```python
# User Group & Role Management with Access Control and Workflows (FastAPI + SQLAlchemy)

from sqlalchemy import Column, Integer, String, ForeignKey, Boolean, Table
from sqlalchemy.orm import relationship, declarative_base

Base = declarative_base()

# Association table for many-to-many (Role <-> Permission)
role_permissions = Table(
    "role_permissions",
    Base.metadata,
    Column("role_id", ForeignKey("roles.id"), primary_key=True),
    Column("permission_id", ForeignKey("permissions.id"), primary_key=True)
)

# Association for Users <-> Groups
user_groups = Table(
    "user_groups",
    Base.metadata,
    Column("user_id", ForeignKey("users.id"), primary_key=True),
    Column("group_id", ForeignKey("groups.id"), primary_key=True)
)

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True)
    username = Column(String, unique=True)
    groups = relationship("Group", secondary=user_groups, back_populates="users")

class Group(Base):
    __tablename__ = "groups"
    id = Column(Integer, primary_key=True)
    name = Column(String, unique=True)
    roles = relationship("Role", back_populates="group")
    users = relationship("User", secondary=user_groups, back_populates="groups")

class Role(Base):
    __tablename__ = "roles"
    id = Column(Integer, primary_key=True)
    name = Column(String)
    group_id = Column(Integer, ForeignKey("groups.id"))
    group = relationship("Group", back_populates="roles")
    permissions = relationship("Permission", secondary=role_permissions, back_populates="roles")

class Permission(Base):
    __tablename__ = "permissions"
    id = Column(Integer, primary_key=True)
    action = Column(String, unique=True)  # e.g., "view_reports", "edit_users"
    roles = relationship("Role", secondary=role_permissions, back_populates="permissions")

class Workflow(Base):
    __tablename__ = "workflows"
    id = Column(Integer, primary_key=True)
    name = Column(String)
    requires_approval = Column(Boolean, default=True)
    assigned_role = Column(Integer, ForeignKey("roles.id"))

# Permission Check
def has_permission(user: User, action: str) -> bool:
    for group in user.groups:
        for role in group.roles:
            for perm in role.permissions:
                if perm.action == action:
                    return True
    return False

# Workflow Manager
class WorkflowManager:
    def __init__(self, db_session):
        self.db = db_session

    def request_action(self, user: User, workflow_name: str):
        workflow = self.db.query(Workflow).filter_by(name=workflow_name).first()
        if not workflow:
            return "Workflow not found"

        if has_permission(user, "approve_" + workflow_name):
            return "Action auto-approved"

        if workflow.requires_approval:
            return "Action pending approval"
        return "Action executed"

# FastAPI Endpoint
from fastapi import FastAPI, Depends, HTTPException
app = FastAPI()

@app.get("/perform/{action}")
def perform_action(action: str, current_user: User = Depends(get_current_user)):
    if not has_permission(current_user, action):
        raise HTTPException(status_code=403, detail="Access denied")
    return {"status": f"Action {action} performed by {current_user.username}"}
```