MASTER UNIVERSITARIO EN SEGURIDAD INFORMATICA (CIBERSEGURIDAD)

UNIVERSIDAD DE CADIZ

SEGURIDAD EN SISTEMAS DISTRUIBIDOS

MADE BY Adil Hirchi

REST PROJECT: PHARMACY

INTRODUCTION

This REST-based web application manages core operations within a pharmacy. It provides a user-friendly interface with features including:

✓ Inventory consultation.

✓ Product management (add, modify, delete).

✓ Client and user registration.

✓ Activity logging.

✓ System response feedback for all operations.

SECURITY FEATURES

Security features are very important for our API:

- **Protects sensitive data** from unauthorized access.
- **Prevents common attacks** like SQL injection and XSS.
- **Controls access** with authentication.
- **Enables monitoring** through logging and auditing.
- **Ensures compliance** with security regulations.

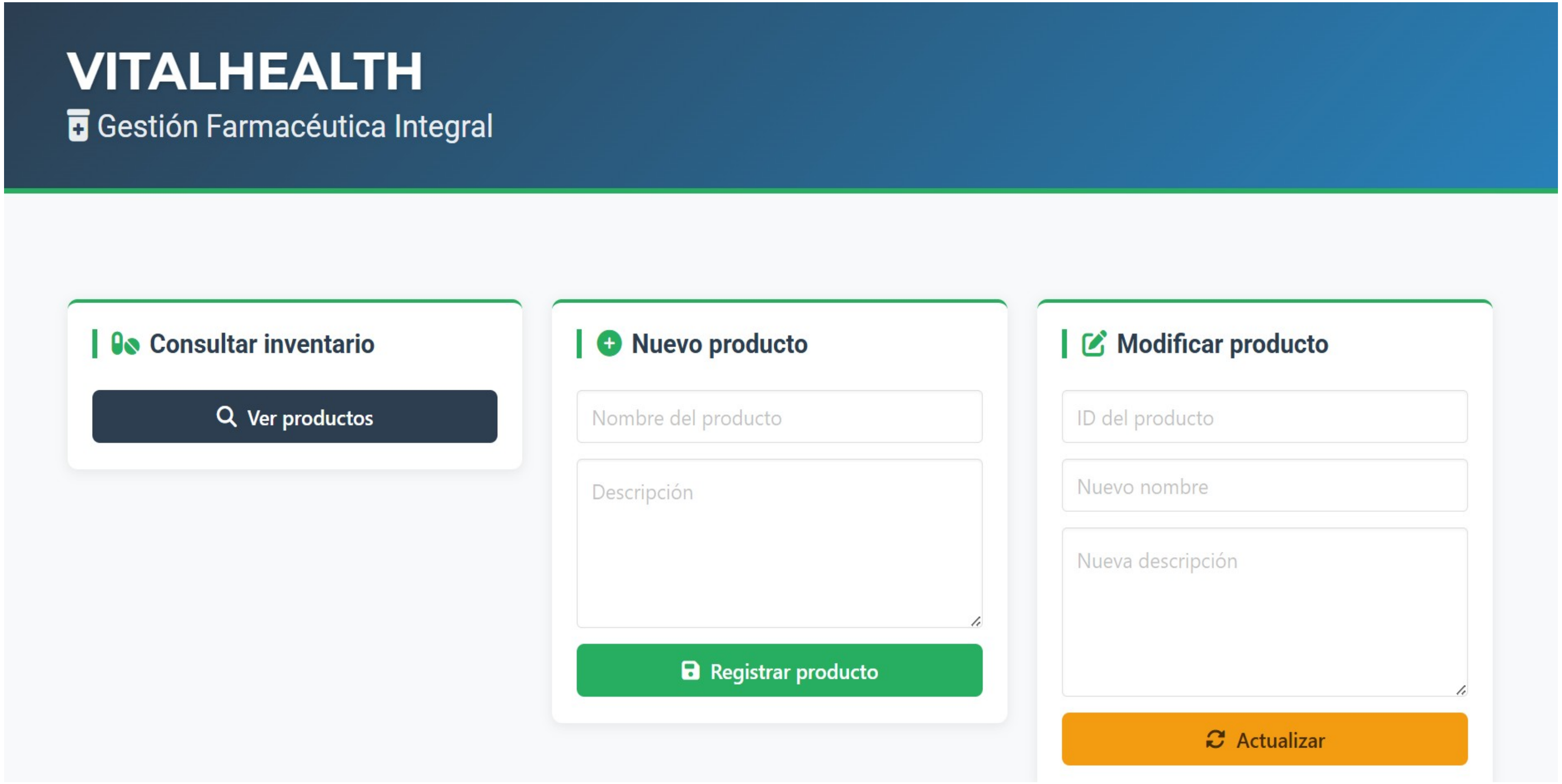


Figure 1: the vitalhealth interface.

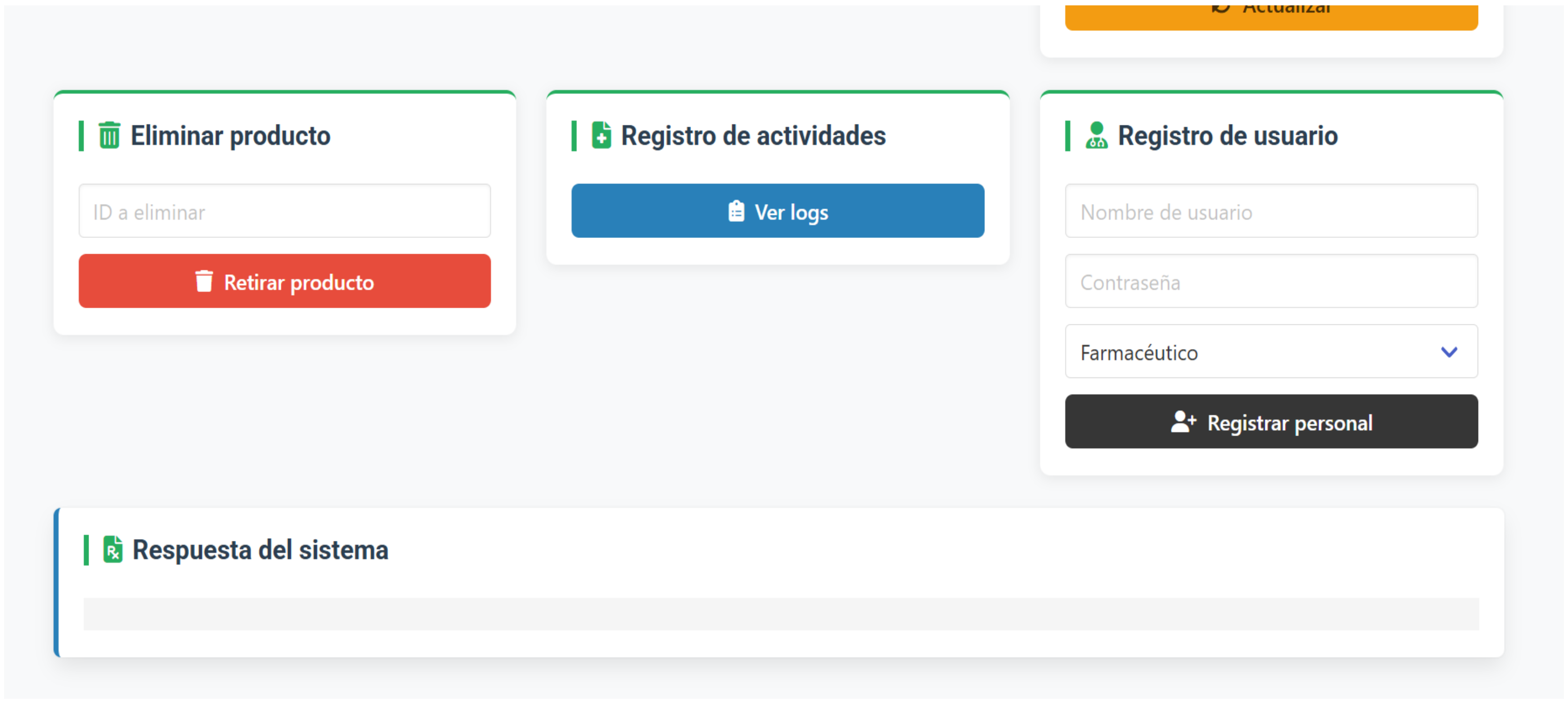
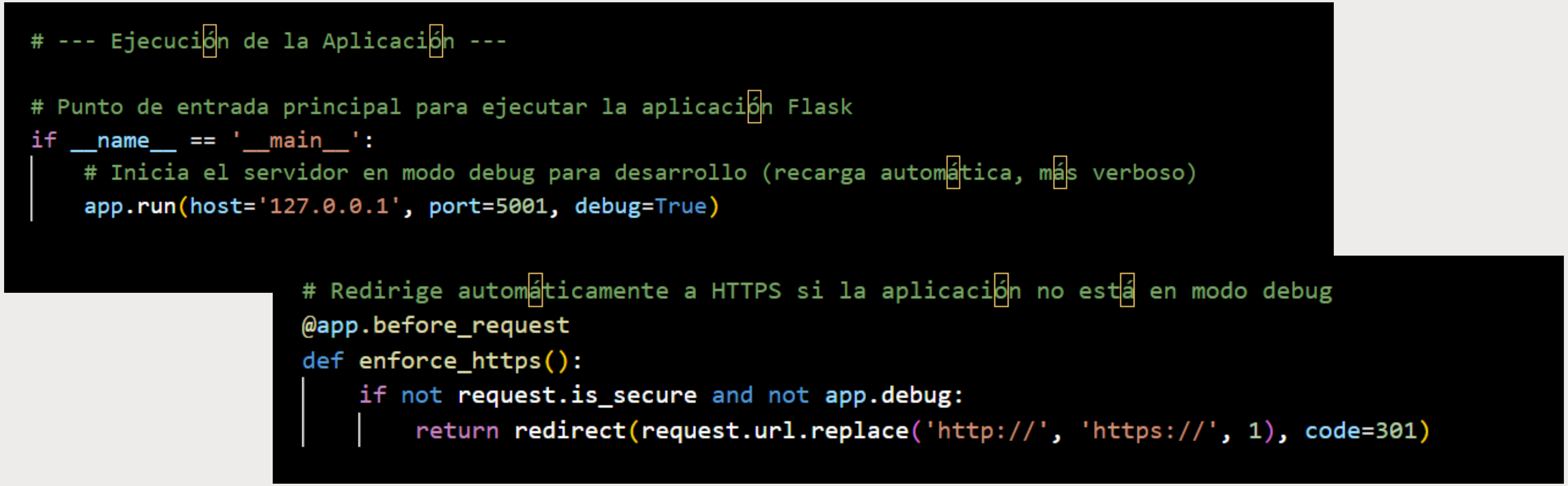


Figure 2: Operations.

HTTPS

HTTPS ensures secure communication between the client and the server by encrypting the data exchange, thereby protecting it from interception or tam attackers (such as man-in-the-middle attacks).



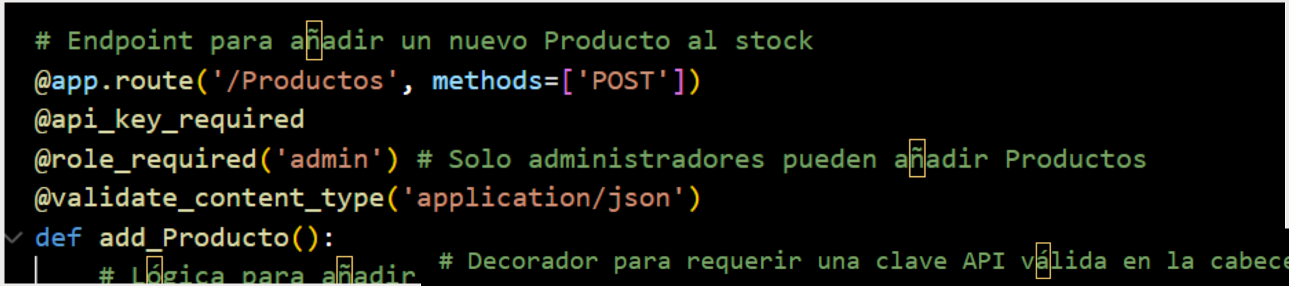
```
# --- Ejecución de la Aplicación ---

# Punto de entrada principal para ejecutar la aplicación Flask
if __name__ == '__main__':
    # Inicia el servidor en modo debug para desarrollo (recarga automática, más verboso)
    app.run(host='127.0.0.1', port=5001, debug=True)
```

```
# Redirige automáticamente a HTTPS si la aplicación no está en modo debug
@app.before_request
def enforce_https():
    if not request.is_secure and not app.debug:
        return redirect(request.url.replace('http://', 'https://', 1), code=301)
```

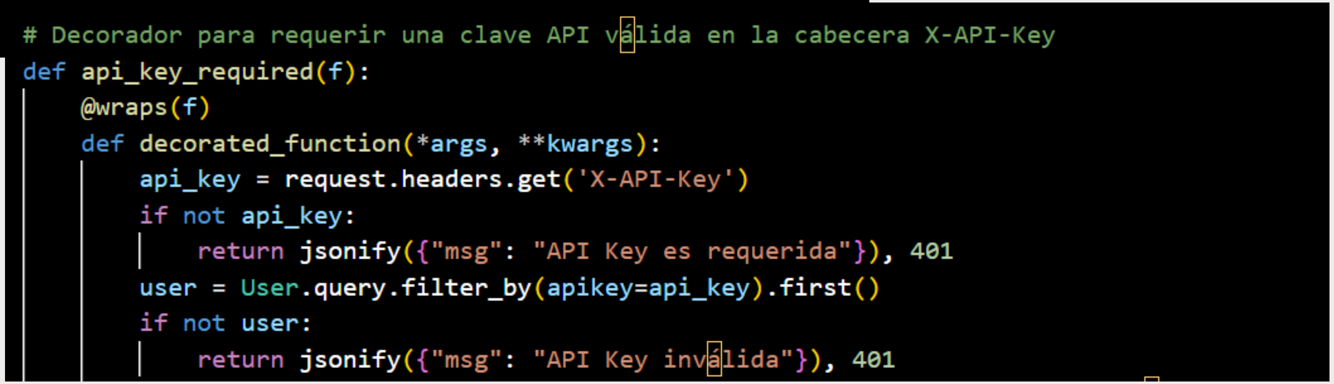
Figure 3: HTTPS configuration.

ACCESS CONTROL

Restricts access to specific routes or features depending on the user's assign



```
# Endpoint para añadir un nuevo Producto al stock
@app.route('/Productos', methods=['POST'])
@api_key_required
@role_required('admin') # Solo administradores pueden añadir Productos
@validate_content_type('application/json')
def add_Producto():
    # Lógica para añadir
```

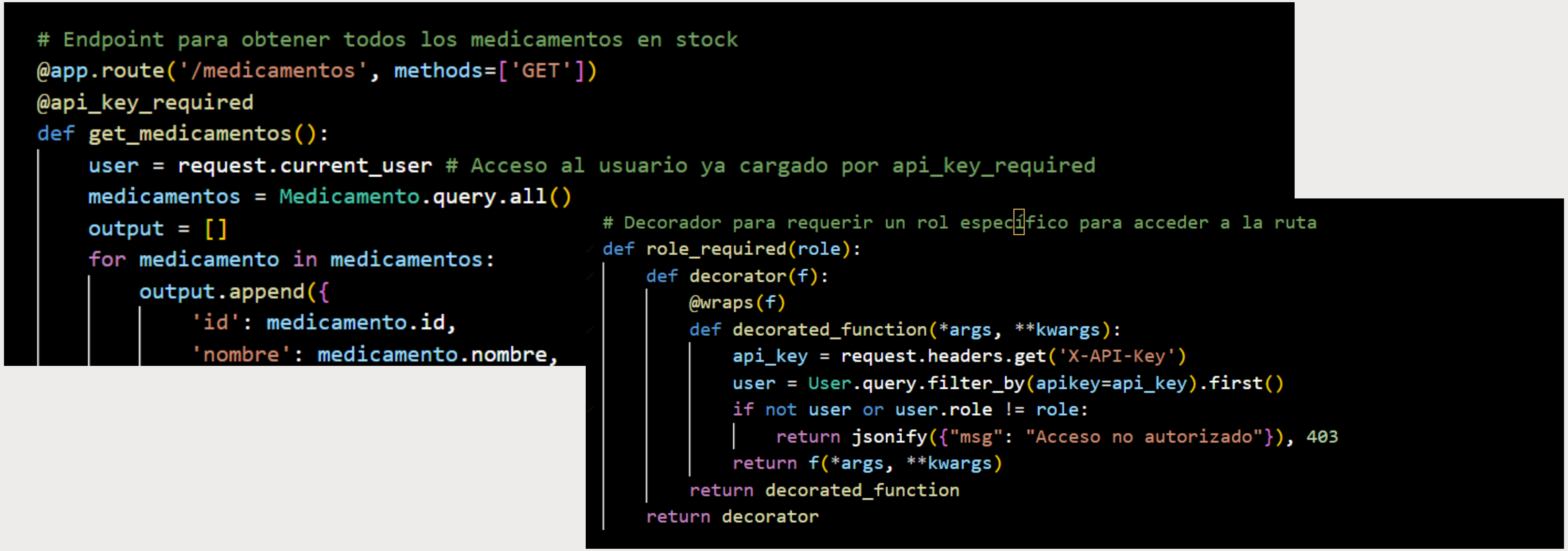


```
# Decorador para requerir una clave API válida en la cabecera X-API-Key
def api_key_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        api_key = request.headers.get('X-API-Key')
        if not api_key:
            return jsonify({"msg": "API Key es requerida"}), 401
        user = User.query.filter_by(apikey=api_key).first()
        if not user:
            return jsonify({"msg": "API Key inválida"}), 401
```

Figure 4: Access control.

API KEYS

Enables authentication and authorization of clients consuming the API.



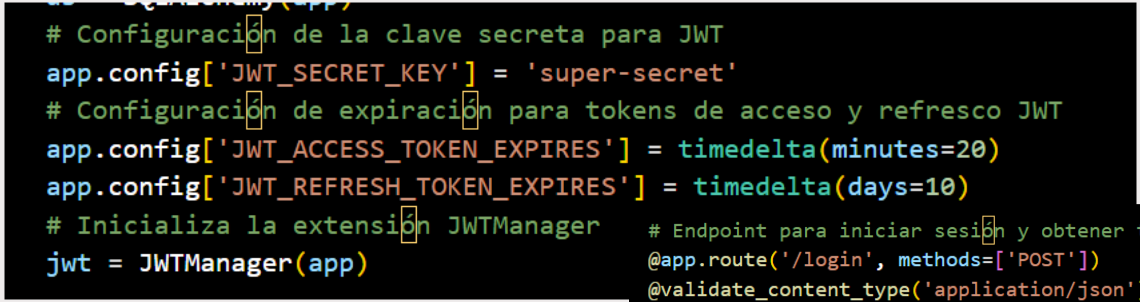
```
# Endpoint para obtener todos los medicamentos en stock
@app.route('/medicamentos', methods=['GET'])
@api_key_required
def get_medicamentos():
    user = request.current_user # Acceso al usuario ya cargado por api_key_required
    medicamentos = Medicamento.query.all()
    output = []
    for medicamento in medicamentos:
        output.append({
            'id': medicamento.id,
            'nombre': medicamento.nombre,
```

```
# Decorador para requerir un rol específico para acceder a la ruta
def role_required(role):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            api_key = request.headers.get('X-API-Key')
            user = User.query.filter_by(apikey=api_key).first()
            if not user or user.role != role:
                return jsonify({"msg": "Acceso no autorizado"}), 403
            return f(*args, **kwargs)
        return decorated_function
    return decorator
```

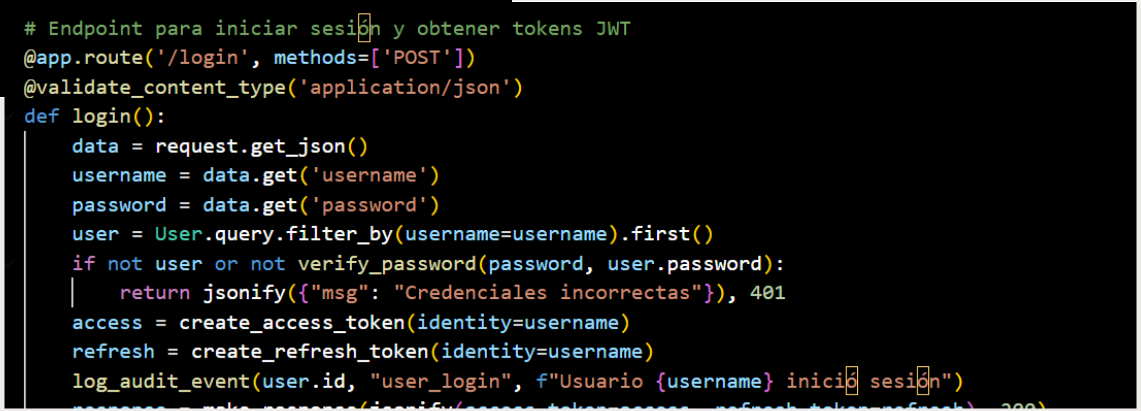
Figure 5: API Keys.

JWT

JWT is used to securely authenticate users and handle session management.



```python
# Configuración de la clave secreta para JWT
app.config['JWT_SECRET_KEY'] = 'super-secret'
# Configuración de expiración para tokens de acceso y refresco JWT
app.config['JWT_ACCESS_TOKEN_EXPIRES'] = timedelta(minutes=20)
app.config['JWT_REFRESH_TOKEN_EXPIRES'] = timedelta(days=10)
# Inicializa la extensión JWTManager
jwt = JWTManager(app)
```

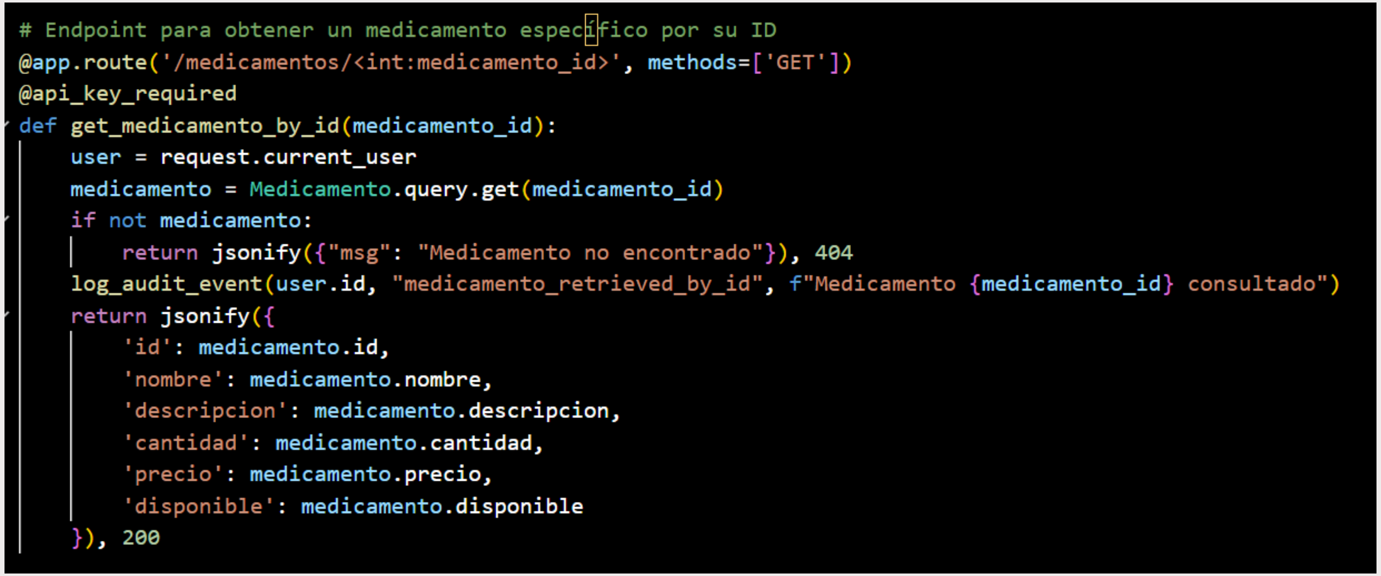


```python
# Endpoint para iniciar sesión y obtener tokens JWT
@app.route('/login', methods=['POST'])
@validate_content_type('application/json')
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')
    user = User.query.filter_by(username=username).first()
    if not user or not verify_password(password, user.password):
        return jsonify({"msg": "Credenciales incorrectas"}), 401
    access = create_access_token(identity=username)
    refresh = create_refresh_token(identity=username)
    log_audit_event(user.id, "user_login", f"Usuario {username} inició sesión")
```

Figure 6: JWT.

RESTRICT HTTP METHODS

Restricts which HTTP methods are permitted on each route.
This is explicitly set within the route definitions(like **methods=['GET', 'POST**

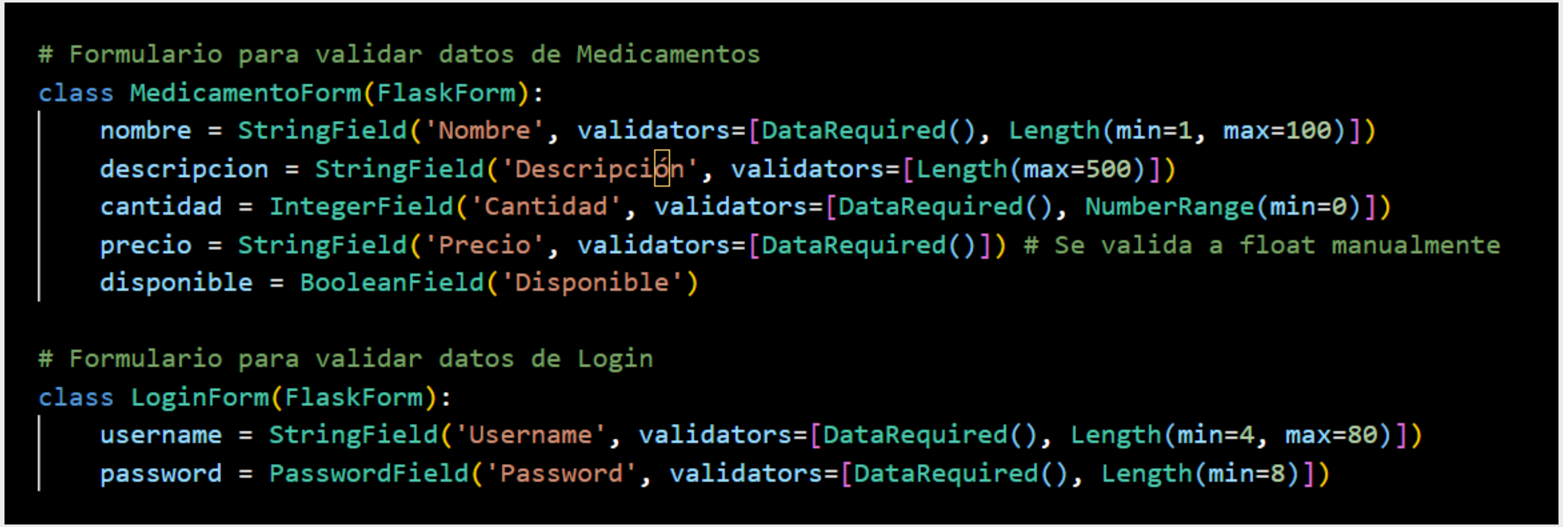


```python
# Endpoint para obtener un medicamento específico por su ID
@app.route('/medicamentos/<int:medicamento_id>', methods=['GET'])
@api_key_required
def get_medicamento_by_id(medicamento_id):
    user = request.current_user
    medicamento = Medicamento.query.get(medicamento_id)
    if not medicamento:
        return jsonify({"msg": "Medicamento no encontrado"}), 404
    log_audit_event(user.id, "medicamento_retrieved_by_id", f"Medicamento {medicamento_id} consultado")
    return jsonify({
        'id': medicamento.id,
        'nombre': medicamento.nombre,
        'descripcion': medicamento.descripcion,
        'cantidad': medicamento.cantidad,
        'precio': medicamento.precio,
        'disponible': medicamento.disponible
    }), 200
```

Figure 7: Restrict http methods.

INPUT VALIDATION

Validates input data to prevent malicious code or injection attacks.

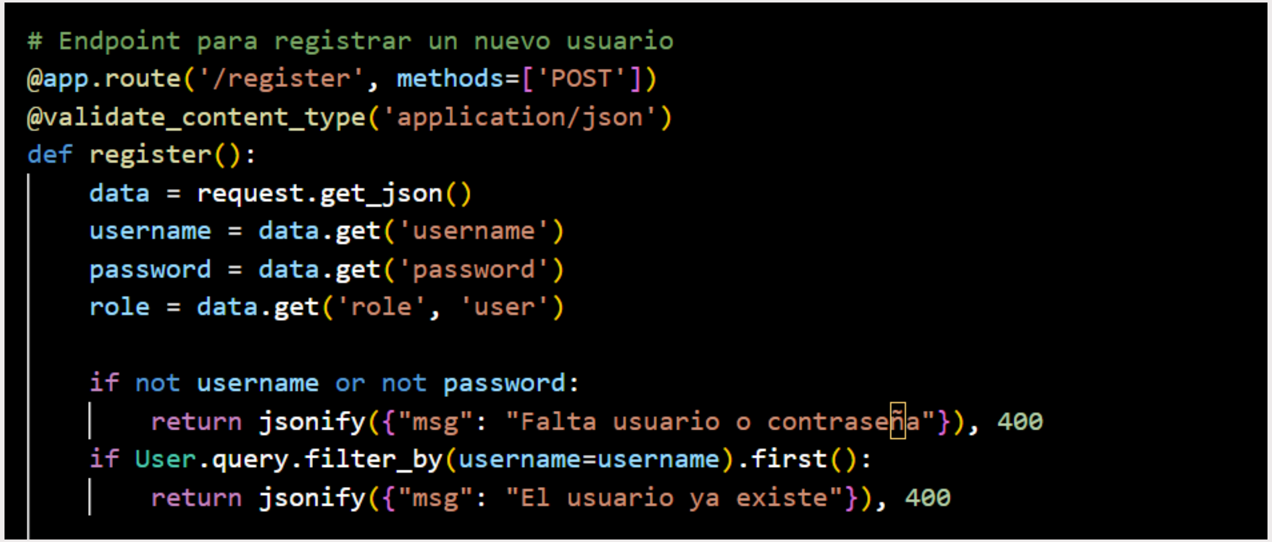


```
# Formulario para validar datos de Medicamentos
class MedicamentoForm(FlaskForm):
    nombre = StringField('Nombre', validators=[DataRequired(), Length(min=1, max=100)])
    descripcion = StringField('Descripción', validators=[Length(max=500)])
    cantidad = IntegerField('Cantidad', validators=[DataRequired(), NumberRange(min=0)])
    precio = StringField('Precio', validators=[DataRequired()]) # Se valida a float manualmente
    disponible = BooleanField('Disponible')

# Formulario para validar datos de Login
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=4, max=80)])
    password = PasswordField('Password', validators=[DataRequired(), Length(min=8)])
```

Figure 8: Input validation.

VALIDATE CONTENT TYPES

Ensures that requests have the expected content type (e.g. application/json).

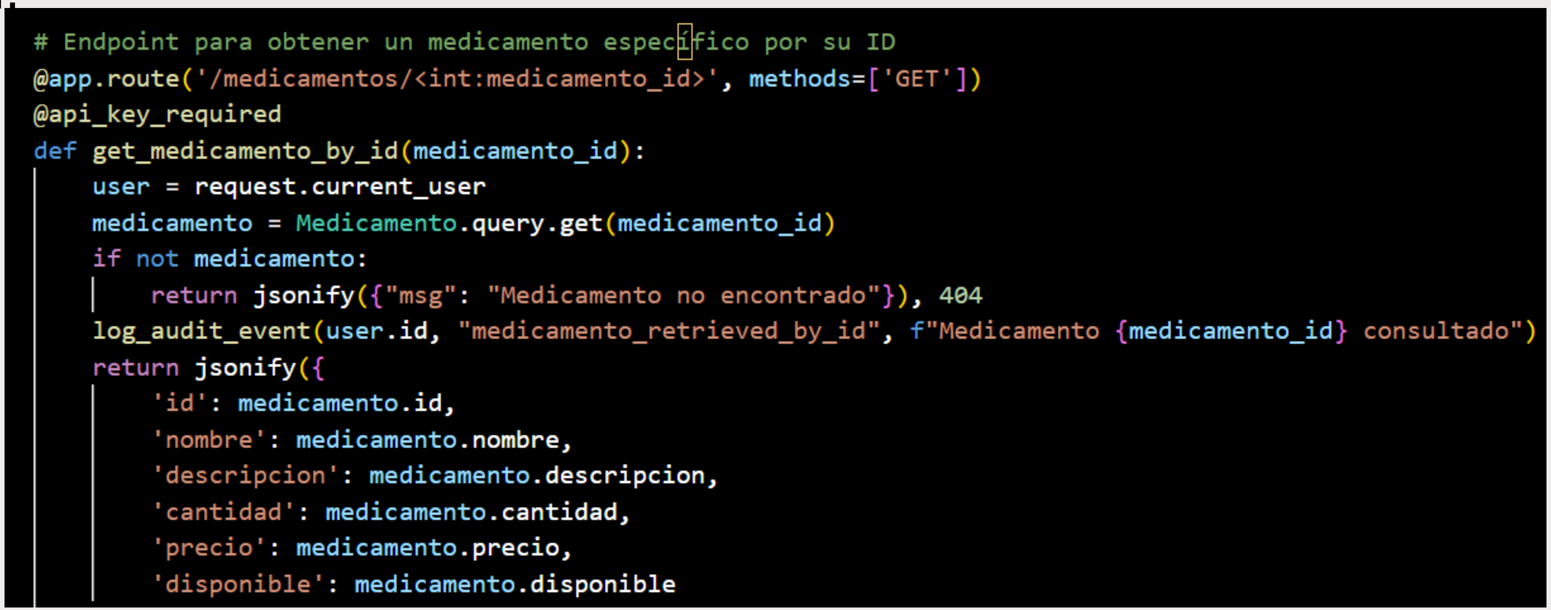


```python
# Endpoint para registrar un nuevo usuario
@app.route('/register', methods=['POST'])
@validate_content_type('application/json')
def register():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')
    role = data.get('role', 'user')

    if not username or not password:
        return jsonify({"msg": "Falta usuario o contraseña"}), 400
    if User.query.filter_by(username=username).first():
        return jsonify({"msg": "El usuario ya existe"}), 400
```

Figure 9: Validate content types.

MANAGEMENT ENDPOINTS

Provides an interface for administrative tasks, such as adding medicaments or
them.

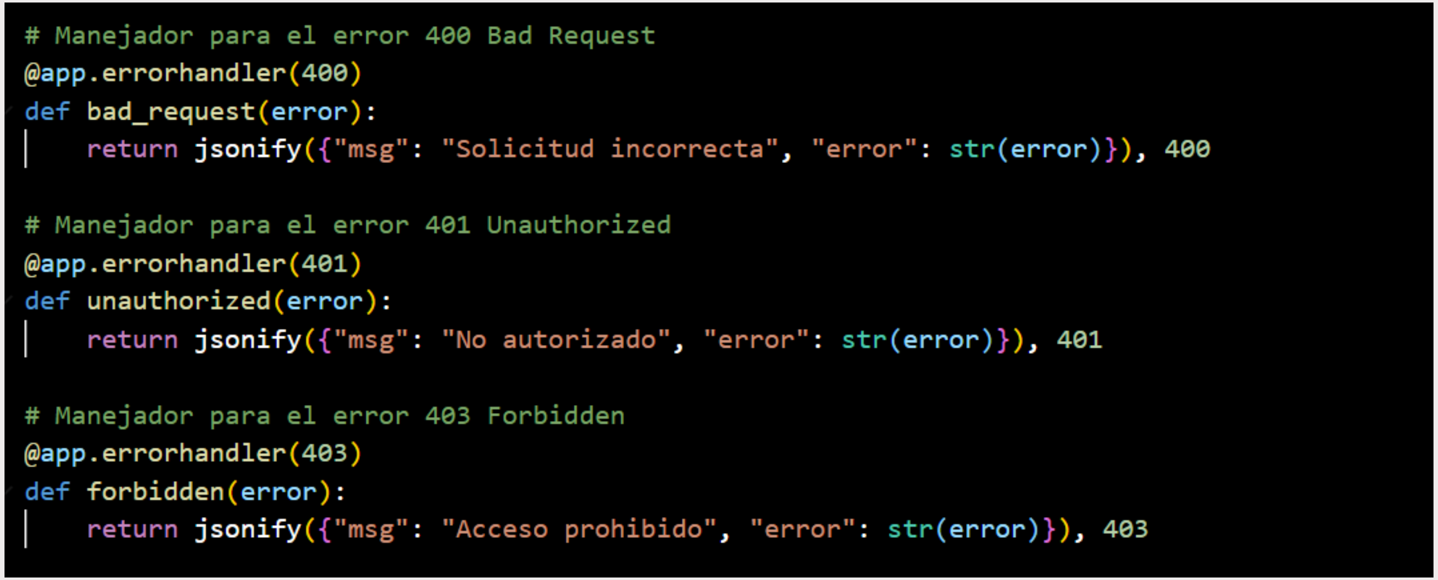


```
# Endpoint para obtener un medicamento específico por su ID
@app.route('/medicamentos/<int:medicamento_id>', methods=['GET'])
@api_key_required
def get_medicamento_by_id(medicamento_id):
    user = request.current_user
    medicamento = Medicamento.query.get(medicamento_id)
    if not medicamento:
        return jsonify({"msg": "Medicamento no encontrado"}), 404
    log_audit_event(user.id, "medicamento_retrieved_by_id", f"Medicamento {medicamento_id} consultado")
    return jsonify({
        'id': medicamento.id,
        'nombre': medicamento.nombre,
        'descripcion': medicamento.descripcion,
        'cantidad': medicamento.cantidad,
        'precio': medicamento.precio,
        'disponible': medicamento.disponible
```

Figure 10: Management endpoints.

ERROR HANDLING

Handles errors gracefully by returning appropriate HTTP status codes and clea
informative messages.

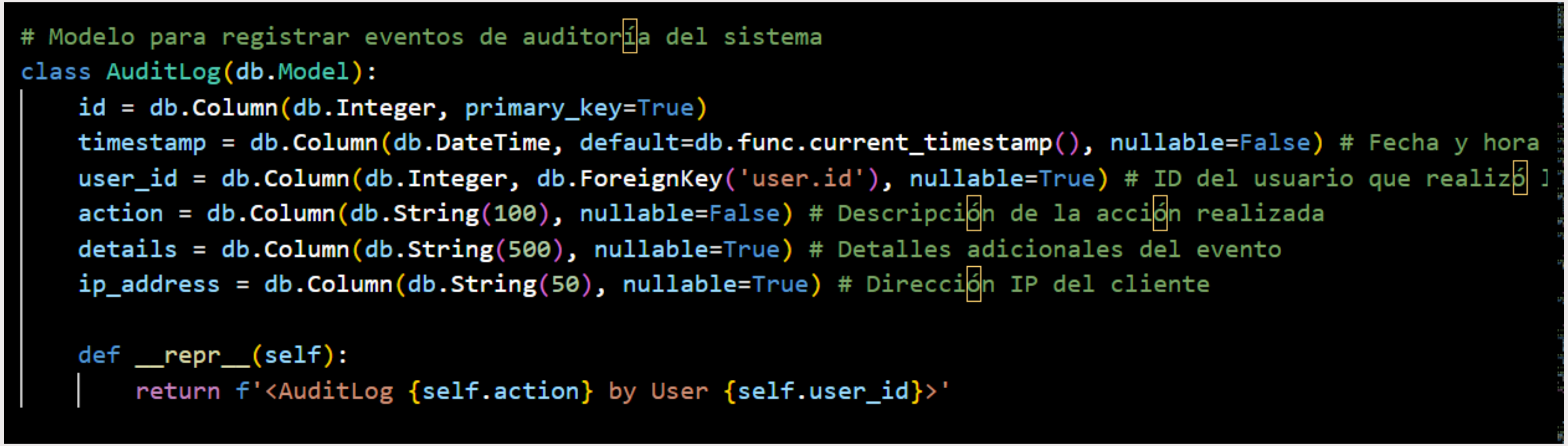


```python
# Manejador para el error 400 Bad Request
@app.errorhandler(400)
def bad_request(error):
    return jsonify({"msg": "Solicitud incorrecta", "error": str(error)}), 400

# Manejador para el error 401 Unauthorized
@app.errorhandler(401)
def unauthorized(error):
    return jsonify({"msg": "No autorizado", "error": str(error)}), 401

# Manejador para el error 403 Forbidden
@app.errorhandler(403)
def forbidden(error):
    return jsonify({"msg": "Acceso prohibido", "error": str(error)}), 403
```

Figure 11: Error handling.

AUDIT LOGS

Logs important events, such as user logins and administrative actions. This enables monitoring and auditing of system activity, which is essential for and responding to security incidents.

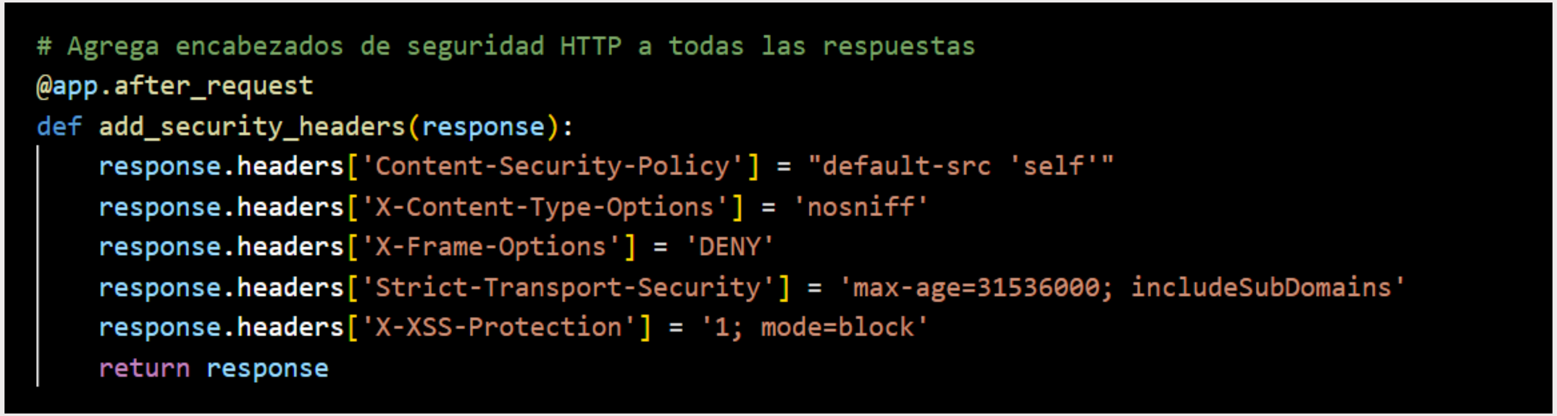


```
# Modelo para registrar eventos de auditoría del sistema
class AuditLog(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    timestamp = db.Column(db.DateTime, default=db.func.current_timestamp(), nullable=False) # Fecha y hora
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True) # ID del usuario que realizó
    action = db.Column(db.String(100), nullable=False) # Descripción de la acción realizada
    details = db.Column(db.String(500), nullable=True) # Detalles adicionales del evento
    ip_address = db.Column(db.String(50), nullable=True) # Dirección IP del cliente

    def __repr__(self):
        return f'<AuditLog {self.action} by User {self.user_id}>'
```

Figure 12: Audit logs.

SECURITY HEADERS

Adds HTTP security headers to protect against common attacks.

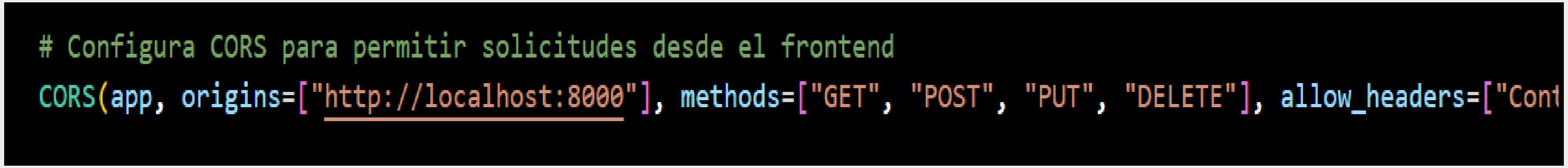


```python
# Agrega encabezados de seguridad HTTP a todas las respuestas
@app.after_request
def add_security_headers(response):
    response.headers['Content-Security-Policy'] = "default-src 'self'"
    response.headers['X-Content-Type-Options'] = 'nosniff'
    response.headers['X-Frame-Options'] = 'DENY'
    response.headers['Strict-Transport-Security'] = 'max-age=31536000; includeSubDomains'
    response.headers['X-XSS-Protection'] = '1; mode=block'
    return response
```

Figure 13: Security headers

CROSS-ORIGIN RESSOURCE SHARING

Using flask_cors.CORS, the API is secured by only allowing requests from **http://localhost:8000** domain which prevents unauthorized Access and CSF

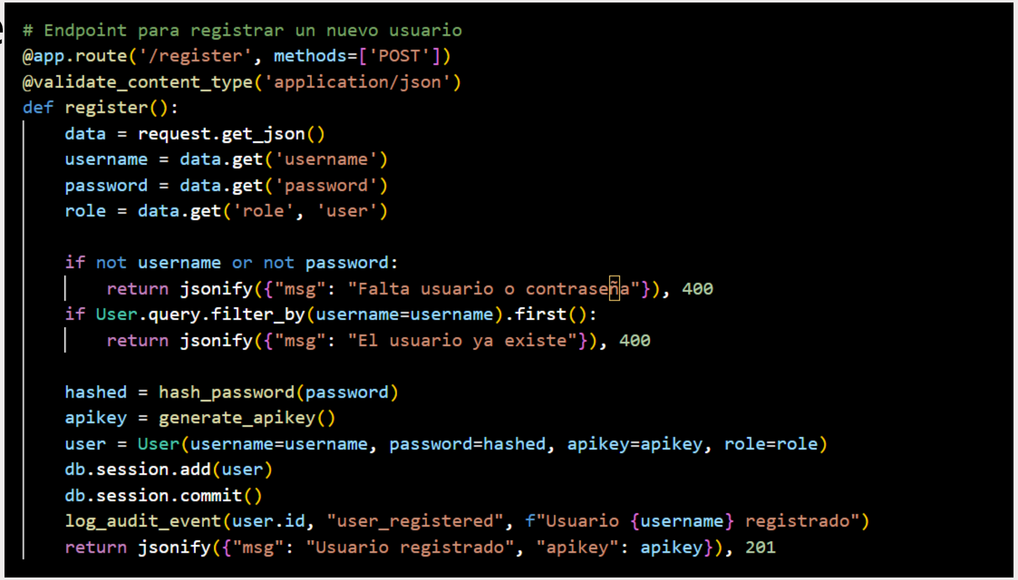


```
# Configura CORS para permitir solicitudes desde el frontend
CORS(app, origins=["http://localhost:8000"], methods=["GET", "POST", "PUT", "DELETE"], allow_headers=["Cont
```

Figure 14: Cross-Origin ressource sharing.

SENSITIVE INFORMATION USING HTTP REQUESTS

Prevent sensitive information (passwords, API keys) from being exposed in U
logs, handled by sending sensitive data in the body of requests and protects
sensitive data leakage

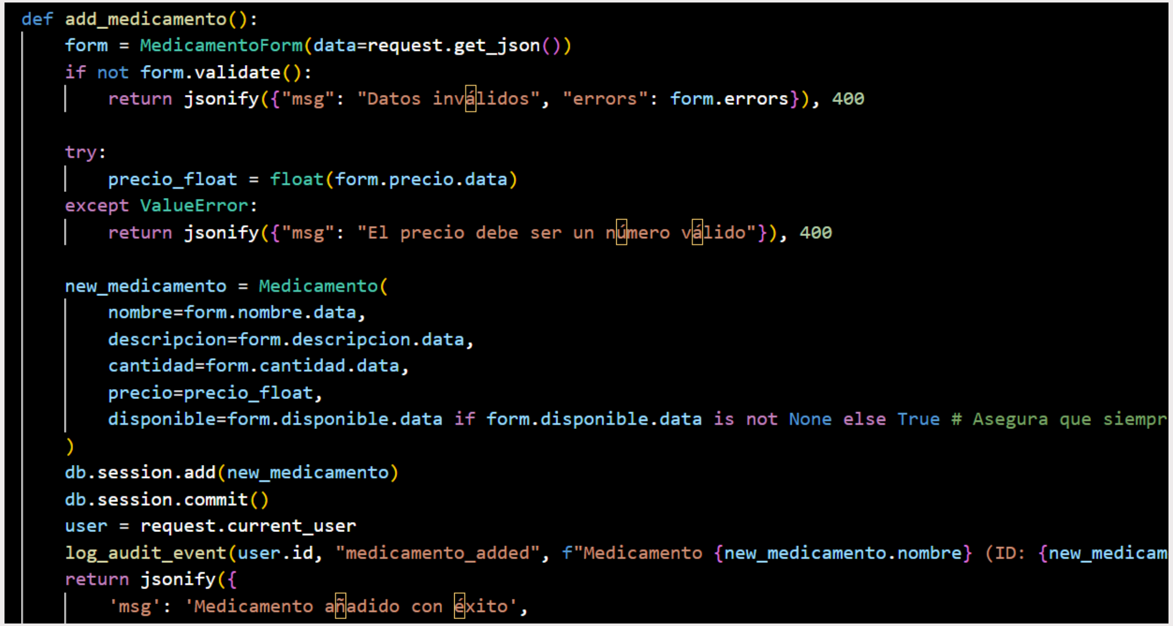


```
# Endpoint para registrar un nuevo usuario
@app.route('/register', methods=['POST'])
@validate_content_type('application/json')
def register():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')
    role = data.get('role', 'user')

    if not username or not password:
        return jsonify({"msg": "Falta usuario o contraseña"}), 400
    if User.query.filter_by(username=username).first():
        return jsonify({"msg": "El usuario ya existe"}), 400

    hashed = hash_password(password)
    apikey = generate_apikey()
    user = User(username=username, password=hashed, apikey=apikey, role=role)
    db.session.add(user)
    db.session.commit()
    log_audit_event(user.id, "user_registered", f"Usuario {username} registrado")
    return jsonify({"msg": "Usuario registrado", "apikey": apikey}), 201
```

Figure 15: Sensitive information using http requests.

HTTP RETURN CODES

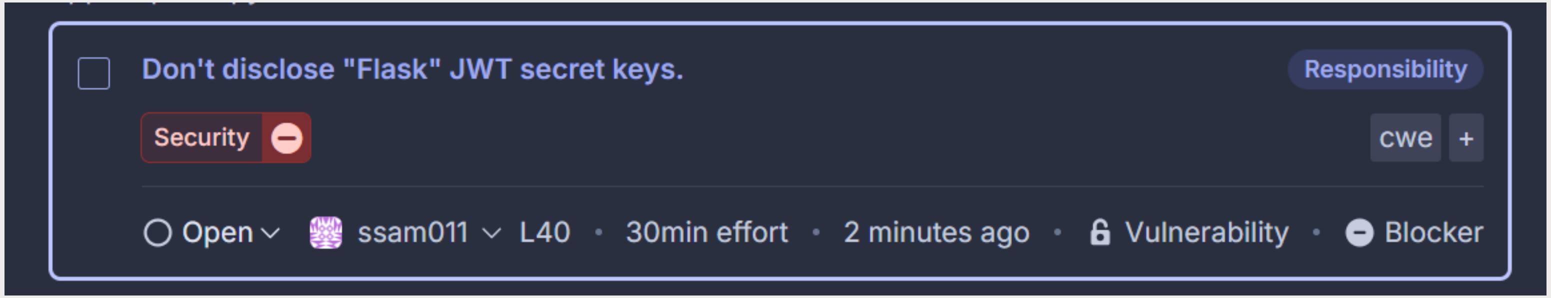
For clear client feedback, all routes return proper HTTP status codes, following
practices.



```
def add_medicamento():
    form = MedicamentoForm(data=request.get_json())
    if not form.validate():
        return jsonify({"msg": "Datos inválidos", "errors": form.errors}), 400

    try:
        precio_float = float(form.precio.data)
    except ValueError:
        return jsonify({"msg": "El precio debe ser un número válido"}), 400

    new_medicamento = Medicamento(
        nombre=form.nombre.data,
        descripcion=form.descripcion.data,
        cantidad=form.cantidad.data,
        precio=precio_float,
        disponible=form.disponible.data if form.disponible.data is not None else True # Asegura que siempr
    )
    db.session.add(new_medicamento)
    db.session.commit()
    user = request.current_user
    log_audit_event(user.id, "medicamento_added", f"Medicamento {new_medicamento.nombre} (ID: {new_medicam
    return jsonify({
        'msg': 'Medicamento añadido con éxito',
```

Figure 16: http return codes.

EVALUATION ATTACKS WITH SONARQUBE

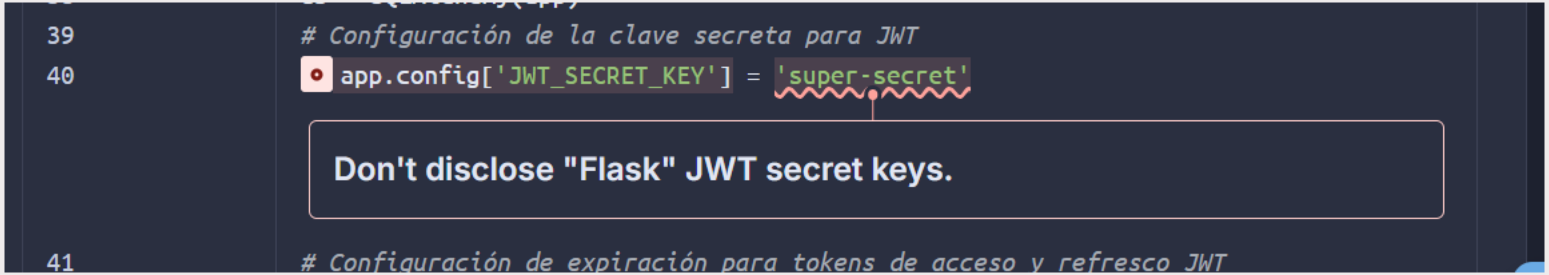

Don't disclose "Flask" JWT secret keys.

Responsibility

Security

cwe +

Open ssam011 L40 · 30min effort · 2 minutes ago · Vulnerability · Blocker

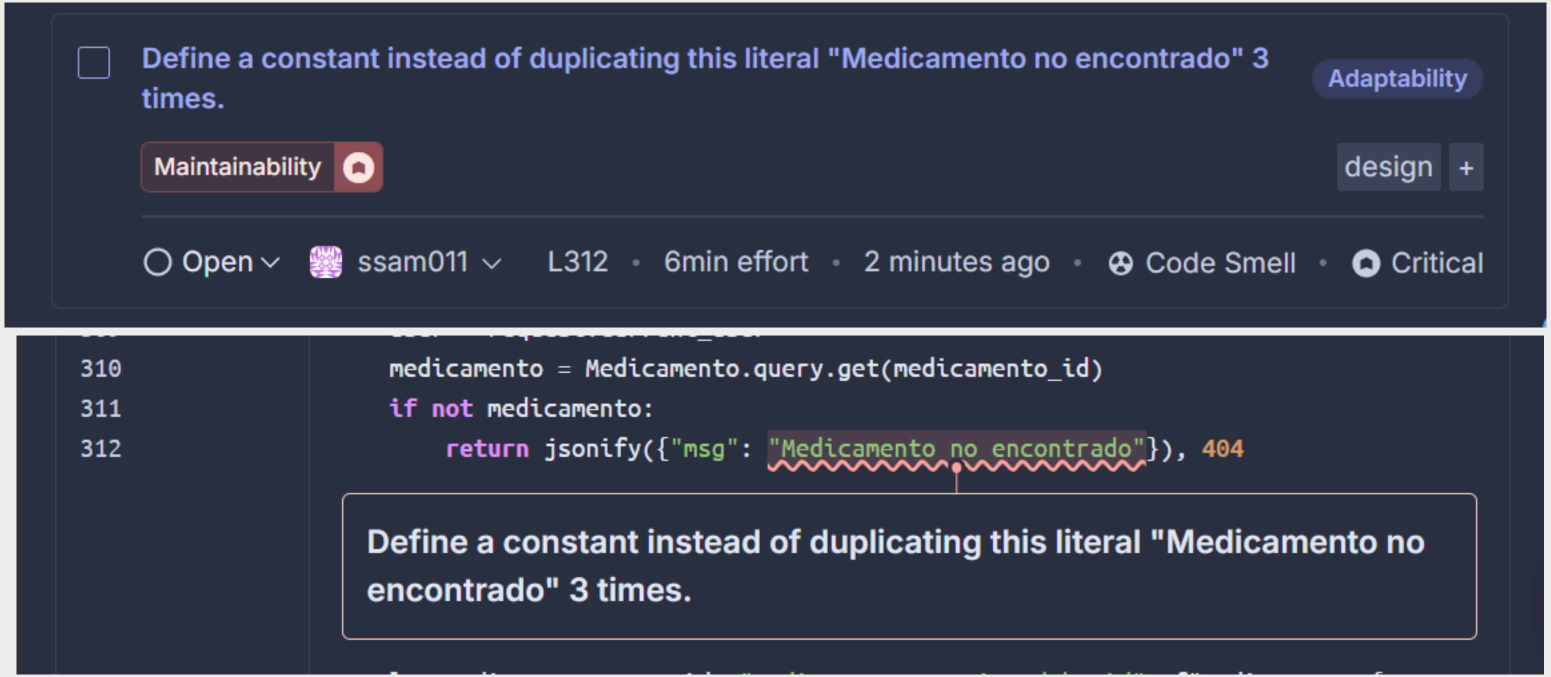

```
# Configuración de la clave secreta para JWT
app.config['JWT_SECRET_KEY'] = 'super-secret'
```

Don't disclose "Flask" JWT secret keys.

```
# Configuración de expiración para tokens de acceso y refresco JWT
```

Figure 17: JWT secret keys.

EVALUATION ATTACKS WITH SONARQUBE



Define a constant instead of duplicating this literal "Medicamento no encontrado" 3 times.

Adaptability

Maintainability

design +

Open ssam011 L312 • 6min effort • 2 minutes ago • Code Smell • Critical

```
    medicamento = Medicamento.query.get(medicamento_id)
    if not medicamento:
        return jsonify({"msg": "Medicamento no encontrado"}), 404
```

Define a constant instead of duplicating this literal "Medicamento no encontrado" 3 times.

Figure 18: Duplicating literal.

DEMO AND CONCLUSION

THANK YOU!