Análisis y Diseño del Software — Implementación de una API para J&B (Bogotá)

Resumen rápido: - Proyecto: Implementación de una API para la gestión de usuarios y tareas de la Empresa J&B en Bogotá. - Propósito de este documento: cubrir Pautas para el desarrollo (análisis, diseño, implementación) con diagramas en Mermaid.

Pautas para el desarrollo del proyecto — Análisis del Software

1. Nombre Proyecto

   Implementación de una API para la gestión de datos operativos de la Empresa J&B (Bogotá)

2. Descripción del Problema

   J&B necesita una API centralizada que permita gestionar usuarios y tareas internas, con acceso controlado y endpoints claros para crear, consultar, actualizar y eliminar tareas, así como autenticar usuarios. Actualmente la gestión es manual o distribuida, lo que dificulta la trazabilidad y la automatización de procesos operativos.

   Solución: desarrollar una API REST que exponga endpoints para gestión de usuarios y tareas, con validación, autenticación y documentación mínima para consumidores internos.

3. Justificación

   - Facilitar el acceso y la gestión de tareas por equipos internos.
   - Reducir trabajo manual y mejorar trazabilidad de actividades.
   - Mejorar seguridad del acceso mediante autenticación y roles.

4. Objetivos

   a. Objetivo General

   Desarrollar e implementar una API segura, escalable y mantenible que permita la gestión de usuarios y tareas para J&B.

   b. Objetivos Específicos

   - Diseñar el esquema de datos y modelo relacional necesario para almacenar usuarios y tareas.
   - Implementar endpoints REST para CRUD sobre tareas y gestión de usuarios (registro y autenticación).
   - Añadir autenticación y autorización para asegurar acceso restringido.

- Crear documentación técnica y pruebas (unitarias y funcionales) que garanticen calidad.
- Desplegar una instancia de prueba y definir pasos para producción.

5. Alcance del Proyecto

   - Incluye: diseño del modelo de datos, API REST con autenticación, endpoints CRUD para tareas y gestión de usuarios, pruebas unitarias y funcionales, documentación y scripts de despliegue mínimo.
   - Excluye (fuera de alcance inicial): integración con sistemas externos no implementados actualmente, dashboards o análisis avanzados.

6. Diagrama de Procesos (estado actual)

```
flowchart LR
  A[Origenes: procesos internos] --> B[Registro manual / Herramientas ad-hoc]
  B --> C[Operaciones gestionan tareas]
  C --> D[Seguimiento y reportes internos]
  D --> E[Decisiones operativas]
  style A fill:#f9f,stroke:#333,stroke-width:1px
  style B fill:#ffebcd
  style C fill:#ffe4e1
  style D fill:#e6ffe6
  style E fill:#d3f8d3
```

   Nota: el diagrama refleja el proceso actual sin una API centralizada para gestión de tareas.

7. Requerimientos

   - Requerimientos Funcionales (RF)
     - RF1: Permitir autenticación (login) de usuarios con token JWT.
     - RF2: Registrar usuarios (administrador) y gestionar roles.
     - RF3: CRUD de entidades principales (tareas y usuarios básicos).
     - RF4: Endpoint de salud (health-check).
     - RF5: Registro (logging) de eventos y auditoría básica.
   - Requerimientos No Funcionales (RNF)
     - RNF1: API documentada (OpenAPI/Swagger o README detallado).
     - RNF2: Respuesta en tiempos aceptables (p. ej. < 500ms para consultas simples).
     - RNF3: Seguridad: transporte por TLS (si aplica), almacenamiento de contraseñas hasheadas.
     - RNF4: Pruebas: cobertura mínima para funcionalidades críticas.
     - RNF5: Despliegue reproducible (Dockerfile/Procfile presentes).

8. Historias de Usuario

   - HU1: Como usuario, quiero autenticarme para consumir los endpoints protegidos.

  - Criterios de aceptación: login con credenciales válidas devuelve token JWT con expiración; token inválido deniega acceso.
- HU2: Como administrador, quiero registrar y gestionar usuarios para controlar acceso.
  - Criterios: crear usuario con rol, listar usuarios, eliminar/actualizar.
- HU3: Como usuario, quiero crear y gestionar tareas para llevar seguimiento de actividades.
  - Criterios: endpoints CRUD para tareas, permisos para acceder solo a tareas propias.
- HU4: Como desarrollador, quiero un endpoint health-check para monitorización.
  - Criterios: /v1/ping devuelve 200 y estado del servicio.

9. Diagrama de Entidad–Relación (propuesta básica)

```
erDiagram
  USERS {
    INTEGER id PK
    VARCHAR username
    VARCHAR email
    VARCHAR password_hash
    DATETIME created_at
  }

  TASKS {
    INTEGER id PK
    VARCHAR title
    TEXT description
    INTEGER user_id FK
    BOOLEAN completed
    DATETIME created_at
    DATETIME updated_at
  }

  USERS ||--o{ TASKS : owns
```

Diseño del Software

1. Diagrama de Clases (esquema simplificado)

```
classDiagram
  class User {
    +int id
    +string username
    +string email
```

```
  +string password_hash
  +datetime created_at
  +set_password(password)
  +verify_password(password)
}
class Task {
  +int id
  +string title
  +string description
  +bool completed
  +datetime created_at
  +datetime updated_at
  +int user_id
  +to_dict()
}

User "1" --> "*" Task : owns
```

2. Diagrama de Casos de Uso (simplificado)

```
flowchart LR
  actorUser[[Usuario]]
  actorAdmin[[Administrador]]

  actorUser --> UC1[Consultar / Gestionar tareas]
  actorUser --> UC2[Autenticarse]
  actorAdmin --> UC3[Registrar usuarios]

  classDef actor fill:#f8f9fa,stroke:#333
  class actorUser,actorAdmin actor
```

3. Diagrama de Secuencia (ejemplo: petición para obtener una tarea)

```
sequenceDiagram
  participant Client
  participant API
  participant Auth
  participant DB
  Client->>API: GET /v1/tareas (token)
  API->>Auth: validar token
  Auth-->>API: token válido
  API->>DB: consulta tareas del usuario
  DB-->>API: resultados
  API-->>Client: 200 OK (JSON)
```

4. Diagrama de Actividades (flujo de creación de tarea)

```
flowchart TD
  Start([Inicio]) --> Input[Recibir payload de tarea]
```

```
Input --> Auth[Verificar token]
Auth --> Validate[Validar & limpiar datos]
Validate --> Persist[Persistir en base de datos]
Persist --> Notify[Responder con recurso creado]
Notify --> End([Fin])
```

Implementación del Software (especificación)

1. Diagrama de flujo (implementación - ejemplo de endpoint POST /v1/tareas)

```
flowchart TD
  Req[Cliente POST /v1/tareas] --> Auth[Verificar token]
  Auth -->|válido| Validate[Validar payload]
  Validate --> Save[Crear registro en DB]
  Save -->|éxito| Resp[201 con recurso]
  Auth -->|inválido| Err[401 Unauthorized]
  Validate -->|error| Err400[400 Bad Request]
```

2. Implementación del Código Fuente

   - Estructura observada en el repositorio:
     - `src/app.py` — inicialización de la app Flask (flask-openapi3) y registro de blueprints.
     - `src/models/` — modelos: `User`, `Task`.
     - `src/blueprints/v1/` — rutas: `auth.py`, `healt_check.py`, `tasks.py`.
     - `src/schemas/` — validaciones y serialización (pydantic) para `auth` y `tasks`.
     - `src/queries/` — consultas: autenticación y obtención/listado de tareas.
     - `src/commands/` — comandos para crear/actualizar/eliminar tareas y registrar usuarios.
   - Elementos clave implementados (scaffolding actual):
     - Extensiones: `db`, `jwt`, `bcrypt` en `src/config/extensions.py`.
     - Blueprints registrados: `/v1/ping`, `/v1/auth/*`, `/v1/tareas` (endpoints CRUD).
     - Tests: `tests/test_auth.py`, `tests/test_tasks.py`.
   - Buenas prácticas usadas en el proyecto:
     - Uso de Pydantic para esquemas de request/response.
     - Separación de queries y comandos (CQRS-like simple).
     - Manejo centralizado de errores (carpeta `src/errors`).

3. Pruebas unitarias

   - En el proyecto ya existen tests para autenticación y tareas. Recomendación: ejecutar pytest y revisar cobertura para asegurar que las

rutas críticas estén cubiertas.

4. Pruebas funcionales

   - Pruebas end-to-end pueden usar SQLite en `instance/app.db` o una base en memoria según `conftest.py`.
   - Colección Postman (pruebas funcionales)

     La colección `postman_collection.json` incluida en el repositorio contiene los flujos funcionales que cubren los endpoints implementados: health, auth (register/login) y CRUD de tareas. Esta colección actúa como la suite de pruebas funcionales y debe mencionarse explícitamente como tal en el plan de pruebas.

     Qué cubre la colección:

     - Health: `GET /v1/ping` — valora disponibilidad del servicio.
     - Auth: registro de usuario (`POST /v1/auth/register`) y obtención de token (`POST /v1/auth/login`).
     - Tasks: flujo completo Create -> List -> Get -> Update -> Delete sobre `/v1/tareas` usando el token obtenido.

     Requisitos previos antes de ejecutar la colección:

     1. Tener la API corriendo localmente (por defecto `http://localhost:5000`) o actualizar `base_url` en las variables de la colección.
     2. Disponer de una base de datos de test o limpiar `instance/app.db` para evitar conflictos con usuarios/tareas previos.

     Ejecutar la colección localmente (Postman GUI):

     - Importar `postman_collection.json` en Postman.
     - Revisar/ajustar la variable `base_url` (Collection/Environment) a la URL del servicio.
     - Ejecutar la colección con el "Collection Runner".

     Criterios de aceptación (pruebas funcionales):

     - Todas las requests en la colección devuelven los códigos HTTP esperados (200/201/401/404 según el caso).
     - Las aserciones definidas en los tests de Postman pasan (por ejemplo, que la respuesta contenga `data.id` en creación de tareas).
     - El flujo de creación -> uso del token -> CRUD de tareas se completa sin errores.

5. Documento final

   - Entregar este documento con:
     - Diagrama y modelos actualizados (sin referencias a ventas/analytics).
     - Instrucciones de despliegue y ejecución (Dockerfile y Procfile ya incluidos).

– Resultados de pruebas y cómo ejecutarlas (pytest).

Apéndice: Endpoints reales observados

- POST /v1/auth/register — registrar usuario (admin)
- POST /v1/auth/login — obtener token JWT
- GET /v1/ping — health-check
- GET /v1/tareas — listar tareas del usuario autenticado
- GET /v1/tareas/ — obtener detalle de una tarea
- POST /v1/tareas — crear tarea
- PUT /v1/tareas/ — actualizar tarea
- DELETE /v1/tareas/ — eliminar tarea

Historias de Usuario (Detalladas)

1. HU - Login

- Como: Usuario
- Quiero: Autenticación vía credenciales
- Para: Acceder a endpoints protegidos
- Criterios: JWT con expiry, manejo de credenciales inválidas

2. HU - Crear Tarea

- Como: Usuario autenticado
- Quiero: Poder crear una tarea con título y fecha de vencimiento (si aplica)
- Para: Registrar actividades y dar seguimiento
- Criterios: Campos obligatorios validados, 201 en creación

3. HU - Gestionar Tareas

- Como: Usuario autenticado
- Quiero: Listar, actualizar y eliminar mis tareas
- Para: Mantener control de mis actividades
- Criterios: Permisos respetados, respuestas 200/404 según corresponda

---

Document updated on: 2025-11-06