Unit 10: From Distributed Computing to Microarchitectures

Seminar:

API Demonstrations

Task:

In this session, we will expand the API Unit 7 with functionality relevant to your summative code submission. Expand the API created in Unit 7 with functionality relevant to your summative code submission.

You are also welcome to demo your summative code assessment in Week 10 if you are complete by then, but the session is arranged for Week 11 to give you the maximum time available. The demonstrations in this seminar are for the API seminar work, without the assumption that your summative assessment development is complete by then.

Answer:

Purpose of the Code

The Flask application implements a Role-Based Access Control (RBAC) system. This ensures that only authorized users with specific roles can access certain parts of the application. It demonstrates authentication, session management, and role-based permission checks.

Key Features

- Role-Based Access Control:
  The role_required decorator ensures that only users with the appropriate roles can access specific endpoints. This centralizes the logic for role verification, making the code modular and easier to maintain.
- Session Management:
  User sessions are managed through a simple in-memory sessions dictionary. Each session is identified by a token, which is passed in the Authorization header for authentication.
- Mock Database:
  User data is stored in the users list, simulating a simple database. This makes the application self-contained for testing and demonstration purposes.

- User-Friendly Endpoints:
  Endpoints like /login, /logout, /admin, /customer, and /clerk are straightforward, making it clear how to interact with the system.
- Error Handling:
  The application returns appropriate HTTP status codes (401 for unauthorized, 403 for forbidden) and messages to guide users in resolving issues.

Code Walkthrough

- Authentication:
  The /login endpoint verifies a user's credentials. If valid, it generates a session token (token-{user_id}) and stores the session details in the sessions dictionary. This token acts as a temporary pass for accessing protected endpoints.
- Authorization:
  - The role_required decorator checks:
  - If a session token is provided in the Authorization header.
  - If the token is valid and maps to an active user session.
  - If the user has the required role for the requested endpoint.
  - Unauthorized or insufficient permissions return clear error messages.
- Role-Based Endpoints:
  - Three endpoints are implemented for role-based access:
  - /admin (admin-only)
  - /customer (customer-only)
  - /clerk (clerk-only)
  - Each endpoint uses the role_required decorator to enforce role-specific access.
- Logout:
  The /logout endpoint invalidates the user's session token, ensuring they cannot access protected resources until they log in again.
- Home Route:
  The / endpoint provides a quick overview of the available routes and their purposes, making the API self-documenting.

Design Choices

- Flask Framework:
  Flask is lightweight and suitable for small to medium-sized applications. Its simplicity makes it an excellent choice for demonstrating RBAC concepts.
- In-Memory Data Storage:
  Mock users and sessions data are used to simplify the setup. This avoids the complexity of setting up a database for demonstration purposes.
  For production, a persistent database (e.g., PostgreSQL, MongoDB) and secure session management would be necessary.
- Token-Based Authentication:

Tokens simulate real-world authentication systems (e.g., JWT). While these tokens are simplistic here, they demonstrate the concept effectively.

- Separation of Concerns:
  The role_required decorator abstracts the logic for role checking, keeping the endpoint functions clean and focused on their specific tasks.
- Error Handling:
  The application provides clear messages and status codes to guide users and developers in understanding issues.

Possible Improvements

- Security Enhancements:
  - Use hashed passwords (e.g., bcrypt) instead of storing plain text.
  - Replace mock sessions with a robust session store.
  - Add token expiration for enhanced security.
- Database Integration:
  - Replace the in-memory users list with a database for persistence and scalability.
- Role Flexibility:
  - Allow multiple roles per user and implement fine-grained access control using role hierarchies or permissions.
- Scalability:
  - Deploy behind a WSGI server (e.g., Gunicorn) for better performance in production.
- Use rate limiting to prevent abuse of authentication endpoints.

Conclusion

This Flask application demonstrates a simple yet effective implementation of RBAC. It is a great starting point for learning how to build secure and modular APIs. For production-grade use, additional enhancements in security, scalability, and maintainability would be necessary.