Documentación Técnica – Aplicación de Gestión de Tareas

Versión

v1.0 – Especificación funcional, técnica y semántica

---

1. Propósito del sistema

Esta aplicación es una **API REST** para la gestión de tareas personales. Permite que los usuarios se registren, se autentiquen mediante JWT y gestionen sus propias tareas (crear, listar, actualizar y eliminar).

El sistema está diseñado bajo principios de: - separación de responsabilidades - seguridad stateless - validación estricta de datos - manejo centralizado de errores

Esta documentación está escrita con el objetivo de que: 1. Un desarrollador humano pueda entender completamente el sistema. 2. Una inteligencia artificial pueda **reconstruir el backend completo** únicamente a partir de esta especificación.

---

2. Arquitectura general

- Backend: Java + Spring Boot
- Persistencia: JPA / Hibernate
- Seguridad: Spring Security + JWT
- Estilo: API REST stateless
- Autenticación: basada en email y contraseña

No existe sesión en servidor. Toda autenticación se basa en tokens JWT enviados en el header `Authorization`.

---

3. Modelo de dominio (conceptual)

3.1 Usuario

Concepto: Representa a una persona que utiliza la aplicación.

Características: - Se identifica de forma única por su email - Posee credenciales de acceso (email + contraseña encriptada) - Tiene un rol de seguridad - Puede tener múltiples tareas asociadas

Reglas de negocio: - El email es único y obligatorio - La contraseña se almacena siempre en formato encriptado - Un usuario solo puede acceder a sus propias tareas

---

3.2 Tarea

Concepto: Representa una actividad o pendiente creado por un usuario.

Características: - Pertenece a un único usuario - Tiene nombre y descripción - Posee fechas de creación y finalización - Tiene un estado y un nivel de importancia

Reglas de negocio: - Una tarea no puede existir sin un usuario - Solo el usuario propietario puede modificarla o eliminarla - El estado y la importancia son valores controlados por enumeraciones

---

4. Relaciones entre entidades

```
Usuario 1 ──────────── * Tarea
```

- Un Usuario puede tener muchas Tareas
- Una Tarea pertenece a un solo Usuario

La relación es obligatoria desde Tarea hacia Usuario.

---

5. Enumeraciones

Estado

- PENDIENTE
- EN_PROGRESO
- COMPLETADA

Importancia

- BAJA
- MEDIA
- ALTA

Role

- USER

El sistema está preparado para soportar nuevos roles en el futuro.

---

6. Seguridad y Autenticación

6.1 Registro

- Endpoint público
- Valida que el email no exista
- Encripta la contraseña con BCrypt
- Asigna rol USER por defecto

6.2 Login

- Endpoint público
- Valida credenciales
- Genera JWT firmado

6.3 JWT

Contenido del token: - issuer: "tareas" - subject: email del usuario - claim: id del usuario - expiración: 2 horas

El token se envía en el header:

```
Authorization: Bearer <token>
```

---

7. Autorización

- Todo endpoint excepto `/auth/login` y `/auth/registro` requiere autenticación
- El token se valida en un filtro previo
- El usuario autenticado se inyecta en el contexto de seguridad

---

8. Endpoints (visión lógica)

Autenticación

- POST /auth/registro
- POST /auth/login

Usuarios

- GET /usuarios (lista de usuarios)
- GET /usuarios/{id} (obtener usuario por id)
- POST /usuarios (crear usuario – uso interno o futuro admin)
- PUT /usuarios (actualizar datos del usuario autenticado)

- DELETE /usuarios/{id} (eliminar usuario)

El CRUD de Usuario es completo. En el estado actual del sistema, los endpoints sensibles se consideran protegidos y pensados para un rol administrativo futuro. El usuario autenticado solo puede modificar sus propios datos.

Tareas

- POST /tareas
- GET /tareas
- PUT /tareas
- DELETE /tareas/{id}

(Los endpoints de tareas siempre operan sobre el usuario autenticado)

---

9. Manejo de errores

8.1 Dependencias y stack técnico

El proyecto utiliza Maven como gestor de dependencias. Las dependencias están seleccionadas para permitir que una IA pueda regenerar el proyecto completo sin ambigüedades.

Dependencias principales: - spring-boot-starter-web: expone la API REST - spring-boot-starter-data-jpa: persistencia con Hibernate - spring-boot-starter-security: autenticación y autorización - spring-boot-starter-validation: validaciones con annotations - jjwt (JWT): generación y validación de tokens - lombok: reducción de boilerplate - h2 / mysql / postgresql: base de datos (configurable) - spring-boot-starter-test: soporte de testing

Las dependencias de seguridad y JWT son obligatorias para reproducir el comportamiento descrito en esta documentación.

El sistema utiliza un manejador global de excepciones.

Tipos de errores:

Código	Motivo
400	Error de validación / JSON inválido
401	Credenciales inválidas
403	Token inválido
404	Entidad no encontrada
500	Error interno

Formato de respuesta de error: - status - error - detalles - timestamp

---

10. Validaciones

- Campos obligatorios no pueden ser nulos
- Email debe tener formato válido
- Contraseña entre 6 y 10 caracteres
- IDs deben existir

Las validaciones se realizan antes de ejecutar lógica de negocio.

---

11. Flujo de creación de tarea (ejemplo)

1. Usuario se autentica y obtiene JWT
2. Cliente envía POST /tareas con token
3. Sistema valida token
4. Se valida el cuerpo de la solicitud
5. Se asocia la tarea al usuario autenticado
6. Se persiste la tarea
7. Se devuelve respuesta

---

12. Prompt base para IA (generación de código)

> Generá una API REST en Java con Spring Boot para gestión de tareas. El sistema debe manejar usuarios con autenticación JWT, roles, seguridad stateless y validación. Existen las entidades Usuario y Tarea con relación uno a muchos. El usuario se identifica por email. Las tareas pertenecen a un único usuario. Implementar manejo global de errores, DTOs, validaciones, filtros de seguridad y token JWT. El sistema debe cumplir estrictamente con esta documentación.

12.1 Testing y calidad

El sistema cuenta con tests automatizados que validan: - Correcto funcionamiento de los endpoints - Validaciones de entrada (400) - Seguridad (401 / 403) - Manejo de errores global - Persistencia correcta de entidades

Los tests están pensados para ser reproducidos por una IA como parte del proceso de generación de código, utilizando: - @SpringBootTest - @AutoConfigureMockMvc - Tests de controladores - Tests de seguridad

No existen tests innecesarios: cada test valida una regla de negocio o una restricción de seguridad.

13. Estado del proyecto

✔️ Backend completo ✔️ Seguridad implementada ✔️ Manejo de errores ✔️ Preparado para frontend o microservicios

---

Fin de la documentación v1.0