eISSN 2783-6851

VILNIUS TECH
Vilnius Gediminas Technical University

NEW TRENDS in COMPUTER SCIENCES

2026
Volume 4
Issue 1
Pages 40–59
https://doi.org/10.3846/ntcs.2026.26620

A SECURE RESTFUL DEPLOYMENT PATTERN FOR COURSE RECOMMENDATION MICROSERVICES USING PROOF-OF-WORK BLOCKCHAIN AUTHENTICATION

Bernadus Very CHRISTIOKO [1] ✉, Khoirudin [2], Yusup[3]

[1,2]*Informatics Engineering, Universitas Semarang, Semarang, Indonesia*
[3]*Information System, Universitas AKI, Semarang, Indonesia*

Article History:
- received 9 April 2026
- accepted 8 July 2026

Abstract. Deploying machine learning models across distributed environments requires robust architectural patterns that ensure both cross-platform interoperability and secure access control. While Representational State Transfer (REST) APIs are widely adopted for service exposure, existing implementations in cross-machine scenarios remain vulnerable to token spoofing and post-hoc administrative tampering of centralized authorization logs. Therefore, this study presents a secure, deployable architectural pattern that integrates localized machine learning workflows with a decentralized authentication boundary. An online course recommendation system using text preprocessing, Term Frequency–Inverse Document Frequency (TF–IDF) vectorization, and cosine similarity is implemented and utilized explicitly as a functional demonstration case to validate the framework. The core machine learning engine is exposed via a Flask-implemented RESTful service layer, while secure access control is enforced by embedding a Proof-of-Work (PoW) blockchain token authentication mechanism at the API boundary. The empirical results demonstrate that access to protected endpoints is granted only upon successful validation of user tokens against immutable on-chain ledger records, structurally rejecting unauthorized requests. Performance benchmarking indicates that while the decentralized ledger operations introduce an operational latency trade-off (a 1113 ms mining overhead and 2.45-second end-to-end authorized query response), the proposed pattern provides an effective, tamper-evident firewall for delivering secure microservices in distributed client–server environments.

Keywords: REST API, deployment architecture, machine learning microservices, blockchain, proof-of-work authentication, content-based filtering.

✉Corresponding author. E-mail: ***very@usm.ac.id***

1. Introduction

Since the advancement of the global shift toward digital education accelerated by the global pandemic, online learning has become an integral paradigm in modern higher educational landscape. This paradigm shift has led to an exponential growth of online courses across various massive open online course (MOOC) platforms, such as Udemy, edX, and Coursera. However, this rapid expansion has inadvertently triggered an information overload, with thousands of courses available across diverse domains and skill categories. Consequently, learners face a significant challenge in filtering and identifying the most relevant courses that align with their specific academic or professional needs. To mitigate this information surplus and enhance the personalization of digital learning, the deployment of an effective Recommendation System has emerged as a crucial solution (Fayyaz et al., 2020; Thakur, 2021).

In the educational field, content-based recommendation systems are the most used technique for personalizing content, such as in the e-learning process (Bajpai et al., 2022). In recent years, recommendation systems used to support teaching and learning processes have received greater attention (da Silva et al., 2023; Pal & Dahiya, 2023). A content-based recommendation system is a type of recommendation system that suggests items to users based on the similarity between the suggested items and items that have been liked or chosen previously (Anand & Nath, 2020; Mohanty et al., 2020). Recommendation Systems use machine learning algorithms to provide product or service recommendations to users (Parvatikar & Parasar, 2021; Portugal et al., 2018). Several machine learning techniques have been used to build recommendation systems, including Term Frequency (TF) and Inverse Document Frequency (IDF) techniques (Javed et al., 2021; Li, 2024; Ni et al., 2021). In the education domain, the TF-IDF method and cosine similarity are widely used to obtain recommendations, classifications, and retrievals. Previous studies have used these techniques to generate seminar recommendations (Yusuf & Cherid, 2020), recommend admissions for new students at private campuses (Apriani et al., 2021), classify thesis documents (Wahyuni et al., 2017), classify journals (Habibi & Cahyo, 2020), and test the similarity of final project title sentences (Mawanta et al., 2021).

Recently, Python has become a dominant ecosystem for machine learning, data science, and scientific computing because of its extensive libraries and productivity benefits, where data processing, model building, and performance evaluation are typically conducted during the training phase (Raschka et al., 2020). In recommendation systems, models can be implemented and validated in Python; however, operational use requires an interface that enables other applications to consume the model outputs.

Several studies have addressed this architectural integration by exposing recommender system functionalities through standard RESTful APIs, facilitating cross-platform consumption by web or mobile clients (García & Bellogín, 2018; Jain & Kakkar, 2019; Kumar et al., 2022). Nevertheless, these conventional API frameworks predominantly operate under the assumption of a centrally trusted network environment, focusing primarily on interoperability and functional interface exposure. Consequently, existing literature falls short of providing a detailed security mechanism for endpoints in true cross-machine deployment scenarios. Traditional centralized access control lists (ACLs) and database-backed authentication layers remain highly vulnerable to post-hoc administrative tampering and token spoofing at the network boundary. Without a verifiable, immutable ledger to audit access logs, ensuring absolute data integrity across distributed nodes becomes inherently problematic.

To address these critical architectural vulnerabilities, this study proposes a deployable security pattern utilizing decentralized ledger operations to reinforce cross-machine authorization boundaries. Rather than claiming a definitive resolution to all network vulnerabilities, this research introduces a robust, integration-ready framework that pairs a Python-trained machine learning model with a Proof-of-Work (PoW) blockchain verification layer (Rani & Bhambay, 2023). In this context, an online course recommendation engine is implemented and utilized explicitly as a functional demonstration case to validate the structural feasibility, systemic interoperability, and security enforcement of the proposed model within a distributed client–server environment.

2. Methodology

This study follows a systematic six-phase methodology to ensure scientific rigor and technical auditability, as shown in Figure 1. Phases 1 and 2 focus on conceptualizing the framework by identifying research gaps in secure cross-machine access and establishing the mathematical foundations for TF-IDF and cosine similarity metrics. Phase 3 involves the acquisition of the Coursera Courses Dataset 2021 and a preprocessing pipeline that isolates high-density features (Course Name and Skills) to mitigate semantic noise.

Subsequently, Phases 4 and 5 cover the operationalization of the prototype as a Flask-based RESTful service integrated with a blockchain security boundary, followed by empirical evaluations of mining latency and API enforcement. Finally, Phase 6 synthesizes the findings, balancing the trade-offs between security integrity and operational performance.

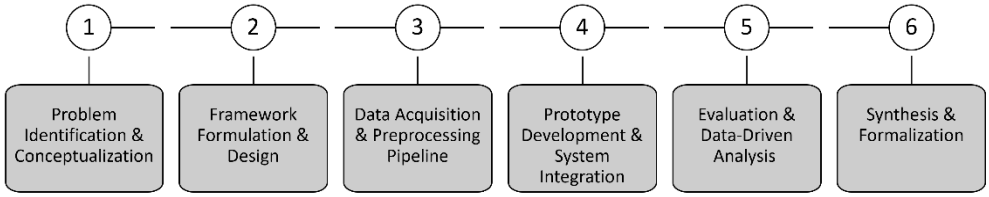

Figure 1. Research stages

2.1. Proposed system framework and architecture

The conceptual framework and operational architecture of the proposed system are illustrated in Figure 2. The diagram is divided into two operational panels to reflect both the process and technical components. Panel A illustrates the research workflow, starting from dataset preparation, recommendation model construction, and REST API development. Meanwhile, Panel B shows the main REST API components and their interaction, including the REST client, the REST API server, the recommendation module, and the blockchain-based authentication module.

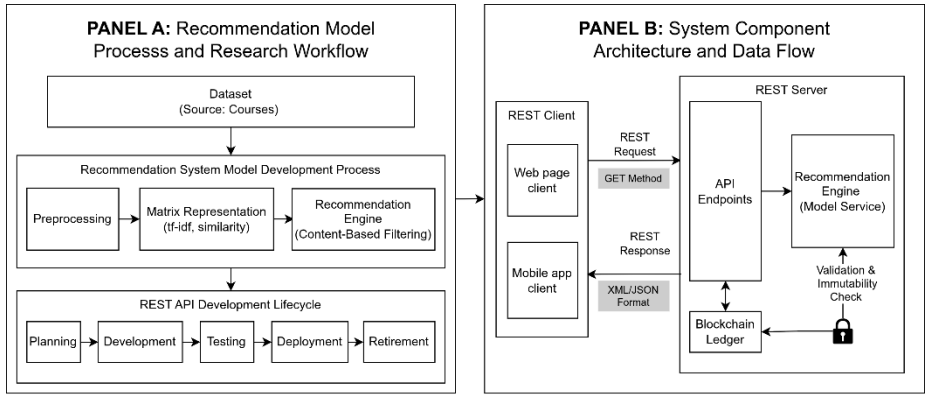

Figure 2. Proposed framework of the secure RESTful course recommendation system

2.2. Data acquisition and preprocessing

Following the research workflow illustrated in Figure 1, the data acquisition and preprocessing phase is conducted as the initial step in preparing data for the recommendation system model. This phase involves the systematic acquisition of the dataset, which serves as the primary empirical basis for the recommendation engine and textual data cleansing, to ensure high-quality data input for vectorization and similarity computation.

1. Dataset

To build the content-based recommendation model, we utilized the Coursera Courses Dataset 2021, a public repository containing 3,522 academic course profiles, as shown in Figure 3. The unstructured textual dataset features a tabular format with seven main attributes: Course Name (string), University (string), Difficulty Level (categorical), Course Rating (numerical), Course URL (string), Course Description (string), and Skills (string). Feature selection was conducted to isolate Course Name and Skills as the core textual attributes for indexing, as they represent the most explicit domain-specific signals for keyword alignment (Kapoor, 2021).

Coursera.csv (5.28 MB)

Detail Compact Column

Course Na...	University	Difficulty L...	Course Ra...	Course URL	Course De...	Skills
Write A Feature Length Screenplay For Film Or Television	Nichigan State University	Beginner	4.8	https://www.coursera.org/learn/write-a-feature-length-screenplay-for-film-or-television	Write a Full Length Feature Film Soript In this course, you will write a complete, feature-length s...	Drama Comedy peering screenwriting film Document Review dialogue creative writing Writing u...
Business Strategy: Business Model Canvas Analysis with Miro	Coursera Project Network	Beginner	4.8	https://www.coursera.org/learn/canvas-analysis-miro	By the end of this guided project, you will be fluent in identifying and creating Business Model Can...	Finance business plan persons (user experience) business model canves Planning Business projec...

Figure 3. Coursera courses dataset 2021

2. Preprocessing

The recommender system model requires a structured preprocessing pipeline to prepare the unstructured text for vectorization and similarity computation. During the initial feature selection stage, only Course Name and Skills are retained, as they represent the primary textual signals for content-based matching. The subsequent cleaning pipeline, consisting of case folding, empty-string handling, and stopword removal, is applied specifically to the Skills field to standardize character casing, manage missing data, and eliminate high-frequency terms that contribute negligible semantic value. By reducing noise and improving textual consistency, this stage ensures that the dataset is optimized for accurate similarity-based retrieval in the subsequent modeling phases.

2.3. Prototype development and system integration

This phase operationalizes the conceptual framework by orchestrating the structural integration between the core machine learning components and the deployment infrastructure, as mapped in the technical architecture. The implementation synthesizes the recommendation system model into a cohesive RESTful service layer, enabling the delivery of the engine to heterogeneous clients.

1. Recommendation system model

The Recommendation System model was developed through a series of machine learning modeling stages and implemented using the Python programming language. The modeling pipeline consists of the following stages.

- Matrix Representation

After preprocessing, the cleaned text was transformed into numerical features using a matrix-based representation. Specifically, this study applies Term Frequency–Inverse Document Frequency (TF–IDF) to produce a Compressed Sparse Row (CSR) matrix, which efficiently stores sparse term-weight vectors for each document. A cosine similarity function is then computed over the TF–IDF vectors to generate a similarity matrix that is used to retrieve the most relevant items. TF–IDF is a statistical weighting scheme that measures the importance of a term (or term group) in a document relative to the entire corpus; terms that appear frequently in a particular document but less frequently across documents receive higher weights (Wahyuni et al., 2017).

Term Frequency (TF) is the number of occurrences of a certain term t in document d, as formulated in Eq. (1):

$$TF_{t,d} = \begin{cases} 1+\log\left(f_{t,d}\right), & f_{t,d} > 0 \\ 0, & f_{t,d} = 0 \end{cases}. \tag{1}$$

Meanwhile, the Inverse Document Frequency (IDF) is the number of documents d in the corpus divided by the frequency of the text, as shown in Eq. (2):

$$IDF_t = \log\left(\frac{N}{df_t}\right). \tag{2}$$

TF–IDF determines the significance level of a term compared to the text in a sequence or corpus. To calculate TF–IDF, we used Eq. (3) below:

$$TFIDF_{t,d} = TF_{t,d} \times IDF_t. \tag{3}$$

In Eqs. (1)–(3), t denotes a term, d denotes a document, $f_{t,d}$ is the raw frequency of term t in document d, df_t is the document frequency of term t, and N is the total number of documents in the corpus. Once the TF–IDF vectors are obtained, cosine similarity is used to measure the similarity between two vectors in the same feature space. For non-negative TF–IDF weights, the similarity value is bounded between 0 and 1, where values closer to 1 indicate a higher similarity. The cosine similarity is computed using Eq. (4):

$$Similarity = \cos\left(\theta\right) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n}\left(A_i\right)^2} \cdot \sqrt{\sum_{i=1}^{n}\left(B_i\right)^2}}. \tag{4}$$

In Eq. (4), A and B are TF–IDF vectors, A_i and B_i are the weights of the i-th term in vectors A and B, respectively, and n is the number of terms (dimensions) in the vector space.

- Recommender Engine

The recommendation engine is designed as a high-level functional module that facilitates keyword-based retrieval within the proposed framework. This module operates as a core search feature by processing user-provided keywords as dynamic input parameters to identify relevant course candidates. The retrieval logic is executed through automated similarity calculations that compare the input vector against the pre-computed feature matrix generated in the previous stage.

By prioritizing 'Course Name' and 'Skills' as high-density features, the engine ensures a precise keyword alignment while minimizing the semantic noise often associated with lengthy course descriptions. This modular design allows the recommendation logic to remain decoupled from the delivery layer, ensuring that the engine consistently returns semantically relevant course records as a structured data output.

2. REST API development

An API enables various software to interoperate, and because an API is also software, its development should follow a clear life cycle. In this study, a REST API was developed through the stages of the API lifecycle as follows (Bigelow, 2020):

- Planning, API development begins with planning and design. At this stage, we identified which recommendation functions should be exposed and specified the API interface using OpenAPI Standard (OAS) v3 (OpenAPI Initiative, 2020).
- Development, during the development stage, the API was implemented in Python. Because the machine learning model in the recommendation system is executed in Python, the API is also written in Python using the FLASK framework.
- Testing, at this stage, the API is tested to ensure that the functions of the recommendation system exposed through the API work according to the plans and designs, using the Postman client.
- Deployment, after the API is tested and stabilized, and is in accordance with the plan and design, the API is ready to be deployed on the production server and becomes accessible to clients via REST.

3. REST server API security

In this study, blockchain authentication was implemented as a token registration and verification workflow. Before accessing the protected recommendation endpoints, the client creates a new blockchain transaction containing a unique token, which is then mined into a new block using the Proof-of-Work (PoW) mechanism. After mining, the complete chain can be retrieved for verification purposes. To access a secure API endpoint, the client includes the unique token in the HTTP Authorization header; the server validates the token against the blockchain record and returns recommendations only when the token is valid (Drescher, 2017; van Flymen, 2020). Figure 4 illustrates the use of the Proof-of-Work (PoW) blockchain in REST API authentication.

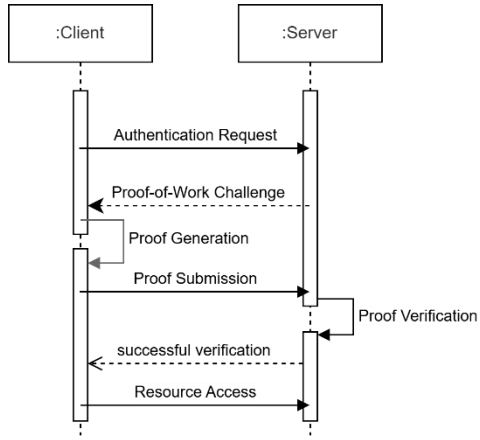

Figure 4. Proof-of-Work blockchain in REST API authentication

The blockchain layer in this study was designed to enforce authenticated access and provide integrity for token registration records through PoW-confirmed blocks. Accordingly, the security scope focuses on preventing unauthorized access to protected endpoints by requiring validation before response delivery. Other aspects of API security (e.g., confidentiality of traffic, network-level threats) are outside the scope of the blockchain mechanism and should be addressed through standard API hardening and secure transport in deployment environments.

2.4. Evaluation and data-driven analysis

This phase evaluates the operational interactions between the REST Client and the REST Server to validate systemic efficiency and security enforcement. The focus of this empirical evaluation is centered on verifying the behavioral correctness of inbound requests, specifically utilizing the HTTP GET method, and the structuring of outbound responses in the standardized JSON format. Furthermore, the analysis provides data-driven assessments of operational metrics, explicitly measuring system latency during token processing and the computational effectiveness of the server-side, Proof-of-Work (PoW) based authentication mechanism.

3. Result

This section presents the experimental findings and performance validation of the proposed framework, focusing on the integration of the recommendation engine and the decentralized authentication layer. The results are categorized into the functional performance of the recommendation system model and the operational evaluation of the blockchain-secured REST API.

3.1. Recommendation system model

The implementation of the recommendation system model follows a structured machine learning pipeline designed to transform raw course data into actionable insights. This process is divided into three phases.

1. Data preprocessing

Using the dataset as shown in Figure 3, the preprocessing pipeline's execution resulted in a refined dataset ready for vectorization and similarity analysis. Initial feature selection reduced the dimensionality of the dataset by focusing exclusively on the Course Name and Skills attributes, which were identified as high-density features that contain the most explicit information for keyword alignment. These attributes provide a concise yet semantically rich representation of the course content, which is crucial for accurate similarity matching. In contrast, the inclusion of Course Description was intentionally omitted because it often introduces significant noise, such as generic academic terms and repetitive stopwords, that contributes little to the core semantic value. On large-scale datasets, such noise can lead to a dilution of the feature vector, ultimately decreasing the Cosine Similarity scores and reducing the precision of the recommendation results. Therefore, prioritizing high-density attributes ensures a more robust and computationally efficient retrieval process. The outputs of each preprocessing step are shown in Figure 5.

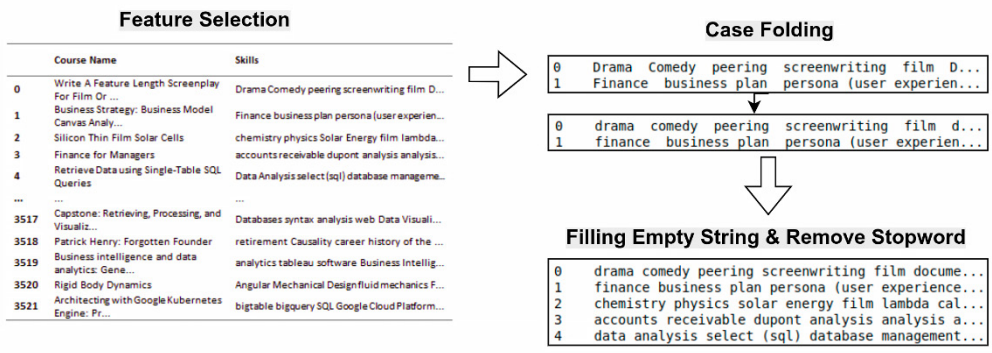

Figure 5. Data preprocessing results

Overall, Figure 5 highlights that preprocessing is a critical step in text-based recommendations. By standardizing the text and removing non-informative tokens, the dataset becomes more consistent and meaningful, which reduces noise and supports more accurate similarity-based retrieval in the later stages.

2. Similarity matrix analysis

After preprocessing, the processed textual corpus was mathematically vectorized into a Compressed Sparse Row (CSR) matrix using the TF-IDF scheme formulated in Eqs. (1)–(3), which efficiently stores sparse term-weight vectors for each document. Table 1 presents an excerpt from the CSR matrix, showing document–term indices and their corresponding TF–IDF values. For example, a term with index 3029 in document 0 has a TF–IDF value of 0.15640432, indicating its relative importance within that document compared to the corpus. The variation in TF–IDF weights across document–term pairs demonstrates that TF–IDF effectively differentiates informative terms based on frequency and distribution.

Table 1. Representation of the Compressed Sparse Row (CSR) matrix

Index (doc, word)	TF–IDF Value
(0, 3029)	0.15640432
(0, 440)	0.15640432
...	...
(3521, 1181)	0.065595692
(3521, 6538)	0.129972711

Next, cosine similarity (Eq. (4)) is computed over the TF–IDF vectors to obtain the similarity matrix shown in Table 2 in matrix form. The matrix quantifies the degree of similarity between the documents based on their TF-IDF vector representations. A maximum value of 1 in the table indicates perfect similarity between the documents, whereas a minimum value of 0 indicates no similarity. For instance, document A at index 0 has maximum similarity with itself (a value of 1) and a value of 0.033 with document B at index 2, indicating a slight similarity between them. This similarity matrix between documents enables the identification of relevant documents based on content and the refinement of search results, thereby increasing the efficiency and accuracy of the recommendation outcomes.

Table 2. Cosine similarity matrix

Doc	0	1	2	...	3519	3520	3521
0	1	0	0.033	...	0	0	0
1	0	1	0	...	0.0827	0	0.006
2	0.033	0	1	...	0	0.142	0.003
...	...	...	...	...	...	...	...
3521	0	0.006	0.003	...	0.0141	0.003	1

3. Recommendation engine

As illustrated in Algorithm 1, the recommendation engine is implemented in Python as a keyword-based module by processing keywords sent through function parameters, utilizing the TF-IDF matrix representation and similarity from the previous stage.

Algorithm 1 Recommendation Engine

Require: title, cos

```
1    idx ← indices[title]
2    cos_scores ← list(enumerate(cos[idx]))
3    cos_scores ← sorted(cos_scores, key=lambda x: x[1], reverse=True)
4    cos_scores ← cos_scores[1:6]
5    course_indices ← [i[0] for i in cos_scores]
```

Return df_org.iloc[course_indices]

The Recommendation function in Algorithm 1 aims to provide recommendations based on course titles. This function requires a cosine similarity matrix (cos) and a series data dictionary (indices), where the keys are course titles and the values are the corresponding numerical indices of the courses in the dataset (df) in a dataframe format. Line 1 searches for the numerical index corresponding to the title in the index dictionary and stores it in

idx. Line 2 creates a list of (index, cosine value) pairs for the row in the cosine matrix that matches idx. Line 3 sorts the cos_scores list based on the cosine value (x[1]) from highest to lowest, and line 4 takes the top five values from the cos_scores. Meanwhile, line 5 creates a course_indices list containing only the course indices from the top five values in cos_scores, and line 6 returns the course records from the dataset dataframe (df_org) corresponding to the course_indices as a recommendation list.

3.2. REST API and security integration

The operationalization of the proposed framework is realized through the systematic integration of a RESTful service layer and a decentralized security boundary. This section details the lifecycle of the API implementation, transitioning from conceptual design to a production-ready environment secured by blockchain technology.

1. Planning

The interoperability of the proposed architecture is formally established through a structured API design that adheres to the OpenAPI Standard (OAS), providing a language-agnostic interface for diverse client platforms. As summarized in Table 3, the system's endpoints are logically bifurcated into two primary functional layers: a decentralized authentication layer (Blockchain) and a recommendation service layer (Recommender System). The technical specifications, including input parameters and the corresponding response schemas and the functional roles of each endpoint in maintaining a secure and auditable distributed environment. The authentication endpoints utilize blockchain technology to enforce secure access through Proof-of-Work (PoW) based transaction logging and immutable block mining, ensuring a robust security boundary. Simultaneously, the recommender endpoints facilitate granular content retrieval, ranging from comprehensive course inventory access to specific filtering by university or difficulty level, and culminating in keyword-driven recommendation generation. This dual-layered design ensures that the system supports heterogeneous client requirements while maintaining verifiable access integrity.

Table 3. API design (OpenAPI)

Category	Endpoint	Method	Key Parameters (Input)	Abstracted Response Schema (Output)	Research Role and Functional Objective
Decentralized Authentication (Blockchain)	`/new_transaction`	`POST`	sender, recipient, amount (JSON Body)	message, status_code	Facilitates the registration of user-specific access tokens as immutable ledger entries.
	`/mine`	`GET`	None	index, proof, previous_ hash, transactions	Validates pending token transactions through the Proof-of-Work (PoW) consensus mechanism.

End of Table 3

Category	Endpoint	Method	Key Parameters (Input)	Abstracted Response Schema (Output)	Research Role and Functional Objective
	`/chain`	`GET`	None	chain (array of blocks), length	Provides a transparent audit trail of the entire sequence of authorization events.
Recommender System Service Layer	`/recommendations`	`GET`	key (query), Authorization (header)	List of Recommended Courses (JSON Array)	Executes semantic matching via TF-IDF and Cosine Similarity based on dynamic user keywords.
	`/courses`	`GET`	Authorization (header)	Complete Course Inventory (JSON Array)	Handles the retrieval of the primary course inventory with protected access enforcement.
	`/univ & /levl`	`GET`	key (query)	Filtered Course Records (JSON Array)	Enables granular course filtering based on specific university categories or difficulty levels.
	`/universities & /levels`	`GET`	None	Categorical Metadata List (JSON Array)	Retrieves categorical metadata to support system-wide navigation and filtering.

2. Development

The development phase involved the implementation of a RESTful service layer using the Flask framework to operationalize the recommendation engine and blockchain authentication logic. The system architecture is designed to expose core functionalities as language-agnostic endpoints, ensuring that each request-response cycle follows a standardized JSON data format. The implementation logic is formally represented through the functional workflows illustrated in Figure 6 and Figure 7.

As modeled in Figure 6, the recommendation service implements a middleware-based authorization interceptor. Every incoming GET request at the `/recommendations` or `/courses` endpoints triggers an extraction of the token from the HTTP Authorization header. If the token is absent or fails validation against the blockchain record, the system immediately terminates the request with a `401 Unauthorized` response. Upon successful verification, the engine executes the semantic matching module, utilizing TF-IDF vectorization and cosine similarity, to retrieve and rank course candidates, which are then serialized into a JSON array for the client.

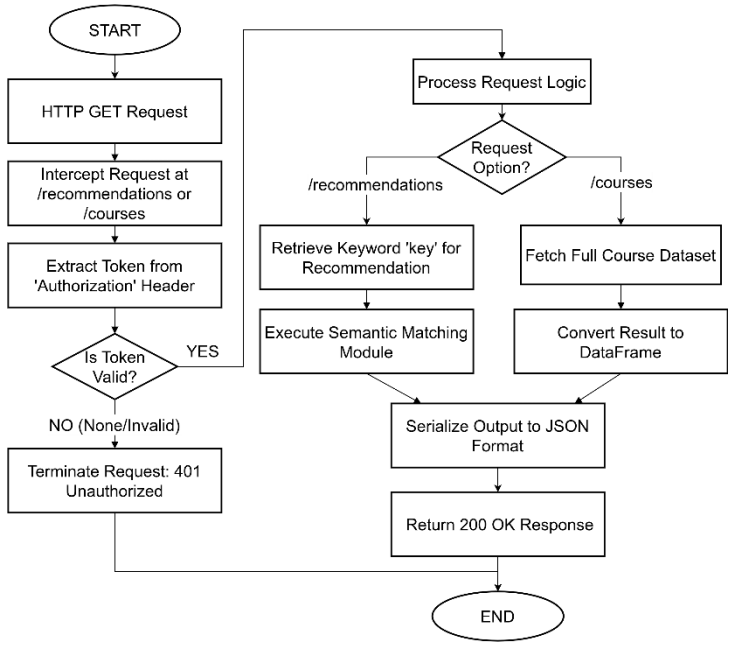

Figure 6. Protected recommendation workflow

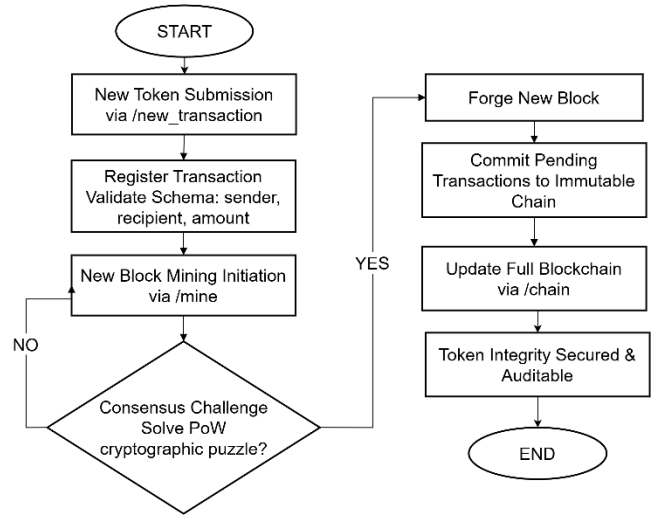

Figure 7. Blockchain security and ledger operations

The decentralized security layer is operationalized through a dual-pipeline routing interface as shown in Figure 7.

- Transaction Pipeline: The `/new_transaction` endpoint functions as a structural validator that mandates the presence of `sender`, `recipient`, and `amount` fields

within the JSON payload. Valid transactions are queued into the pending pool, returning a `201 Created` status.

- Consensus Pipeline: The `/mine` endpoint initiates the Proof-of-Work (PoW) consensus mechanism, solving a cryptographic puzzle to forge a new block. Once the proof is validated, the pending transactions are committed to the immutable chain, and the ledger state is updated to ensure a tamper-evident audit trail of all authorization events.

3. Testing

Comprehensive integration testing was conducted to verify the behavioral correctness of the API endpoints. Utilizing functional test logs, the system ensures that inbound requests comply with the defined schema and that the blockchain authentication mechanism accurately enforces access control.

- Testing and validation of blockchain authentication mechanism

To validate the functionality of the blockchain authentication mechanism, integration testing was conducted on the transaction registration endpoint. The test was performed by sending a `POST` request to the `/new_transaction` endpoint carrying the raw transaction payload in JSON format. The test results show that the system accurately processes the request and provides status responses in accordance with the HTTP protocol, as summarized in Table 4.

Table 4. Functional test results for transaction registration endpoint

Test Component	Specification / Value
Endpoint URL	`http://127.0.0.1:5000/new_transaction`
HTTP Method	`POST`
Request Payload (JSON)	`{"sender": "user1", "recipient": "api_gateway", "amount": "unique_token_123"}`
Response Status	`201 Created`
Response Format	`application/json`
Response Body	`{"message": "Transaction will be added to Block 2"}`
Response Latency	`9 ms`
Validation Result	`Successful - Transaction was correctly queued in the pending pool.`

- Testing and Validation of Recommendation Engine via REST API

Testing was conducted to ensure that the system can generate relevant course recommendations based on keyword input via the RESTful protocol. This validation includes examination of the query parameter schema and the structure of the resulting JSON response. The results of functional testing for the recommendation endpoint are summarized in Table 5.

In these tests, the REST API server returned responses in the JSON format. The blockchain endpoint `/new transaction` returns a response in the form of a request status message, whereas the recommendation endpoint `/recommendations` returns a list of course recommendations based on the provided keyword. Secure recommendation access is enforced through token validation: requests with valid tokens receive 200 OK, whereas invalid tokens are rejected with an "Unauthorized" response.

Table 5. Functional validation of the course recommendation endpoint

Test Component	Specification / Value
Endpoint URL	`http://127.0.0.1:5000/recommendations`
HTTP Method	`GET`
Query Parameter	`key=Data mining`
Response Status	`200 OK`
Response Format	`application/json`
Key Output Attributes	`Course Name, University, Difficulty Level, Course Rating, Course URL`
Response Latency	`796 ms`
Validation Result	`Successful - The system returned semantically relevant course data.`

The functional tests in Tables 4 and 5 were conducted on a local server to verify endpoint correctness, response schema, and access-control behaviour. In contrast, the performance benchmark in Table 6 was conducted after deployment to the production server to observe end-to-end latency under remote client–server access conditions. This distinction is important because production latency includes additional network communication and cloud-hosting overhead that are not fully reflected in local testing.

4. Deployment architecture

After testing, the recommendation system model and blockchain authentication service are deployed in a cloud-based server environment to facilitate cross-machine access. This Blockchain-Secured Course Recommender Service Deployment architecture separates computational logic from client applications, utilizing an API gateway to enforce token validation at the system boundary.

The main components in this deployment of architecture include:

- Production Server, Utilizes Python-based hosting services to run a Flask application that integrates the ML model and blockchain logic.

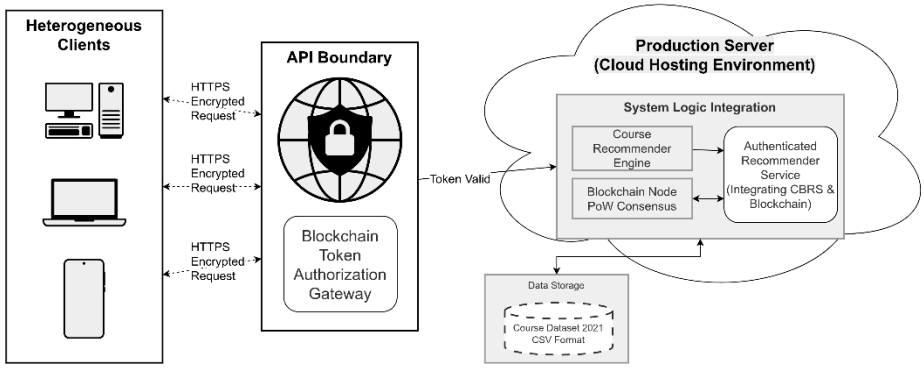

Figure 8. Blockchain-secured course recommender service deployment architecture

- Data Storage, Stores course datasets in a structured format (CSV), accessed directly by the recommendation engine at runtime.
- API Boundary, The interface between the server and clients implementing RESTful protocols and secured with token validation in the authorization header.
- Heterogeneous Clients, Various client devices (computers, laptops, smartphones) that interact with the server via encrypted HTTP requests.

Figure 8 shows the hosting dashboard interface used to deploy the Flask-based REST files.

5. Operational performance evaluation and latency benchmarking

Operational performance measurement was conducted to evaluate the response behavior of the deployed RESTful recommender service under cross-machine access conditions. The Flask-based REST API was deployed on a PythonAnywhere production server, while requests were sent from an external machine using Postman and curl to simulate a distributed client–server scenario.

The evaluated endpoints consisted of the blockchain authentication endpoints (`/new_transaction`, `/mine`, and `/chain`) and the protected recommendation endpoint (`/recommendations`). For the blockchain workflow, `/new_transaction` was tested using a JSON payload containing `sender`, `recipient`, and `amount` fields, where the `amount` value represented a unique access token. The `/mine` endpoint was then invoked to commit the pending transaction into a new block using the Proof-of-Work mechanism, while `/chain` was used to retrieve the complete ledger state. For the recommendation workflow, `/recommendations` was tested using the query parameter `key=Data Science` and a valid token in the HTTP Authorization header.

Latency was measured as the end-to-end elapsed time from sending the HTTP request to receiving the complete JSON response. Thus, the reported values include network round-trip time, server-side processing, blockchain validation or ledger operations, recommendation computation when applicable, and JSON serialization. Since repeated benchmarking was not conducted, the latency values are presented as representative observations from remote production execution, not as average measurements. Invalid-token requests were tested only to verify access-control enforcement and were excluded from the authorized latency observation because they terminated before recommendation computation. Therefore, the results should be interpreted as indicative operational observations of the deployed prototype rather than statistically generalized performance benchmarks.

Table 6 summarizes the observed latency of the blockchain authentication endpoints and the protected recommendation endpoint. These endpoints represent token registration, Proof-of-Work block mining, ledger retrieval, and authorized recommendation retrieval, respectively.

The observations in Table 6 show that the Proof-of-Work authentication workflow introduces additional latency, particularly during ledger retrieval and authorized recommendation access. The `/mine` endpoint required 1113 ms to execute the PoW process and commit the token transaction into a new block, while the `/recommendations` endpoint required 2450 ms to process an authorized request involving token validation and recommendation retrieval. Since these values were obtained as representative observations rather than averaged repeated measurements, they primarily indicate the

Table 6. Representative operational latency observations on production server

Endpoint	HTTP Method	Target Payload / System Operation	Computational Metric / Cryptographic Output	Observed Latency	Protocol Status
`/new_transaction`	`POST`	Ingestion of client identity and unique access token	Queued into the pending transaction pool	921 ms	201 Created
`/mine`	`GET`	Execution of server-side Proof-of-Work (PoW) consensus	Cryptographic Proof: 35089 Block Index: 3	1113 ms	200 OK
`/chain`	`GET`	Extraction of full ledger sequence for state validation	Array of validated blocks from genesis	3060 ms	200 OK
`/recommendations`	`GET`	On-chain authorization check and similarity evaluation	5-item JSON course array (Query key: 'Data Science')	2450 ms	200 OK

operational behavior of the deployed prototype. Future performance evaluation should involve repeated trials, concurrent request scenarios, and statistical reporting such as mean, standard deviation, minimum, maximum, and confidence intervals to provide more generalizable performance evidence.

4. Discussions

4.1. Analysis of the decentralized authorization boundary

The empirical results demonstrate that implementing a decentralized verification layer at the API boundary resolves a fundamental vulnerability in cross-machine 'Recommender-as-a-Service' models. By forcing token validation directly against immutable blockchain transactions prior to model evaluation, the server successfully rejects unauthorized data requests with a strict `401 Unauthorized` response. Unlike traditional centralized access control lists (ACLs) or databases, which are susceptible to post-hoc administrative tampering, the Proof-of-Work (PoW) ledger ensures a permanent, auditable ledger of token registration events.

A critical analysis of the operational data reveals a distinct trade-off between architectural security and execution latency. The server-side consensus layer required a mining latency of 1113 ms to forge a new block. From a security perspective, this 1.11-second delay acts as an effective, deterministic rate-limiting firewall at the network perimeter. Rate limiting is a core defense against overload and DDoS, stabilizing APIs and protecting backends in cloud and

microservice systems (Dharma & Muslikh, 2025; Kalyanasundaram et al., 2025; Lyu et al., 2025; Manoharan, 2024). It penalizes malicious actors attempting automated brute-force token generation by requiring high computational costs per block, while remaining entirely transparent to valid users since mining occurs once during initial client registration.

Conversely, the 2.45-second delay measured at the `/recommendations` endpoint includes the cross-machine network round-trip time (RTT), authorization extraction, ledger state validation, and TF-IDF calculation. While empirical studies across distributed architectures demonstrate that effective rate limiting and standard security layers often add merely milliseconds, not seconds, to request latency, the larger overhead observed in this framework is a direct consequence of decentralized on-chain verification (Kalyanasundaram et al., 2025; Podduturi, 2024; Sultana & Khan, 2025). This cumulative overhead represents a necessary and acceptable trade-off for distributed e-learning microservices where absolute access log integrity and tamper-evident auditability are strictly prioritized over sub-second execution speeds.

4.2. Comparison with prior works

The proposed architecture "Recommender-as-a-Service" balances cross-platform interoperability with decentralized access enforcement by decoupling heavy server-side model computation from lightweight client applications via a structured REST API lifecycle. While foundational frameworks in the literature, such as those by García and Bellogín (2018) and Kumar et al. (2022), optimized functional service exposure and web crawling for cross-platform consumption, they predominantly operate under the assumption of a centrally trusted network. These traditional approaches fall short of securing endpoints in true cross-machine deployments, leaving exposed APIs vulnerable to token spoofing and unauthorized harvesting.

This study successfully bridges this gap by embedding a Proof-of-Work (PoW) blockchain token authentication layer into the Flask routing middleware. Unlike traditional architecture relying on vulnerable centralized databases or access control lists (ACLs) prone to post-hoc administrative tampering, this integrated framework mandates that access tokens be registered as immutable ledger transactions validated at request time via the HTTP authorization header (Drescher, 2017; van Flymen, 2020). Although this introduction of a decentralized boundary imposes an operational latency trade-off compared to unsecured systems, it presents a novel security blueprint where distributed recommendation delivery is structurally bound to tamper-evident access integrity.

5. Conclusions

This study introduces a secure, decentralized a RESTful "Recommender-as-a-Service" framework that successfully integrates a text-based machine learning engine with a Proof-of-Work blockchain token authentication mechanism. The system achieves reliable content-based course filtering by vectorizing high-density attributes (Course Name and Skills) through a structured TF-IDF and Cosine Similarity pipeline. Beyond proving functional client-server interoperability over a cloud environment, the empirical validation confirms that the

deployment of on-chain token validation provides absolute access control, ensuring that only cryptographically verified requests trigger model execution.

Despite its architectural benefits, certain limitations must be addressed. First, the recommendation engine utilizes a pure content-based approach, which relies heavily on textual metadata and may not capture dynamic user interactions as effectively as collaborative methods. Second, the computational overhead of the PoW consensus mechanism makes it less scalable for high-frequency token generation or resource-constrained server infrastructure. Third, security validation was limited to functional endpoint tests and lacked formal penetrative threat assessments. Future research will focus on strengthening the token lifecycle by implementing automated expiration policies and rotational mechanics to mitigate replay attacks. Additionally, we aim to investigate lighter consensus frameworks, such as permission or Proof-of-Stake configurations, to reduce mining latency while maintaining decentralized auditability.

Acknowledgements

The authors thank the Institute for Research and Community Service (LPPM), Universitas Semarang, for supporting this research under Contract No. 004/USM.H7.LPPM/L/2023. This study was conducted as a collaborative research project involving researchers from Universitas Semarang and Universitas AKI. The authors also thank all parties who supported the implementation of this research.

Funding

This research was funded by the Institute for Research and Community Service (LPPM), Universitas Semarang, under Contract No. 004/USM.H7.LPPM/L/2023.

Author contributions

Bernadus Very Christioko contributed to the writing and preparation of the manuscript and conducted the experiments and model testing. Khoirudin contributed to the writing of the manuscript and the research implementation as a contributing author. Yusup contributed to the research implementation and manuscript review. All authors read and approved the final version of the manuscript.

Disclosure statement

The authors declare that there is no conflict of interest related to this study.

References

Anand, P. B., & Nath, R. (2020). Content-based recommender systems. In S. N. Mohanty, J. M. Chatterjee, S. Jain, A. A. Elngar, & P. Gupta (Eds.), *Recommender system with machine learning and artificial intelligence* (pp. 165–195). Wiley. https://doi.org/10.1002/9781119711582.ch9

Apriani, A., Zakiyudin, H., & Marzuki, K. (2021). Penerapan Algoritma Cosine Similarity dan Pembobotan TF-IDF System Penerimaan Mahasiswa Baru pada Kampus Swasta. *Jurnal Bumigora Information Technology* (*BITe*), *3*(1), 19–27. https://doi.org/10.30812/bite.v3i1.1110

Bajpai, P., Suryawanshi, N., Orse, A., Srivastava, H., Patil, M., & Lavale, B. (2022). E-learning recommendation system. *International Journal for Research in Applied Science & Engineering Technology* (*IJRASET*), *10*, 2321–9653. https://doi.org/10.22214/ijraset.2022.42706

Bigelow, S. J. (2020). *5 stages of an API lifecycle explained*. https://www.techtarget.com/searchapparchitecture/feature/5-stages-of-an-API-lifecycle-explained

da Silva, F. L., Slodkowski, B. K., da Silva, K. K. A., & Cazella, S. C. (2023). A systematic literature review on educational recommender systems for teaching and learning: Research trends, limitations and opportunities. *Education and Information Technologies*, *28*(3), 3289–3328. https://doi.org/10.1007/s10639-022-11341-9

Dharma, B., & Muslikh, A. R. (2025). Implementasi rate limiting dan Bot Telegram untuk mitigasi serangan HTTP GET Flood. *Journal of Information System and Application Development*. https://doi.org/10.26905/jisad.v3i1.15398

Drescher, D. (2017). Blockchain basics: A non-technical introduction in 25 steps. In *Blockchain basics*: *A non-technical introduction in 25 steps*. Apress Media LLC. https://doi.org/10.1007/978-1-4842-2604-9

Fayyaz, Z., Ebrahimian, M., Nawara, D., Ibrahim, A., & Kashef, R. (2020). Recommendation systems: Algorithms, challenges, metrics, and business opportunities. *Applied Sciences*, *10*(21), Article 7748. https://doi.org/10.3390/app10217748

García, I., & Bellogín, A. (2018, June). Towards an open, collaborative REST API for recommender systems. In *RecSys 2018 – 12th ACM Conference on Recommender Systems* (pp. 504–505). Association for Computing Machinery. https://doi.org/10.1145/3240323.3241615

Habibi, M., & Cahyo, P. W. (2020). Journal classification based on abstract using cosine similarity and support vector machine. *JISKA* (*Jurnal Informatika Sunan Kalijaga*), *4*(3), 185–192. https://doi.org/10.14421/jiska.2020.43-06

Jain, H., & Kakkar, M. (2019). Job recomemendation system based on ML & DM techniques using RESful API and Android IDE. In *2019 9th International Conference on Cloud Computing*, *Data Science & Engineering* (*Confluence*) (pp. 416–421). IEEE. https://doi.org/10.1109/CONFLUENCE.2019.8776964

Javed, U., Shaukat, K., Hameed, I. A., Iqbal, F., Alam, T. M., & Luo, S. (2021). A review of content-based and context-based recommendation systems. *International Journal of Emerging Technologies in Learning*, *16*(3), 274–306. https://doi.org/10.3991/ijet.v16i03.18851

Kalyanasundaram, T., Panchalingam, K., Jegatheesan, T., Wijayasiri, A., & Perera, S. (2025). Load balancer filter-based approach to enable distributed API rate limiting. In *2025 37th Conference of Open Innovations Association* (*FRUCT*) (pp. 75–85). IEEE. https://doi.org/10.23919/fruct65909.2025.11008311

Kapoor, K. (2021). *Coursera courses dataset 2021* [Data set]. https://www.kaggle.com/datasets/khusheekapoor/coursera-courses-dataset-2021

Kumar, N., Gupta, M., Sharma, D., & Ofori, I. (2022). Technical job recommendation system using APIs and Web crawling. *Computational Intelligence and Neuroscience*, *2022*, Article 7797548. https://doi.org/10.1155/2022/7797548

Li, M. (2024). Recommendation system building based on CNN and TF-IDF approaches. *Highlights in Science*, *Engineering and Technology*, *92*, 178–187. https://doi.org/10.54097/633gjj39

Lyu, N., Wang, Y., Cheng, Z., Zhang, Q., & Chen, F. (2025). Multi-objective adaptive rate limiting in microservices using deep reinforcement learning. In *Proceedings of the 4th International Conference on Artificial Intelligence and Intelligent Information Processing* (pp. 862–869). Association for Computing Machinery. https://doi.org/10.1145/3778534.3778668

Manoharan, M. (2024). API rate limiting mechanisms in SaaS applications: A systematic analysis of DDoS protection strategies. *International Journal of Scientific Research in Computer Science*, *Engineering and Information Technology*, *10*(6), 1787–1798. https://doi.org/10.32628/cseit241061223

Mawanta, I., Gunawan, T. S., & Wanayumini, W. (2021). Uji Kemiripan Kalimat Judul Tugas Akhir dengan Metode Cosine Similarity dan Pembobotan TF-IDF. *Jurnal Media Informatika Budidarma*, *5*(2), 726–738. https://doi.org/10.30865/mib.v5i2.2935

Mohanty, S. N., Chatterjee, J. M., Jain, S., Elngar, A. A., & Gupta, P. (2020). Content-based recommender systems. In F. Ricci, L. Rokach, B. Shapira, & P. Kantor (Eds.), *Recommender system with machine learning and artificial intelligence* (pp. 167–195). Springer. https://doi.org/10.1002/9781119711582

Ni, J., Cai, Y., Tang, G., & Xie, Y. (2021). Collaborative filtering recommendation algorithm based on TF-IDF and user characteristics. *Applied Sciences*, *11*(20), Article 9554. https://doi.org/10.3390/app11209554

OpenAPI Initiative. (2020). *OpenAPI specification* (*Version 3.0.3*) [Specification]. https://spec.openapis.org/oas/v3.0.3.html

Pal, N., & Dahiya, O. (2023). Analysis of educational recommender system techniques for enhancing student's learning outcomes. In *2023 3rd International Conference on Innovative Practices in Technology and Management* (*ICIPTM*) (pp. 1–5). IEEE. https://doi.org/10.1109/ICIPTM57143.2023.10118132

Parvatikar, S., & Parasar, D. (2021). Recommendation system using machine learning. *International Journal of Artificial Intelligence and Machine Learning*, *1*(1), 24–30. https://doi.org/10.51483/IJAIML.1.1.2021.24-30

Podduturi, S. M. (2024). Security and performance optimization in microservices for real-time data systems. *International Journal for Multidisciplinary Research*, *6*(6). https://doi.org/10.36948/ijfmr.2024.v06i06.33239

Portugal, I., Alencar, P., & Cowan, D. (2018). The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, *97*, 205–227. https://doi.org/10.1016/j.eswa.2017.12.020

Rani, P., & Bhambay, R. (2023). A comparative survey of consensus algorithms based on proof of work. In P. Dutta, S. Chakrabarti, A. Bhattacharya, S. Dutta, & V. Piuri (Eds.), *Emerging technologies in data mining and information security* (pp. 261–268). Springer Nature Singapore. https://doi.org/10.1007/978-981-19-4193-1_25

Raschka, S., Patterson, J., & Nolet, C. (2020). Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, *11*(4), Article 193. https://doi.org/10.3390/info11040193

Sultana, R., & Khan, R. (2025). AI-based rate limiting for cloud infrastructure: Implementation guide. *Journal of Computer Science and Technology Studies*, *7*(3), 370–380. https://doi.org/10.32996/jcsts.2025.7.3.43

Thakur, A. (2021). Automated online course recommendation system using collaborative filtering. *International Journal for Research in Applied Science and Engineering Technology*, *9*(2), 222–229. https://doi.org/10.22214/ijraset.2021.33043

van Flymen, D. (2020). *Learn blockchain by building one*: *A concise path to understanding cryptocurrencies*. Springer International Publishing. https://doi.org/10.1007/978-1-4842-5171-3

Wahyuni, R. T., Prastiyanto, D., & Supraptono, E. (2017). Penerapan Algoritma Cosine Similarity dan Pembobotan TF-IDF pada Sistem Klasifikasi Dokumen Skripsi. *Jurnal Teknik Elektro Universitas Negeri Semarang*, *9*(1), 18–23. https://journal.unnes.ac.id/nju/index.php/jte/article/download/10955/6659

Yusuf, M., & Cherid, A. (2020). Implementasi Algoritma Cosine Similarity Dan Metode TF-IDF Berbasis PHP Untuk Menghasilkan Rekomendasi Seminar. *Jurnal Ilmiah Fakultas Ilmu Komputer*, *9*(1), 8–16. https://publikasi.mercubuana.ac.id/index.php/fasilkom/article/view/8830