GESTOR DE TAREAS

Documentación Técnica

Versión 1.0

Fecha: Julio 2025

Autor: Equipo de Desarrollo

Contents

MANUAL TÉCNICO

1. INTRODUCCIÓN

El presente manual técnico describe la arquitectura, componentes y funcionamiento interno del sistema Gestor de Tareas desarrollado con Flask. Esta aplicación permite gestionar usuarios y tareas de manera eficiente, empleando una arquitectura modular que favorece el mantenimiento y la escalabilidad del sistema. El backend está desarrollado en Python usando Flask y SQLAlchemy, y el frontend en HTML, CSS y JavaScript.

2. OBJETIVOS DEL MANUAL

Objetivo general:

• Proveer una guía técnica para facilitar la implementación, capacitación y mantenimiento del sistema.

Objetivos específicos:

• Describir la arquitectura general del sistema.

• Explicar la estructura del proyecto, configuración y uso de los modelos, rutas y servicios.

• Detallar la instalación y despliegue del sistema.

3. ARQUITECTURA DEL SISTEMA

El sistema utiliza una arquitectura MVC (Modelo-Vista-Controlador), donde Flask actúa como el controlador central. La información se almacena en una base de datos relacional gestionada con SQLAlchemy. El despliegue se realiza en Render.

Componentes principales:

• Flask: Framework principal del backend.

• SQLAlchemy: ORM para interacción con la base de datos.

• HTML, CSS, JavaScript: Tecnologías del frontend.

• Render: Plataforma de despliegue continuo.

4. ESTRUCTURA DEL PROYECTO

El proyecto sigue una organización modular basada en Blueprints de Flask. La estructura base es:

• /app: núcleo del sistema.

• /app/routes: contiene los Blueprints de usuarios y tareas.

• /app/models: define los modelos de datos.

• /app/services: lógica de negocio.

• /templates: vistas HTML con Jinja2.

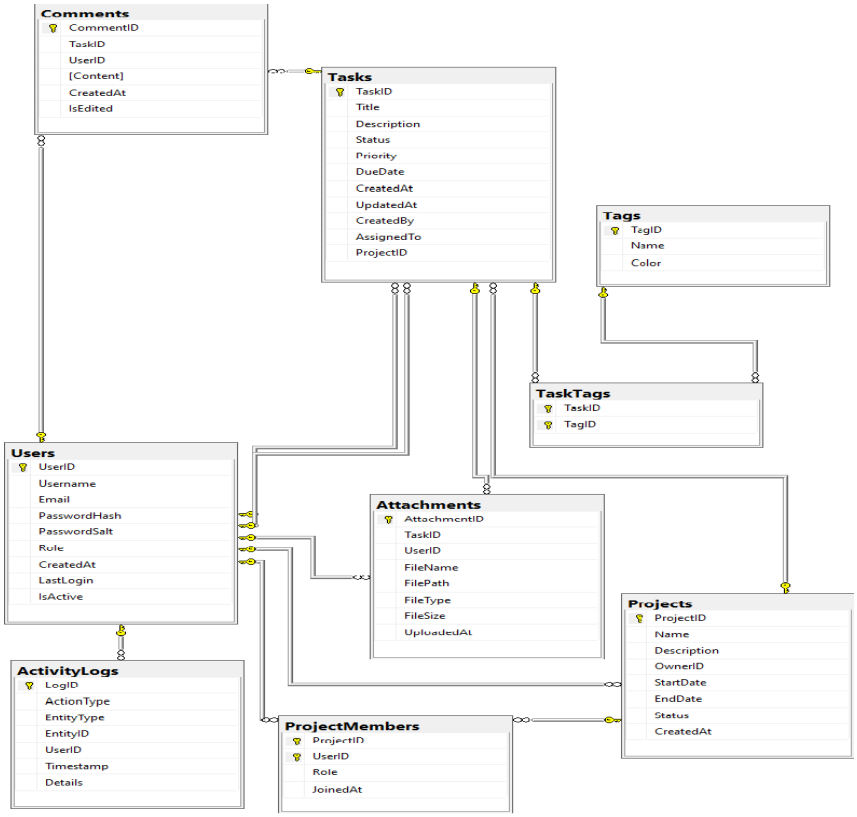
• /static: recursos estáticos (CSS, JS).

• config.py: configuración por entorno.

• run.py: archivo de entrada de la aplicación.

5. BASE DE DATOS

La base de datos se gestiona con SQLAlchemy. Se definen modelos como Usuario y Tarea, los cuales incluyen relaciones y validaciones.



6. ENTORNO DE DESARROLLO

Requisitos:

• Python 3.10+

• SQL Server o SQLite (según entorno).

• pip y entorno virtual.

Pasos para configurar:

1. Clonar el repositorio desde GitHub.

2. Crear entorno virtual y activar.

3. Instalar dependencias desde requirements.txt.

4. Configurar archivo .env con variables necesarias.

5. Ejecutar migraciones con Flask-Migrate.

6. Ejecutar run.py para iniciar el sistema localmente.

7. DESPLIEGUE EN RENDER

El sistema está preparado para desplegarse en Render. Los pasos incluyen:

• Conectar el repositorio de GitHub a Render.

• Definir las variables de entorno necesarias.

• Render se encarga del despliegue continuo.

8. CONCLUSIÓN

El Gestor de Tareas es un sistema robusto y modular que permite la gestión efectiva de tareas y usuarios. Gracias a su arquitectura limpia, uso de Blueprints y ORM, es fácil de mantener, escalar y extender.

MEDIDAS DE SEGURIDAD IMPLEMENTADAS

1. Autenticación y Gestión de Sesiones

• Uso de Flask-Login para gestionar sesiones de usuario.

• Decorador @login_required para proteger rutas críticas.

• Redirección automática al login si no hay sesión activa.

• Implementación de cierre automático de sesión tras 2 horas de inactividad con app.before_request.

2. Autorización por Roles

• Decorador @require_role que verifica si el usuario tiene permiso para acceder.

• Uso de current_user.rol para restringir acceso a funciones específicas.

• Plantillas que ajustan su contenido con current_role y funciones de permisos.

3. Protección contra CSRF

• Uso de Flask-WTF y WTForms con tokens CSRF automáticos.

• Validación en form.validate_on_submit().

• Configuración de app.config['WTF_CSRF_ENABLED'] = True.

4. Gestión de Contraseñas y Tokens

• Contraseñas almacenadas de forma segura con algoritmos hash (bcrypt).

• Implementación de tokens únicos para recuperación de contraseña con tiempo de caducidad.

5. Envío Seguro de Correos Electrónicos

• Configuración de Flask-Mail con TLS para cifrado en el envío.

• Variables sensibles (MAIL_PASSWORD, etc.) gestionadas mediante archivo .env.

6. Validación de Datos de Entrada

• Validadores en WTForms: DataRequired, Email, Length, NumberRange, etc.

• Validaciones personalizadas en modelos usando @validates.

• Validación de tipo y nombre de archivos con werkzeug.utils.secure_filename.

7. Seguridad en Archivos Subidos

• Archivos subidos se almacenan en carpetas individuales por solicitante.

• Acceso controlado por @login_required y @require_role.

• Validación de extensiones y tipo MIME.

8. Protección de la Base de Datos

• Uso de SQLAlchemy para evitar inyecciones SQL.

• Gestión de migraciones con Alembic.

• Claves primarias y foráneas definidas para asegurar integridad referencial.

9. Auditoría y Registro de Actividades

• Decoradores y eventos para registrar operaciones CRUD en tablas de auditoría.

• Registro de datos antes y después de cada cambio.

• Logging de errores configurado con logging.basicConfig.

REQUERIMIENTOS DEL SISTEMA

1. Introducción

Este documento especifica los requerimientos funcionales y técnicos del sistema 'Gestor de Tareas'. El propósito es establecer una base clara para el desarrollo, implementación y mantenimiento del sistema.

2. Objetivos del Proyecto

Objetivo General:

• Diseñar e implementar un sistema web para la gestión eficiente de tareas y usuarios.

Objetivos Específicos:

• Facilitar el registro, edición y eliminación de tareas.

• Implementar control de acceso por roles.

• Ofrecer reportería básica sobre tareas y usuarios.

3. Alcance del Sistema

Incluye:

• Gestión de usuarios (creación, edición, autenticación).

• Gestión de tareas (CRUD).

• Sistema de roles con permisos diferenciados.

• API REST para operaciones administrativas.

No incluye:

• Integración con sistemas externos.

• Módulos de notificaciones push o móviles.

4. Requerimientos Funcionales

• RF1: El sistema debe permitir registrar nuevos usuarios con rol definido.

• RF2: El sistema debe permitir iniciar y cerrar sesión con autenticación segura.

• RF3: El sistema debe permitir registrar, consultar, editar y eliminar tareas.

• RF4: El sistema debe mostrar las tareas asignadas por usuario.

• RF5: El sistema debe restringir el acceso a funcionalidades según rol.

5. Requerimientos No Funcionales

• RNF1: El sistema debe estar desarrollado en Python 3.10+ con Flask.

• RNF2: La base de datos debe ser Microsoft SQL Server o SQLite en desarrollo.

• RNF3: El tiempo de respuesta del sistema no debe superar los 2 segundos por solicitud.

• RNF4: Debe seguir estándares OWASP para asegurar protección contra ataques comunes.

• RNF5: El sistema debe ser desplegable automáticamente vía Render.

6. Requerimientos del Entorno

Servidor de Aplicaciones:

• Python 3.10+, Flask 2.3+

• Sistema operativo: Linux o Windows Server

Base de Datos:

• SQL Server 2019+ o SQLite (modo local)

Cliente:

• Navegador compatible: Chrome, Firefox, Edge (versión actualizada)

• Resolución mínima: 1366x768

7. Seguridad y Roles

Roles del sistema:

• Administrador: Acceso total a usuarios y tareas.

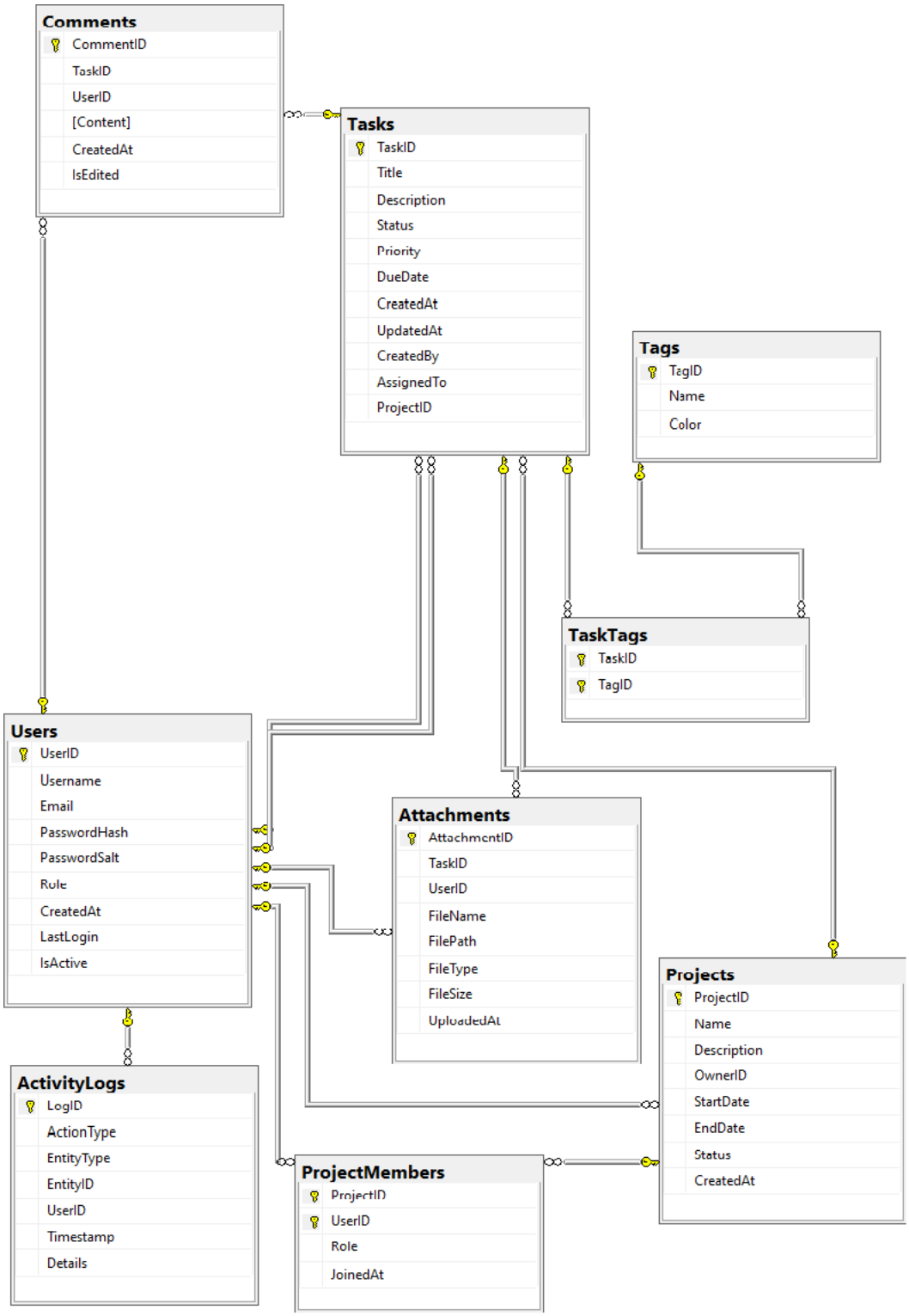
• Usuario estándar: Puede gestionar solo sus propias tareas.

Políticas de seguridad:

• Contraseñas hash con bcrypt.

• Expiración de sesión por inactividad.

• Restricción de acceso a rutas protegidas.



DOCUMENTACIÓN DE ENDPOINTS

1. Introducción

Esta documentación describe los endpoints implementados en el sistema 'Gestor de Tareas', desarrollado en Flask. Cada ruta está diseñada para permitir la interacción con los recursos del sistema a través de una API RESTful.

2. Autenticación

• `POST /login`: Permite iniciar sesión con credenciales de usuario.

• `POST /register`: Permite registrar un nuevo usuario en el sistema.

3. Gestión de Usuarios

• `GET /usuarios`: Devuelve la lista de todos los usuarios registrados.

• `GET /usuarios/<id>`: Devuelve los datos de un usuario específico por su ID.

• `POST /usuarios`: Crea un nuevo usuario (requiere autenticación y rol administrador).

• `PUT /usuarios/<id>`: Actualiza los datos de un usuario existente.

• `DELETE /usuarios/<id>`: Elimina un usuario por su ID.

4. Gestión de Tareas

• `GET /tareas`: Devuelve una lista de todas las tareas del sistema.

• `GET /tareas/<id>`: Devuelve los detalles de una tarea específica.

• `POST /tareas`: Crea una nueva tarea (requiere autenticación).

• `PUT /tareas/<id>`: Actualiza la información de una tarea existente.

• `DELETE /tareas/<id>`: Elimina una tarea por su ID.

5. Seguridad y Control de Acceso

• Todas las rutas sensibles utilizan el decorador `@login_required`.

• El acceso a funciones administrativas se controla mediante `@require_role('admin')`.

• Los tokens de sesión y cookies son gestionados por Flask-Login.

6. Formato de Respuesta

Todas las respuestas de la API siguen el siguiente formato estándar:

```
{
   "status": "success",
   "data": {...},
   "message": "Operación realizada con éxito"
}
```

7. Códigos de Estado

• 200 OK – Solicitud procesada correctamente.

• 201 Created – Recurso creado correctamente.

• 400 Bad Request – Error en los datos enviados.

8. Conclusión

La API del Gestor de Tareas está diseñada para ser segura, clara y fácil de utilizar. Cada endpoint implementa validaciones y controles de acceso necesarios para garantizar la integridad de los datos.