
WEEK 1 – DAY 5

AUTHORIZATION MODELS

(Very Detailed, Easy-to-Explain, Interview + Practical Ready)

---

1. Authentication vs Authorization (First, Clear the Confusion)

This question is asked in **almost every interview**.

Authentication

👉 Answers: **"Who are you?"**
Example: Login with username & password

Authorization

👉 Answers: **"What are you allowed to do?"**
Example: Can you view admin panel?

Real-Life Analogy

- Authentication → Showing ID at office entrance
- Authorization → Accessing manager's cabin

Important Rule:

Authentication happens first, authorization comes next.

---

2. Why Authorization Is Extremely Important

Even if authentication is strong:

- A normal user should NOT access admin APIs
- A student should NOT modify marks
- An employee should NOT access HR salary data

If authorization is weak → **Privilege Escalation attacks happen**.

---

3. RBAC – Role-Based Access Control

3.1 What Is RBAC?

Access is granted based on a **user's role**.

User → Role → Permissions

3.2 Common Roles

- Admin
- Manager
- Employee
- User

3.3 RBAC Example

Role	Permissions
Admin	Create, Read, Update, Delete
Manager	Read, Update
User	Read

3.4 How RBAC Works (Step-by-Step)

1. User logs in
2. System checks user role
3. Access is granted or denied

3.5 Advantages of RBAC

- Simple to understand
- Easy to manage
- Widely used in companies

3.6 Disadvantages of RBAC

- Not flexible for complex conditions
- Too many roles can become messy

Interview Line (Very Strong)

“RBAC is simple and scalable, but can become rigid in complex environments.”

---

4. ABAC – Attribute-Based Access Control

4.1 What Is ABAC?

Access is based on **attributes**, not just roles.

Attributes can belong to:

- User (department, age)
- Resource (owner, classification)
- Environment (time, location)

4.2 ABAC Example

Rule:

Allow access if:

user.department == "HR"

AND resource.type == "Payroll"

AND time < 6 PM

4.3 Why ABAC Is Powerful

- Very flexible
- Context-aware
- Fine-grained control

4.4 Disadvantages

- Complex to implement
- Harder to debug

Interview Comparison

“RBAC answers WHO, ABAC answers WHO + WHEN + WHERE + CONDITIONS.”

---

5. ACL – Access Control Lists

5.1 What Is an ACL?

A list attached to a **resource** that defines:

Who can access this resource and what they can do

5.2 ACL Example (File System)

User	Permission
Alice	Read
Bob	Read, Write
Charlie	No access

5.3 Where ACLs Are Used

- File systems (Linux permissions)
- Network devices
- Cloud storage

5.4 Pros & Cons

✅ Fine-grained
❌ Hard to manage at scale

Interview Line

“ACLs are resource-centric, while RBAC is user-role-centric.”

---

6. Least Privilege (MOST IMPORTANT PRINCIPLE)

6.1 What Is Least Privilege?

Users should have **only the minimum permissions needed** to do their job.

Nothing more. Nothing extra.

6.2 Why Least Privilege Matters

- Limits damage if account is compromised
- Prevents accidental misuse
- Reduces attack surface

6.3 Example

❌ Bad:

- Normal user has admin access

✅ Good:

- User can only view their own data

Interview Line (Must Remember)

“Least privilege minimizes the blast radius of security incidents.”

---

7. Authorization Attacks (Interview Awareness)

You should be aware of:

- Privilege escalation
- IDOR (Insecure Direct Object Reference)
- Broken access control (OWASP Top 10)

---

8. Hands-On: Implement RBAC in Flask (Step-by-Step)

Below is **simple, clean, interview-safe RBAC implementation** using Flask.

8.1 Basic Flask Setup

```
from flask import Flask, jsonify, request
from functools import wraps

app = Flask(__name__)
```

8.2 Mock User Database

```
users = {
    "kavya": {"role": "admin"},
    "arjun": {"role": "user"}
}
```

8.3 Role-Based Decorator (CORE LOGIC)

```
def role_required(required_role):
    def decorator(f):
        @wraps(f)
        def wrapper(username, *args, **kwargs):
            user = users.get(username)
            if not user:
                return jsonify({"error": "User not found"}), 404

            if user["role"] != required_role:
                return jsonify({"error": "Access denied"}), 403

            return f(username, *args, **kwargs)
        return wrapper
    return decorator
```

8.4 Protected Routes

```
@app.route("/admin/<username>")
@role_required("admin")
def admin_dashboard(username):
    return jsonify({"message": "Welcome Admin"})
@app.route("/user/<username>")
@role_required("user")
def user_dashboard(username):
    return jsonify({"message": "Welcome User"})
```

8.5 Testing Scenarios

User	Endpoint	Result
kavya (admin)	/admin/kavya	Allowed
arjun (user)	/admin/arjun	Denied

User	Endpoint	Result
arjun (user)	/user/arjun	Allowed

---

9. How This Is Used in Real Projects (Important)

In real apps:

- Roles stored in DB
- Decorators check roles
- Combined with authentication (JWT / session)
- Logging on access denial