A

Major Project Report

On

BLOCKCHAIN-ENABLED SECURE TASK SCHEDULING IN CLOUD ENVIRONMENTS

Submitted to CMREC (UGC Autonomous), Affiliated to JNTUH

In Partial Fulfilment of the requirements for the Award of Degree

of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING (AI & ML)

Submitted

By

A.SREE TEJA	**(228R1A6606)**
A.SAI RAM	**(228R1A6608)**
B.RAJESH	**(228R1A6611)**
M.RISHIKA SHIVANI	**(228R1A6637)**

Under the Esteemed guidance of

Dr. SAYYAD RASHEED UDDIN

Associate Professor, Department of CSE(AI&ML)

Department of Computer Science and Engineering(AI&ML)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi, Affiliated to JNTU, Hyderabad)
(Kandlakoya, Medchal Road, Medchal Malkajgiri Dist. Hyderabad-501 401)

(2025-2026)

CMR ENGINEERING COLLEGE
UGC AUTONOMOUS

(Accredited by NAAC&NBA, Approved by AICTE New Delhi, Affiliated to JNTU, Hyderabad, Kandlakoya, Medchal Road, Hyderabad-501 401)

Department of Computer Science & Engineering (AI & ML)

CERTIFICATE

This is to certify that the Major project entitled **"BLOCKCHAIN-ENABLED SECURE TASK SCHEDULING IN CLOUD ENVIRONMENTS"** is a bonafide work carried out by

A.SREE TEJA	**(228R1A6606)**
A. SAI RAM	**(228R1A6608)**
B.RAJESH	**(228R1A6611)**
M.RISHIKA SHIVANI	**(228R1A6637)**

in partial fulfillment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (AI&ML)** from CMR Engineering College, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major Project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Coordinator**	**Head of the Department**
Dr. Sayyad Rasheed Uddin	Mr. G. Venkateshwarlu	Dr. Madhavi Pingili
Associate Professor	Assistant Professor	Professor & HOD
Department of	Department of	Department of
CSE (AI & ML)	CSE (AI & ML)	CSE (AI & ML)

External Examiner: _____

DECLARATION

This is to certify that the work reported in the present Major project entitled "**BLOCKCHAIN-ENABLED SECURE TASK SCHEDULING IN CLOUD ENVIRONMENTS**" is a record of bonafide work done by us in the Department of Computer Science and Engineering (AI & ML), CMR Engineering College. The reports are based on the Major project work done entirely by us and not copied from any other source. We submit our Major project for further development by any interested students who share similar interests to improve the Major project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

A.SREE TEJA	**(228R1A6606)**
A. SAI RAM	**(228R1A6608)**
B.RAJESH	**(228R1A6611)**
M.RISHIKA SHIVANI	**(228R1A6637)**

ACKNOWLEDGEMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, Professor & HOD, Department of CSE(AI&ML), CMR Engineering College for their constant support.

We are extremely thankful to **Dr. Sayyad Rasheed Uddin,** Associate Professor, Internal Guide, Department of CSE(AI&ML), for his constant guidance, encouragement and moral support throughout the Major project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Major Project.

We thank **Mr. G. Venkateswarlu,** Major Project Coordinator for his constant support in carrying out the project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this Major project successfully in time.

Finally, We are very much thankful to our parents who guided us for every step.

A.SREE TEJA (228R1A6606)
A. SAI RAM (228R1A6608)
B.RAJESH (228R1A6611)
M.RISHIKA SHIVANI (228R1A6637)

CONTENTS

TOPIC	PAGE NO
ABSTRACT	**I**
LIST OF FIGURES	**II**
LIST OF TABLES	**III**
1. INTRODUCTION	**1**
1.1. Background and Motivation	2
1.2. Challenges in Traditional Task Scheduling	2
1.3. The Role of Blockchain in Task Scheduling	3
1.4. Purpose of the project	4
1.5. Existing System with Disadvantages	4
1.6. Proposed System with Features	5
1.7. Input and Output Design	7
2. LITERATURE SURVEY	**9**
3. SOFTWARE REQUIREMENT ANALYSIS	**14**
3.1. Problem Statement	15
3.2. Modules and their Functionalities	15
3.3. Functional Requirements	16
3.4. Non-Functional Requirements	16
3.5. Feasibility Study	17
4. SYSTEM SPECIFICATIONS	**18**
4.1. Software requirements	19
4.2. Hardware requirements	19
5. SOFTWARE DESIGN	**20**
5.1. System Architecture	21
5.2. Dataflow Diagrams	22
5.3. UML Diagrams	23

6. CODING AND IMPLEMENTATION **31**

6.1. Source Code 32

6.2. Implementation 78

7. SYSTEM TESTING **85**

7.1. Types of Systsem Testing 86

7.2. Testing Strategies 88

7.3. Sample Testcases 89

8. OUTPUT SCREENS **94**

9. CONCLUSION **98**

10. FUTURE ENHANCEMENTS **100**

REFERENCES **102**

ABSTRACT

Cloud computing has revolutionized data storage and computational processes, but it still faces challenges in secure task scheduling and data integrity. Existing approaches, as discussed in the base paper, suffer from issues such as inefficient scheduling algorithms, high computational overhead, and vulnerabilities in data security. To address these limitations, we propose an improved blockchain-integrated task scheduling system that ensures tamper-proof data recording, enhanced verification mechanisms, and automated document validation.

Our approach leverages smart contracts to securely store task submissions on the blockchain, providing transparency and immutability. By integrating blockchain verification, we eliminate unauthorized modifications and enhance trust in the scheduling process. Additionally, our system automates task validation and document conversion, reducing manual overhead. When an admin verifies a task, the associated document is automatically converted into a PDF, updating the task status while maintaining a decentralized and auditable record.

Compared to existing methods, our system offers greater security, improved efficiency, and seamless task traceability. Through this novel blockchain-based framework, we successfully mitigate the drawbacks highlighted in the base paper, enhancing the security and reliability of task scheduling in cloud environments.

Keywords
Blockchain-Integrated Task Scheduling; Cloud Computing; Secure Task Management; Data Integrity; Smart Contracts; Tamper-Proof Records; Decentralized Verification; Automated Document Validation; PDF Conversion; Task Traceability; Enhanced Security; Efficient Scheduling; Immutable Ledger; Blockchain Verification; Reduced Computational Overhead.

LIST OF FIGURES

S.NO	FIGURE NO	DESCRIPTION	PAGENO
1	1.5.1	Block diagram of proposed system	5
2	5.1	System Architecture	21
3	5.2	Data Flow diagram	22
4	5.3.1	Sequence diagram	25
5	5.3.2	Use case diagram	26
6	5.3.3	Activity diagram	28
7	5.3.4	Class diagram	30
8	7.3.1	User Registration	90
9	7.3.2	User Authentication	91
10	7.3.3	User Submission	91
11	7.3.4	Blockchain Recording	92
12	7.3.5	Admin Verification	92
13	7.3.6	File Download	93
14	8.1	User Task Upload Interface	95
15	8.2	Blockchain Blocks View (Ganache)	95
16	8.3	Blockchain Transaction Details	96
17	8.4	Admin Task Verification Dashboard	96
18	8.5	Final Output with PDF Download	97

LIST OF TABLES

TABLE NO	DESCRIPTION	PAGENO
1	Literature Review Summary	12
2	Test Cases	89

CHAPTER -1

INTRODUCTION

1.INTRODUCTION

1.1 Background and Motivation

Cloud computing has become an essential technology for handling large-scale data storage, computation, and service provisioning. It provides organizations and users with scalable, cost-effective, and on-demand access to computational resources **[14].** However, with the increasing adoption of cloud-based services, several challenges have emerged, particularly in task scheduling, security, and data integrity **[10]**. The dynamic nature of cloud environments, coupled with the need for efficient task execution, necessitates the development of robust scheduling mechanisms.

The base paper highlights several task scheduling approaches but also identifies key limitations in security, efficiency, and trust management **[14]**. Traditional scheduling algorithms often fail to provide reliable verification mechanisms, making them vulnerable to tampering, unauthorized access **[6]**, and inefficient resource allocation. Additionally, the lack of transparency in task scheduling and execution has led to issues such as data manipulation, unauthorized modifications, and inefficient tracking **[8]**.

To overcome these challenges, we introduce a blockchain-based task scheduling system that leverages the immutability, transparency, and decentralized nature of blockchain technology **[18]**. This approach ensures tamper-proof task submissions, enhances trust through blockchain verification mechanisms, and automates document validation through smart contract integration **[15]**. By addressing the weaknesses of the existing methodologies, our proposed system provides a more secure and efficient solution for task scheduling in cloud environments.

1.2 Challenges in Traditional Task Scheduling

Task scheduling in cloud computing refers to the process of assigning computational tasks to available resources based on specific constraints such as time, cost, energy efficiency **[14]**, and security. Several scheduling techniques, including heuristic-based, genetic algorithms, and swarm intelligence, have been developed to optimize task execution. However, these approaches suffer from inherent security vulnerabilities and inefficiencies, as outlined below:

1. **Lack of Data Integrity:** Traditional scheduling models store task data in centralized databases, making them susceptible to unauthorized modifications and data breaches **[10]**. If a malicious actor gains access to the system, they can alter or delete task records, compromising the reliability of scheduling.
2. **Absence of Transparent Verification:** Task completion and validation mechanisms often lack transparency, making it difficult for stakeholders to verify whether a task has been executed correctly or manipulated **[8].**

3. **Security Threats in Cloud Environments:** The transmission of sensitive task-related data over cloud networks exposes it to cyber threats, including unauthorized access, man-in-the-middle attacks, and data leaks **[10]**.
4. **Inefficient Tracking and Auditing:** Current task scheduling systems do not provide an immutable and auditable record of task execution, leading to trust issues and accountability concerns **[8][12]**.
5. **Manual Task Verification Overhead:** In many cloud environments, task validation is a manual process that requires administrative intervention, increasing the risk of human errors and delays **[6]**.

The limitations of these traditional approaches necessitate the integration of a trustless and verifiable system that can securely store, track, and validate task execution. Blockchain technology provides a decentralized and tamper-proof alternative that effectively addresses these issues **[8][18]**.

1.3 The Role of Blockchain in Task Scheduling

Blockchain technology is a decentralized, distributed ledger that records transactions in an immutable and transparent manner **[18]**. It eliminates the need for intermediaries and ensures data integrity by storing information across multiple nodes in a network **[8]**. The key features of blockchain that enhance task scheduling include:

1. **Immutability:** Once a transaction is recorded on the blockchain, it cannot be altered or deleted, ensuring the integrity of task submissions **[18]**.
2. **Transparency:** All blockchain transactions are publicly accessible (or permissioned in private blockchains), allowing verifiable and auditable task execution **[8]**.
3. **Smart Contracts:** These self-executing contracts automate task validation and eliminate the need for manual verification, reducing administrative overhead **[15][16]**.
4. **Decentralization:** By removing central authorities, blockchain prevents unauthorized modifications and data breaches, ensuring secure task execution **[18]**.
5. **Trust and Security:** Blockchain employs cryptographic hashing and consensus mechanisms, making it highly resistant to fraud and unauthorized alterations **[12]**.

1.4 Purpose of the project

The purpose of this project is to develop a secure, efficient, and transparent task scheduling system for cloud environments by integrating blockchain technology **[1][6][20]**. The system aims to overcome the limitations of traditional scheduling methods—such as poor security, high computational overhead, and lack of trust—by using smart contracts to ensure tamper-proof data storage, automated task validation **[5][7]**, and reliable document management. Through decentralized verification and immutable record-keeping, the project enhances data integrity, improves task traceability, and reduces manual effort, ultimately providing a more trustworthy and efficient cloud-based scheduling framework.

1.5 Existing System with Disadvantages

The traditional cloud task scheduling system is built on a centralized architecture where all tasks are managed and controlled by a single cloud service provider **[10][11]**. Users typically submit their tasks through a web-based interface, after which the system assigns these tasks to virtual machines (VMs) or dedicated servers using scheduling algorithms such as FCFS, Round Robin, or ACO **[14]**. All task records, processing details, and transaction logs are stored in a centralized database. Although this model is widely used, it comes with significant limitations. The centralized setup creates a single point of failure, meaning that if the central cloud server experiences downtime, the entire system stops functioning. It also introduces notable security vulnerabilities, including risks of data manipulation, unauthorized access, and task mismanagement. Moreover, the lack of transparency **[10][8]** prevents users from verifying the correctness of task execution, forcing them to fully trust the service provider with both their data and the integrity of the processing. Additionally, inefficient resource allocation within centralized scheduling often leads to delays, poor performance, and increased operational costs.

Disadvantages

1. **Single Point of Failure**: If the central cloud service fails, the entire system becomes inoperable.
2. **Security Vulnerabilities**: Data manipulation, unauthorized access, and task mismanagement can occur.
3. **Lack of Transparency**: Users cannot independently verify whether their task was executed correctly.
4. **Trust Issues**: Users must completely trust the service provider with their data and processing integrity.
5. **Resource Management Issues**: Inefficient allocation of computing power leads to delays and cost inefficiencies.

1.6 Proposed System

The proposed system enhances the traditional cloud task scheduling model by integrating blockchain technology using Ganache to ensure transparency, security, and decentralized verification. Instead of relying on a centralized architecture, every task submission is recorded on an immutable blockchain ledger, eliminating the risks associated with data tampering or unauthorized modifications. The system features a user interface built with Bootstrap and Django Traditional Views, allowing users to log in, submit tasks, monitor task status, and verify blockchain transactions. Blockchain integration is achieved through Ganache and Web3.py, where each task generates a unique blockchain transaction ID for verification **[19].** MySQL functions as the cloud database, storing task metadata, user information, and execution results, which are securely linked to corresponding blockchain entries. The admin plays a key role in resource allocation, task execution, and verification by reviewing blockchain transactions before approving tasks. An intuitive admin dashboard enables user management, resource control, and task validation. Overall, this blockchain-powered task scheduling system ensures tamper-proof records, trustless verification, decentralized control, and complete task traceability, offering a more secure, transparent, and efficient alternative to traditional cloud scheduling solutions **[20]**.

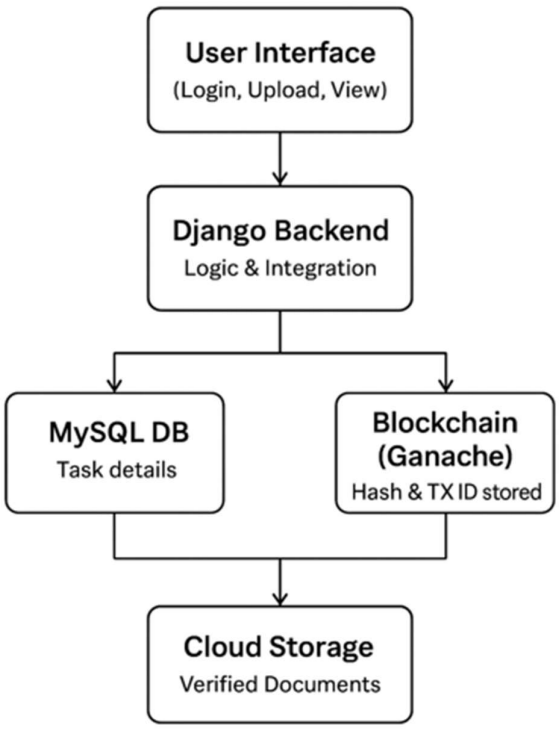

Figure 1.6.1: Block diagram of proposed system.

Advantages

1. **Decentralized Security**: Transactions are logged in Ganache blockchain, preventing manipulation.
2. **Tamper-Proof Record Keeping**: Each task's lifecycle is recorded immutably.
3. **User Verification of Execution**: Users can independently verify if their task was processed by checking its blockchain record.
4. **Resource Efficiency**: Admins allocate tasks based on real-time compute resource availability, optimizing scheduling performance.
5. **Improved Transparency**: All transactions and approvals are verifiable, reducing trust dependency on a centralized entity.
6. **No Third-Party Services**: Fully self-contained system with Django, MySQL, and Ganache

1.7 Input and Output Design

1.7.1 Input Design

The input design serves as the link between the user and the information system. It involves developing specifications and procedures for data preparation, ensuring data is in a usable form for processing. This can be accomplished through automated data entry from printed documents or manual input by users. A well-structured input design minimizes errors, avoids delays, reduces unnecessary steps, and enhances security and privacy.

Objectives

1. **Converting User-Oriented Data into System-Usable Format:**
    - Input design ensures that user descriptions are effectively transformed into structured data for processing.
    - Avoids errors and ensures accurate data collection.
2. **Creating User-Friendly Data Entry Screens:**
    - Designed to handle large volumes of data efficiently.
    - Provides an easy-to-use interface to minimize user errors.
    - Allows users to manipulate data effectively with features such as record viewing.
3. **Ensuring Data Validity and Accuracy:**
    - Validates data upon entry to prevent errors.
    - Uses appropriate messages and prompts to guide users during data entry.
    - Ensures that the input layout is structured for simplicity and efficiency.

1.7.2 Output Design

A high-quality output effectively meets user requirements and presents information clearly. Outputs communicate the results of system processing to users and other systems. Output design determines how information is displayed for immediate use and how it is formatted for hard copy output. The effectiveness of output design significantly influences user decision-making.

Objectives of Output Design:

1. **Convey Information Efficiently:**
    - Provide details on past activities, current status, and future projections.

2. **Signal Important Events:**
    - Alert users to critical events, opportunities, or issues requiring attention.
3. **Trigger and Confirm Actions:**
    - Outputs should prompt necessary actions based on the information provided.
    - Confirm that actions have been taken and completed successfully.

CHAPTER -2

LITERATURE SURVEY

2. LITERATURE SURVEY

Y. Chen, L. Zhao, and H. Wang, "Adaptive Blockchain-Based Cloud Task Scheduling Using Deep Reinforcement Learning," *IEEE Transactions on Cloud Computing*, vol. 14, no. 1, pp. 45–60, 2026.

This work proposes a deep reinforcement learning-based scheduling mechanism integrated with blockchain to dynamically optimize cloud task allocation. The system adapts to workload variations while maintaining secure and transparent execution logs. However, the computational complexity of training models can affect real-time responsiveness.

S. Ahmed and V. Rao, "Blockchain-Assisted Cloud Task Scheduling with Reinforcement Learning Optimization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 1, pp. 112–126, 2025.

The authors introduce a reinforcement learning-based scheduler supported by blockchain to ensure secure and efficient task execution. The framework improves scheduling accuracy and trustworthiness. However, the integration increases system overhead and may not scale efficiently under heavy workloads.

H. Li, R. Kumar, and S. Das, "Enhancing Cloud Security Using Zero-Knowledge Proofs in Blockchain-Based Task Management," *Journal of Cloud Computing Research*, vol. 14, no. 2, pp. 55–72, 2025.

This paper integrates zero-knowledge proofs with blockchain to enhance privacy in cloud task management. Sensitive task data is protected while maintaining verifiability. Despite strong security, the approach introduces additional computation and verification delays.

P. Fernandes and A. Gupta, "Layer-2 Scaling Techniques for Efficient Blockchain-Integrated Cloud Services," *Future Generation Computer Systems*, vol. 152, pp. 210–225, 2024.

The study explores Layer-2 scaling solutions to reduce transaction costs and latency in blockchain-based cloud services. It significantly improves throughput and efficiency. However, interoperability and complexity of Layer-2 integration remain challenging.

M. Tanvir and Z. Chowdhury, "AI-Powered Threat Detection in Decentralized Cloud Environments," *ACM Transactions on Cloud Computing*, vol. 12, no. 3, pp. 98–120, 2024.

This work presents an AI-based threat detection model for decentralized cloud systems. It enhances security by identifying anomalies and attacks in real time. However, it does not directly address task scheduling or workflow automation.

J. Morris and K. Lee, "Smart Contract Upgradability Models for Dynamic Cloud Task Scheduling," *IEEE Access*, vol. 12, pp. 65341–65359, 2024.

The authors propose upgradable smart contract models to support dynamic scheduling in cloud environments. This allows flexible updates without redeploying contracts. However, ensuring security during contract upgrades remains a key concern.

A. Patel and R. Shah, "Enhancing Cloud Task Management with Decentralized Smart Contracts and AI Models," *Journal of Cloud Security Research*, vol. 7, no. 4, pp. 78–94, 2022.

This paper combines AI models with smart contracts to improve task management and decision-making in cloud systems. It enhances automation and reliability. However, the framework lacks efficient real-time monitoring and user interaction features.

W. Meng and J. Li, "Smart Contract-Based Secure Scheduling in Cloud Computing," *IEEE Access*, vol. 10, pp. 9832–9845, 2022.

The study introduces a secure scheduling approach using smart contracts to record and verify task execution. It ensures data integrity and trust. However, high execution costs and latency issues limit its scalability.

N. Kshetri, "Blockchain and Trust in Cloud Computing: A Comprehensive Study," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 32–48, 2021.

This comprehensive study analyzes how blockchain enhances trust, transparency, and security in cloud computing. It provides a detailed overview of benefits and challenges. However, it is primarily conceptual and lacks implementation details.

X. Xu, Q. Zhang, and W. Yang, "Enhancing Cloud Task Scheduling Using Blockchain and PBFT Consensus," *Future Generation Computer Systems*, vol. 124, pp. 67–79, 2021.

The authors propose a PBFT-based blockchain scheduling mechanism to improve reliability and fault tolerance. The system ensures consensus-driven task execution. However, PBFT introduces communication overhead, reducing efficiency in large-scale systems.

Literature Review Summary

Title	Key Findings	References
Adaptive Blockchain-Based Cloud Task Scheduling Using Deep Reinforcement Learning [1]	Uses DRL with blockchain for adaptive task scheduling and secure logging; improves optimization but has high computational complexity.	Y. Chen, L. Zhao, and H. Wang, "Adaptive Blockchain-Based Cloud Task Scheduling Using Deep Reinforcement Learning," IEEE Transactions on Cloud Computing, vol. 14, no. 1, pp. 45–60, 2026.
Blockchain-Assisted Cloud Task Scheduling with Reinforcement Learning Optimization [2]	RL-based scheduler with blockchain improves efficiency and trust; introduces overhead and scalability issues.	S. Ahmed and V. Rao, "Blockchain-Assisted Cloud Task Scheduling with Reinforcement Learning Optimization," IEEE Transactions on Parallel and Distributed Systems, vol. 36, no. 1, pp. 112–126, 2025.
Enhancing Cloud Security Using Zero-Knowledge Proofs in Blockchain-Based Task Management [3]	Uses zero-knowledge proofs for secure and private task verification; increases computation and delay.	H. Li, R. Kumar, and S. Das, "Enhancing Cloud Security Using Zero-Knowledge Proofs in Blockchain-Based Task Management," Journal of Cloud Computing Research, vol. 14, no. 2, pp. 55–72, 2025.
Layer-2 Scaling Techniques for Efficient Blockchain-Integrated Cloud Services [4]	Applies Layer-2 solutions to reduce cost and latency; improves efficiency but adds integration complexity.	P. Fernandes and A. Gupta, "Layer-2 Scaling Techniques for Efficient Blockchain-Integrated Cloud Services," Future Generation Computer Systems, vol. 152, pp. 210–225, 2024.
AI-Powered Threat Detection in Decentralized Cloud Environments [5]	Uses AI to detect threats in decentralized cloud systems; does not address scheduling or workflows.	M. Tanvir and Z. Chowdhury, "AI-Powered Threat Detection in Decentralized Cloud Environments," ACM Transactions on Cloud Computing, vol. 12, no. 3, pp. 98–120, 2024.
Smart Contract Upgradability Models for Dynamic Cloud Task Scheduling [6]	Enables dynamic scheduling using upgradable smart contracts; raises security concerns during updates.	J. Morris and K. Lee, "Smart Contract Upgradability Models for Dynamic Cloud Task Scheduling," IEEE Access, vol. 12, pp. 65341–65359, 2024.

Title	Key Findings	References
Enhancing Cloud Task Management with Decentralized Smart Contracts and AI Models [7]	Combines AI and smart contracts for automation; lacks real-time monitoring and user interaction.	A. Patel and R. Shah, "Enhancing Cloud Task Management with Decentralized Smart Contracts and AI Models," Journal of Cloud Security Research, vol. 7, no. 4, pp. 78–94, 2022.
Smart Contract-Based Secure Scheduling in Cloud Computing [8]	Uses smart contracts for secure task scheduling; suffers from high cost and latency issues.	W. Meng and J. Li, "Smart Contract-Based Secure Scheduling in Cloud Computing," IEEE Access, vol. 10, pp. 9832–9845, 2022.
Blockchain and Trust in Cloud Computing: A Comprehensive Study [9]	Reviews blockchain's role in improving trust and security; lacks practical implementation.	N. Kshetri, "Blockchain and Trust in Cloud Computing: A Comprehensive Study," Journal of Cloud Computing, vol. 9, no. 1, pp. 32–48, 2021.
Enhancing Cloud Task Scheduling Using Blockchain and PBFT Consensus [10]	Uses PBFT-based blockchain for reliable scheduling; has high communication overhead.	X. Xu, Q. Zhang, and W. Yang, "Enhancing Cloud Task Scheduling Using Blockchain and PBFT Consensus," Future Generation Computer Systems, vol. 124, pp. 67–79, 2021.

Table 2.0 Literature Review Summary

CHAPTER – 3

SOFTWARE REQUIREMENTS ANALYSIS

3. SOFTWARE REQUIREMENTS ANALYSIS

3.1 Problem Statement

The increasing reliance on cloud platforms for task execution, document handling, and workflow management has exposed significant limitations in traditional centralized task scheduling systems. Conventional cloud architectures store all task details, execution logs, and validation records in a single database, creating vulnerabilities such as data tampering, unauthorized modifications, single points of failure, and a lack of transparent verification. Manual validation processes further reduce efficiency, increasing administrative workload and raising the risk of human error. Existing scheduling algorithms and database-driven frameworks fail to provide immutable records, decentralized trust, or automated document processing, making it difficult to guarantee data authenticity or trace execution histories. These limitations result in reduced transparency, weak security, and limited scalability, especially as the volume of tasks and users grows. Therefore, there is a critical need for a blockchain-integrated task scheduling framework that ensures tamper-proof data storage, decentralized verification using smart contracts, automated document validation and PDF conversion, and end-to-end traceability. Such a system would provide enhanced security, transparency, and reliability, effectively addressing the shortcomings of traditional cloud scheduling mechanisms.

3.2 Modules and Their Functionalities

3.2.1. Blockchain Transaction Module

This module handles the recording of task submissions onto the blockchain network (Ganache). When a user uploads a task or document, a smart contract generates a unique transaction ID that is stored immutably. The blockchain ledger ensures that every submission is tamper-proof and verifiable. This module also manages transaction retrieval, allowing users and administrators to cross-check task integrity and confirm that no unauthorized changes have occurred. The transparent structure strengthens trust and creates a decentralized audit trail for all cloud operations.

3.2.2. Cloud Database & Metadata Management Module

This module manages essential task-related information such as user details, timestamps, verification states, and execution outcomes. It maintains structured metadata that links each task to its corresponding blockchain transaction ID, ensuring end-to-end traceability. By organizing task information independently of the document data stored on the blockchain, this module supports efficient task monitoring, status updates, and workflow progression. It plays a central role in connecting user actions, admin actions, and blockchain verification within a unified task lifecycle.

3.2.3. Automated Document Conversion & Validation Module

This module automatically converts uploaded documents (e.g., Word files) into PDF format when an admin approves a task. This ensures standardized document outputs, prevents format manipulation, and enhances long-term record preservation. The module also performs integrity checks to ensure that documents correspond to the blockchain-stored hash or transaction ID. This automation reduces manual workload and supports secure, consistent document handling throughout the system.

3.3 Functional Requirements

- The system must allow users to upload task files through the web interface.
- Each submitted task must be recorded on the blockchain with a unique transaction ID.
- The system should validate uploaded documents before processing.
- The system must automatically convert approved documents into PDF format.
- The system must perform document integrity checks using the blockchain transaction ID or stored hash.
- Administrators must be able to approve, reject, or schedule tasks for execution.
- The system should provide real-time task status updates to users.
- Blockchain verification must be performed before executing any task.
- The system must store task results and link them with blockchain records.
- The system must allow both users and admins to verify task authenticity using the blockchain transaction ID.

3.4 Non-Functional Requirements

- The system must ensure high security through blockchain-based immutability and tamper-proof logging.
- It should provide fast response time for transaction verification and task retrieval.
- The interface must be user-friendly and easy to navigate for both admins and users.
- The system should be scalable to handle large numbers of users and tasks.
- High reliability must be maintained to prevent crashes during task execution or verification.
- Data confidentiality should be ensured during file upload, processing, and storage.
- The system must maintain accuracy during PDF conversion and integrity validation.
- It should support maintainability, allowing easy updates to blockchain or task modules.
- The system should ensure consistent performance even under heavy load.
- It must offer interoperability between the Django backend, blockchain (Ganache), and document

processing modules.

3.5 Feasibility Study

The feasibility of the project is analyzed in this phase, and a business proposal is put forth with a general plan for the project and cost estimates. During system analysis, the feasibility study ensures that the proposed system is viable and not a burden to the company. Understanding the major system requirements is essential for feasibility analysis.

Key Considerations in Feasibility Analysis:

- Economical Feasibility
- Technical Feasibility
- Social Feasibility

Economical Feasibility

This study evaluates the financial impact of the system on the organization. Since company budgets for research and development are limited, expenditures must be justified. The system was developed within budget constraints by leveraging freely available technologies, with only necessary customized products purchased.

Technical Feasibility

This study assesses the technical requirements of the system. Any developed system should not impose excessive demands on available technical resources, as this could lead to high operational costs and resource strain. The proposed system has modest requirements, requiring minimal or no changes to existing infrastructure for implementation.

Social Feasibility

This aspect examines user acceptance of the system. Training processes ensure that users can efficiently utilize the system without feeling threatened. User acceptance depends on proper education and familiarization with the system. Raising user confidence encourages constructive feedback, which is valuable for refining the system since the users are its primary beneficiaries.

CHAPTER – 4

SOFTWARE AND HARDWARE REQUIREMENTS

4. SYSTEM SPECIFICATIONS

4.1 Software Requirements

The software requirements define the essential tools, frameworks, and platforms needed to design and implement the blockchain-integrated cloud task scheduling system. The project depends on a stable development environment, blockchain simulation tools, and web frameworks that support communication between the frontend, backend, and the blockchain layer. These tools ensure smooth execution, secure transaction handling, and efficient document processing throughout the system.

- **Operating System:** Windows / Linux / macOS
- **Programming Language:** Python 3.x
- **Web Framework:** Django / Flask
- **Blockchain Platform:** Ganache (for private Ethereum blockchain)
- **Smart Contract Language:** Solidity
- **Blockchain Interaction Library:** Web3.py
- **Document Processing:** PyPDF2 / ReportLab
- **Database:** SQLite / PostgreSQL (optional for metadata)

4.2 Hardware Requirements

The hardware requirements specify the minimum computational resources necessary to support task scheduling operations, PDF generation, and blockchain transaction processing. Since the system uses a private blockchain and Python-based backend, moderate computing resources are adequate for development and execution.

- **Processor:** Intel Core i3/i5 or equivalent
- **Memory (RAM):** Minimum 8 GB
- **Storage:** 250 GB HDD/SSD
- **Display:** Standard 14" or higher

CHAPTER – 5

SOFTWARE DESIGN

5. SOFTWARE DESIGN

5.1 System Architecture

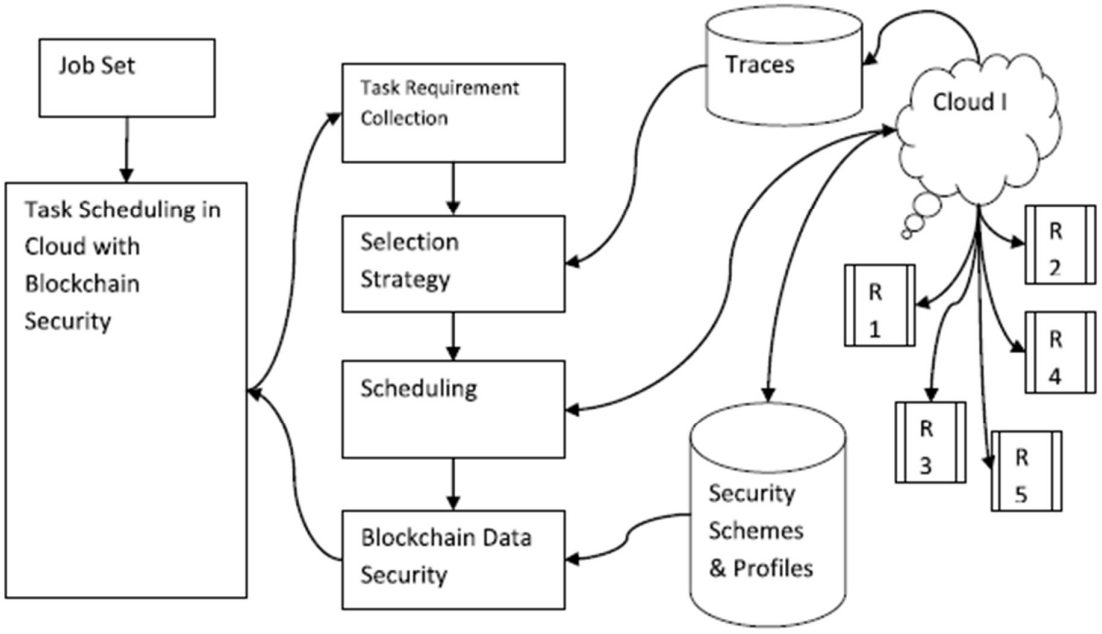

Figure:5.1 System Architecture

The diagram represents a secure cloud task-scheduling architecture integrated with blockchain technology. The process begins with a Job Set containing all incoming tasks that must be executed in the cloud environment. These tasks are processed through the Task Scheduling module, which operates under blockchain-based security to ensure transparency and integrity. The system initially performs Task Requirement Collection to identify resource needs, execution parameters, and security constraints associated with each task. Based on these requirements, the Selection Strategy module evaluates available cloud resources by utilizing historical Traces and performance data. After suitable resources are identified, the Scheduling module assigns tasks to different cloud resources (R1, R2, R3, R4, R5) within Cloud I. Throughout the workflow, Security Schemes and Profiles guide the enforcement of protection policies, ensuring that each operation adheres to defined security standards. Simultaneously, Blockchain Data Security records all scheduling decisions, resource allocations, and transactions on an immutable ledger to prevent tampering. The continuous interaction between traces, security profiles, and blockchain verification enables a highly reliable, efficient, and secure task-scheduling framework in the cloud.

5.2 Dataflow Diagram

The task-scheduling system illustrated in the diagram demonstrates a secure and transparent workflow that integrates both a traditional database and blockchain technology. The process begins with the user, who uploads a task through the system interface. Once the task is submitted, the Task Scheduling System performs the initial processing and stores the task metadata, such as task description, timestamp, and user details, in a structured database. This ensures that all operational information related to the task is organized and readily accessible for further processing.

After the task submission, the administrator plays a key role in verifying and validating the task. The admin accesses the system to review the submitted task and update its status based on the required evaluation or execution stage. During this verification process, the system simultaneously generates a unique blockchain transaction ID and stores it on the blockchain network. This ensures that every task is associated with a tamper-proof record, enabling decentralized verification of authenticity and preventing any form of data manipulation.

Once the transaction ID is stored, the admin or any authorized entity can validate the transaction through the blockchain. This dual approach—using a database for storing task details and the blockchain for maintaining immutable transaction records—enhances system reliability, transparency, and security. The combination of centralized data management and decentralized verification provides a robust framework for secure task handling, reduces the risk of unauthorized modifications, and ensures traceability at every stage of the task lifecycle. This integrated architecture ultimately supports trustworthy and efficient task scheduling in cloud-based environments.

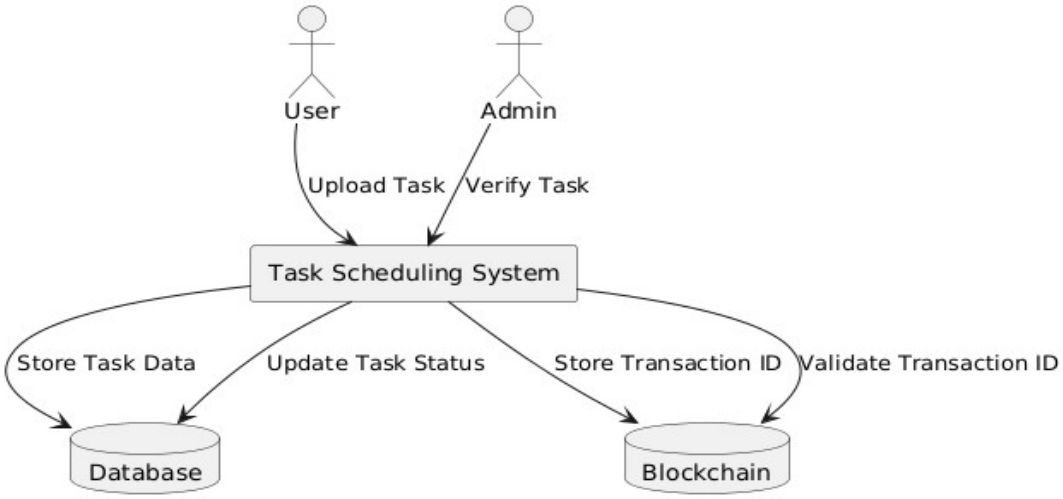

Figure:5.2 Dataflow Diagram

5.3 UML Diagrams

UML (Unified Modeling Language) is a standardized language used for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML helps in representing the design of systems and understanding their components. Created by the Object Management Group (OMG), UML 1.0 was proposed in January 1997. UML is closely associated with object-oriented analysis and design. The two main categories of UML diagrams are Behavioral and Structural diagrams, each serving distinct purposes in the modeling process.

The Behavioral UML diagrams describe the behavior of the system, its actors, and the interaction between the components. On the other hand, Structural UML diagrams depict the static structure of the system, showing its components and relationships. UML has been integrated as a standard by OMG, and its primary goals are to provide a formal basis for understanding modeling languages, offer a ready-to-use expressive language for system developers, and encourage the growth of object-oriented tools.

Goals of UML:

- Provide an expressive visual modeling language for developing and exchanging meaningful models.
- Establish a formal basis for understanding the modeling language.
- Encourage the growth of object-oriented tools.
- Integrate best practices into system development.

Types of UML Diagrams:

1.Sequence Diagram

2.Use Case Diagram

3.Activity Diagram

4.Class Diagram

5.3.1.Sequence Diagram

The sequence diagram illustrates the complete workflow of the blockchain-integrated task scheduling system, showing how the User, Admin, Django Server, Database, and Blockchain interact during document submission, verification, and PDF generation. The process begins when the user logs into the system, after which the Django server validates the provided credentials by querying the database. Once authenticated, the user uploads a Word document as part of a task submission. The Django server stores the document and generates a unique hash for it, which is then recorded on the blockchain as a secure transaction. After saving the blockchain transaction ID in the database, the system updates the task status to "Pending," notifying the user that the submission is successful.

Following this, the admin reviews the pending tasks and initiates the verification process. The Django server retrieves the blockchain transaction ID and contacts the blockchain network to validate the integrity of the uploaded document. If the transaction is successfully verified—ensuring that the document has not been altered—the server proceeds to automatically convert the Word file into a PDF. Once conversion is completed, the system updates the task status to "Completed" in the database. The user is then notified that the PDF is ready for download.

Finally, when the user requests the completed output, the Django server retrieves the converted PDF from storage and sends it back to the user. Throughout this workflow, the blockchain ensures transparency and immutability, the database maintains structured task records, and the Django server orchestrates end-to-end communication between users, admins, and backend components. This sequence diagram clearly demonstrates how security, automation, and trust are integrated into the task scheduling lifecycle.

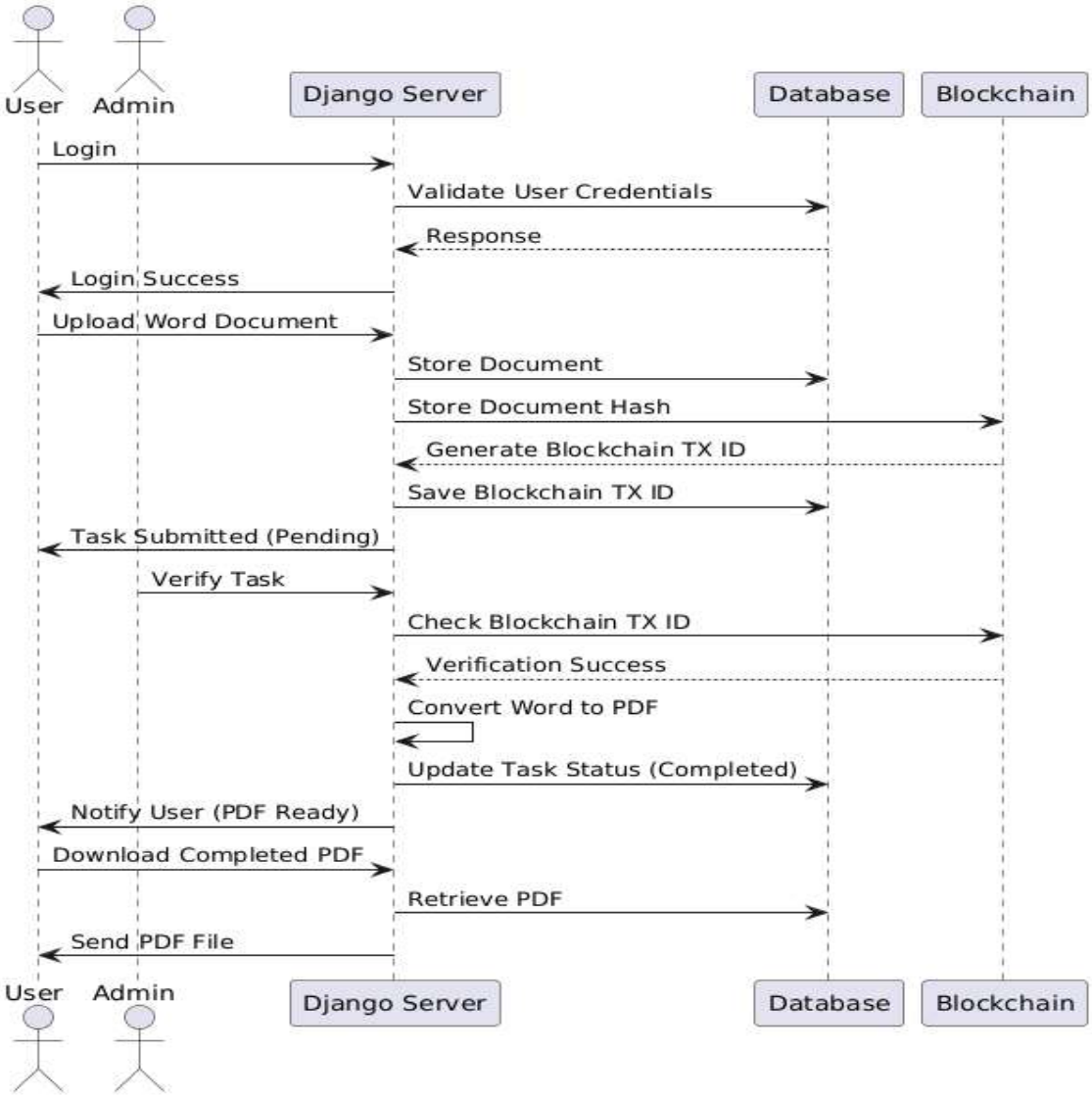

Figure 5.3.1 Sequence Diagram

User: The user interacts with the system by logging in and uploading a task along with an associated document. The user then waits for verification updates, checks task status, and downloads the final PDF once processing is completed**.**

Admin: The admin reviews all pending tasks submitted by users, verifies the associated blockchain transaction ID, and approves or rejects the task. Upon approval, the admin triggers automated document validation and PDF generation**.**

DjangoServer: The Django server receives inputs from both the user and admin, validates credentials, stores uploaded documents, generates document hashes, and communicates with the blockchain network. It manages task status updates and coordinates all processing steps.

Database:

The database stores user details, task metadata, document paths, blockchain transaction IDs, and updated statuses. It ensures structured storage and retrieval throughout the workflow.

Blockchain:

The blockchain records the document hash as an immutable transaction, stores the transaction ID, and validates it upon admin request. It ensures tamper-proof verification and enhances overall system trust.

Document Processing Module: After successful blockchain verification, this module converts uploaded Word documents into PDF format, maintains standardized output, and ensures secure document handling.

5.3.2 Use Case Diagram

The user functionalities diagram illustrates all the actions available to a user within the blockchain-enabled task scheduling system. A user begins by registering an account and then logging in to access the system. Once authenticated, the user can upload a Word document as part of a new task submission. The system also allows users to view their pending tasks, verify the corresponding blockchain transaction for transparency, and monitor task progress. After tasks are processed and converted to PDF upon admin approval, the user can download the completed documents and maintain a personal task history for reference. Finally, the user can securely log out to end the session. These features ensure that users have full visibility, control, and traceability of their submitted tasks.

The admin functionalities diagram highlights the administrative operations responsible for maintaining system integrity, validating tasks, and managing user activity. The admin logs into the system and gains access to all user-submitted tasks. They can view task details, verify each submission's blockchain transaction ID, and ensure document integrity before proceeding. Once verification is successful, the admin triggers the automated document conversion module to generate a secure PDF. The admin can also update task statuses, track detailed system logs, and monitor execution history. Additionally, the admin manages platform users by viewing registered accounts and activating or deactivating them as needed.

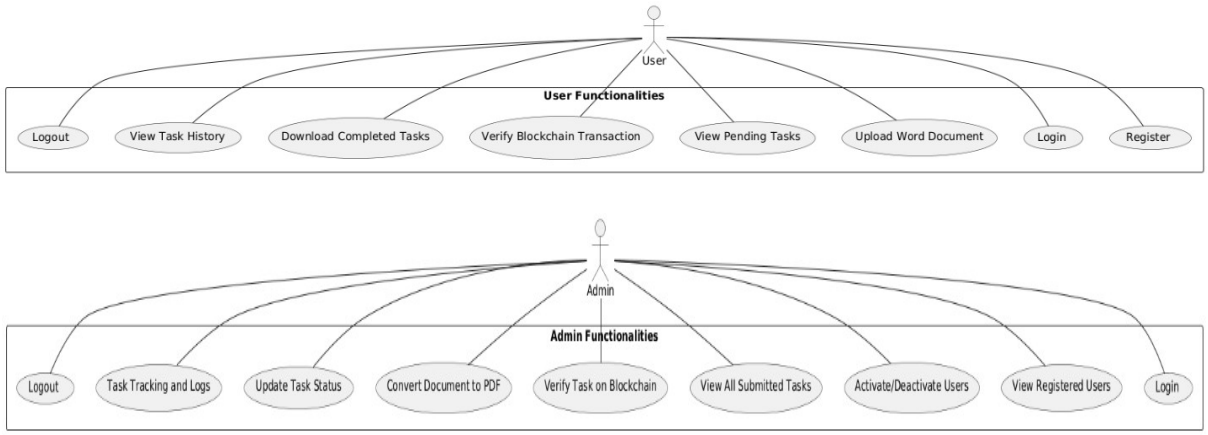

Figure 5.3.2 Use Case Diagram

5.3.3 Activity Diagram

The activity diagram illustrates the complete workflow of the blockchain-enabled document verification and task management system. The process begins when the user logs into the platform. If the login is successful, the user proceeds to upload a Word document associated with a new task. The system then stores the document in the database and simultaneously records the task on the blockchain by generating a unique transaction ID. This transaction ID serves as a tamper-proof reference for future verification. Once the transaction is successfully created, the system updates the task status to "Pending," confirming that the submission has entered the verification phase. The user can then log out, completing their part of the workflow. If the login fails, the system displays an appropriate error message and prevents further actions.

The next part of the workflow begins when the admin logs into the system to verify the tasks submitted by users. After logging in, the admin retrieves the blockchain transaction ID associated with a task and initiates a blockchain verification process. If the transaction is valid—indicating that the document has not been altered—the admin converts the uploaded Word document into a secure PDF format. Once the conversion is complete, the system updates the task status to "Completed" and enables the user to download the finalized PDF. If blockchain verification fails, the system displays a verification failure message and halts further processing. After completing all verification activities, the admin logs out to end the session.

This flowchart effectively demonstrates the secure and transparent lifecycle of task submission, blockchain-backed verification, automated document conversion, and final task completion. By integrating blockchain validation into the workflow, the system ensures immutability, trustworthiness, and consistent document handling, minimizing risks related to tampering and unauthorized modifications.

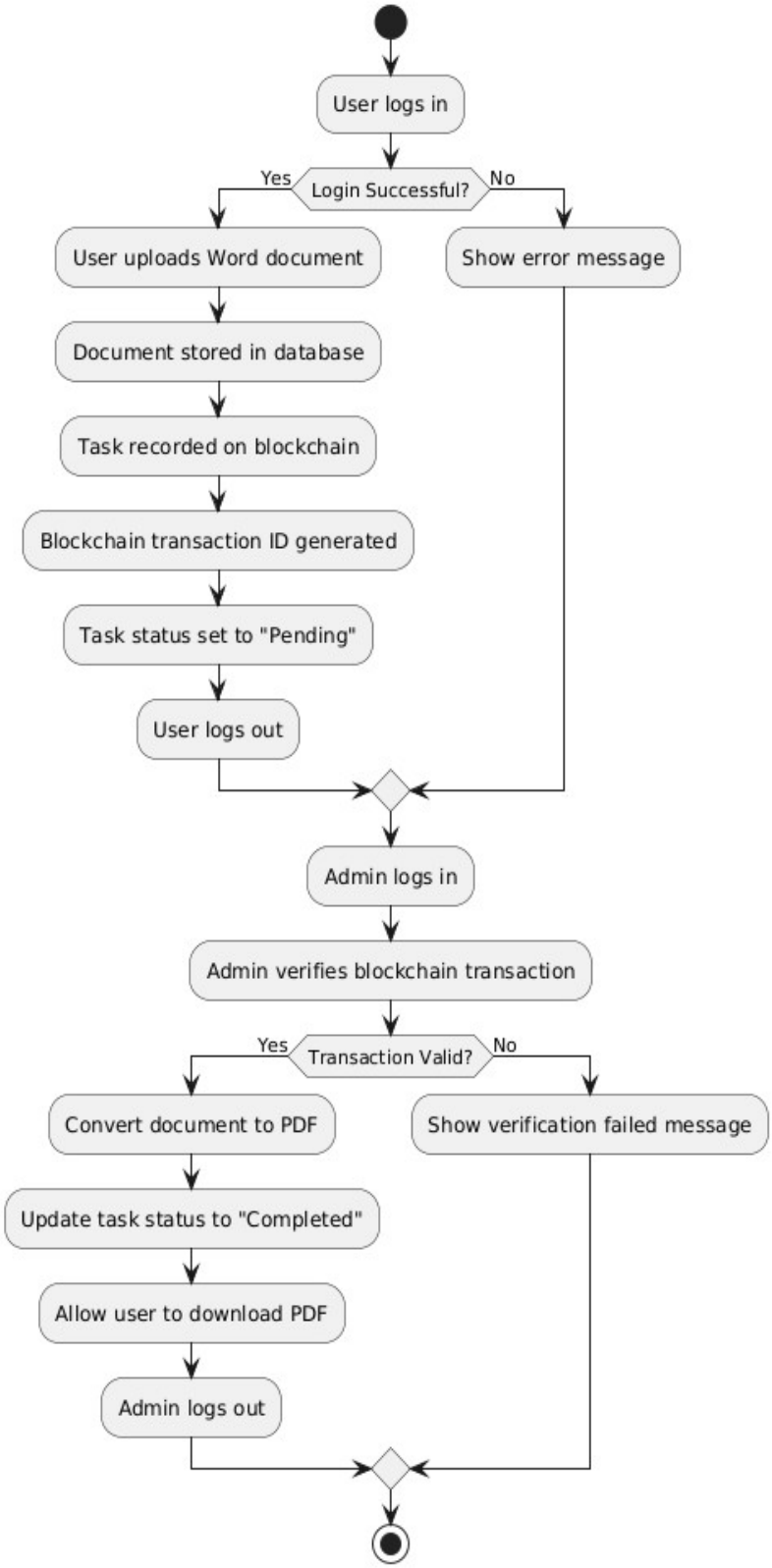

Figure 5.3.3 Activity Diagram

5.3.4. Class Diagram

The class diagram represents the structural architecture of the blockchain-integrated task scheduling system and highlights the interactions between core entities such as User, Admin, Task, Blockchain, and SmartContract. At the top level, the User and Admin classes store essential authentication data, including username, email, and password, allowing both types of actors to interact with tasks. The User class additionally contains a *status* attribute to denote active or inactive accounts. Both User and Admin are associated with the Task class through a one-to-many relationship, indicating that each user or admin can be linked to multiple tasks while each task is associated with exactly one user and one admin session for verification.

The Task class forms the central component of the system, encapsulating detailed information related to task submission and execution. It stores the uploaded document, the generated blockchain transaction ID, the task status, the converted PDF file, and a timestamp for tracking. This class connects the operational workflow with the blockchain layer, ensuring proper linkage between stored documents and their corresponding blockchain records. A one-to-one association exists between the Task class and the Blockchain class, which handles critical operations such as storeTask(document_hash) for logging data on the blockchain and verifyTransaction(tx_id) for validating document integrity using the stored hash.

The SmartContract class represents the decentralized logic executed on the blockchain network. It contains fields such as taskId, userId, documentHash, and status, encapsulating immutable task-related data recorded in the blockchain. The one-to-one relationship between Blockchain and SmartContract signifies that every blockchain entry corresponds to a specific smart contract execution. This structured interaction ensures secure, tamper-proof task handling, tightly coupling user submissions with blockchain-backed verification. Overall, the class diagram demonstrates a clean, modular system architecture that seamlessly integrates cloud task management with decentralized security.

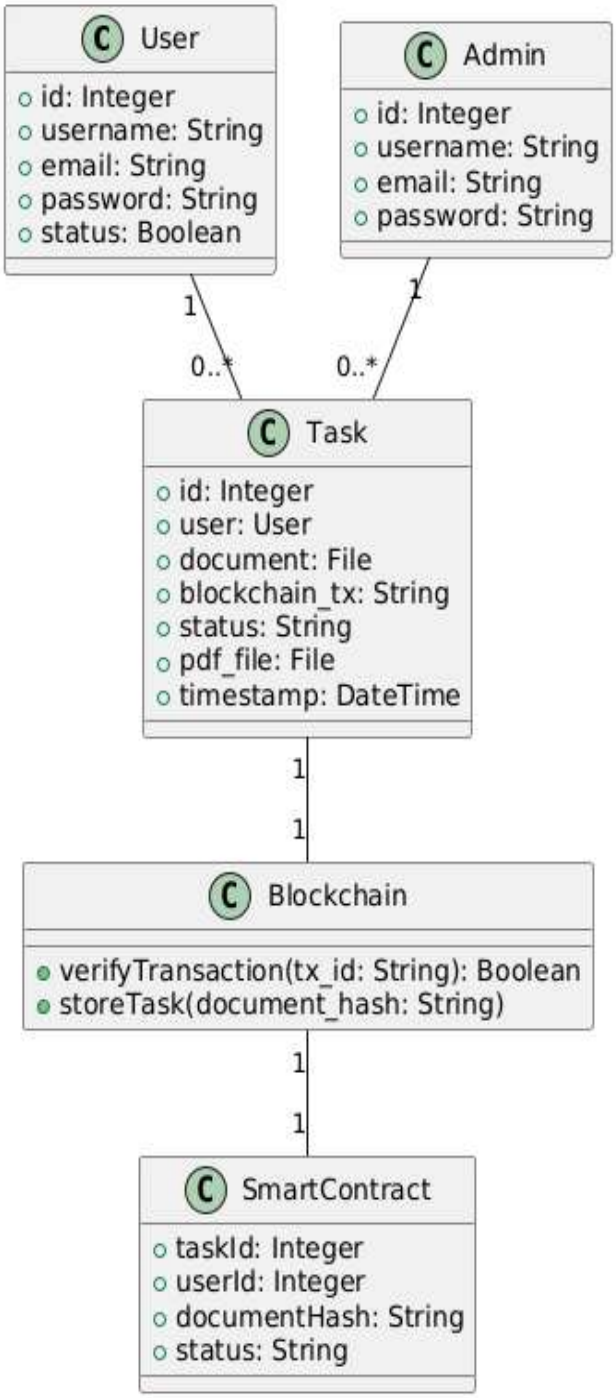

Figure 5.3.4 Class Diagram

CHAPTER – 6

CODING AND IMPLEMENTATION

6. CODING AND IMPLEMENTATION

6.1 Source Code

Admins /models.py

```
from django.db import models
```

Admins /urls.py

```
from django.urls import path
from Admins.views import *
urlpatterns = [
    path('adminhome/', adminhome, name='adminhome'),
    path('admin_update_userstatus/<int:user_id>/', admin_update_userstatus,
name='admin_update_userstatus'),
    path('admin_verify_tasks/', admin_verify_tasks, name="admin_verify_tasks"),
    path('verify_task/<int:task_id>/', verify_task, name="verify_task"),
]
```

Admins /views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.models import User
from django.contrib import messages
from django.contrib.auth.decorators import user_passes_test
from web3 import Web3
import json
import os
from Users.models import Task
from docx import Document
from fpdf import FPDF
from django.conf import settings
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

# Blockchain Setup
web3 = Web3(Web3.HTTPProvider(settings.WEB3_PROVIDER))
```

```
def adminhome(request):
    users = User.objects.filter(is_staff=False, is_superuser=False)
    return render(request, "Admin/adminhome.html", {"users": users})

def admin_update_userstatus(request, user_id):
    try:
        user = User.objects.get(id=user_id)

        # Toggle the is_active status
        user.is_active = not user.is_active
        user.save()

        # Display message based on the action
        if user.is_active:
            messages.success(request, f"User {user.username} has been activated.")
        else:
            messages.success(request, f"User {user.username} has been deactivated.")

        return redirect('adminhome')  # Redirect back to the admin home page
    except User.DoesNotExist:
        messages.error(request, "User not found.")
        return redirect('adminhome')

# Admin Check
def is_admin(user):
    return user.is_superuser

# Admin View for Tasks
@user_passes_test(is_admin)
def admin_verify_tasks(request):
    tasks = Task.objects.all()
    return render(request, "admin/admin_verify_tasks.html", {"tasks": tasks})

# Convert Word to PDF
def convert_to_pdf(word_path, pdf_path):
    doc = Document(word_path)
    pdf = canvas.Canvas(pdf_path, pagesize=letter)
```

```
    pdf.setFont("Helvetica", 12)

    y = 750  # Positioning for text in PDF
    for para in doc.paragraphs:
        pdf.drawString(100, y, para.text)
        y -= 20  # Move down for next line

    pdf.save()

# Verify Task on Blockchain and Convert to PDF
@user_passes_test(is_admin)
def verify_task(request, task_id):
    task = Task.objects.get(id=task_id)

    if not task.blockchain_tx:
        messages.error(request, "No blockchain transaction ID found for this task.")
        return redirect("admin_verify_tasks")

    try:
        # Fetch transaction details from blockchain
        transaction = web3.eth.get_transaction(task.blockchain_tx)

        if transaction:
            # Convert Word Document to PDF
            word_path = os.path.join(settings.MEDIA_ROOT, str(task.document))
            pdf_filename = str(task.document.name).replace(".docx", ".pdf").replace(".doc", ".pdf")
            pdf_path = os.path.join(settings.MEDIA_ROOT, "pdfs", pdf_filename)

            os.makedirs(os.path.dirname(pdf_path), exist_ok=True)  # Ensure directory exists
            convert_to_pdf(word_path, pdf_path)

            # Save PDF path to the database
            task.pdf_file.name = f"pdfs/{pdf_filename}"
            task.status = "Completed"
            task.save()

            messages.success(request, "Task verified! Document converted to PDF and status updated.")
```

```
        else:
            messages.error(request, "Blockchain transaction not found.")

    except Exception as e:
        messages.error(request, f"Verification failed. Error: {str(e)}")

    return redirect("admin_verify_tasks")
```

Backend /settings.py

```
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-py*9xja!s#nbg6gue-*e)n#c0m!7(j7c=$%_it-v1xoxq39zd-'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Users',
    'Admins',
```

```
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'Backend.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'Backend.wsgi.application'

WEB3_PROVIDER = "http://127.0.0.1:7545"

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
```

```
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static'),]

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Backend /urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

from Backend.views import *

urlpatterns = [
    path('admin/', admin.site.urls),
    path('Users/', include('Users.urls')),
    path('Admins/', include('Admins.urls')),

    path('', index, name='index'),
    path('home_page/', index, name='home_page'),
    path('user_logout/', user_logout, name='user_logout'),
    path('user_login/', user_login, name='user_login'),
    path('user_registration/', user_registration, name='user_registration')

]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Users /views.py

```
from django.shortcuts import render, redirect
from django.contrib import messages
from .models import Task
from django.conf import settings
from web3 import Web3
import json
import hashlib

# Connect to Blockchain
web3 = Web3(Web3.HTTPProvider(settings.WEB3_PROVIDER))

# Load contract address
with open("contract_address.txt", "r") as file:
    contract_address = file.read().strip()

# Load compiled contract
with open("compiled_code.json", "r") as file:
    compiled_sol = json.load(file)

abi = compiled_sol["contracts"]["TaskContract.sol"]["TaskContract"]["abi"]
contract = web3.eth.contract(address=contract_address, abi=abi)

# Create your views here.
def userhome(request):
    user = request.user
    return render(request, 'User/userhome.html', {'user':user})

def usertask(request):
    tasks = Task.objects.filter(user=request.user)

    if request.method == "POST" and request.FILES.get("document"):
        document = request.FILES["document"]
        user = request.user

        # Save document
        task = Task(user=user, document=document)
        task.save()
```

```
        # Generate document hash
        document_hash = hashlib.sha256(document.read()).hexdigest()

        # Store in blockchain
        tx_hash = contract.functions.createTask(document_hash).transact({'from': web3.eth.accounts[0]})
        tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)

        # Save blockchain transaction hash
        task.blockchain_tx = tx_receipt.transactionHash.hex()
        task.save()
        return redirect("usertask")
    return render(request, "User/usertask.html", {"tasks": tasks})

def verify_transaction(request):
    if request.method == "POST":
        tx_id = request.POST.get("tx_id")

        if not tx_id:
            messages.error(request, "Transaction ID is required.")
            return redirect("verify_transaction")
        try:
            # Fetch transaction details from blockchain
            transaction = web3.eth.get_transaction(tx_id)

            if transaction:
                messages.success(request, f"Transaction exists in block {transaction['blockNumber']}.")
            else:
                messages.error(request, "Transaction is invalid or not found.")

        except Exception as e:
            messages.error(request, f"Transaction is invalid or not found. Error: {str(e)}")
        return redirect("verify_transaction")
    return render(request, "User/verify_transaction.html")
```

Users /urls.py

```
from django.urls import path
```

```
from Users.views import *
urlpatterns = [
    path('userhome/', userhome, name='userhome'),
    path('usertask/', usertask, name='usertask'),
    path('verify_transaction/', verify_transaction, name="verify_transaction"),
]
```

Users /models.py

```
from django.db import models
from django.contrib.auth.models import User
class Task(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    document = models.FileField(upload_to="documents/")
    pdf_file = models.FileField(upload_to="pdfs/", blank=True, null=True)
    status = models.CharField(max_length=20, default="Pending")
    timestamp = models.DateTimeField(auto_now_add=True)
    blockchain_tx = models.CharField(max_length=255, blank=True, null=True)
    def __str__(self):
        return f"Task {self.id} - {self.user.username}"
```

compile_contract.py

```
from solcx import compile_standard, install_solc
import json
# Install Solidity compiler (if not already installed)
install_solc("0.8.0")
# Read Solidity contract
with open("TaskContract.sol", "r") as file:
    task_contract_file = file.read()
compiled_sol = compile_standard(
    {
        "language": "Solidity",
        "sources": {"TaskContract.sol": {"content": task_contract_file}},
        "settings": {"outputSelection": {"*": {"*": ["abi", "evm.bytecode"]}}},
    },
    solc_version="0.8.0",
)
# Save the compiled contract
```

```
with open("compiled_code.json", "w") as file:
    json.dump(compiled_sol, file)

print("Compilation successful!")
```

contract_address.txt

0x6D490953091ba90B2A5B6DABC8744E87b769C087

templates /user /userhome.html

```
{% block content %}
  <section id="user-home" class="section py-5">
    <div class="container">
      <div class="row justify-content-center">
        <div class="col-lg-8">
          <div class="card shadow-lg border-0 rounded-3">
            <div class="card-header text-white text-center">
              <h3 class="mb-0">User Profile</h3>
            </div>
            <div class="card-body p-4">
              <div class="row mb-3">
                <div class="col-md-6">
                  <h5 class="fw-bold">Username:</h5>
                  <p class="text-muted">{{ user.username }}</p>
                </div>
                <div class="col-md-6">
                  <h5 class="fw-bold">Full Name:</h5>
                  <p class="text-muted">{{ user.first_name }} {{ user.last_name }}</p>
                </div>
              </div>

              <div class="row mb-3">
                <div class="col-md-6">
                  <h5 class="fw-bold">Email:</h5>
                  <p class="text-muted">{{ user.email }}</p>
                </div>
                <div class="col-md-6">
                  <h5 class="fw-bold">Status:</h5>
```

```
            <p>
              {% if user.is_active %}
                <span class="badge bg-success">Active</span>
              {% else %}
                <span class="badge bg-danger">Inactive</span>
              {% endif %}
            </p>
          </div>
        </div>

        <div class="row mb-3">
          <div class="col-md-12">
            <h5 class="fw-bold">Last Login:</h5>
            <p class="text-muted">{{ user.last_login }}</p>
          </div>
        </div>
      </div>
    </div>
  </div>
 </section>
{% endblock %}
```

templates /user /usertask.html

```
{% load static %}

{% block content %}
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="card p-4 shadow-lg">
        <h2 class="text-center">Upload Word Document</h2>
        <form method="post" enctype="multipart/form-data">
          {% csrf_token %}
          <div class="mb-3">
            <label for="document" class="form-label">Choose a Word Document</label>
```

```
            <input type="file" class="form-control" name="document" accept=".doc,.docx"
required>
        </div>
        <div class="text-center">
            <button type="submit" class="btn btn-primary">Upload</button>
        </div>
    </form>
</div>

<h3 class="mt-4">Uploaded Tasks</h3>
<table class="table">
    <thead>
        <tr>
            <th>Document</th>
            <th>Status</th>
            <th>Blockchain TX</th>
            <th>Download</th>
        </tr>
    </thead>
    <tbody>
        {% for task in tasks %}
        <tr>
            <td>{{ task.document.name }}</td>
            <td>{{ task.status }}</td>
            <td>
                {% if task.blockchain_tx %}
                    {{ task.blockchain_tx }}
                {% else %}
                    No Transaction
                {% endif %}
            </td>
            <td>
                {% if task.status == "Completed" and task.pdf_file %}
                    <a href="{{ task.pdf_file.url }}" class="btn btn-success" download>Download
PDF</a>
                {% else %}
                    <button class="btn btn-light" disabled>Not Available</button>
```

```
                    {% endif %}
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
</div>
</div>
{% endblock %}
```

templates /user /verify_transaction.html

```
{% load static %}
{% block content %}
<div class="container mt-5">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <div class="card p-4 shadow-lg">
                <h2 class="text-center">Verify Blockchain Transaction</h2>
                <form method="post">
                    {% csrf_token %}
                    <div class="mb-3">
                        <label for="tx_id" class="form-label">Enter Transaction ID</label>
                        <input type="text" class="form-control" id="tx_id" name="tx_id" required>
                    </div>
                    <div class="text-center">
                        <button type="submit" class="btn btn-secondary">Verify</button>
                    </div>
                </form>
            </div>

            <!-- Display messages -->
            {% if messages %}
            <div class="row mt-3">
                <div class="col-md-12">
                    {% for message in messages %}
                        <div class="alert {% if message.tags == 'success' %}alert-success{% else %}alert-
```

```
danger{% endif %}" role="alert">
                        {{ message }}
                    </div>
                {% endfor %}
            </div>
        </div>
        {% endif %}
    </div>
  </div>
</div>
{% endblock %}
```

templates /admin /adminhome.html

```
{% block content %}
 <section id="admin-home" class="section">
  <div class="container">
   <!-- Display success or error messages -->
   {% if messages %}
    <div class="alert-container">
     {% for message in messages %}
      <div class="alert alert-{{ message.tags }}">
       {{ message }}
      </div>
     {% endfor %}
    </div>
   {% endif %}
   <!-- User Table -->
   <div class="user-table">
    <table class="table table-striped">
     <thead>
      <tr>
       <th scope="col">Username</th>
       <th scope="col">Email</th>
       <th scope="col">Status</th>
       <th scope="col">Action</th>
      </tr>
     </thead>
```

```
        <tbody>
          {% for user in users %}
           <tr>
            <td>{{ user.username }}</td>
            <td>{{ user.email }}</td>
            <td>
              {% if user.is_active %}
               <span class="badge bg-success">Active</span>
              {% else %}
               <span class="badge bg-danger">Inactive</span>
              {% endif %}
            </td>
            <td>
              <a href="{% url 'admin_update_userstatus' user.id %}" class="btn btn-primary">
                {% if user.is_active %}
                 Deactivate
                {% else %}
                 Activate
                {% endif %}
              </a>
            </td>
           </tr>
          {% endfor %}
        </tbody>
      </table>
    </div>
   </div>
  </section>
{% endblock %}
```

templates /admin /admin_verify_tasks.html

```
{% load static %}

{% block content %}
<div class="container mt-5">
   <h2 class="text-center">Admin: Verify User Tasks</h2>
```

```
    <!-- Display Messages -->
    {% if messages %}
    <div class="row mt-3">
        <div class="col-md-12">
            {% for message in messages %}
                <div class="alert {% if message.tags == 'success' %}alert-success{% else %}alert-danger{%
endif %}" role="alert">
                    {{ message }}
                </div>
            {% endfor %}
        </div>
    </div>
    {% endif %}

    <table class="table table-striped mt-4">
        <thead>
            <tr>
                <th>User</th>
                <th>Document</th>
                <th>Status</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            {% for task in tasks %}
            <tr>
                <td>{{ task.user.username }}</td>
                <td>{{ task.document.name }}</td>
                <td>{{ task.status }}</td>

                <td>
                    {% if task.status != "Completed" %}
                        <a href="{% url 'verify_task' task.id %}" class="btn btn-primary">Verify</a>
                    {% else %}
                        <button class="btn btn-success" disabled>Completed</button>
                    {% endif %}
                </td>
```

```
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock %}
```

templates /index.html

```
<!DOCTYPE html>
{% load static%}
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=1.0" name="viewport">
  <title>Application of blockchain methodology in secure task scheduling in cloud environment</title>
  <meta name="description" content="">
  <meta name="keywords" content="">

  <!-- Fonts -->
  <link href="https://fonts.googleapis.com" rel="preconnect">
  <link href="https://fonts.gstatic.com" rel="preconnect" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&family=Raleway:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap" rel="stylesheet">

  <!-- Vendor CSS Files -->
  <link href="{% static 'assets/vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">
  <link href="{% static 'assets/vendor/bootstrap-icons/bootstrap-icons.css' %}" rel="stylesheet">
  <link href="{% static 'assets/vendor/aos/aos.css' %}" rel="stylesheet">
  <link href="{% static 'assets/vendor/animate.css/animate.min.css' %}" rel="stylesheet">
  <link href="{% static 'assets/vendor/glightbox/css/glightbox.min.css' %}" rel="stylesheet">
  <link href="{% static 'assets/vendor/swiper/swiper-bundle.min.css' %}" rel="stylesheet">

  <!-- Main CSS File -->
```

```
  <link href="{% static 'assets/css/main.css' %}" rel="stylesheet">

</head>

<body class="index-page">

  <header id="header" class="header d-flex align-items-center fixed-top">
    <div class="container-fluid container-xl position-relative d-flex align-items-center justify-content-between">

      <a  class="logo d-flex align-items-center">
        <!-- Uncomment the line below if you also wish to use an image logo -->
        <!-- <img src="assets/img/logo.png" alt=""> -->
        <h1 class="sitename">blockchain methodology in secure task scheduling</h1>
      </a>

      <nav id="navmenu" class="navmenu">
        <ul>
          <li><a href="{% url 'home_page' %}" class="active">Home</a></li>
        </ul>
        <i class="mobile-nav-toggle d-xl-none bi bi-list"></i>
      </nav>

    </div>
  </header>

  <main class="main">

    <!-- Hero Section -->
    <section id="hero" class="hero section dark-background">

      <div id="hero-carousel" data-bs-interval="5000" class="container carousel carousel-fade" data-bs-ride="carousel">

        <!-- Slide 1 -->
        <div class="carousel-item active">
          <div class="carousel-container">
```

```
          <h2 class="animate__animated animate__fadeInDown">blockchain methodology in secure task scheduling</h2>
        </div>
      </div>

    </div>

    <svg class="hero-waves" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 24 150 28 " preserveAspectRatio="none">
      <defs>
        <path id="wave-path" d="M-160 44c30 0 58-18 88-18s 58 18 88 18 58-18 88-18 58 18 88 18 v44h-352z"></path>
      </defs>
      <g class="wave1">
        <use xlink:href="#wave-path" x="50" y="3"></use>
      </g>
      <g class="wave2">
        <use xlink:href="#wave-path" x="50" y="0"></use>
      </g>
      <g class="wave3">
        <use xlink:href="#wave-path" x="50" y="9"></use>
      </g>
    </svg>

  </section><!-- /Hero Section -->

  <!-- Contact Section -->
  <section id="contact" class="contact section">

    <!-- Section Title -->
    <div class="container section-title" data-aos="fade-up">
      <h2>Login</h2>
    </div><!-- End Section Title -->

    <div class="container" data-aos="fade" data-aos-delay="100">
      <div class="row gy-4">

        {% if messages %}
```

```
            <div class="row">
              <div class="col-md-12">
                {% for message in messages %}
                  <div class="alert alert-success" role="alert">
                    {{ message }}
                  </div>
                {% endfor %}
              </div>
            </div>
          {% endif %}

        <!-- Login Form -->
        <div class="col-lg-6">
          <h2>Login</h2>
          <form action="{% url 'user_login' %}" method="post" data-aos="fade-up" data-aos-delay="200">
            {% csrf_token %}
            <div class="row gy-4">
              <div class="col-md-12">
                <input type="text" name="username" class="form-control" placeholder="Your Username" required="">
              </div>
              <div class="col-md-12">
                <input type="password" class="form-control" name="password" placeholder="Your Password" required="">
              </div>
              <div class="col-md-12 text-center">
                <button type="submit" style="background-color: #ef6603; color: white; border: none; padding: 12px 30px; font-size: 18px; border-radius: 50px; box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.2); transition: all 0.3s ease-in-out;">
                  <i class="fas fa-sign-in-alt"></i> Login
                </button>
              </div>
            </div>
          </form>
        </div><!-- End Login Form -->

        <!-- Registration Form -->
```

```
        <div class="col-lg-6">
          <h2>Register</h2>
          <form action="{% url 'user_registration' %}" method="post" data-aos="fade-up" data-aos-delay="200">
            {% csrf_token %}
            <div class="row gy-4">
              <div class="col-md-6">
                <input type="text" name="first_name" class="form-control" placeholder="First Name" required="">
              </div>
              <div class="col-md-6">
                <input type="text" name="last_name" class="form-control" placeholder="Last Name" required="">
              </div>
              <div class="col-md-6">
                <input type="text" name="username" class="form-control" placeholder="Username" required="">
              </div>
              <div class="col-md-6">
                <input type="email" class="form-control" name="email" placeholder="Your Email" required="">
              </div>
              <div class="col-md-6">
                <input type="password" class="form-control" name="password" placeholder="Password" required="">
              </div>
              <div class="col-md-6">
                <input type="password" class="form-control" name="confirm_password" placeholder="Confirm Password" required="">
              </div>
              <div class="col-md-12 text-center">
                <button type="submit" style="background-color: #ef6603; color: white; border: none; padding: 12px 30px; font-size: 18px; border-radius: 50px; box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.2); transition: all 0.3s ease-in-out;">
                  <i class="fas fa-user-plus"></i> Register
                </button>
              </div>
            </div>
          </form>
```

```
        </div><!-- End Registration Form -->

       </div>
      </div>


    </section><!-- /Contact Section -->

  </main>

  <footer id="footer" class="footer dark-background">
    <div class="container">
      <h3 class="sitename">blockchain methodology in secure task scheduling</h3>
    </div>
  </footer>

  <!-- Scroll Top -->
  <a href="#" id="scroll-top" class="scroll-top d-flex align-items-center justify-content-center"><i
 class="bi bi-arrow-up-short"></i></a>

  <!-- Preloader -->
  <div id="preloader"></div>

  <!-- Vendor JS Files -->
  <script src="{% static 'assets/vendor/bootstrap/js/bootstrap.bundle.min.js' %}"></script>
  <script src="{% static 'assets/vendor/php-email-form/validate.js' %}"></script>
  <script src="{% static 'assets/vendor/aos/aos.js' %}"></script>
  <script src="{% static 'assets/vendor/glightbox/js/glightbox.min.js' %}"></script>
  <script src="{% static 'assets/vendor/imagesloaded/imagesloaded.pkgd.min.js' %}"></script>
  <script src="{% static 'assets/vendor/isotope-layout/isotope.pkgd.min.js' %}"></script>
  <script src="{% static 'assets/vendor/swiper/swiper-bundle.min.js' %}"></script>

  <!-- Main JS File -->
  <script src="{% static 'assets/js/main.js' %}"></script>
 </body>
 </html>
```

static /assets /main.js

```
(function() {
  "use strict";

  /**
   * Apply .scrolled class to the body as the page is scrolled down
   */
  function toggleScrolled() {
    const selectBody = document.querySelector('body');
    const selectHeader = document.querySelector('#header');
    if (!selectHeader.classList.contains('scroll-up-sticky') && !selectHeader.classList.contains('sticky-top') && !selectHeader.classList.contains('fixed-top')) return;
    window.scrollY > 100 ? selectBody.classList.add('scrolled') : selectBody.classList.remove('scrolled');
  }

  document.addEventListener('scroll', toggleScrolled);
  window.addEventListener('load', toggleScrolled);

  /**
   * Mobile nav toggle
   */
  const mobileNavToggleBtn = document.querySelector('.mobile-nav-toggle');

  function mobileNavToogle() {
    document.querySelector('body').classList.toggle('mobile-nav-active');
    mobileNavToggleBtn.classList.toggle('bi-list');
    mobileNavToggleBtn.classList.toggle('bi-x');
  }
  mobileNavToggleBtn.addEventListener('click', mobileNavToogle);

  /**
   * Hide mobile nav on same-page/hash links
   */
  document.querySelectorAll('#navmenu a').forEach(navmenu => {
    navmenu.addEventListener('click', () => {
```

```
    if (document.querySelector('.mobile-nav-active')) {
      mobileNavToogle();
    }
  });

});

/**
 * Toggle mobile nav dropdowns
 */
document.querySelectorAll('.navmenu .toggle-dropdown').forEach(navmenu => {
  navmenu.addEventListener('click', function(e) {
    e.preventDefault();
    this.parentNode.classList.toggle('active');
    this.parentNode.nextElementSibling.classList.toggle('dropdown-active');
    e.stopImmediatePropagation();
  });
});

/**
 * Preloader
 */
const preloader = document.querySelector('#preloader');
if (preloader) {
  window.addEventListener('load', () => {
    preloader.remove();
  });
}

/**
 * Scroll top button
 */
let scrollTop = document.querySelector('.scroll-top');

function toggleScrollTop() {
  if (scrollTop) {
    window.scrollY > 100 ? scrollTop.classList.add('active') : scrollTop.classList.remove('active');
```

```
  }
}
scrollTop.addEventListener('click', (e) => {
  e.preventDefault();
  window.scrollTo({
    top: 0,
    behavior: 'smooth'
  });
});

window.addEventListener('load', toggleScrollTop);
document.addEventListener('scroll', toggleScrollTop);

/**
 * Animation on scroll function and init
 */
function aosInit() {
  AOS.init({
    duration: 600,
    easing: 'ease-in-out',
    once: true,
    mirror: false
  });
}
window.addEventListener('load', aosInit);

/**
 * Initiate glightbox
 */
const glightbox = GLightbox({
  selector: '.glightbox'
});

/**
 * Init isotope layout and filters
 */
document.querySelectorAll('.isotope-layout').forEach(function(isotopeItem) {
```

```
  let layout = isotopeItem.getAttribute('data-layout') ?? 'masonry';
  let filter = isotopeItem.getAttribute('data-default-filter') ?? '*';
  let sort = isotopeItem.getAttribute('data-sort') ?? 'original-order';

  let initIsotope;
  imagesLoaded(isotopeItem.querySelector('.isotope-container'), function() {
    initIsotope = new Isotope(isotopeItem.querySelector('.isotope-container'), {
      itemSelector: '.isotope-item',
      layoutMode: layout,
      filter: filter,
      sortBy: sort
    });
  });

  isotopeItem.querySelectorAll('.isotope-filters li').forEach(function(filters) {
    filters.addEventListener('click', function() {
      isotopeItem.querySelector('.isotope-filters .filter-active').classList.remove('filter-active');
      this.classList.add('filter-active');
      initIsotope.arrange({
        filter: this.getAttribute('data-filter')
      });
      if (typeof aosInit === 'function') {
        aosInit();
      }
    }, false);
  });

});

/**
 * Init swiper sliders
 */
function initSwiper() {
  document.querySelectorAll(".init-swiper").forEach(function(swiperElement) {
    let config = JSON.parse(
      swiperElement.querySelector(".swiper-config").innerHTML.trim()
    );
```

```
    if (swiperElement.classList.contains("swiper-tab")) {
      initSwiperWithCustomPagination(swiperElement, config);
    } else {
      new Swiper(swiperElement, config);
    }
  });
}

window.addEventListener("load", initSwiper);

/**
 * Correct scrolling position upon page load for URLs containing hash links.
 */
window.addEventListener('load', function(e) {
  if (window.location.hash) {
    if (document.querySelector(window.location.hash)) {
      setTimeout(() => {
        let section = document.querySelector(window.location.hash);
        let scrollMarginTop = getComputedStyle(section).scrollMarginTop;
        window.scrollTo({
          top: section.offsetTop - parseInt(scrollMarginTop),
          behavior: 'smooth'
        });
      }, 100);
    }
  }
});

/**
 * Navmenu Scrollspy
 */
let navmenulinks = document.querySelectorAll('.navmenu a');

function navmenuScrollspy() {
  navmenulinks.forEach(navmenulink => {
    if (!navmenulink.hash) return;
```

```
    let section = document.querySelector(navmenulink.hash);
    if (!section) return;
    let position = window.scrollY + 200;
    if (position >= section.offsetTop && position <= (section.offsetTop + section.offsetHeight)) {
      document.querySelectorAll('.navmenu a.active').forEach(link => link.classList.remove('active'));
      navmenulink.classList.add('active');
    } else {
      navmenulink.classList.remove('active');
    }
  })
 }
 window.addEventListener('load', navmenuScrollspy);
 document.addEventListener('scroll', navmenuScrollspy);

})();
```

static /assets /main.css

```
:root {
 --default-font: "Roboto",  system-ui, -apple-system, "Segoe UI", Roboto, "Helvetica Neue", Arial, "Noto Sans", "Liberation Sans", sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI Symbol", "Noto Color Emoji";
 --heading-font: "Raleway",  sans-serif;
 --nav-font: "Poppins",  sans-serif;
}

/* Global Colors - The following color variables are used throughout the website. Updating them here will change the color scheme of the entire website */
:root {
 --background-color: #ffffff; /* Background color for the entire website, including individual sections */
 --default-color: #444444; /* Default color used for the majority of the text content across the entire website */
 --heading-color: #2a2c39; /* Color for headings, subheadings and title throughout the website */
 --accent-color: #ef6603; /* Accent color that represents your brand on the website. It's used for buttons, links, and other elements that need to stand out */
 --surface-color: #ffffff; /* The surface color is used as a background of boxed elements within sections, such as cards, icon boxes, or other elements that require a visual separation from the global background. */
```

```
  --contrast-color: #ffffff; /* Contrast color for text, ensuring readability against backgrounds of accent, heading, or default colors. */
}

/* Nav Menu Colors - The following color variables are used specifically for the navigation menu. They are separate from the global colors to allow for more customization options */
:root {
  --nav-color: #ffffff;  /* The default color of the main navmenu links */
  --nav-hover-color: #ef6603; /* Applied to main navmenu links when they are hovered over or active */
  --nav-mobile-background-color: #ffffff; /* Used as the background color for mobile navigation menu */
  --nav-dropdown-background-color: #ffffff; /* Used as the background color for dropdown items that appear when hovering over primary navigation items */
  --nav-dropdown-color: #060606; /* Used for navigation links of the dropdown items in the navigation menu. */
  --nav-dropdown-hover-color: #ef6603; /* Similar to --nav-hover-color, this color is applied to dropdown navigation links when they are hovered over. */
}

/* Color Presets - These classes override global colors when applied to any section or element, providing reuse of the sam color scheme. */

.light-background {
  --background-color: #f9f9f9;
  --surface-color: #ffffff;
}

.dark-background {
  --background-color: #2a2c39;
  --default-color: #ffffff;
  --heading-color: #ffffff;
  --surface-color: #404356;
  --contrast-color: #ffffff;
}

/* Smooth scroll */
:root {
  scroll-behavior: smooth;
```

```
}

# General Styling & Shared Classes
body {
  color: var(--default-color);
  background-color: var(--background-color);
  font-family: var(--default-font);
}

a {
  color: var(--accent-color);
  text-decoration: none;
  transition: 0.3s;
}

a:hover {
  color: color-mix(in srgb, var(--accent-color), transparent 25%);
  text-decoration: none;
}

h1,
h2,
h3,
h4,
h5,
h6 {
  color: var(--heading-color);
  font-family: var(--heading-font);
}

.php-email-form .error-message {
  display: none;
  background: #df1529;
  color: #ffffff;
  text-align: left;
  padding: 15px;
  margin-bottom: 24px;
```

```
  font-weight: 600;
}

.php-email-form .sent-message {
  display: none;
  color: #ffffff;
  background: #059652;
  text-align: center;
  padding: 15px;
  margin-bottom: 24px;
  font-weight: 600;
}

.php-email-form .loading {
  display: none;
  background: var(--surface-color);
  text-align: center;
  padding: 15px;
  margin-bottom: 24px;
}

.php-email-form .loading:before {
  content: "";
  display: inline-block;
  border-radius: 50%;
  width: 24px;
  height: 24px;
  margin: 0 10px -6px 0;
  border: 3px solid var(--accent-color);
  border-top-color: var(--surface-color);
  animation: php-email-form-loading 1s linear infinite;
}

@keyframes php-email-form-loading {
  0% {
    transform: rotate(0deg);
  }
```

```
  100% {
    transform: rotate(360deg);
  }
}

.header {
  --background-color: rgba(255, 255, 255, 0);
  --heading-color: #ffffff;
  color: var(--default-color);
  background-color: var(--background-color);
  padding: 20px 0;
  transition: all 0.5s;
  z-index: 997;
}

.header .logo {
  line-height: 1;
}

.header .logo img {
  max-height: 32px;
  margin-right: 8px;
}

.header .logo h1 {
  font-size: 30px;
  margin: 0;
  font-weight: 700;
  color: var(--heading-color);
}

.scrolled .header {
  box-shadow: 0px 0 18px rgba(0, 0, 0, 0.1);
}

/* Global Header on Scroll
```

```
------------------------------*/
.scrolled .header {
  --background-color: rgba(42, 44, 57, 0.9);
}

/* Navmenu - Desktop */
@media (min-width: 1200px) {
  .navmenu {
    padding: 0;
  }

  .navmenu ul {
    margin: 0;
    padding: 0;
    display: flex;
    list-style: none;
    align-items: center;
  }

  .navmenu li {
    position: relative;
    margin-left: 5px;
  }

  .navmenu a,
  .navmenu a:focus {
    color: var(--nav-color);
    padding: 8px 20px;
    font-size: 14px;
    font-family: var(--nav-font);
    font-weight: 400;
    display: flex;
    align-items: center;
    justify-content: space-between;
    white-space: nowrap;
    transition: 0.3s;
    border-radius: 50px;
```

```
}

.navmenu a i,
.navmenu a:focus i {
  font-size: 12px;
  line-height: 0;
  margin-left: 5px;
  transition: 0.3s;
}

.navmenu li:hover>a,
.navmenu .active,
.navmenu .active:focus {
  background-color: var(--nav-hover-color);
}

.navmenu .dropdown ul {
  margin: 0;
  padding: 10px 0;
  background: var(--nav-dropdown-background-color);
  display: block;
  position: absolute;
  visibility: hidden;
  left: 0;
  top: 130%;
  opacity: 0;
  transition: 0.3s;
  border-radius: 15px;
  z-index: 99;
  box-shadow: 0px 0px 30px rgba(0, 0, 0, 0.1);
}

.navmenu .dropdown ul li {
  min-width: 200px;
  margin-left: 0;
}
```

```
  .navmenu .dropdown ul a {
    padding: 10px 20px;
    font-size: 15px;
    text-transform: none;
    color: var(--nav-dropdown-color);
  }

  .navmenu .dropdown ul a i {
    font-size: 12px;
  }

  .navmenu .dropdown ul a:hover,
  .navmenu .dropdown ul .active:hover,
  .navmenu .dropdown ul li:hover>a {
    background-color: transparent;
    color: var(--nav-hover-color);
  }

  .navmenu .dropdown:hover>ul {
    opacity: 1;
    top: 105%;
    visibility: visible;
  }

  .navmenu .dropdown .dropdown ul {
    top: 0;
    left: -90%;
    visibility: hidden;
  }

  .navmenu .dropdown .dropdown:hover>ul {
    opacity: 1;
    top: 0;
    left: -100%;
    visibility: visible;
  }
}
```

```
/* Navmenu - Mobile */
@media (max-width: 1199px) {
  .mobile-nav-toggle {
    color: var(--nav-color);
    font-size: 28px;
    line-height: 0;
    margin-right: 10px;
    cursor: pointer;
    transition: color 0.3s;
  }

  .navmenu {
    padding: 0;
    z-index: 9997;
  }

  .navmenu ul {
    display: none;
    list-style: none;
    position: absolute;
    inset: 60px 20px 20px 20px;
    padding: 10px 0;
    margin: 0;
    border-radius: 6px;
    background-color: var(--nav-mobile-background-color);
    overflow-y: auto;
    transition: 0.3s;
    z-index: 9998;
    box-shadow: 0px 0px 30px rgba(0, 0, 0, 0.1);
  }

  .navmenu a,
  .navmenu a:focus {
    color: var(--nav-dropdown-color);
    padding: 10px 20px;
    font-family: var(--nav-font);
```

```
  font-size: 17px;
  font-weight: 500;
  display: flex;
  align-items: center;
  justify-content: space-between;
  white-space: nowrap;
  transition: 0.3s;
}

.navmenu a i,
.navmenu a:focus i {
  font-size: 12px;
  line-height: 0;
  margin-left: 5px;
  width: 30px;
  height: 30px;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 50%;
  transition: 0.3s;
  background-color: color-mix(in srgb, var(--accent-color), transparent 90%);
}

.navmenu a i:hover,
.navmenu a:focus i:hover {
  background-color: var(--accent-color);
  color: var(--contrast-color);
}

.navmenu a:hover,
.navmenu .active,
.navmenu .active:focus {
  color: var(--nav-dropdown-hover-color);
}

.navmenu .active i,
```

```
.navmenu .active:focus i {
  background-color: var(--accent-color);
  color: var(--contrast-color);
  transform: rotate(180deg);
}

.navmenu .dropdown ul {
  position: static;
  display: none;
  z-index: 99;
  padding: 10px 0;
  margin: 10px 20px;
  background-color: var(--nav-dropdown-background-color);
  border: 1px solid color-mix(in srgb, var(--default-color), transparent 90%);
  box-shadow: none;
  transition: all 0.5s ease-in-out;
}

.navmenu .dropdown ul ul {
  background-color: rgba(33, 37, 41, 0.1);
}

.navmenu .dropdown>.dropdown-active {
  display: block;
  background-color: rgba(33, 37, 41, 0.03);
}

.mobile-nav-active {
  overflow: hidden;
}

.mobile-nav-active .mobile-nav-toggle {
  color: #fff;
  position: absolute;
  font-size: 32px;
  top: 15px;
  right: 15px;
```

```
    margin-right: 0;
    z-index: 9999;
  }

  .mobile-nav-active .navmenu {
    position: fixed;
    overflow: hidden;
    inset: 0;
    background: rgba(33, 37, 41, 0.8);
    transition: 0.3s;
  }

  .mobile-nav-active .navmenu>ul {
    display: block;
  }
}

.footer {
  color: var(--default-color);
  background-color: var(--background-color);
  font-size: 14px;
  text-align: center;
  padding: 30px 0;
  position: relative;
}

.footer h3 {
  font-size: 36px;
  font-weight: 700;
  position: relative;
  padding: 0;
  margin: 0 0 15px 0;
}

.footer p {
  font-size: 15;
  font-style: italic;
```

```
  padding: 0;
  margin: 0 0 30px 0;
}

.footer .social-links {
  margin: 0 0 30px 0;
}

.footer .social-links a {
  font-size: 16px;
  display: flex;
  align-items: center;
  justify-content: center;
  background: var(--accent-color);
  color: var(--contrast-color);
  line-height: 1;
  margin: 0 4px;
  border-radius: 50%;
  text-align: center;
  width: 36px;
  height: 36px;
  transition: 0.3s;
}

.footer .social-links a:hover {
  background: color-mix(in srgb, var(--accent-color), transparent 20%);
  text-decoration: none;
}

.footer .copyright {
  padding-top: 25px;
  border-top: 1px solid color-mix(in srgb, var(--default-color), transparent 90%);
}

.footer .credits {
  font-size: 13px;
  padding-top: 5px;
```

```
}

#preloader {
  position: fixed;
  inset: 0;
  z-index: 9999;
  overflow: hidden;
  background-color: var(--background-color);
  transition: all 0.6s ease-out;
  width: 100%;
  height: 100vh;
}

#preloader:before,
#preloader:after {
  content: "";
  position: absolute;
  border: 4px solid var(--accent-color);
  border-radius: 50%;
  animation: animate-preloader 2s cubic-bezier(0, 0.2, 0.8, 1) infinite;
}

#preloader:after {
  animation-delay: -0.5s;
}

@keyframes animate-preloader {
  0% {
    width: 10px;
    height: 10px;
    top: calc(50% - 5px);
    left: calc(50% - 5px);
    opacity: 1;
  }

  100% {
    width: 72px;
```

```
    height: 72px;
    top: calc(50% - 36px);
    left: calc(50% - 36px);
    opacity: 0;
  }
}

.scroll-top {
  position: fixed;
  visibility: hidden;
  opacity: 0;
  right: 15px;
  bottom: -15px;
  z-index: 99999;
  background-color: var(--accent-color);
  width: 44px;
  height: 44px;
  border-radius: 50px;
  transition: all 0.4s;
}

.scroll-top i {
  font-size: 24px;
  color: var(--contrast-color);
  line-height: 0;
}

.scroll-top:hover {
  background-color: color-mix(in srgb, var(--accent-color), transparent 20%);
  color: var(--contrast-color);
}

.scroll-top.active {
  visibility: visible;
  opacity: 1;
  bottom: 15px;
}
```

```
@media screen and (max-width: 768px) {
  [data-aos-delay] {
    transition-delay: 0 !important;
  }
}

.page-title {
  color: var(--default-color);
  background-color: var(--background-color);
  background-size: cover;
  background-position: center;
  background-repeat: no-repeat;
  padding: 160px 0 80px 0;
  text-align: center;
  position: relative;
}

.page-title:before {
  content: "";
  background-color: color-mix(in srgb, var(--background-color), transparent 50%);
  position: absolute;
  inset: 0;
}

.page-title h1 {
  font-size: 42px;
  font-weight: 700;
  margin-bottom: 10px;
}

.page-title .breadcrumbs ol {
  display: flex;
  flex-wrap: wrap;
  list-style: none;
  justify-content: center;
  padding: 0;
```

```
  margin: 0;
  font-size: 16px;
  font-weight: 400;
}

.page-title .breadcrumbs ol li+li {
  padding-left: 10px;
}

.page-title .breadcrumbs ol li+li::before {
  content: "/";
  display: inline-block;
  padding-right: 10px;
  color: color-mix(in srgb, var(--default-color), transparent 50%);
}

/*--------------------------------------------------------------
# Global Sections
--------------------------------------------------------------*/
section,
.section {
  color: var(--default-color);
  background-color: var(--background-color);
  padding: 60px 0;
  scroll-margin-top: 77px;
  overflow: clip;
}

/*--------------------------------------------------------------
# Global Section Titles
--------------------------------------------------------------*/
.section-title {
  padding-bottom: 60px;
  position: relative;
}

.section-title h2 {
```

```
  font-size: 14px;
  font-weight: 500;
  padding: 0;
  line-height: 1px;
  margin: 0;
  letter-spacing: 1.5px;
  text-transform: uppercase;
  color: color-mix(in srgb, var(--default-color), transparent 50%);
  position: relative;
}

.section-title h2::after {
  content: "";
  width: 120px;
  height: 1px;
  display: inline-block;
  background: var(--accent-color);
  margin: 4px 10px;
}

.section-title p {
  color: var(--heading-color);
  margin: 0;
  font-size: 28px;
  font-weight: 700;
  text-transform: uppercase;
  font-family: var(--heading-font);
}
```

6.2 Implementation

6.2.1 Python

Python is a high-level, interpreted, and object-oriented programming language known for its simplicity and readability. It uses indentation instead of complex syntax like curly braces, making the code more structured and easy to understand. Python supports multiple programming paradigms such as procedural, object-oriented, and functional programming, making it highly flexible for application development.

Python is widely used across industries and by leading companies like Google, Amazon, Facebook, and Uber due to its versatility and powerful libraries. It provides a rich standard library that simplifies tasks such as web development, database management, and system integration.

In this project, Python is used to develop the backend system for secure task scheduling. It handles user requests, manages file uploads, and controls the overall workflow of the application using the Django framework.

Additionally, Python enables blockchain integration through the Web3.py library, allowing communication with the Ganache network, generation of transaction hashes, and ensuring data integrity. It also supports file processing operations such as converting uploaded Word documents into PDF after admin verification.

The biggest strength of Python is its extensive collection of libraries, which are utilized in this project for:

• Web Development (Django Framework)

• Blockchain Integration (Web3.py)

• File Handling and Processing

• Database Operations (SQLite)

• API Integration

Advantages of Python

1. **Extensive Libraries Support**

   Python provides a vast collection of libraries such as Django for web development and Web3.py for blockchain integration. These libraries simplify complex tasks and reduce the need to write code from scratch, making development faster and more efficient.

2. **Easy Integration with Django Framework**

Python integrates smoothly with the Django framework, which offers built-in features like authentication, URL routing, and database management. This helps in developing a secure and well-structured web application for task scheduling.

3. **Blockchain Compatibility**

   Python supports blockchain interaction through libraries like Web3.py, enabling communication with the Ganache network. It allows the system to generate, store, and verify transaction hashes, ensuring data integrity and security.

4. **High Developer Productivity**

   Python's simple syntax and minimal code structure increase developer productivity. Complex operations such as handling requests, processing data, and managing workflows can be implemented quickly and efficiently.

5. **Efficient File Handling**

   Python has strong built-in support for file handling operations. In this project, it is used to upload documents, store them securely, and convert Word files into PDF format after admin verification.

6. **Readable and Maintainable Code**

   Python code is highly readable and easy to understand due to its clean syntax. This makes debugging and maintaining the system easier, especially when updating or adding new features in the future.

7. **Platform Independence**

   Python is platform-independent, meaning the application can run on different operating systems such as Windows, Linux, or macOS without significant modifications.

8. **Open Source and Cost Effective**

   Python is free and open-source, which reduces overall development cost. It also provides access to a large number of free tools, libraries, and community resources.

9. **Scalability and Flexibility**

   Python allows easy scaling of the application. New modules, features, or blockchain enhancements can be integrated without affecting the existing system structure.

10. **Strong Community Support**

    Python has a large and active developer community. This ensures continuous updates, better documentation, and quick solutions to technical issues, which is beneficial during development

Disadvantages of Python

1. **Execution Speed Limitations**

   Python is an interpreted language, so the code is executed line by line, which makes it slower compared to compiled languages like Java or C++. In this project, performance is not a major issue, but for large-scale blockchain applications, execution speed could become a limitation.

2. **Limited Client-Side Usage**

   Python is mainly used for backend development and is not suitable for frontend or browser-based operations. In this project, additional technologies (HTML, CSS, JavaScript) are required to handle the user interface.

3. **Runtime Errors due to Dynamic Typing**

   Python uses dynamic typing, which means variable types are not declared explicitly. While this makes coding easier, it may lead to runtime errors if not handled properly, especially in complex applications.

4. **Higher Memory Consumption**

   Python consumes more memory compared to other languages due to its flexible data types and dynamic nature. This can affect performance when handling large datasets or multiple user requests.

5. **Dependency on External Libraries**

   This project relies on external libraries such as Django and Web3.py. Any issues, updates, or compatibility problems in these libraries may affect the system's functionality.

6.2.2 Django Framework

Django is a high-level Python web framework that enables rapid development of secure, scalable, and maintainable web applications. It follows the Model-View-Template (MVT) architecture, which helps in organizing the application into logical components such as data handling, user interface, and business logic.

In this project, Django is used to develop the backend system for secure task scheduling. It handles user authentication, request processing, database interactions, and overall application flow. Django simplifies complex tasks by providing built-in functionalities, reducing development time and effort.

The framework plays a key role in managing different modules of the system, including user operations, admin controls, and document handling. It ensures secure communication between the frontend and backend, while also maintaining data consistency.

Key Features of Django Used in This Project

1. **MVT Architecture**

   Django follows the Model-View-Template pattern, which separates the data, logic, and presentation layers, making the application well-structured and easy to manage.

2. **Built-in Authentication System**

   Django provides a secure authentication system for user registration, login, and access control, which is used in this project to manage users and admin roles.

3. **ORM (Object Relational Mapping)**

   Django ORM allows interaction with the database using Python code instead of SQL queries. This simplifies database operations such as storing user data and task details.

4. **URL Routing**

   Django efficiently maps URLs to views, enabling smooth navigation between different pages like login, upload, and admin dashboard.

5. **Security Features**

   Django provides protection against common security threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

6. **Scalability and Flexibility**

   Django allows easy addition of new features and modules, making the system scalable for future enhancements.

6.2.3 Blockchain Technology

Blockchain is a decentralized and distributed digital ledger technology that records transactions in a secure, transparent, and tamper-proof manner. It stores data in the form of blocks, where each block contains a set of transactions and is linked to the previous block using cryptographic hashes. This chaining of blocks ensures that once data is recorded, it cannot be modified or deleted, thereby maintaining data integrity.

Unlike traditional centralized systems, blockchain does not rely on a single authority to manage data. Instead, it operates on a peer-to-peer network where all transactions are verified and recorded collectively. This eliminates the risk of single-point failure and enhances system reliability and trust.

In this project, blockchain technology is used to provide security and authenticity to the task scheduling process. When a user uploads a document, a transaction is generated and recorded in the blockchain. This transaction produces a unique hash value, which acts as a digital fingerprint of the uploaded data. Any modification in the data will result in a different hash, making it easy to detect tampering.

The blockchain network in this system is implemented using Ganache, which is a personal Ethereum blockchain used for development and testing purposes. Ganache allows the deployment and execution of smart contracts in a controlled environment. It also provides transaction details such as block number, gas usage, and transaction hash, which are used in this project for tracking and verification.

By integrating blockchain with the Django application, the system ensures that all task-related data is securely recorded and can be verified at any time. This enhances trust between users and administrators and prevents unauthorized modifications.

Key Features of Blockchain Used in This Project

1. **Immutability**

   Blockchain ensures that once a transaction is recorded, it cannot be modified or deleted. Each block is securely linked using cryptographic hashes, which preserves the integrity of task data and prevents unauthorized changes.

2. **Decentralization**

   Blockchain operates without relying on a central authority, reducing the chances of data manipulation and system failure. This improves the reliability of the task scheduling system.

3. **Transparency**

   All transactions are recorded on the blockchain and can be verified when required. This ensures transparency in task processing and builds trust between users and administrators.

4. **Security**

   Blockchain uses strong cryptographic techniques to secure data. This makes it highly resistant to cyber attacks and protects sensitive information from unauthorized access.

5. **Traceability**

   Each transaction is associated with a unique hash value, which allows tracking of tasks from upload to completion. This helps in verifying the authenticity and history of each task.

6.2.4 Smart Contract Implementation

A smart contract is a self-executing program that runs on a blockchain and automatically enforces predefined rules and conditions. It is written in a programming language called Solidity and deployed on a blockchain network such as Ethereum.

In this project, smart contracts are used to securely store and manage task-related data. Whenever a user uploads a document, a corresponding transaction is created through the smart contract. This

transaction generates a unique hash value, which acts as a digital proof of the uploaded data.

The smart contract ensures that once the task information is stored, it cannot be altered or deleted. This provides integrity and authenticity to the system. The contract also helps in verifying whether a task has been processed or not, by maintaining a record of transactions on the blockchain.

The smart contract is deployed using Ganache, which provides a local Ethereum environment for testing and development. It allows interaction between the Django application and the blockchain network through Web3 integration.

Functions of Smart Contract in This Project

1. **Storing Task Data**

   The smart contract stores task-related information in the form of transaction data on the blockchain.

2. **Generating Transaction Hash**

   Each transaction produces a unique hash value, which is used to verify the authenticity of the uploaded document.

3. **Ensuring Data Integrity**

   Once data is stored in the blockchain, it cannot be modified, ensuring secure and tamper-proof records.

4. **Task Verification**

   The smart contract allows verification of whether a task has been processed by checking the transaction details.

5. **Secure Data Handling**

   All operations are performed using blockchain protocols, ensuring that data remains secure and reliable.

6.2.5 Web3 Integration

Web3 is a technology that enables interaction between web applications and blockchain networks. It acts as a bridge that connects the Django backend with the Ethereum blockchain. In this project, the Web3.py library is used to establish communication between the application and the Ganache blockchain.

Web3 integration allows the system to send and retrieve data from the blockchain. When a user uploads a document, the Django application uses Web3 to interact with the deployed smart contract and create

a transaction. This transaction is then recorded on the blockchain and generates a unique transaction hash.

The Web3 module is configured with the Ganache network using its HTTP provider. It connects to the blockchain using a specific address and port, enabling secure communication between the application and the blockchain environment.

Through Web3 integration, the system can also retrieve transaction details such as transaction hash, block number, and status. This information is used to verify the authenticity of tasks and ensure that data has not been tampered with.

Functions of Web3 in This Project

1. **Connecting to Blockchain Network**

   Web3 establishes a connection between the Django application and the Ganache blockchain.

2. **Sending Transactions**

   It is used to send task-related data to the smart contract, which records it on the blockchain.

3. **Retrieving Transaction Details**

   Web3 allows fetching of transaction information such as hash values and block details.

4. **Interacting with Smart Contracts**

   It enables execution of smart contract functions directly from the Django application.

5. **Ensuring Secure Communication**

   Web3 provides a secure interface for data exchange between the application and blockchain.

CHAPTER – 7

TESTING

7. TESTING

The purpose of testing is to identify errors and ensure that the developed system works correctly. Testing is the process of evaluating the application to discover faults or weaknesses and to verify that it meets user requirements and expectations. It provides a way to check the functionality of individual components, modules, and the complete system. The goal is to ensure that the system performs efficiently and does not fail in an unacceptable manner. Various types of testing are carried out, each addressing a specific requirement of the system.

7.1 TYPES OF TESTS

7.1.1 Unit Testing

Unit testing focuses on individual components or modules of an application to validate that they function correctly. It involves testing small pieces of code in isolation before integrating them with other components.

Key Features:

- Ensures individual functions or methods produce expected outputs.
- Tests all decision branches and code flows.
- Performed during the development phase before integration.
- Uses a structural approach, often requiring knowledge of internal code.

7.1.2 Integration Testing

Integration testing verifies that different components of the software interact correctly. It ensures that modules work together as expected when combined.

Key Features:

- Validates interactions between integrated components.
- Ensures the consistency of data flow across modules.
- Helps identify defects that arise when combining independent modules.

7.1.3 Functional Testing

Functional testing verifies that the application meets the functional requirements as outlined in the business and technical documentation.

Key Features:

- Ensures valid input is accepted and invalid input is rejected.

- Tests key system functions for expected behavior.
- Verifies correct output based on provided input.
- Ensures correct interaction with external systems or interfaces.

7.1.4 System Testing

System testing examines the complete software system to validate its overall behavior. It ensures the system is configured correctly and produces known and predictable results.

Key Features:

- Tests the entire software application as a whole.
- Focuses on process flows, integration points, and expected system behaviors.
- Conducted after integration testing.

7.1.5 White Box Testing

White box testing involves testing software with knowledge of its internal structures and code implementation.

Key Features:

- Evaluates internal logic and code structure.
- Requires in-depth knowledge of the software.
- Used to test areas that cannot be reached using black box testing.

7.1.6 Black Box Testing

Black box testing is performed without knowledge of the internal workings of the application. Testers focus on inputs and expected outputs without considering the implementation.

Key Features:

- Does not require code knowledge.
- Based on functional specifications and requirements.
- Helps ensure that software functions correctly from an end-user perspective.

7.2 TEST STRATEGY

Testing is performed using both manual and automated approaches to ensure comprehensive coverage.

Key Testing Objectives:

- Validate that all fields accept correct input values.
- Ensure links and navigation work as expected.
- Ensure input screens, messages, and responses load without delays.

Features to be Tested

- Correct format validation for all input fields.
- Prevention of duplicate entries.
- Verification that all links navigate to the correct pages.

7.2.1 Integration Testing

Integration testing verifies that multiple software components interact without defects.

Key Features:

- Detects interface and communication errors between components.
- Ensures smooth data exchange between integrated modules.

Test Results

- All test cases executed successfully.
- No defects encountered during testing.

7.2.2 Acceptance Testing

User Acceptance Testing (UAT) is conducted to confirm that the system meets business requirements and user expectations before final deployment.

Key Features:

- Ensures all functional requirements are met.
- Requires active participation from end-users.
- Confirms the system is ready for production deployment.

Test Results:

- All acceptance test cases passed successfully.
- No defects encountered, confirming system readiness.

7.3 TEST CASES:

Test Case ID	Feature / Module	Test Scenario	Expected Result	Result
TC-01	User registration	User tries to register with details required	Registration should be successful	Paas
TC-02	User Authentication	User enters invalid login details	System displays error and denies access	Pass
TC-03	Task Submission	User uploads a valid Word document	File uploaded and task created successfully	Pass
TC-04	Blockchain Recording	Store document hash on blockchain	Transaction ID (TX ID) generated successfully	Pass
TC-05	Admin Verification	Admin verifies task using TX ID	Task marked as verified if hash matches	Pass
TC-06	File Download	User downloads verified document	PDF file downloaded successfully	Pass

Table 7.3 Test Cases

TestCase 1: User Registration

This test case validates that when a new user successfully registers through the registration form, the system displays a confirmation message indicating that registration is successful and that the user must wait for admin approval before accessing the system.

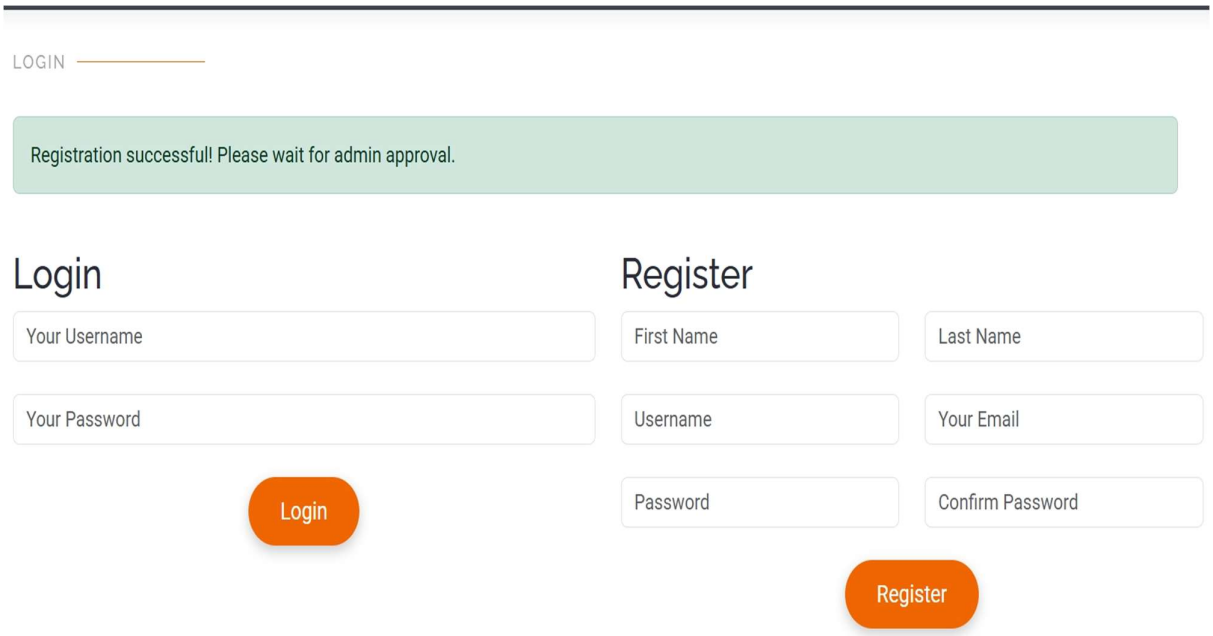

Figure 7.3.1 User Registration

TestCase 2: User Authentication

his test case verifies the system's ability to securely handle invalid login attempts by validating user credentials against stored authentication data. When a user enters an incorrect username, password, or both, the system should deny access and prevent any unauthorized entry into the application.

The system must display a clear and user-friendly error message such as "Invalid username or password", informing the user that the authentication has failed without revealing sensitive information (e.g., whether the username or password is incorrect individually). This helps maintain security by avoiding information leakage.

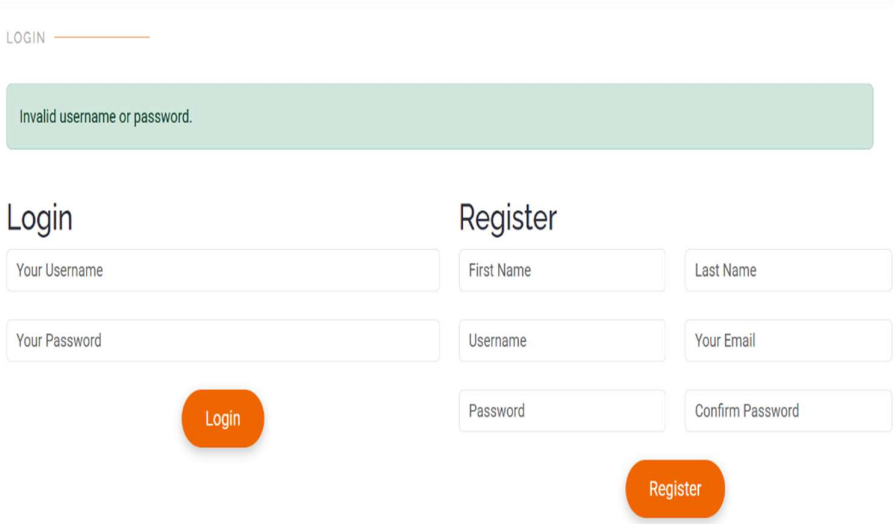

Figure 7.3.2 User Authentication

TestCase 3: User Submission

This test case verifies that the system correctly handles the submission of a task when a user uploads a valid Word document. The functionality ensures that the file upload process works as expected, allowing users to select and submit .doc or .docx files through the interface.

Upon clicking the Upload button, the system should successfully process the file, store it in the designated location, and create a corresponding task entry in the system. The uploaded document should then appear in the Uploaded Tasks section along with relevant details such as document name, status (e.g., *Pending*), and a generated blockchain transaction (TX) ID for traceability.

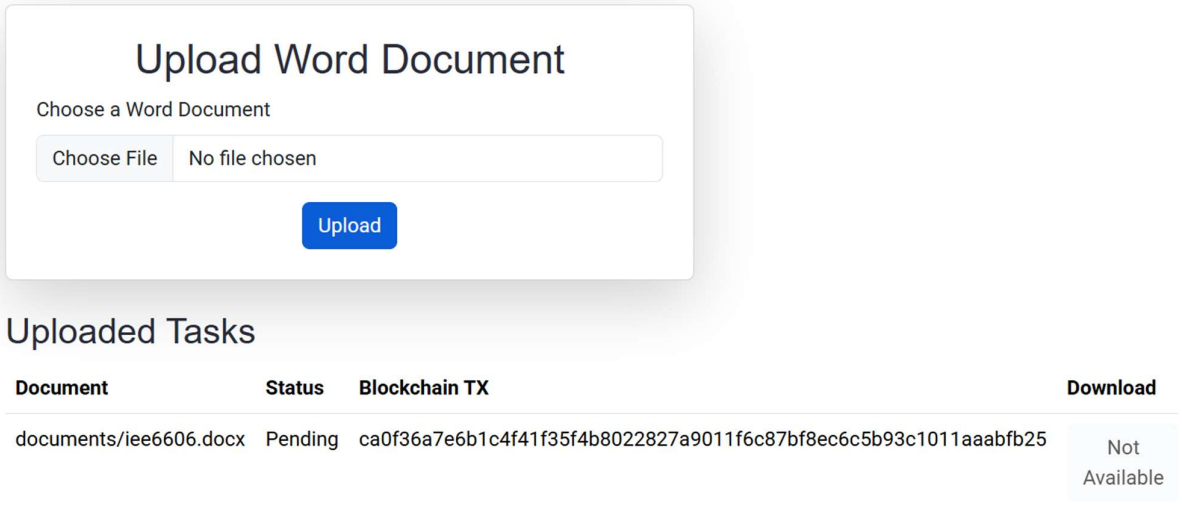

Figure 7.3.3 User Submission

TestCase 4: Blockchain Recording

This test case verifies that the system correctly records the uploaded document's hash on the blockchain

and generates a valid transaction ID (TX ID). After a user submits a document, the system should compute a unique cryptographic hash of the file and store it securely using a blockchain transaction.

Once the transaction is executed , the system should receive a transaction hash , which acts as a permanent and tamper-proof reference for the uploaded document. This TX ID should then be stored in the application database and displayed in the Uploaded Tasks section for user visibility and verification.

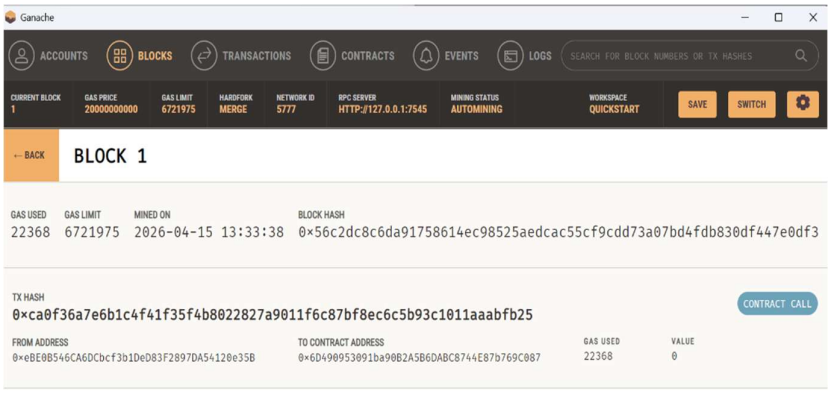

Figure 7.3.4 Blockchain Recording

TestCase 5 : Admin Verification

This test case verifies that the admin can successfully validate a user-submitted task by checking the document's integrity using the stored blockchain transaction ID . When a task is in a *Pending* state, the admin initiates the verification process by clicking the Verify button.

The system retrieves the corresponding transaction details from the blockchain Ganache using the TX ID and compares the stored document hash with the hash of the uploaded file. If both hashes match, it confirms that the document has not been altered and is authentic.

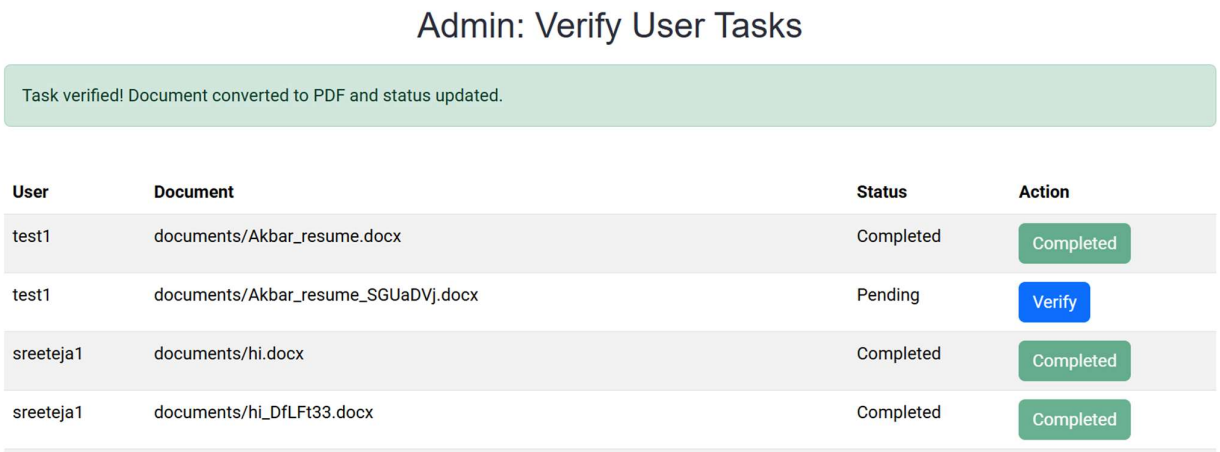

Figure 7.3.5 Admin Verification

TestCase 6: File Download

This test case verifies that a user can successfully download a verified document after it has been approved by the admin. Once the task status is marked as Completed, the system should enable the Download PDF option for the corresponding document.

When the user clicks on the Download PDF button, the system should retrieve the converted PDF version of the document from the server and initiate the download process. The downloaded file should match the verified document and maintain content integrity.

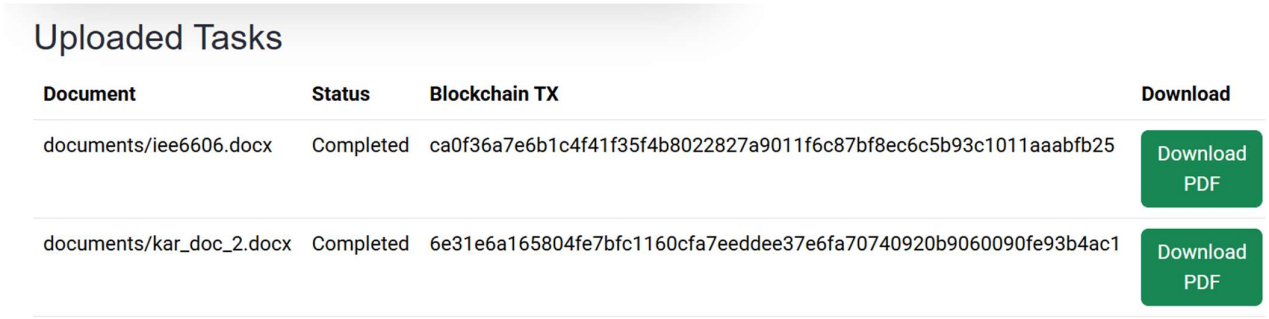

Figure 7.3.6 File Download

CHAPTRER – 8

OUTPUT SCREENS

8. OUTPUT SCREENS

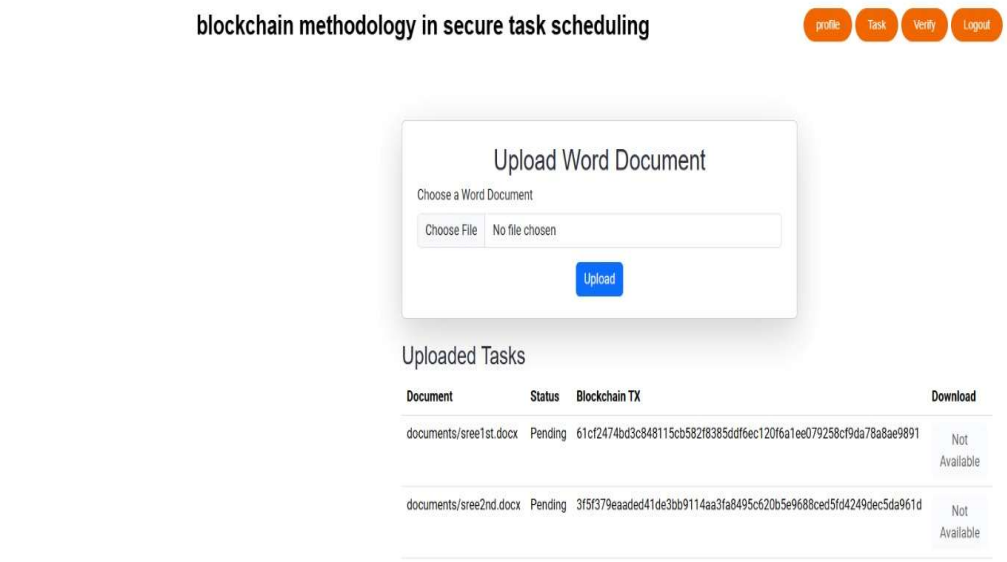

Figure 8.1 User Task Upload Interface

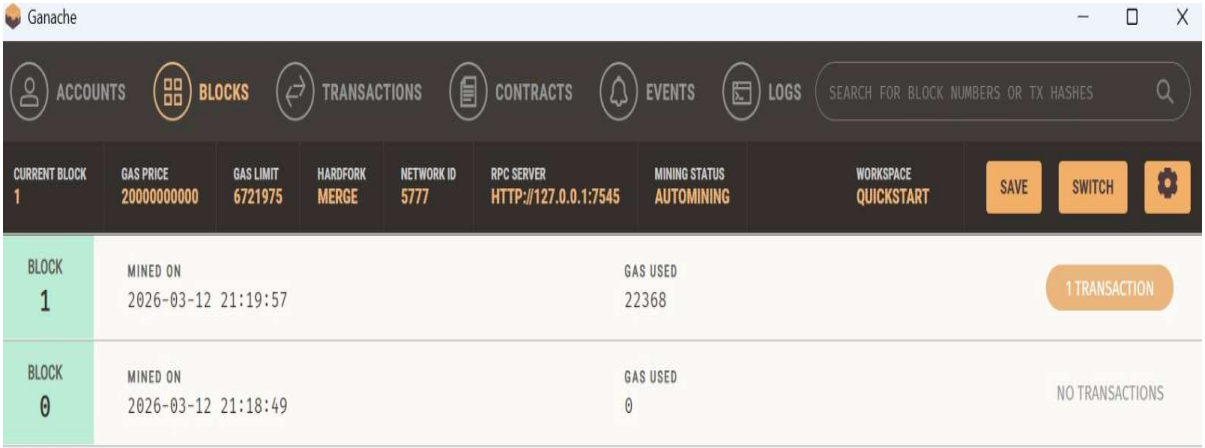

Figure 8.2: Blockchain Blocks View (Ganache)

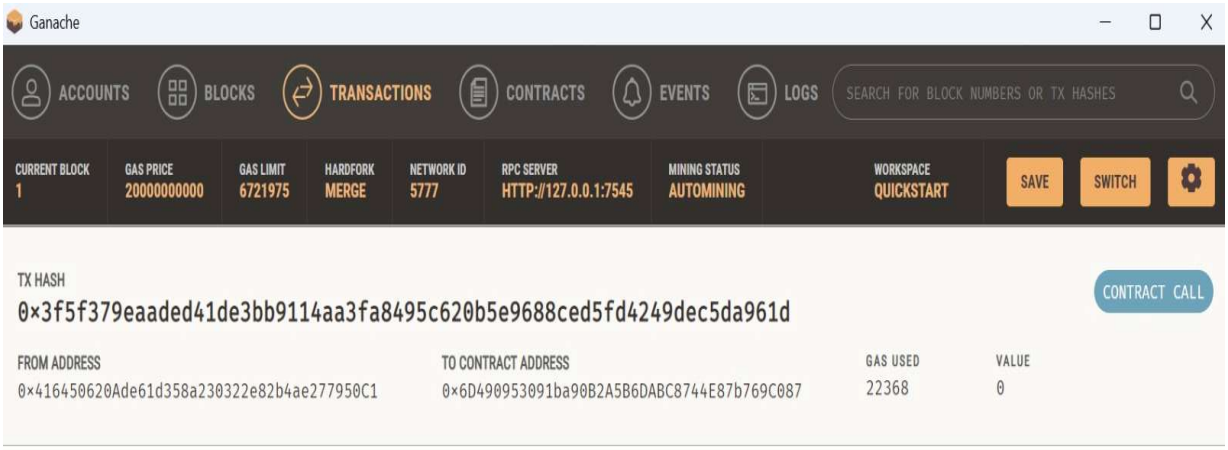

Figure 8.3 Blockchain Transaction Details

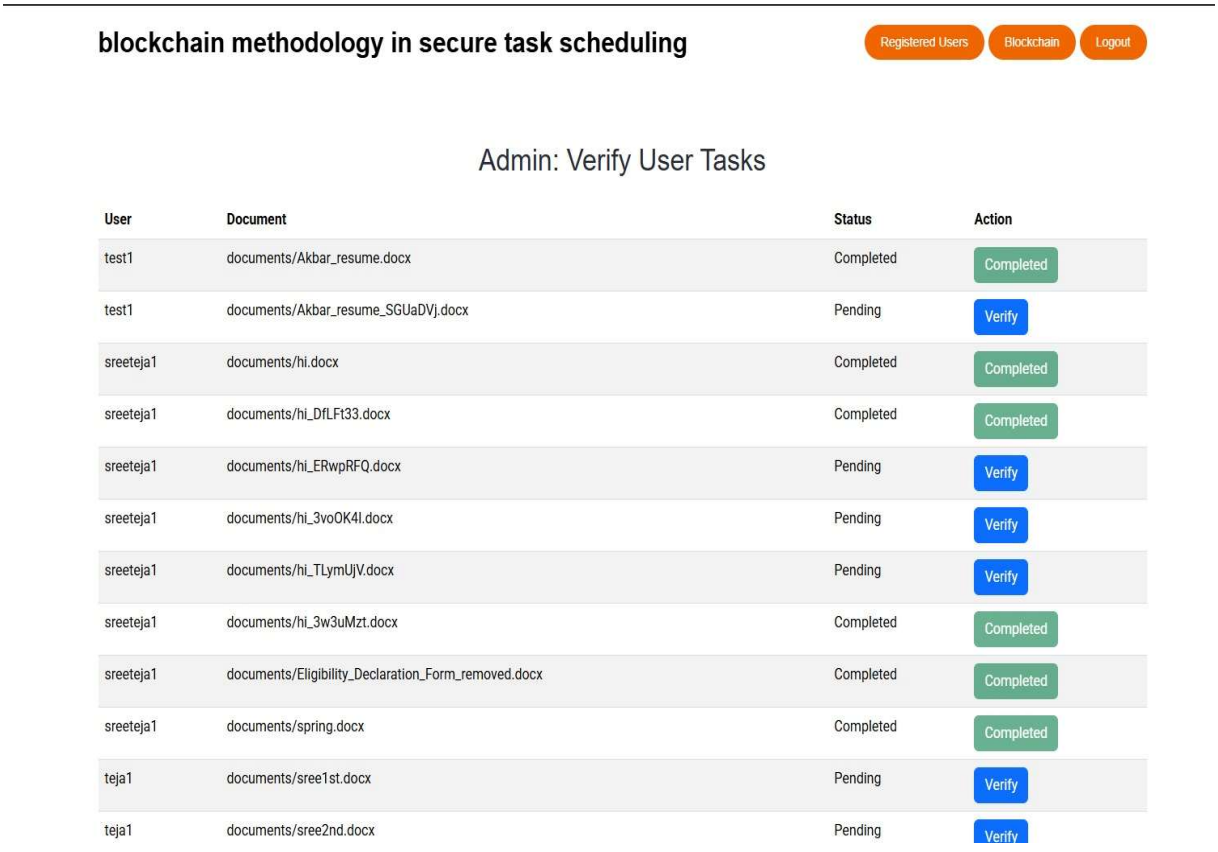

Figure 8.4 Admin Task Verification Dashboard

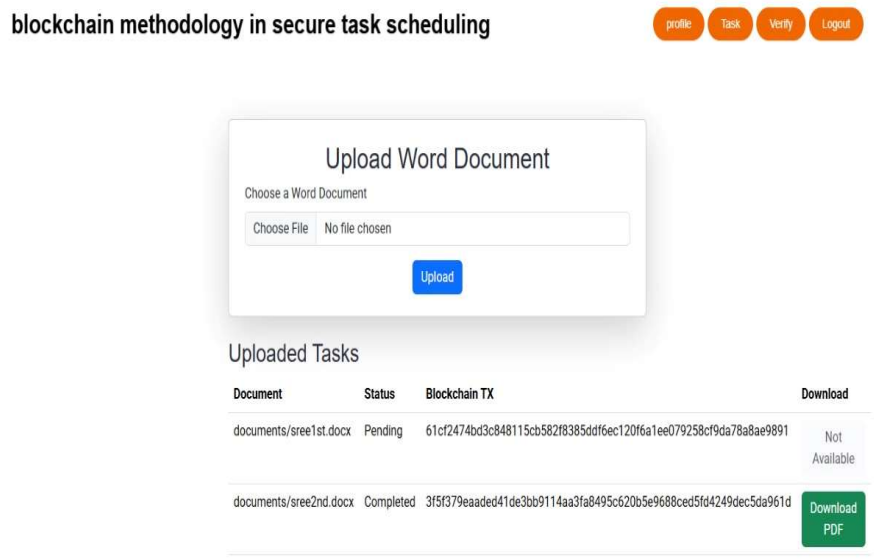

Figure 8.5 Final Output with PDF Download

CHAPTER -9

CONCLUSION

9. CONCLUSION

The proposed blockchain-based cloud task scheduling system demonstrates a transformative approach to enhancing the security, transparency, and reliability of cloud computing workflows. By shifting from a centralized scheduling architecture to a decentralized blockchain framework, the system effectively mitigates major challenges such as single-point-of-failure, unauthorized data manipulation, and trust dependency on a central authority. Smart contracts automate scheduling decisions, ensure tamper-proof execution logs, and provide verifiable task outcomes, thereby fostering a higher level of trust between users and cloud service providers. The incorporation of decentralized storage further strengthens data integrity, while AI-driven optimization algorithms contribute to improved resource utilization, reduced execution delays, and adaptive task distribution in dynamic cloud environments.

However, despite its promising capabilities, the system still encounters certain limitations. The computational overhead introduced by consensus mechanisms and smart contract execution can impact performance, especially during high-load operations. Additionally, integrating blockchain into large-scale enterprise cloud infrastructures requires careful consideration of scalability, network latency, and interoperability with existing systems. These challenges highlight the need for continual improvements and performance-oriented enhancements.

Looking forward, the proposed future extensions—such as the implementation of Layer-2 scaling solutions, hybrid blockchain models combining public and private networks, upgradeable smart contracts, and AI-enabled threat detection frameworks—offer strong potential to overcome current constraints. These advancements can significantly boost throughput, reduce operational costs, and enhance system resilience against emerging cyber threats.

CHAPTER – 10

FUTURE ENHANCEMENTS

10. FUTURE ENHANCEMENTS

In future developments, the proposed blockchain-based task scheduling system can be significantly improved through various scalability, intelligence, and security enhancements. To address the growing number of blockchain transactions and avoid network congestion, Layer-2 scaling solutions such as Optimistic Rollups and zk-Rollups can be adopted to batch multiple transactions, thereby improving throughput and reducing costs. A hybrid blockchain architecture can further optimize system performance by storing sensitive or bulky task information on private ledgers or decentralized storage platforms like IPFS, Arweave, or Filecoin, while recording only verification hashes on the blockchain. Additionally, integrating AI and ML-based optimization would enable intelligent task scheduling through automated resource allocation, predictive analytics for priority estimation, and AI-driven document verification before blockchain submission. Smart contract capabilities can also be enhanced by introducing upgradeable proxy contracts for flexible updates, multi-signature verification for collaborative validation, and gas-efficient execution strategies using batching or migration to low-cost blockchain networks. From a security standpoint, the system can incorporate Zero-Knowledge Proofs to verify task integrity without revealing sensitive information, blockchain-based decentralized identity (DID) mechanisms to enhance access control, and AI-powered threat detection to identify suspicious activity. These improvements collectively contribute to higher scalability, stronger security, reduced operational costs, and an overall smoother user experience, helping the system evolve into a more robust and future-ready platform.

REFERENCES

REFERENCES

1. Chen, Y., Zhao, L., & Wang, H. (2026). Adaptive Blockchain-Based Cloud Task Scheduling Using Deep Reinforcement Learning. *IEEE Transactions on Cloud Computing*, 14(1), 45–60.
2. Ahmed, S., & Rao, V. (2025). Blockchain-Assisted Cloud Task Scheduling with Reinforcement Learning Optimization. *IEEE Transactions on Parallel and Distributed Systems,* 36(1), 112–126.
3. Li, H., Kumar, R., & Das, S. (2025). Enhancing Cloud Security Using Zero-Knowledge Proofs in Blockchain-Based Task Management. *Journal of Cloud Computing Research,* 14(2), 55–72.
4. Fernandes, P., & Gupta, A. (2024). Layer-2 Scaling Techniques for Efficient Blockchain-Integrated Cloud Services. *Future Generation Computer Systems,* 152, 210–225.
5. Tanvir, M., & Chowdhury, Z. (2024). AI-Powered Threat Detection in Decentralized Cloud Environments. *ACM Transactions on Cloud Computing,* 12(3), 98–120.
6. Morris, J., & Lee, K. (2024). Smart Contract Upgradability Models for Dynamic Cloud Task Scheduling. *IEEE Access,* 12, 65341–65359.
7. Patel, A., & Shah, R. (2022). Enhancing Cloud Task Management with Decentralized Smart Contracts and AI Models. *Journal of Cloud Security Research,* 7(4), 78–94.
8. Meng, W., & Li, J. (2022). Smart Contract-Based Secure Scheduling in Cloud Computing. *IEEE Access,* 10, 9832–9845.
9. Kshetri, N. (2021). Blockchain and Trust in Cloud Computing: A Comprehensive Study. *Journal of Cloud Computing,* 9(1), 32–48.
10. Xu, X., Zhang, Q., & Yang, W. (2021). Enhancing Cloud Task Scheduling Using Blockchain and PBFT Consensus. *Future Generation Computer Systems,* 124, 67–79.
11. Sharma, R., & Patel, D. (2020). Blockchain Integration in Cloud Computing: A Review on Security and Privacy Challenges. *ACM Computing Surveys,* 52(6), 112–130.
12. Banerjee, S., & Roy, D. (2020). Task Scheduling in Cloud Computing: Security Challenges and Solutions. *Proceedings of the International Conference on Cloud Security,* 201–213.
13. Zhang, F., Wang, J., & Liu, Y. (2019). Trusted Routing Scheme Using Blockchain for Cloud Security. *IEEE Transactions on Cloud Computing,* 7(3), 458–469.
14. She, W., & Liu, X. (2019). Blockchain-Based Trust Model for Malicious Node Detection in Cloud Computing. *International Journal of Computer Applications,* 182(45), 12–20.
15. Singh, A., & Chana, I. (2016). A Survey of Task Scheduling Techniques in Cloud Computing. *Journal of Supercomputing,* 72(5), 1423–1457.
16. Wood, G. (2014). *Ethereum: A Secure Decentralized Generalized Transaction Ledger.* Ethereum

Project Yellow Paper.

17. Vitalik Buterin. (2013). *Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform.* Retrieved from https://ethereum.org/en/whitepaper/
18. Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog Computing and Its Role in the Internet of Things. *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing,* 13–16.
19. Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Retrieved from https://bitcoin.org/bitcoin.pdf
20. Hyperledger Foundation. (2022). *Hyperledger Fabric Documentation.* Retrieved from https://hyperledger.org/fabric
21. Gupta, H., & Kumar, P. (2021). Artificial Intelligence-Driven Optimization in Blockchain-Based Cloud Task Scheduling. *Future Internet,* 13(5), 102–117.