Power System Technology
ISSN:1000-3673

Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

Design and Development of a Scalable Policy Management Module in a Meta University Framework

Ravindra M. Huddar*[1] , Dr. V. S. Kumbhar[2], Akhilesh R. Huddar[3], Dr. Kabir Kharade[4], Dr. V. I. Pujari[5]

[1*]Research scholar, Dept. of Computer Science, Shivaji University, Kolhapur

[2]Assistant Professor, Dept. of Computer Science, Shivaji University, Kolhapur

[3]B.Tech. Student, Data science, Kolhapur Institute of Technology, Kolhapur

[4]Assistant Professor, Dept. of Computer Science, Shivaji University, Kolhapur

[5]Assistant Professor, Dept. of Computer Applications, D. Y. Patil school of Engineering and management, D.Y. Patil Education Society, (Deemed to be University) Kolhapur

Corresponding Author: **Ravindra M. Huddar,** Research scholar, Dept. of Computer Science, Shivaji University, Kolhapur, E-mail:- alc.33210010@gmail.com

Abstract*:*

The notion of a meta university promotes the concept of inter-space cooperation and resource sharing; thus, a policy management system needs to be in place to enhance their security and governance. This paper details the policy management module design for the meta university system, the system development by implementation of Flask for web application and the Open Policy Agent for policy management. The module integrates role-based access control (RBAC) mechanism policies created in Rego for dynamic and hierarchical level access control management to limit and permit certain actions to certain persons only depending on the role assigned to them. Unit and performance testing have been done and the modules results given effectiveness and scalability evaluation of the policy management mechanism on the meta university framework contributes to the wider educational technology agenda as it offers a useful policy management application for meta university framework.

Keywords*:* Meta university framework, RBAC, Open Policy Agent, Scalability, Higher education, Inter-university collaboration

I. Introduction

The higher education system is in changing trends across the globe due to technological improvements and the increasing demand for effective, quick and collaborative learning. Therefore, the meta university conception has been introduced as a solution for such

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

challenges, enhancing inter-university relationships as well as interactions between universities and their environments. By offering such framework, it is possible for the academic institutions to combine their efforts, resources and courses in an appealing way that makes educational satisfaction more rewarding to the learners.

A sound policy management mechanism is at the heart of the sustainability of the meta university framework. Policy management provides an ability to control access and actions within the system depending on the user role whether a faculty member or a student. It plays a critical role in the protection of network security, enforcing policy compliance and functional operation of the entire interconnected system. In the absence of an effective and adaptable policy management framework, there is a danger that the framework will suffer from unwarranted usage, incompetence and inefficiencies towards meeting its objectives.

The paper describes the design, implementation and testing of a specific policy management module for the meta university framework. In this module, web application development is with Flask, and policy enforcement is with Open Policy Agent (OPA) and the Rego-based Role-Based Access Control (RBAC) policies introduction. This enables a highly efficient and flexible control of the framework's access and actions.

Open Policy Agent (OPA)

Open Policy Agent (OPA) is a policy-approving tool that is freely available to all and enforces organizational policies. The OPA engine was commercialized for the first time in 2018 as the project under the cloud native computing sandbox which went on to be an open domain based policy engine designed for pervasive policy implementation in any organization. The organization has complex policy writing and policy adherence is critical for the organization's success. In OPA implemented policies are non-conflicting and consistent with the policies of the Organization. The query is passed on to OPA along with any relevant information required for the Answer. OPA than processes the policies and data and gives the results for the query.

This research has three main goals:

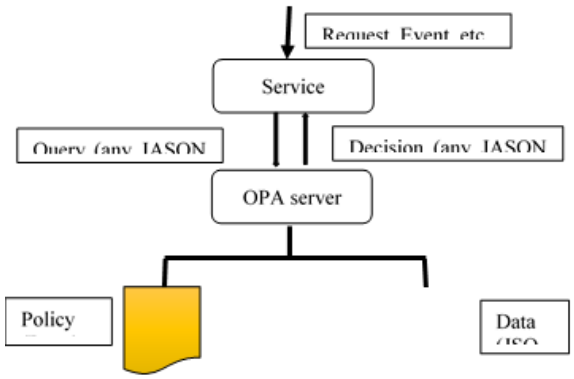

Figure.1 working of OPA

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

1. Develop a highly flexible and easily extendable policy management system that can meet the peculiarities of a meta university ecosystem.

2. Develop this system based on contemporary web platforms and policy engines, making it easy to deploy and upgrade.

3. Conduct extensive testing and demonstration of the proposed approach, especially its capability to control user roles and actions in a secure manner.

The rest of this paper is organized as follows: First, we review the relevant literature on policy management solutions, paying special attention to their integration within educational systems. Then we introduce the concept of a meta-university, and explain the importance of the policy management module in such a system. We then describe the design and implementation of the policy enforcement module, both from a technical standpoint and its integration with Flask and OPA. Following this we present the results of our testing and validation efforts highlighting the module's performance and scalability. Finally, we state the conclusions we have drawn from our report, and give some ideas on how the work could be developed in the future in order to make the module even more powerful.

II. Literature review

The term policy management in educational systems means determining and putting into place rules that govern access to resources, data, and how users perform actions. These management approaches have started with Role Based Access Control (RBAC) and Attribute Based Access Control (ABAC). These approaches have been researched and applied in many fields including education.

RBAC as an example means a mechanism where threshold access levels are given to users based on roles rather than individuals. This eases the load of managing user permissions Cole, P. (2007). In this situation, changing the roles of some users will not who can access the deletions. For Large and dynamic users bases System of Managing Access Recourse Based on Role (Sandhu et .al, 1996). In the educational context, RBAC systems have been used to protect students' faculty and administrative access control enhancing security. Fernandez et al. (2014) studied how RBAC can be used in e-learning systems and proved its success in controlling permissions and roles in order to use educational systems. Their work emphasized the effectiveness of RBAC in regulating user access based on user classification as well as in implementing and controlling security measures.

ABAC provides a more detailed control by taking into consideration the attributes of users, resources and environment as well. It has however found its use in different educational systems for enabling flexible management of access rights. Hu et al. (2015) considered ABAC for cloud e-learning applications also. It is pointed out that ABAC is especially suitable for such dynamic systems, and is able to address the complex requirements of access control in

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

educational systems. Rajpoot et al. (2016) researched using ABAC principles in educational cloud services, proving it is useful for supporting needs in dynamic and fine-grained access control scenarios. Their findings affirm that there are merits to utilizing ABAC in online education platforms that have various users engaging with the platform and accessing different educational resources.

Open Policy Agent (OPA) is an open-source, convenient, policy engine usable in any context, singular policy framework aims both definition and enforcement of policies OPA's centralization of policies means that policy decisions are made at a higher level than the application code decoupling policies from applications making it easier to manage policies. The policy language implemented in OPA called Rego enables policies to be composed allowing for specific policies.

Most of the reviewed literature revolves around the implementation of OPA in contemporary managerial practices or problems. For instance, Torstensson et al. (2019) analyzed the performance of OPA systems on the cloud outlining the deployment of policies on the intervening microservices. Policy dynamics is one of the challenges that OPA makes reality for applications today which offer extreme policy management. Anderson et al. (2020) focused on the use of OPA for policies in existing systems, particularly, Kubernetes. They provided evidence that OPA could be used to provide fine-gained access control policies in containerized applications, thereby maintaining safe and compliant business practices. These results further advocate that OPA can be practically used in designing systems that are complex and have dynamic attributes with regards to security and compliance. Leino et al. (2021) investigated potential applications of OPA from an API security perspective. Their study showed that OPA can be a flexible and scalable way of controlling access to APIs - allowing organizations to extend the security policies over the API ecosystem. This shows that OPA, while aimed primarily at enterprises, has found application in a number of other activities such as educational systems.

In their research Hwang et al. (2005) introduced a policy-based management approach for e-learning environments, mainly stressing the necessity for dynamic and context-sensitive policy implementation. Their study shows how flexible and adaptable policies are paramount in the management of educational resources and users. Kumar et al. (2018) analyzed policy based management techniques for the educational cloud environments, highlighting the issue of centralized policy enforcement as a key factor of security and compliance. Their research discusses the advantages of the policy-based approach in controlling the access to geographically distributed educational resources.

Identity and Access Management (IAM) tools offer educational systems a complete user identities and access control management. Those can often integrate RBAC, ABAC, or some other access modules. Gonzalez et al. (2017) looked through the IAM solutions designed for

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

educational institutions with an emphasis on management of users, their roles, and permissions in a flexible and effective way. The findings from their research underscore the contribution of IAM tools in facilitating safe and compliant access to educational materials. Zhalo et al. (2020) studied the application of IAM tools in the provision of access control in e-learning systems. The findings from their research indicate that IAM solutions perform well in the provision of access control in the educational sector.

In this review of literature on policy management in educational systems, several tools and models such as RBAC, ABAC, OPA policies, and policy management systems have been discussed. Each tool offers unique features and can be implemented in various ways in educational settings. In this regard, even though access control approaches have been built on established models such as RBAC and ABAC, which have their benefits, the modern tool, OPA, offers a better and easier management of policies in the educational sector which can be considered complex. It is found that these tools need to be further developed and strove to further their applicability in education systems.

III. Meta University overview

The conceptualization of a Meta University is an integrated approach towards education with the aim of subsuming many different universities into one platform while enhancing interactivity of the learning experience. The said framework has adopted some sophisticated technology for the tailoring of the educational experience, administrative work, respective policy advocacy and implementation. It is aimed to provide a continuous and versatile learning space for students, teaching staff and even the administration.

Following diagram shows various components of Meta University framework

Figure 2 Meta University framework components

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

Key components

Other than that, they are interlinked; each of the elements on its own builds the lateral staircase for the attainment of the core objectives. There are 6 Components for a meta-university:

1. Learning Management System (LMS) – Intended for managing the learning process-a varied term used to define the boundaries of learning objects, including lists of courses, assessments, and assignments.

2. Personalization Engine-the artificial intelligence system that explores the student data given purposefully for the identification of the student and provides the courses to be taken and the learning maps in due course.

3. Policy Management Module-A comprehensive and leveraged access management solution for the enforcement of access control policy-centralized around Open Policy Agent (OPA) for compliant security.

4. Credit Management Module-A system designed to control and supervise the credit loads of the students by overseeing the transfer of credits appropriately if undertaken from other institutions.

5. Networking Sites-Interaction and communication resources are basic in nature for both teachers and students across various institutions.

6. Admin instrumentation-course management system using one platform where the course administration takes place and where the following occur: approval of courses, assignment of students and faculties, etc.

User Roles

The Meta University framework provides a variety of user role types, with specific effects on privileges and duties. The roles are hence classified:

1. Meta University administrator: oversees the entire framework. One of the main duties is to supervise member universities for adherence to laid-down policies.
2. Member university administrator: Oversees the running of a specific member university and carries out local administrative responsibilities, usually in consultation with the Meta university administrator.
3. Course coordinator: Acts on behalf of the Member university Administrator to coordinate academic courses, teachers, and students in the member university.
4. Faculty: Responsible for the design and delivery of course content, evaluation of student work, and constructive feedback.
5. Student: interacts with course content and materials and keeps records of grades.
6. Outside helper or guest: Usually, each student has a guest role, although this user only can see the meta university dashboard.

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

IV. Policy Management module

The policy management module is one of the components of the Meta University structure. It is responsible for making sure that access control policies are effectively applied. This module integrates the OPA to specify how policies will be understood and managed in practice and equally to protect users based on how advanced they are in their various roles with actions available to the users given their roles.

Structural Elements of the Policy Management Module

Client application - The screen used by different users to carry out the actions is called the client application.

Policy Enforcement Layer – The component of the architecture that interacts with the client application and all subsequent components. Policies comprise two components – Policy Enforcement Layer Culture and Operations Middleware. a) Web framework in the meaning of policy enforcement layer is aware of all web requests, provides all web responses, and specifies a particular Api that will begin interacting with certain endpoints. In our research work we used Python Flask as web framework b) Authorisation middleware it is a piece of software that sits between the internet and the application and processes every incoming request, checking whether authorisation is necessary and, if so, performing it by consulting the OPA to establish whether the request can be allowed or not. The OPA is a policy agent that is used only in the Flask application to ensure authorisation is performed on every incoming request before it is processed by all the controller logic.

3. **OPA (Open Policy Agent)** – A policy decision point that decides whether a request is to be granted or restricted based on policies enforced in Rego files. The policy decision may be affirmative to allow, or negative to deny the request.

4. **Police Store** - Holds the policies written in Rego language and define the actions permitted to specific roles.

5. **Database** – Custodian of the records pertaining to various classes of users and connects to the Policy Enforcement Layer.

Following diagram visualizes components and interactions in policy management module

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

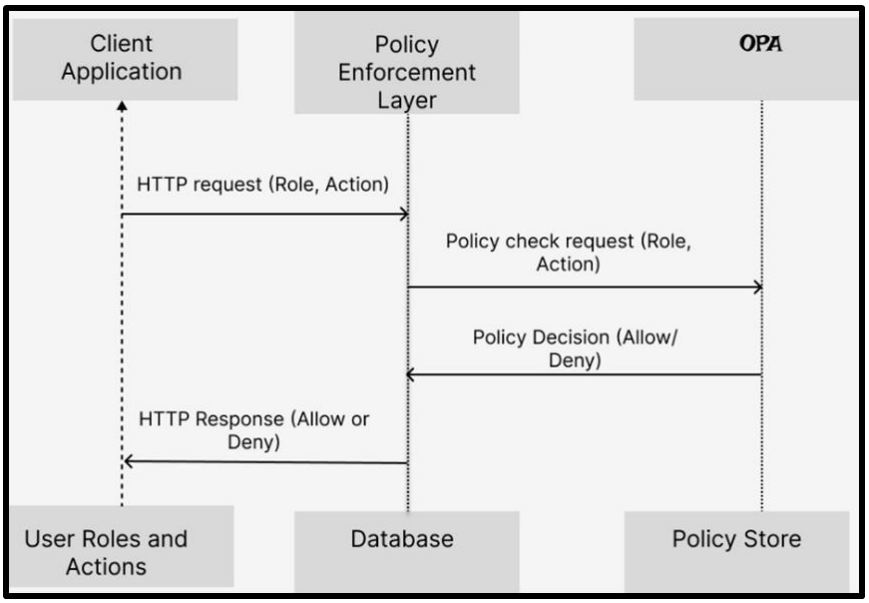

Figure 3 Policy management module component

A) Application Architecture

The policy management components discussed above will be implemented in python, OPA with MySQL as backend. The multi-tier application architecture employed is shown in fig. 4

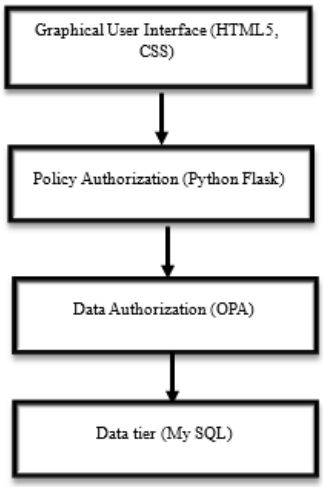

Figure 4 Multi-Tier Application architecture

B. Implementation of Policy management module

Folder structure

The different files employed in the module design along with the different folders in which they reside is depicted in Figure 4.

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

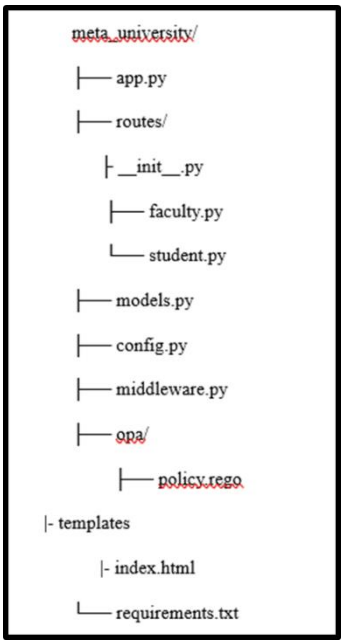

Figure 5 Directory structure

1. app.py

This is the main entry point for flask application. It initializes the Flask application, registers Blueprints, and set up middleware. It also defines the root route (‘/’) which can serve as a welcome page or landing page.

2. Routes/

This directory contains the route definitions for different parts of the application,

3. Metaadmin.py

This file defines routes related to meta admin actions, such as approval to the member university admin, and all actions needed for management of meta university

4. Memberadmin.py

This file defines routes related to Member admin actions, such as allocation of Manager and approval of Manager actions and maintaining users of that university

5. Manager.py

This file defines routes related to Manager actions, like creating courses, allocation of courses to the faculty.

6. Faculty.py

This file defines routes related to faculty actions, such as creating course material, enrolling students, and grading student assignment and quizzes.

7. Student.py

This file defines routes related to student actions, such as taking admission, accessing course materials, attempting assignment and test.

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

8. Models.py

This file is intended to define the data models for the application, It uses ORM to define database models here.

9. Config.py

This file is used to store configuration settings for flask application like database connection strings.

10. Middleware.py

This file contains middleware functions that run before each request to handle tasks like authorization. For e.g. authorize_request checks the user's role and action against the policies defined in OPA.

11. OPA/

This directory contains OPA policies that define access control rules. These policies are written in the Rego language.

12. Requirements.txt

This file lists all the python dependencies for the project.

13. Index.html – It provides page where user provides details and view the result of policy management module.

Define policies in opa/policies.rego file

```
package meta_university
default allow = false
allow {
   input.user.role == "faculty"
   input.action == "create_course"
}
allow {
   input.user.role == "faculty"
   input.action == "enroll_student"
}

allow {
   input.user.role == "faculty"
   input.action == "grade_test"
}
allow {
   input.user.role == "student"
   input.action == "take_admission"
}
allow {
   input.user.role == "student"
   input.action == "access_material"
}
allow {
   input.user.role == "student"
   input.action == "attempt_test"
}
```

Figure.6 policy.rego file

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

Define app.py

```
from flask import Flask, request, render_template, jsonify
from routes.faculty import faculty_bp
from routes.student import student_bp
from middleware import opa_authorize, authorize_request
app = Flask(__name__)
# Register middleware
app.before_request(authorize_request)
# Register Blueprints
app.register_blueprint(faculty_bp, url_prefix='/faculty')
app.register_blueprint(student_bp, url_prefix='/student')
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        role = request.form['role']
        action = request.form['action']
        result = opa_authorize(role, action)
        return render_template('index.html', result=result)
    return render_template('index.html', result=None)

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 7 Flask web server app.py file

Define middleware.py

```
import requests
from flask import request, jsonify
def opa_authorize(role, action):
    data = {
        "input": {
            "user": {
                "role": role
            },
            "action": action
        }
    }
    print(f"Sending data to OPA: {data}")
        response = requests.post("http://localhost:8181/v1/data/meta_university/allow",
json=data)
    print(f"Response status code: {response.status_code}")
    print(f"Response content: {response.json()}")
    result = response.json()
    return result.get('result', False)

def authorize_request():
    if request.path=='/':
        return None
    role = request.headers.get('Role')
    action = request.endpoint
    if not role or not action:
        return jsonify({"message": "Unauthorized"}), 403
    if not opa_authorize(role, action):
        return jsonify({"message": "Unauthorized"}), 403
```

Figure 8 Middleware authorize

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

Define all the routes py files

For example, faculty.py will contain faculty route action

```
from flask import Blueprint, jsonify

faculty_bp = Blueprint('faculty', __name__)

@faculty_bp.route('/create_course', methods=['POST'])
def create_course():
    return jsonify({"message": "Course created"}), 201

@faculty_bp.route('/enroll_student', methods=['POST'])
def enroll_student():
    return jsonify({"message": "Student enrolled"}), 201

@faculty_bp.route('/grade_test', methods=['POST'])
def grade_test():
    return jsonify({"message": "Test graded"}), 201
```

Figure 9 Faculty route

C. Evaluation and Results

- **Set up the Environment**

Make sure you have python and Flask installed. Install the required dependencies listed in 'requirements.txt' by running the following command

```
pip install –r requirements.txt
```

- **Start the OPA server**

opa run –server opa/policy.rego

```
D:\meta_university>opa run --server opa/policy.rego
{"addrs":[":8181"],"diagnostic-addrs":[],"level":"info","msg":"Initializing server. OPA is running on a public (0.0.0.0)
 network interface. Unless you intend to expose OPA outside of the host, binding to the localhost interface (--addr loca
lhost:8181) is recommended. See https://www.openpolicyagent.org/docs/latest/security/#interface-binding","time":"2024-07
-31T20:01:08+05:30"}
```

Figure 10 OPA server

- **Start Flask application**

```
D:\meta_university>python app1.py
 * Serving Flask app 'app1'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 129-889-499
```

Figure 11 web request response server

- **Visit the site localhost:5000 it displays index.html file**

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

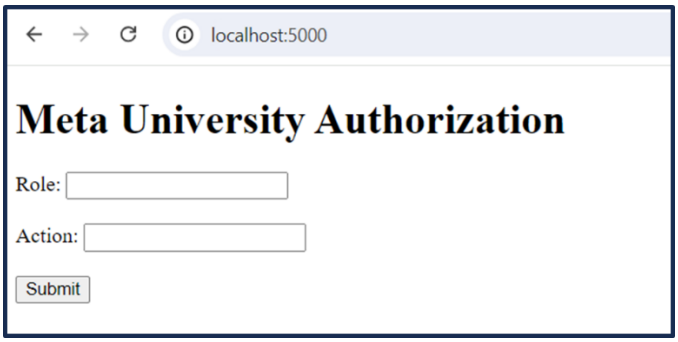

Figure 12 User UI to provide information to the authorization server

- **Server provides response to the page using get method**

```
D:\meta_university>python app1.py
 * Serving Flask app 'app1'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 129-889-499
127.0.0.1 - - [31/Jul/2024 20:05:47] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Jul/2024 20:05:47] "GET /favicon.ico HTTP/1.1" 403 -
```

Figure 13 Web Server Request processing

- **provide the user role and action like to perform and request is sent to opa server for to check the policy rules and based on that allows or Denay's action provided by user.**

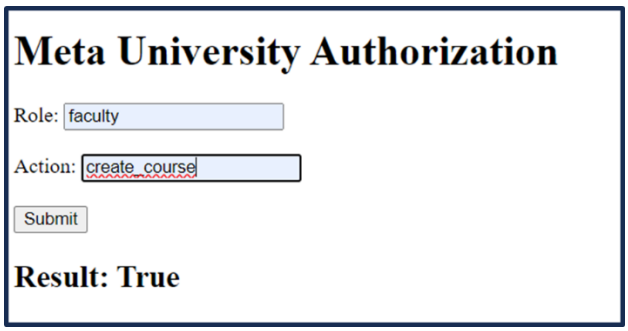

Figure 14 Role=faculty and action = create_course

- **Request is sent to OPA server**

```
D:\meta_university>opa run --server opa/policy.rego
{"addrs":[":8181"],"diagnostic-addrs":[],"level":"info","msg":"Initializing server. OPA is running on a public (0.0.0.0)
 network interface. Unless you intend to expose OPA outside of the host, binding to the localhost interface (--addr loca
lhost:8181) is recommended. See https://www.openpolicyagent.org/docs/latest/security/#interface-binding","time":"2024-07
-31T20:01:08+05:30"}
{"client_addr":"[::1]:53687","level":"info","msg":"Received request.","req_id":1,"req_method":"POST","req_path":"/v1/dat
a/meta_university/allow","time":"2024-07-31T20:10:26+05:30"}
{"client_addr":"[::1]:53687","level":"info","msg":"Sent response.","req_id":1,"req_method":"POST","req_path":"/v1/data/m
eta_university/allow","resp_bytes":16,"resp_duration":2.4121,"resp_status":200,"time":"2024-07-31T20:10:26+05:30"}
```

Figure 15 OPA server checks Authorization

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

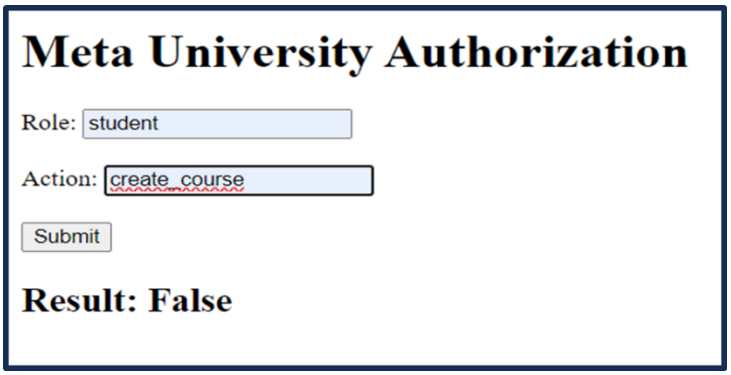

Figure 16 Role=student action=create_course

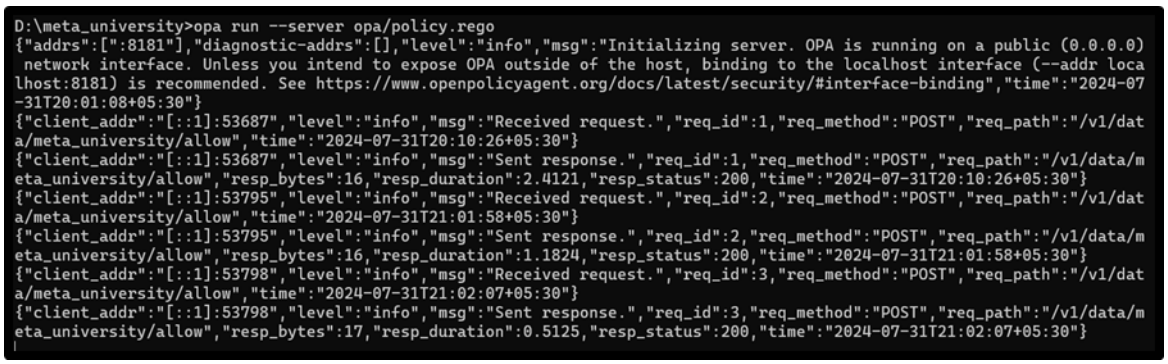

Figure 17 OPA server request for policy execution

The policy management module was evaluated for its effectiveness and scalability in managing access control within the meta university framework. The evaluation involved testing various user actions and verifying that the policies were correctly enforced. The results demonstrate that the module effectively manages permissions and scales to accommodate users and policies.

Flow of policy management module

Whenever a client takes a certain action within Meta University, such action request is captured by the system. The system then prepares an authorization request to the OPA server with all the necessary context for the given user, such as the user's role, the action performed and others. This authorization request is directed to the OPA server which is located in a cloud environment. This request is however made in a way (JSON) that OPA understands. The OPA server checks the request against the policies that have been deployed in OPA using the Rego language. OPA handles the input and directly issues a simple response that is formatted in JSON specifying whether the action is permitted or not {'true' or 'false'}. The framework draws the judgment from the OPA, for instance, if the verdict is positive, the user is allowed to perform the activity while if the verdict is negative, the user is prohibited from performing the activity, and may even inform the user of lack of sufficient permissions.

Power System Technology
ISSN:1000-3673
Received: 06-01-2025 Revised: 15-02-2025 Accepted: 05-03-2025

REFERENCES

1. Sandhu, R. S., Coyne, E. J., Feinstein, H. L. & Youman, C. E. (1996), Role-based control models, IEEE Computer, 29(2), 38-47.
2. Fernandez, E. B., Mujica, S. R. & Ortega, A. C. (2014). RBAC in e-learning, international journal of Computer Applications,107(1), 15-20
3. Hu, V. C., Ferraiolo, D. F., Kuhn, R., & Schnitzer, A. (2015), Attribute-based access control, Computer, 48(2), 85-88.
4. Rajpoot, Q., Alharbi, A., & Walters, R. J. (2016), Attribute-based access control for cloud computing. Journal of Cloud Computing, 5(1), 14.
5. Torstensson, M., Torell, M., & Sturegard, J. (2019), Policy-based management for cloud-native applications. IEEE International conference on Cloud Computing Technology and Science (CloudCom) (pp. 148-153)
6. Anderson, E., Smith, J., & Wallace, P, (2020). Policy enforcement for Kubernetes with Open Policy Agent, IEEE International Conference on Cloud Engineering (IC2E) (pp. 104-111)
7. Leino, K. R., Bharti, P., & Singhm A. K. (2018). Polic based management for educational cloud environments, IEEE international conference on Cloud Computing in Emerging Markets (CCEM) (pp. 33-39)
8. Hwang. G. J., Tsai, C., & Yang, S. J. H. (2005), Criteria, strategies and research issues of context-aware ubiquitous learning, Educational Technology & Society, 8(4), 81-91
9. Kumar, P., Bharti, P., & Singh, A. K. (2018). Policy-based management for educational cloud environments, IEEE international conference of Cloud Computing in Emerging Markets (CCEM) (pp. 33-39)
10. Gonzalez, C., Bravo J., & Navarro, L. (2017), Identity and access management for educational environments using blockchain technology. IEEE Frontiers in Education Conference (FIE) (pp. 1-8) IEEE.