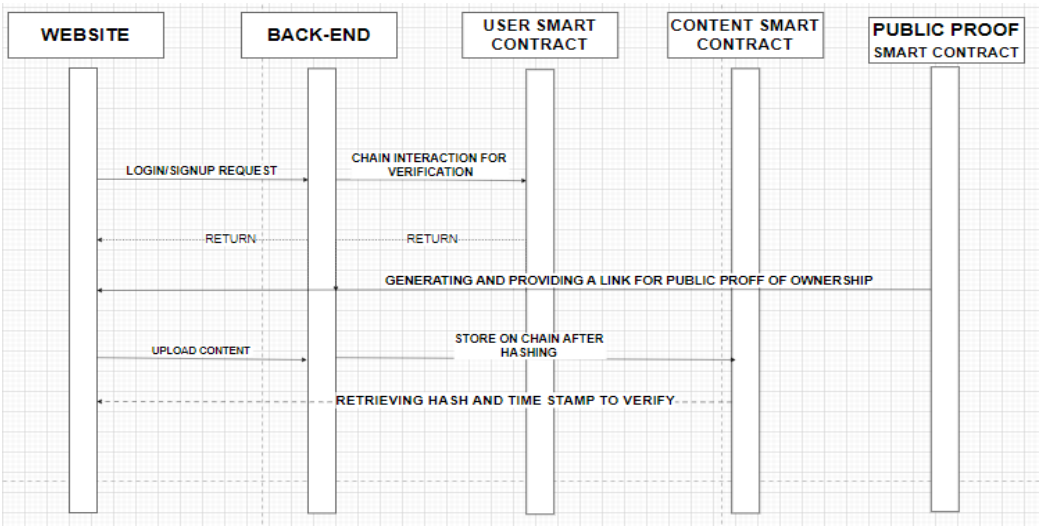


Figure 4.2.1: Workflow sequence diagram

4.2.7 UML class diagram

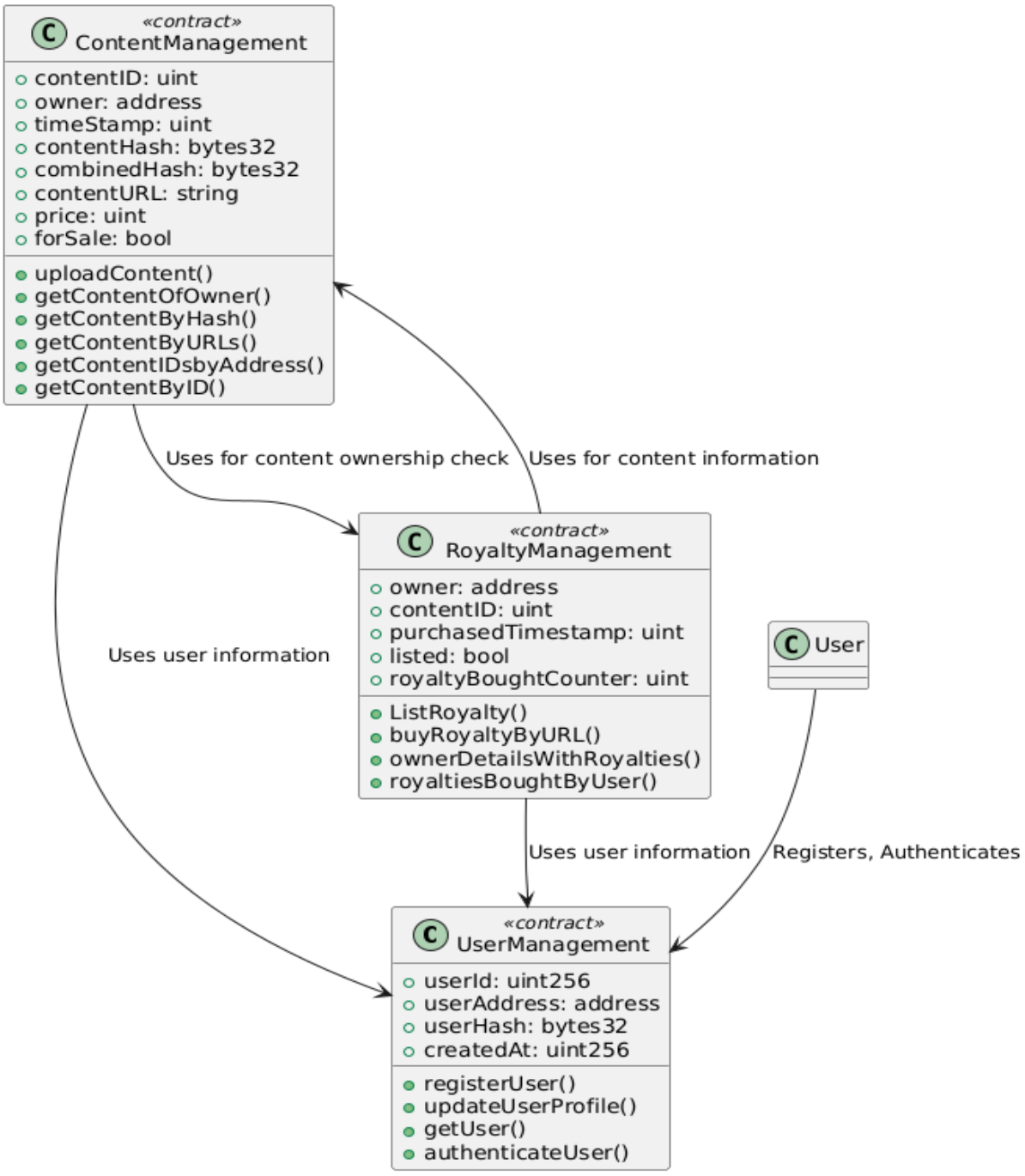


Figure 4.3.2: UML

4.3 Design Approach

4.3.1 Blockchain:

Smart Contracts:

- Written in Solidity.
- Function for uploading of content, proving ownership and management of royalty.

Network:

- Currently deployed on Vanarchain test-net .

4.3.2 Frontend Design

- Clean and responsive UI built on top of React (typescript + vite) and CSS.
- Wallet connectivity and user authentication using MetaMask Auth.
- Components for content management, ownership verification, and the ability to purchase royalties.

4.3.3 Backend Design

- Node.js server processes API requests from the frontend.
- Uses Ethers.js for smart contract interactions.

4.3.4 Database Design

MongoDB stores:

- Hashed email addresses.
- Hashed wallet addresses.
- Metadata for optimal user organization and management.

CHAPTER 5

IMPLMENTATION

5.1 Implementation

5.1.1 Technology Stack

The system utilizes the following technologies:

- **Frontend**: React, TypeScript.
- **Backend**: Node.js, Express.js, MongoDB.
- **Blockchain**: Ethereum, Solidity, Ethers.js.

5.1.2 Smart Contract Implementation

- **usermanagement**: Registers user on-chain, manages on-chain authentication.
- **contentmanagemnt**: Uploads hashes and relative metadate on chain, functions for retrieval of data stored.
- **publicproof/royalitymanagement**: Handles royalty management, royalites bought/sold.

5.1.3 Deployment:

- Contracts deployed on vanarchain test-net.

5.1.4 Backend Implementation

5.1.5 API Endpoints:

- **/api/login**: Authenticates users.
- **/api/register**: registration for new users.

5.1.6 Blockchain Integration:

- Ethers.js is used to interact with smart contracts for on-chain operations.

5.1.7 Frontend Implementation

Key Features:

- User-friendly interface for uploading content and managing royalties.
- MetaMask integration for wallet-based authentication.

Components:

Login, Register, Services Page, My Content, Content Display.

CHAPTER 6

USER MANUAL

6.1 System Requirements

Use DATAMARK with the following criteria in place:

6.1.1 Hardware Requirements

- A computer or cell phone with Internet.

6.1.2 Software Requirements

- **Browser**: Chrome, Firefox, Edge, etc.
- **Extensions**: MetaMask, or any other EVM wallet

6.2 Installation and configuration

Step 1. Download MetaMask.

- Visit https://metamask.io.
- Download and install the browser extension.

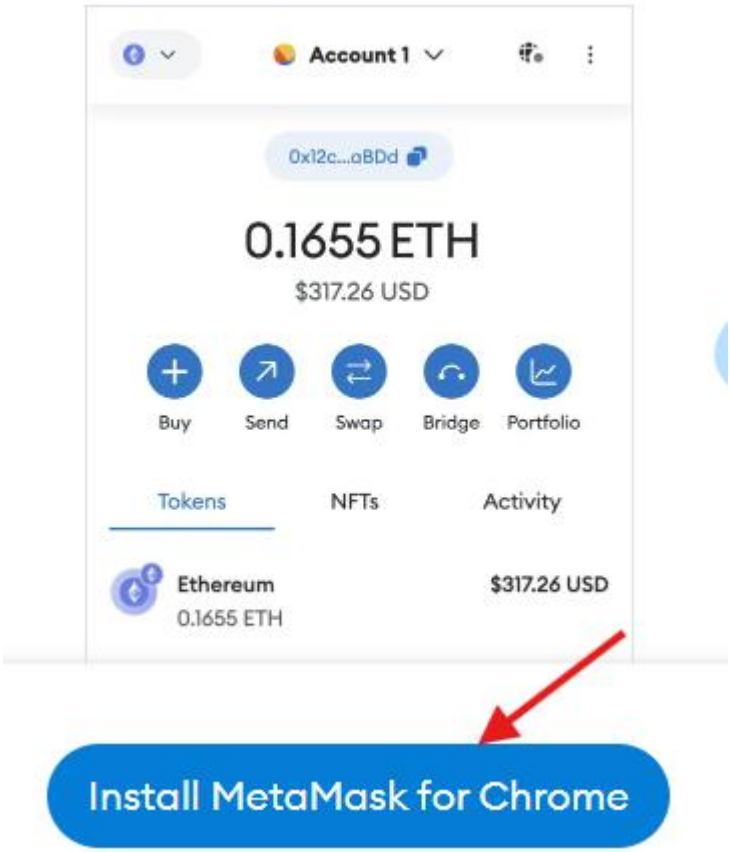


Figure 6.1.1: MetaMask Extension

Step 2: Connect MetaMask to Vanar Test-net(optional)

- Open MetaMask and create/import a wallet.
- Switch to the Vanarchain test-net under network settings.

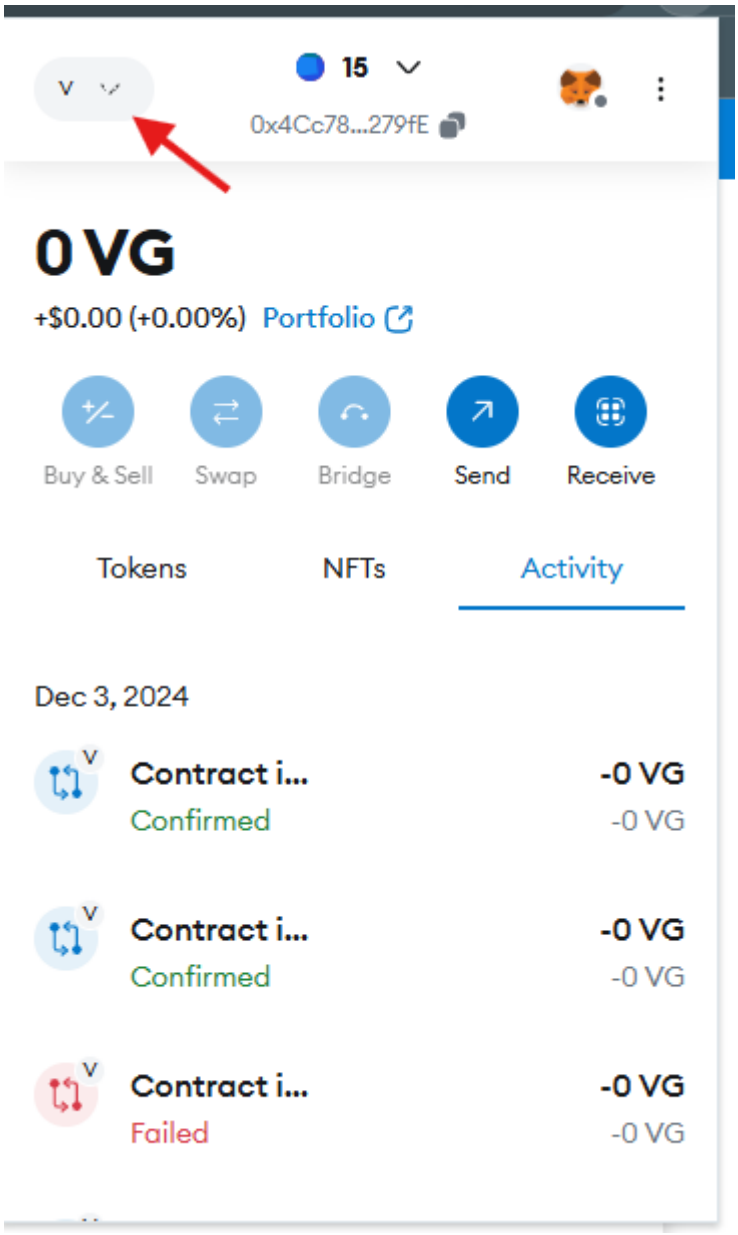


Figure 6.2.2: MetaMask Extension (switch network)

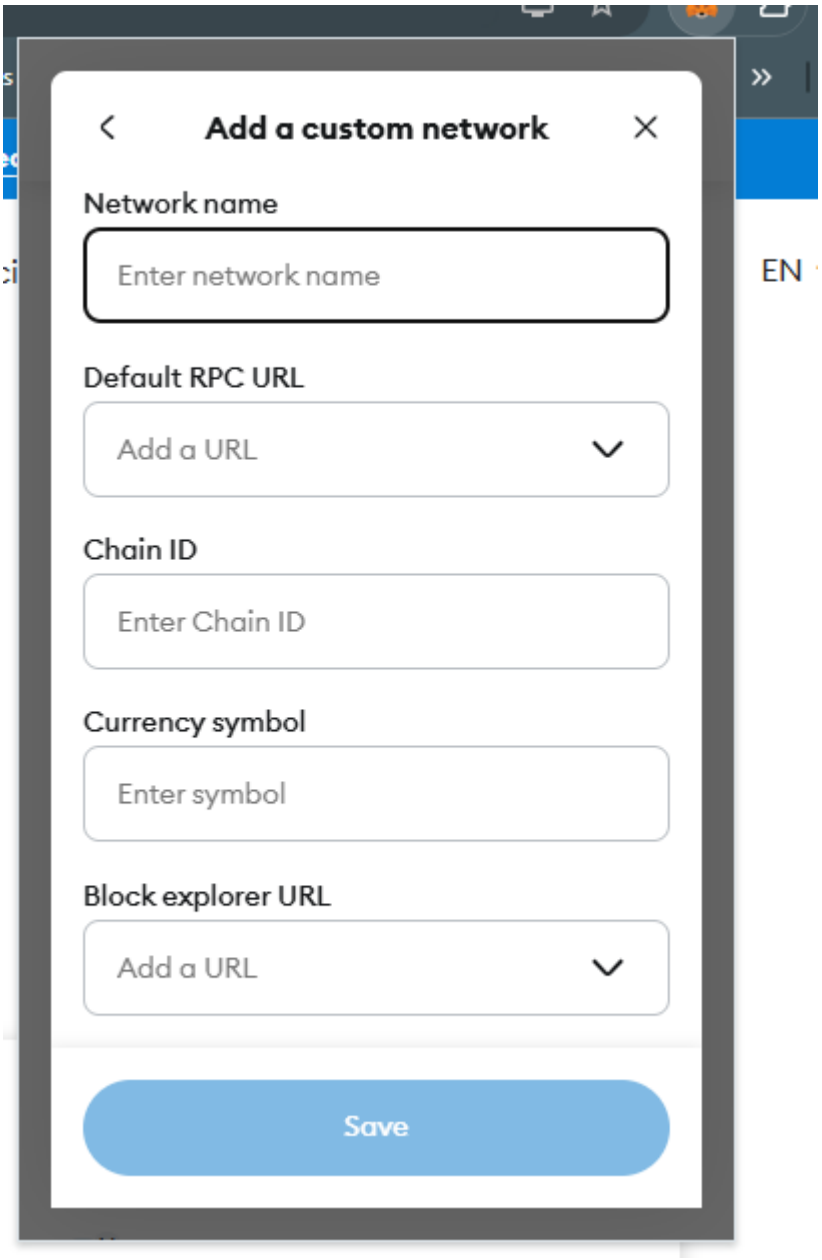


Figure 6.3.3: MetaMask Extension (switch network)

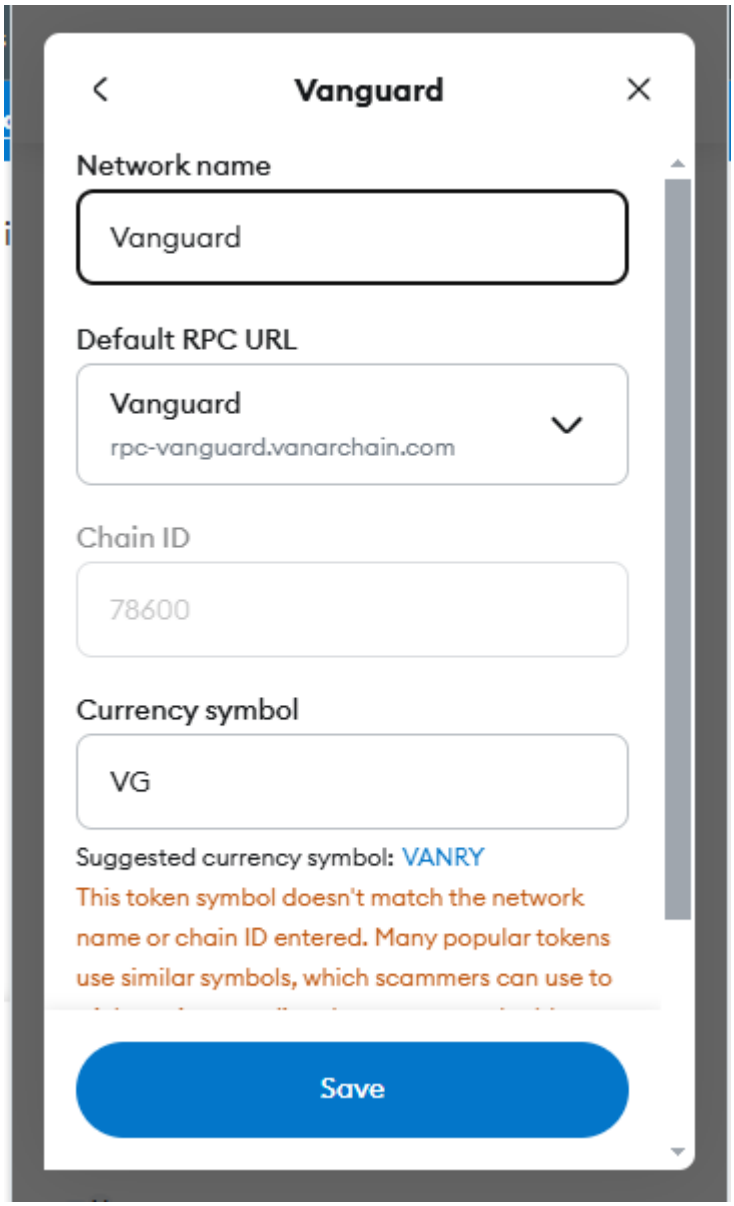


Figure 6.4.4: MetaMask Extension (switch network)

Step 3: Access DATAMARK:

- Visit the DATAMARK platform.
- Ensure MetaMask is connected when prompted.

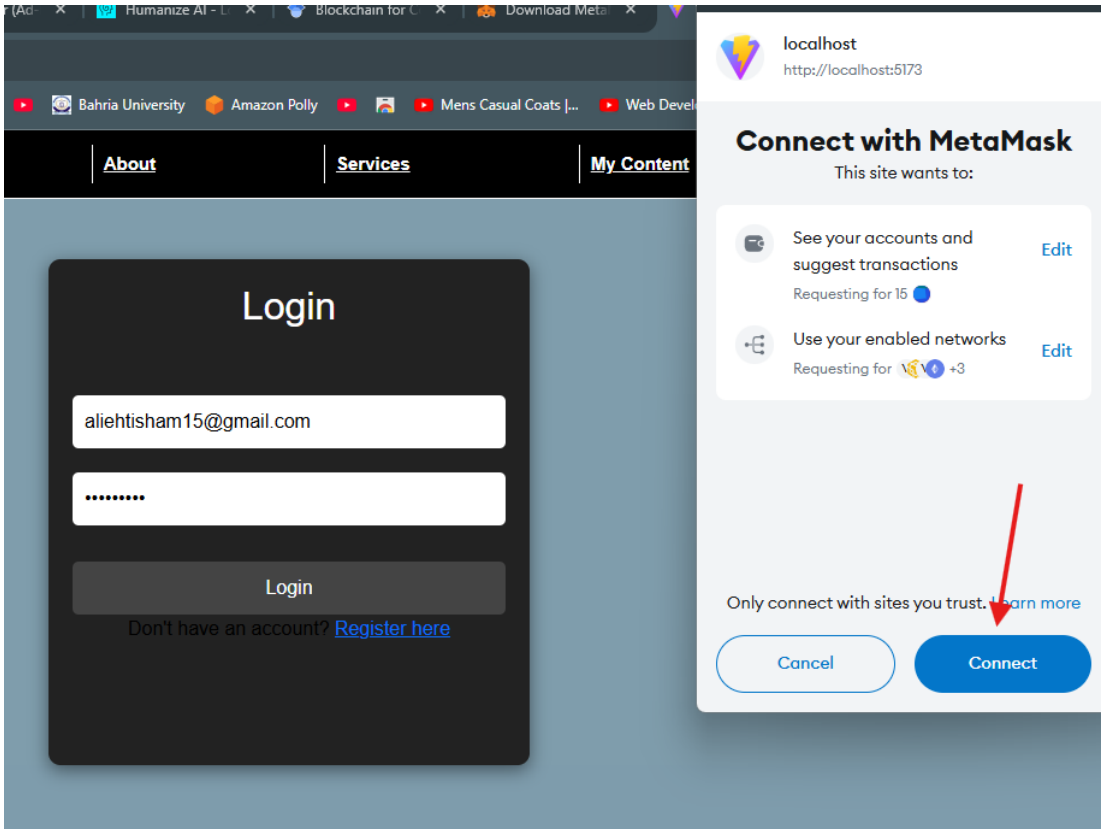


Figure 6.5.5: Connect wallet

6.3 Features and Navigation

6.3.1 Registration

- Navigate to the Register page.
- Enter the details requested (email, password).
- Connect your MetaMask wallet.
- Click the Register button.

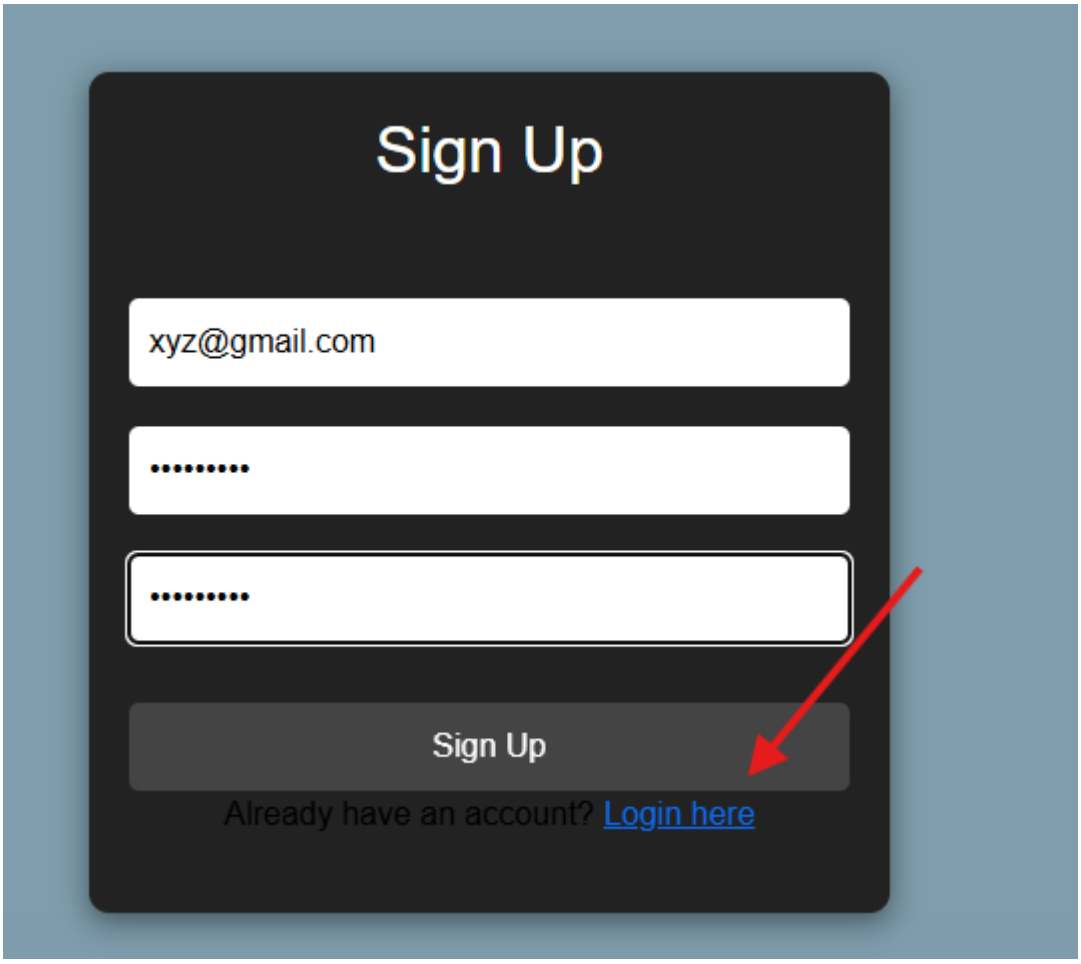


Figure 6.6.6: Signup

6.3.2 Login

- Click on the "Login" link.
- Use your registered email and the password.
- Connect your MetaMask wallet to access the dashboard.

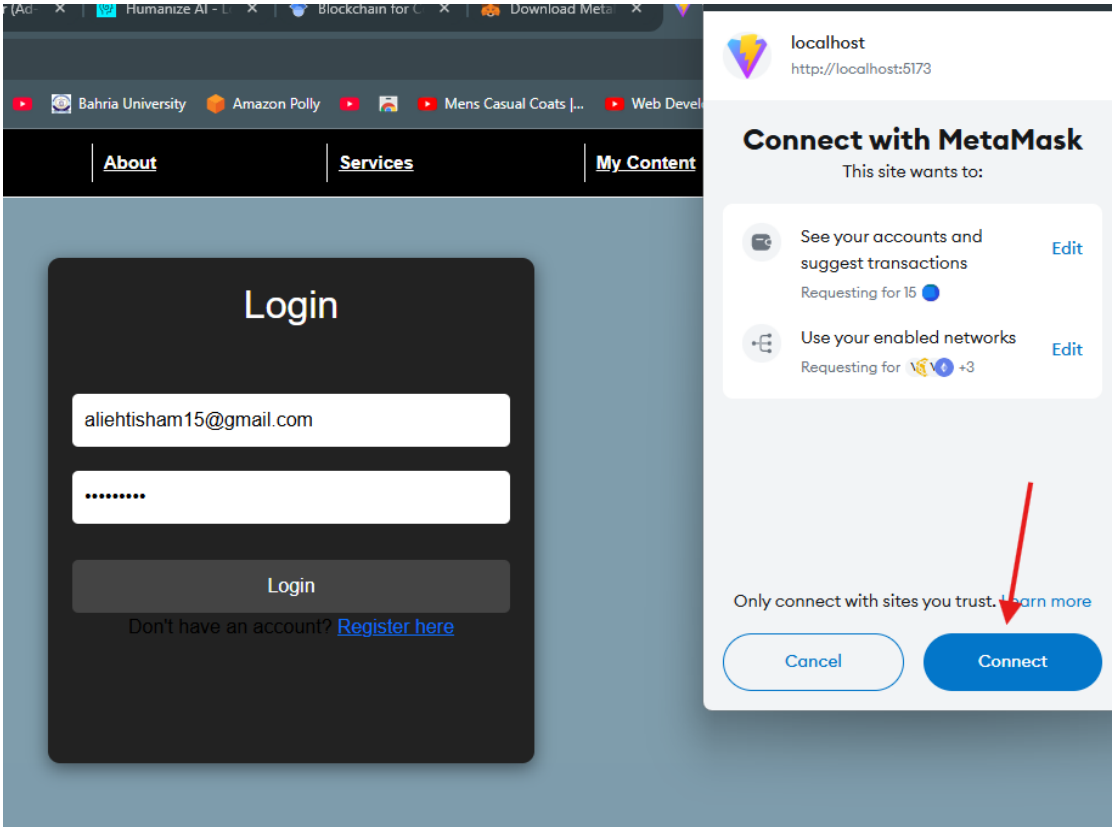


Figure 6.7.7: Login

6.3.3 Upload Content

- Click on the Services page.
- Use the drag and drop facility or click Browse Files to upload a file.
- Set the content to For Sale and enter a price if appropriate.
- Click Upload to store your content's details (hash, URL, etc.) on the blockchain.

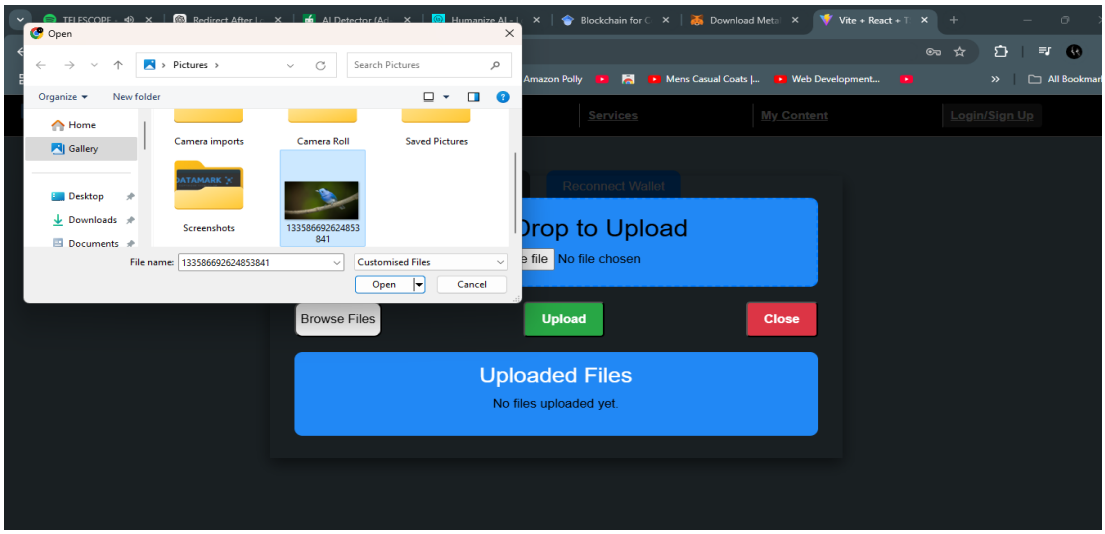


Figure 6.8.8: Upload content

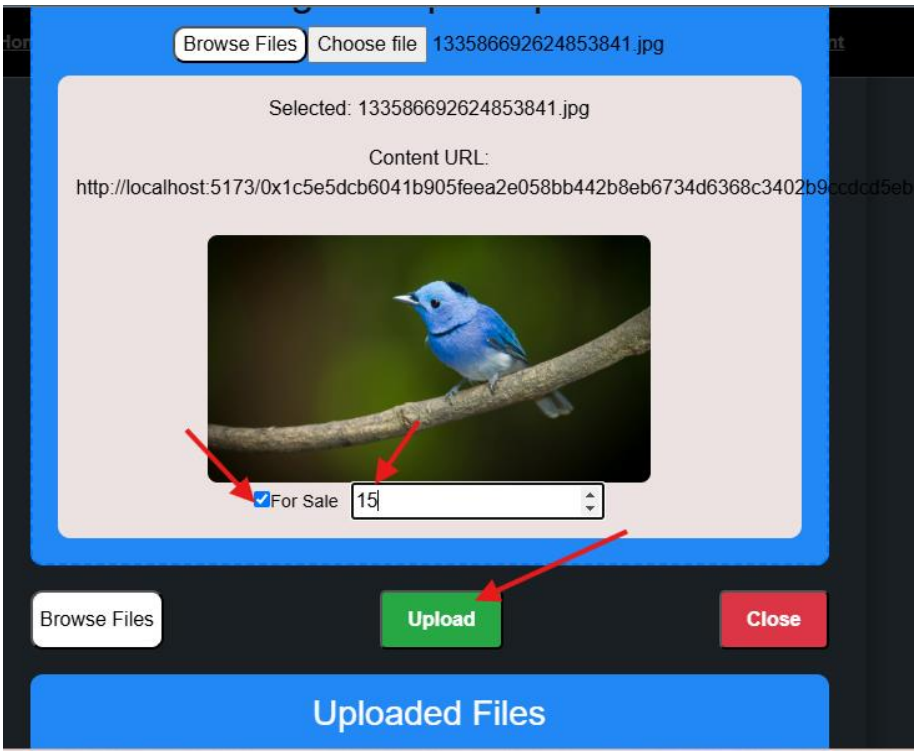


Figure 6.9.9: Upload content (Services page)

6.3.4 View Your Content

- Navigate to the **My Content** page.
- View all your registered content, including:
  Content ID, Ownership Details, Content Hash, Combined Hash, URL, and Price.
- Copy any data by clicking on the respective table cell (a message will confirm copying).

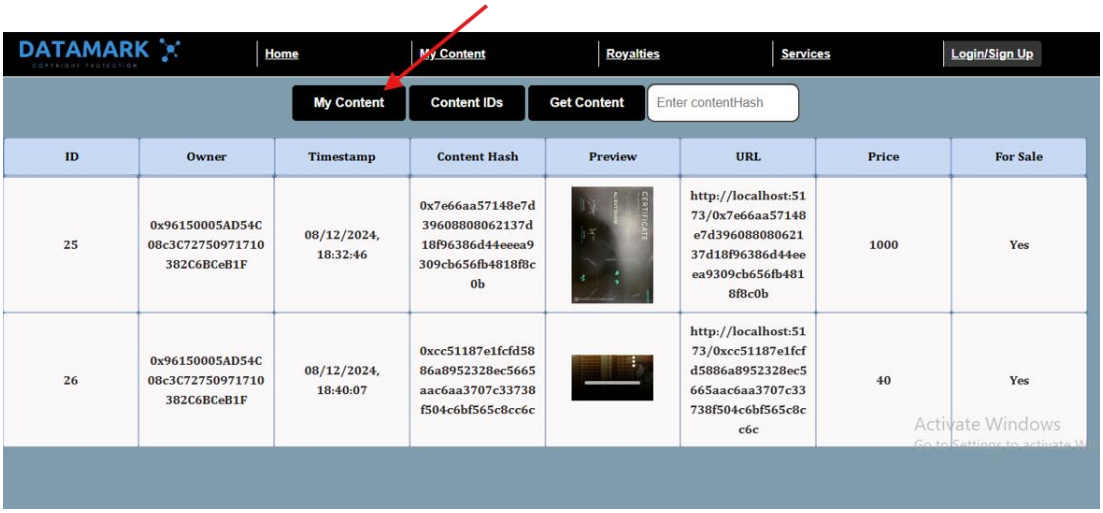


Figure 6.10.10: My content

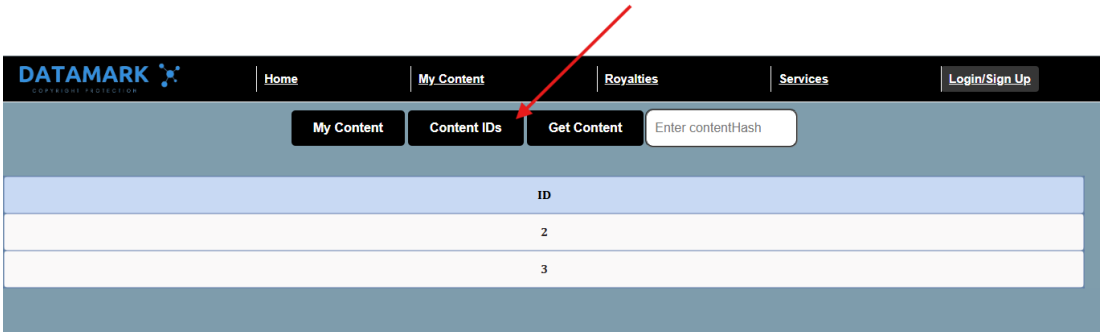


Figure 6.11.11: Content ID's

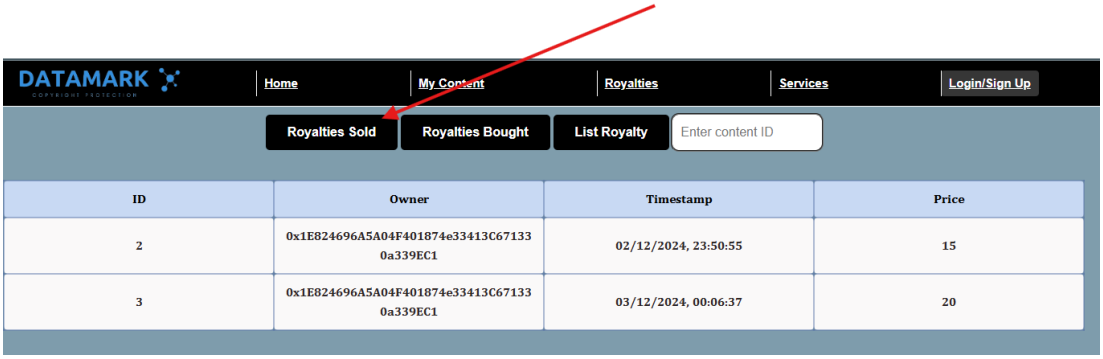


Figure 6.12.12:Royalties

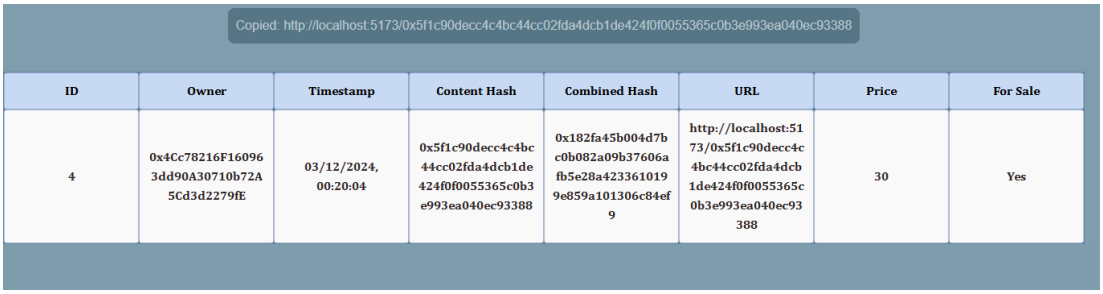


Figure 6.13.13: on click (copy)

6.3.5 List Royalty

- Navigate to the Royalties Page from the navbar.
- Enter the content id in the input field to list the content for royalty.
- Click the button list royalty to list the royalty

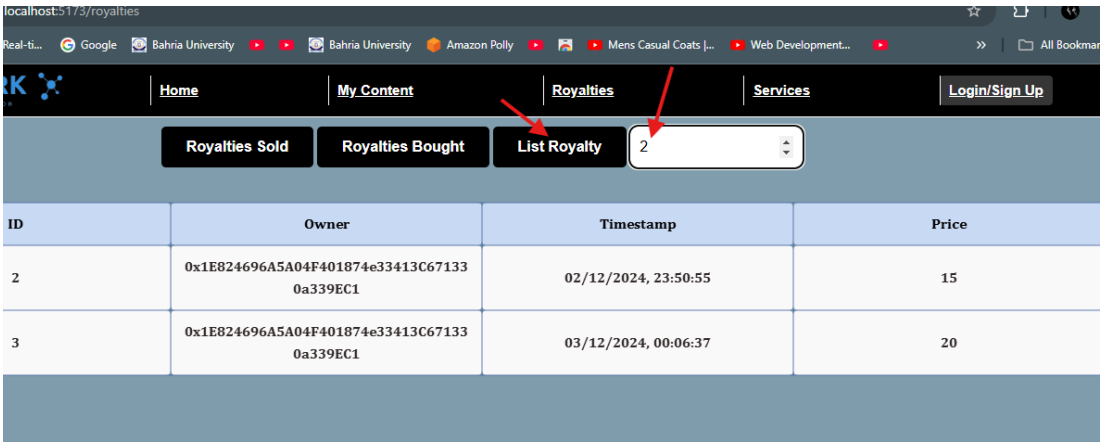


Figure 6.14.14: List Royalty

6.3.6 Purchase Royalty

- Open the content URL shared by the creator.
- Click Buy Royalty.
- Complete the transaction with MetaMask.

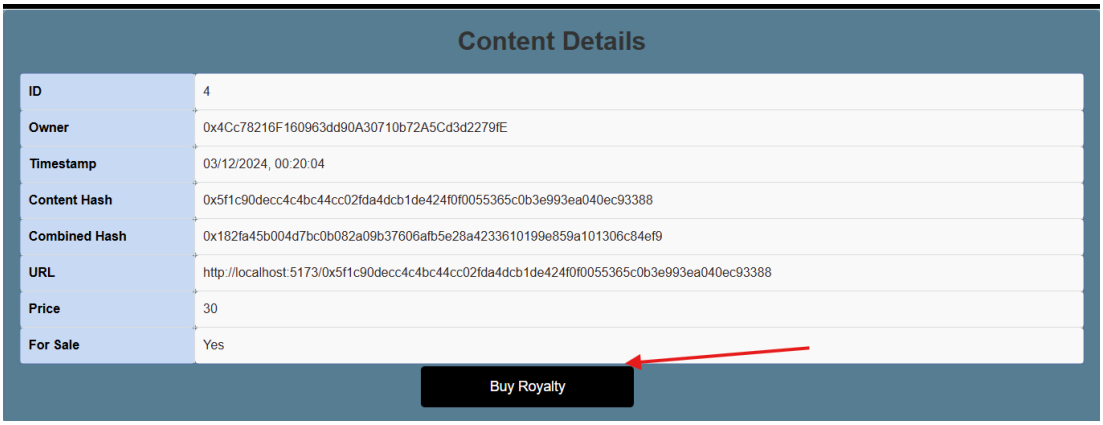
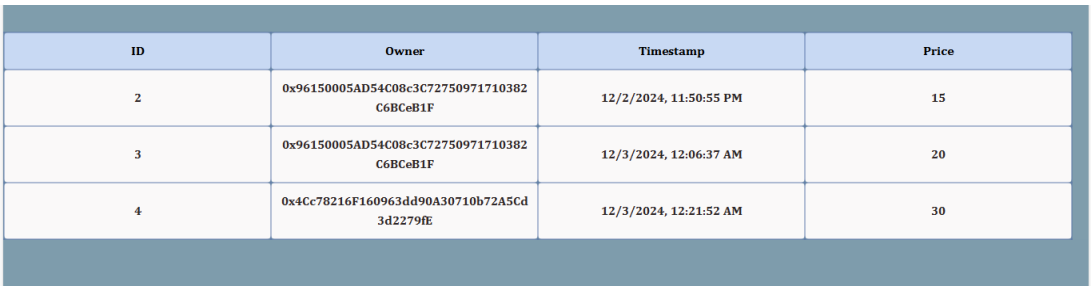


Figure 6.15.15: Buy Royalty



ID	Owner	Timestamp	Price
2	0x96150005AD54C08c3C72750971710382C6BCeB1F	12/2/2024, 11:50:55 PM	15
3	0x96150005AD54C08c3C72750971710382C6BCeB1F	12/3/2024, 12:06:37 AM	20
4	0x4Cc78216F160963dd90A30710b72A5Cd3d2279fE	12/3/2024, 12:21:52 AM	30

Figure 6.16.16: Royalties Bought

6.4 Troubleshooting

Issue 1: MetaMask is not connecting at login.
Solution: Ensure that MetaMask is set up or if one needs to switch the network manually.

Issue 2: Content is not uploaded.
Solution: Check file format maybe not supported. Also, check your internet connection and wallet connection.

Issue 3: Transaction failed while purchasing royalty. Ensure your wallet has enough Vanry for a transaction.

CHAPTER 7

CONCLUSION AND RECOMMENDATIONS

7.1 Conclusions

The DATAMARK project effectively addresses the burning need for the protection of ownership in digital content in the growing tidal wave of online content production. Utilizing blockchain technology, this system provides an unmodifiable and transparent platform for intellectual property rights registration, verification, and management. Its great achievements include:

- A secure content registration process that ties user identity with content hashes to validate ownership robustly.
- A transparent mechanism of public proof, third parties can verify ownership, request royalty transactions.
- A user-friendly interface designed for content creators facilitates the uploading, management, and monetization of their intellectual property.

This project demonstrates the practicality of blockchain for intellectual property management and paves the way for its adoption in the creative industry. It effectively addresses challenges such as unauthorized use of content, lack of transparent ownership verification, and difficulty in enforcing royalties.

However, the current implementation is limited Vanarchain test-net and relies

on MetaMask for wallet integration. These limitations highlight opportunities for further improvement.

7.2 Recommendations

7.2.1 Enhancements of Current Operations

- The wallet compatibility with popular wallets such as WalletConnect and Coinbase Wallet.
- Make it possible for creators to change the royalty price of registered content.
- Inform the creator as soon as possible if/when royalties are bought or sold.

7.2.2 Multi-chain Support

- Support for alternative blockchain networks like Polygon and Binance Smart Chain to reduce gas fees and improve scalability.
- Cross-chain interoperability for broader accessibility.

7.2.3 Usability and accessibility

- Create a mobile version of the platform to retain the moving creators
- Enhance the user interface with intuitive navigation, advanced search filters, and improved visual design for better user experience.

7.2.4 Advanced Features

- Use AI driven copyright infringement detection, allowing creators to identify unauthorized use of their content.

7.2.5 Integration and Partnerships

- Develop APIs to enable connectors to DATAMARK for content registration to platforms such as YouTube, Shutterstock and Getty Images.

7.2.6 Integration and Partnerships

- APIs for integrating DATAMARK with existing platforms like YouTube, Shutterstock, or Getty Images for seamless content registration and verification.
- Partner with legal firms to establish blockchain-based copyright registration as legally recognized evidence in court proceedings.

REFERENCES

Electronic Sources from Internet:

1. Ethereum Foundation, "Ethereum Documentation,". Available: https://ethereum.org.
2. Solidity Team, "Solidity Documentation," Available: https://docs.soliditylang.org/en/v0.8.28/
3. MongoDB Inc., "MongoDB Documentation," Available: https://www.mongodb.com/docs/.
4. Ethers.js Team, "Ethers.js Documentation," Available: https://docs.ethers.org/v6/
5. Remix IDE, "Remix: Ethereum IDE for Smart Contract Development," Available: https://remix.ethereum.org/.
6. Hardhat Team, "Hardhat Documentation," Available: https://hardhat.org/docs.
7. Stack Overflow, "Developer Discussions and Solutions," Available: https://stackoverflow.com/.
8. OpenAI, "Understanding Cryptographic Hashing and Blockchain Concepts," Available: https://openai.com/

APPENDICES

APPENDIX A: Smart Contract Codes

User Management smart contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

contract usermanagement{
struct User{
uint256  userId;
address  userAddress;
bytes32  userHash;
uint256  createdAt;
}
mapping(address=> User) internal users;
mapping(bytes32=>address)private userHashToAddress;
address[] public userAddresses;
uint256  private userCount;

event UserRegistered(address userAddress, uint256 userId, bytes32 userHash);

   event UserProfileUpdated(address userAddress, bytes32 newUserHash);

function registeruser(bytes32 _userhash) public {
   require(users[msg.sender].userAddress==address(0),"User Already Registered with this wallet address!");
   require(userHashToAddress[_userhash]==address(0),"User Hash already registered with other wallet address");
```

```
 userCount++;
 users[msg.sender]=User({
   userId:userCount,
   userAddress:msg.sender,
   userHash:_userhash,
   createdAt:block.timestamp

 }
 )
 ;
 userAddresses.push(msg.sender);
 userHashToAddress[_userhash]=msg.sender;
 emit UserRegistered(msg.sender, userCount, _userhash);

}
function updateUserProfile(bytes32 _newUserHash) public {
        require(users[msg.sender].userAddress != address(0), "User not found");
        users[msg.sender].userHash = _newUserHash;

        emit UserProfileUpdated(msg.sender, _newUserHash);
    }

    function getUser(address _userAddress) public view returns (User memory) {
        require(users[_userAddress].userAddress != address(0), "User not found");
        return users[_userAddress];
    }
    function authenticateUser(address _userAddress,bytes32 _hash) public view returns
(bool) {
        require(userHashToAddress[_hash]==_userAddress,"This wallet address linked
with other username");
        return users[_userAddress].userAddress != address(0);
    }

}
```

Content Management Smart Contract:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

import "./userManagement.sol";

contract ContentManagement {
    usermanagement private userManagementContract;

    struct Content {
        uint contentID;
        address owner;
        uint timeStamp;
        bytes32 contentHash;
        bytes32 combinedHash;
        string contentURL;
        uint price;
        bool forSale;
    }

    mapping(address => uint[]) internal contentIDsbyAddress;
    mapping(uint => Content) internal contentbyID;
    mapping(bytes32 => address) internal contentOwnerByAddress;
    mapping(string => uint) internal contentByURLs;
    uint public contentCounter;

    event contentUploaded(uint _contentID, address _owner, bytes32 _contentHash,
uint _timeStamp, bytes32 _combinedHash, string _contentURL);
    event contentDetails(address _userAddress, uint _contentID, bytes32 _contentHash,
uint timestamp);

    constructor(address _userManagementAddress) {
        userManagementContract = usermanagement(_userManagementAddress);
```

```
    }

    modifier auth(bytes32 userHash) {
        require(userManagementContract.authenticateUser(msg.sender, userHash), "You need to be registered first before uploading any content");
        _;
    }

    function uploadContent(bytes32 _contentHash, bytes32 _combinedHash, string memory _contentURL, bool _forSale, uint _price, bytes32 userHash) public auth(userHash) {
        require(contentOwnerByAddress[_contentHash] == address(0), "Content already exists and cannot be uploaded again");

        contentCounter++;
        uint cost = _forSale ? _price : 0;

        Content memory newContent = Content({
            contentID: contentCounter,
            owner: msg.sender,
            timeStamp: block.timestamp,
            contentHash: _contentHash,
            combinedHash: _combinedHash,
            contentURL: _contentURL,
            forSale: _forSale,
            price: cost
        });

        contentIDsbyAddress[msg.sender].push(contentCounter);
        contentbyID[contentCounter] = newContent;
        contentOwnerByAddress[_contentHash] = msg.sender;
        contentByURLs[_contentURL] = contentCounter;
```

```
        emit contentUploaded(contentCounter, msg.sender, _contentHash, block.timestamp, _combinedHash, _contentURL);
    }

    function getContentOfOwner(bytes32 userHash) public auth(userHash) view returns (Content[] memory) {
        uint[] memory userContentIDs = contentIDsbyAddress[msg.sender];
        require(userContentIDs.length > 0, "There isn't any content uploaded yet");

        Content[] memory userContents = new Content[](userContentIDs.length);
        for (uint i = 0; i < userContentIDs.length; i++) {
            userContents[i] = contentbyID[userContentIDs[i]];
        }
        return userContents;
    }

    function getContentByHash(bytes32 _contentHash, bytes32 userHash) public auth(userHash) view returns (Content memory) {
        address owner = contentOwnerByAddress[_contentHash];
        require(owner != address(0), "Content does not exist");

        uint[] memory ownerContentIDs = contentIDsbyAddress[owner];
        for (uint i = 0; i < ownerContentIDs.length; i++) {
            Content memory content = contentbyID[ownerContentIDs[i]];
            if (content.contentHash == _contentHash) {
                return content;
            }
        }

        revert("Content not found");
    }

    function getContentByURLs(string memory _contentURL) public  view returns (Content memory) {
```

```
        uint contentID = contentByURLs[_contentURL];
        require(contentID != 0, "Content does not exist");
        return contentbyID[contentID];
    }

    function getContentIDsbyAddress(address _owner, bytes32 userHash) public
auth(userHash) view returns (uint[] memory) {
        return contentIDsbyAddress[_owner];
    }
     function getContentByID(uint _contentID) public view returns (Content memory)
{
        return contentbyID[_contentID];
    }
     function getContentByurls(string memory _contenturl) public view returns (uint) {
        return contentByURLs[_contenturl];
    }

    function getContentidbyaddress(address _owner)public view returns(uint[]
memory){
        return contentIDsbyAddress[_owner];
    }
}
```

Royality Management Smart Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

import "./contentManagement.sol";

contract RoyaltyManagement {

    struct Royalty {
```

```
        address owner;
        uint contentID;
        address[] royaltyBuyer;
        uint purchasedTimestamp;
        bool listed;
    }
     usermanagement private userManagementContract;
    ContentManagement private contentmanagement;

    constructor(address _contentmanagement, address _userMAddr) {
        contentmanagement = ContentManagement(_contentmanagement);
        userManagementContract = usermanagement(_userMAddr);
    }

    modifier auth(bytes32 _userhash){

require(userManagementContract.authenticateUser(msg.sender,_userhash),"This
wallet address linked with other username");
        _;
    }
    mapping(uint => Royalty) internal royalties;
    mapping(address => uint256[]) internal addressToInt;
    mapping(uint => mapping(address => bool)) internal hasPurchased;

    uint royaltyBoughtCounter;

    event royaltyListed(uint contentID);
    event royaltedPurchased(uint contentID, address royaltyBuyer, uint price, uint
_timeStamp);

    function ListRoyalty(uint _contentID,bytes32 _userhash) public auth(_userhash) {
        ContentManagement.Content memory content =
contentmanagement.getContentByID(_contentID);
        Royalty memory royal=royalties[_contentID];
```

```
        require(content.owner==msg.sender, "You are not the owner of this content");
        require(!royal.listed,"content already listed");

        royalties[_contentID] = Royalty({
            owner: msg.sender,
            contentID: _contentID,
            royaltyBuyer: new address[](0) ,
            purchasedTimestamp: 0,
            listed:true
        });
        emit royaltyListed(_contentID);
    }

function buyRoyaltyByURL(string memory _contentURL,bytes32 _userhash) public auth(_userhash) payable {
    uint contentid = contentmanagement.getContentByurls(_contentURL);
    Royalty memory royal = royalties[contentid];
    require(contentid != 0, "content does not exist");

    ContentManagement.Content memory cnt = contentmanagement.getContentByID(contentid);
    require(cnt.forSale, "The content is not for sale");
    require(!hasPurchased[contentid][msg.sender], "You already purchased the royalty of this content");
    require(msg.value == cnt.price, "Incorrect value sent");
    require(royal.listed, "Content is not listed for sale");
    require(cnt.owner != msg.sender, "Being owner, you cannot purchase your own content");

    Royalty storage royalty = royalties[contentid];
    royalty.royaltyBuyer.push(msg.sender);  // Add buyer to royaltyBuyer list
    royalty.purchasedTimestamp = block.timestamp;  // Set timestamp of purchase

    // Transfer payment to the owner
```

```
    payable(royalty.owner).transfer(msg.value);

    // Add this content ID to the user's purchase record
    addressToInt[msg.sender].push(contentid);

    // Mark as purchased for this user
    hasPurchased[contentid][msg.sender] = true;

    emit royaltedPurchased(contentid, msg.sender, cnt.price, block.timestamp);
}


    function ownerDetailsWithRoyalties(bytes32 _userhash) public auth(_userhash) view returns (uint[] memory, address[] memory, uint[] memory, uint[] memory) {
        uint[] memory contentids = contentmanagement.getContentidbyaddress(msg.sender);
        uint totalLength = 0;

        for (uint i = 0; i < contentids.length; i++) {
            totalLength += royalties[contentids[i]].royaltyBuyer.length;
        }

        uint[] memory royaltyPrices = new uint[](totalLength);
        uint[] memory timestamps = new uint[](totalLength);
        address[] memory buyerAddresses = new address[](totalLength);
        uint[] memory contentIDs = new uint[](totalLength);

        uint counter = 0;
        for (uint i = 0; i < contentids.length; i++) {
            uint CntID = contentids[i];
            Royalty memory royalty = royalties[CntID];
            ContentManagement.Content memory cnt = contentmanagement.getContentByID(CntID);
```

```
            for (uint j = 0; j < royalty.royaltyBuyer.length; j++) {
                if (hasPurchased[CntID][royalty.royaltyBuyer[j]]) {
                    royaltyPrices[counter] = cnt.price;
                    timestamps[counter] = royalty.purchasedTimestamp;
                    buyerAddresses[counter] = royalty.royaltyBuyer[j];
                    contentIDs[counter] = CntID;
                    counter++;
                }
            }
        }

        return (contentIDs, buyerAddresses, timestamps, royaltyPrices);
    }

function royaltiesBoughtByUser() public view returns (address[] memory, uint[] memory, uint[] memory, uint[] memory) {
    uint[] memory purchasedContentIDs = addressToInt[msg.sender];  // Retrieve the list of content IDs the user has bought

    uint purchasedCount = purchasedContentIDs.length;
    address[] memory owners = new address[](purchasedCount);
    uint[] memory contentIDs = new uint[](purchasedCount);
    uint[] memory prices = new uint[](purchasedCount);
    uint[] memory timestamps = new uint[](purchasedCount);

    uint counter = 0;

    // Loop through the purchased content IDs for the user
    for (uint i = 0; i < purchasedCount; i++) {
        uint contentID = purchasedContentIDs[i];
        if (hasPurchased[contentID][msg.sender]) {
            Royalty memory royalty = royalties[contentID];
            ContentManagement.Content memory cnt = contentmanagement.getContentByID(contentID);
```

```
            owners[counter] = royalty.owner;
            contentIDs[counter] = royalty.contentID;
            prices[counter] = cnt.price;
            timestamps[counter] = royalty.purchasedTimestamp;
            counter++;
        }
    }

    return (owners, contentIDs, prices, timestamps);
}
}
```

DATAMARK

ORIGINALITY REPORT

8%	4%	4%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Polytechnics Mauritius Student Paper	1%
2	www.investopedia.com Internet Source	1%
3	digitalcollection.utem.edu.my Internet Source	1%
4	Gavin Zheng, Longxiang Gao, Liqun Huang, Jian Guan. "Ethereum Smart Contract Development in Solidity", Springer Science and Business Media LLC, 2021 Publication	1%
5	dev.to Internet Source	1%
6	Submitted to University of Limerick Student Paper	1%
7	Raghavan, Vijay Anand. "Security Threats from Data Poisoning Attacks in Deep Learning Vision Systems", The George Washington University, 2023 Publication	<1%

8	Submitted to The University of Manchester Student Paper	<1 %
9	Zhe Ma, Xuhesheng Chen, Tiejiang Sun, Xukang Wang, Ying Cheng Wu, Mengjie Zhou. "Blockchain-Based Zero-Trust Supply Chain Security Integrated with Deep Reinforcement Learning for Inventory Optimization", Future Internet, 2024 Publication	<1 %
10	Submitted to University of Birmingham Student Paper	<1 %
11	Submitted to Stuyvesant High School Student Paper	<1 %
12	Submitted to Visvesvaraya Technological University Student Paper	<1 %
13	www.studymode.com Internet Source	<1 %
14	Submitted to University of Greenwich Student Paper	<1 %
15	publications.ossrea.net Internet Source	<1 %
16	Submitted to Postgraduate Institute of Medicine Student Paper	<1 %

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?

The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?

Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.