Actividad: Modulo 3 - Lección 2

app.py

```python
# Importar las librerías necesarias de Flask y sus extensiones
from flask import Flask, render_template, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, login_user, logout_user, login_required, current_user
from flask_principal import (
    Principal, Permission, RoleNeed,
    identity_loaded, Identity, identity_changed, AnonymousIdentity
)
from models import db, User, Role
from auth import auth_blueprint, login_manager

# Inicializar la aplicación Flask
app = Flask(__name__)
app.config['SECRET_KEY'] = 'clave_super_secreta'  # Clave secreta para sesiones
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'  # Base de datos SQLite

# Inicializar extensiones
db.init_app(app)
login_manager.init_app(app)
principal = Principal(app)

# Registrar el blueprint de autenticación
app.register_blueprint(auth_blueprint)

# Definir permisos basados en roles
admin_permission = Permission(RoleNeed('admin'))
editor_permission = Permission(RoleNeed('editor'))

# Cargar identidad y asociar los roles al usuario actual
@identity_loaded.connect_via(app)
def on_identity_loaded(sender, identity):
    identity.user = current_user
    if hasattr(current_user, 'role'):
        identity.provides.add(RoleNeed(current_user.role.name))

# Ruta principal pública
@app.route('/')
```

```python
def index():
    return 'Bienvenido. <a href="/admin">Ir a Admin</a> | <a href="/logout">Cerrar sesión</a>'

# Ruta protegida: solo accesible por usuarios con rol "admin"
@app.route('/admin')
@admin_permission.require(http_exception=403)
def admin_panel():
    return 'Panel de Administración (solo admins)'

# Ruta protegida: solo accesible por usuarios con rol "editor"
@app.route('/editor')
@editor_permission.require(http_exception=403)
def editor_panel():
    return 'Panel de Edición (solo editores)'

# Ruta para cerrar sesión
@app.route('/logout')
@login_required
def logout():
    logout_user()  # Finaliza la sesión
    identity_changed.send(app, identity=AnonymousIdentity())  # Resetea la identidad de permisos
    return redirect(url_for('index'))

# Ejecutar la aplicación
if __name__ == '__main__':
    with app.app_context():
        db.create_all()  # Crear las tablas si no existen

        # Crear roles y usuarios solo si no existen
        if not User.query.first():
            admin_role = Role(name='admin')
            editor_role = Role(name='editor')
            db.session.add_all([admin_role, editor_role])
            db.session.commit()

            admin_user = User(username='admin', password='1234', role_id=admin_role.id)
            editor_user = User(username='editor', password='1234', role_id=editor_role.id)
            db.session.add_all([admin_user, editor_user])
            db.session.commit()

    app.run(debug=True)
```

auth.py

```python
def load_user(user_id):
    return User.query.get(int(user_id))

# Ruta para iniciar sesión (GET para mostrar formulario, POST para enviar datos)
@auth_blueprint.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # Obtener datos del formulario
        username = request.form.get('username')
        password = request.form.get('password')

        # Buscar usuario por nombre
        user = User.query.filter_by(username=username).first()

        # Validar contraseña (no se usa hash aquí por simplicidad)
        if user and user.password == password:
            login_user(user)  # Inicia sesión
            # Asociar identidad del usuario con Flask-Principal
            identity_changed.send(request._get_current_object(), identity=Identity(user.id))
            return redirect(url_for('index'))

        return 'Credenciales incorrectas.'

    # Mostrar formulario simple
    return '''
        <form method="POST">
            Usuario: <input type="text" name="username"><br>
            Contraseña: <input type="password" name="password"><br>
            <input type="submit" value="Iniciar sesión">
        </form>
    '''
```

models.py

```python
# Importar la extensión de base de datos y el mixin de usuario
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin

# Inicializar SQLAlchemy (se activa en app.py)
db = SQLAlchemy()

# Modelo de tabla de roles
class Role(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # ID del rol
    name = db.Column(db.String(50), unique=True)  # Nombre único del rol (ej: "admin", "editor")

# Modelo de tabla de usuarios
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)  # ID del usuario
    username = db.Column(db.String(50), unique=True, nullable=False)  # Nombre de usuario
    password = db.Column(db.String(50), nullable=False)  # Contraseña simple (idealmente usar hash)
    role_id = db.Column(db.Integer, db.ForeignKey('role.id'))  # Clave foránea al rol
    role = db.relationship('Role')  # Relación ORM para acceder al objeto Role directamente
```

requirements.txt

```
Flask
Flask-Login
Flask-Principal
Flask-SQLAlchemy
```

test_requests.http

```
### Prueba 1: Iniciar sesión como ADMIN
Send Request
POST http://localhost:5000/login
Content-Type: application/x-www-form-urlencoded

username=admin&password=1234

### Prueba 2: Acceder al panel de admin (requiere rol 'admin')
Send Request
GET http://localhost:5000/admin
Cookie:

### Prueba 3: Acceder al panel de editor (no debe tener acceso)
Send Request
GET http://localhost:5000/editor
Cookie:

### Prueba 4: Cerrar sesión
Send Request
GET http://localhost:5000/logout
Cookie:

### Prueba 5: Iniciar sesión como EDITOR
Send Request
POST http://localhost:5000/login
Content-Type: application/x-www-form-urlencoded

username=editor&password=1234

### Prueba 6: Acceder al panel de editor (requiere rol 'editor')
Send Request
GET http://localhost:5000/editor
Cookie:

### Prueba 7: Acceder al panel de admin (no debe tener acceso)
Send Request
GET http://localhost:5000/admin
Cookie:
```