Estrategia de versionado de API y migraciones de BD

Jheferson Danni Checa Díaz

Agricapital (Prueba técnica)

2025

1. Estrategia de Versionado de API

El versionado de la API es esencial para permitir que los consumidores (como el front-end, otros servicios o aplicaciones de terceros) sigan utilizando versiones anteriores mientras se introducen nuevas funcionalidades o cambios que rompen la compatibilidad (breaking changes).

Estrategia Propuesta: Versionado por Prefijo en la URL (Path Versioning)

Esta estrategia es clara, explícita y ampliamente adoptada:

- **Implementación:**
    - Cada versión principal de la API se indicará directamente en la URL de los endpoints. Por ejemplo:
        - api.agricapital.com/v1
        - api.agricapital.com/v2
    - Para el backend (FastAPI), esto se lograría fácilmente con el APIRouter de FastAPI, especificando un prefijo para cada versión.
- **Ciclo de Vida de las Versiones:**
    - **v1 (Versión Actual):** Es la versión inicial y estable.
    - **v2 (Nueva Versión):** Se introduciría cuando haya **cambios incompatibles** (ej. eliminación de un campo obligatorio en un DTO de solicitud, cambio de un tipo de dato, cambio significativo en la lógica de un endpoint que altere su respuesta esperada).
        - Ambas versiones (v1 y v2) coexistirían durante un período de transición.
        - Se comunicaría claramente a los consumidores la disponibilidad de v2 y la fecha de deprecación de v1.
    - **Deprecación y Eliminación:** Las versiones antiguas (v1) serían marcadas como "deprecated" (obsoletas) tras un período de notificación adecuado (ej. 3-6 meses). Durante este tiempo, aún recibirían soporte para errores críticos, pero no para nuevas funcionalidades. Finalmente, la versión obsoleta sería eliminada.
    - **Cambios Compatibles (Non-breaking changes):**

        - Para cambios que no rompen la compatibilidad (ej. adición de nuevos campos opcionales, nuevos endpoints sin afectar los existentes), se aplicarían directamente a la versión actual sin necesidad de crear una nueva versión principal. Estos se considerarían actualizaciones **menores** o **de parche** y se comunicarían a través de notas de lanzamiento detalladas.

- **Ventajas de esta Estrategia:**
    - **Claridad:** Los consumidores saben exactamente qué versión están usando.
    - **Simplicidad:** Fácil de implementar y mantener en el código.
    - **Coexistencia:** Permite que diferentes clientes consuman diferentes versiones simultáneamente, facilitando las actualizaciones del front-end o de terceros.

2. Estrategia Propuesta: Migraciones Esquemáticas Controladas (Alembic)

Para el Backend, que utiliza **PostgreSQL** y **FastAPI**, el uso de una herramienta de migración de bases de datos como **Alembic** (comúnmente usada con SQLAlchemy, que es la base de los ORMs en Python con FastAPI) es el estándar.

- **Principio:** Cada cambio en el esquema de la base de datos (ej. añadir una tabla, modificar una columna, crear un índice) se define como una "migración" en un script dedicado. Estos scripts son versionados junto con el código de la aplicación.
- **Proceso:**
    1. **Desarrollo Local:** Cuando un desarrollador necesita modificar el esquema de la base de datos (ej. añadir una columna para un nuevo requisito de solicitud), lo hace a través de la herramienta de migración (Alembic).
    2. **Generación de Migración:** La herramienta genera automáticamente un script de migración (.py) que contiene las operaciones para "subir" (aplicar) y "bajar" (revertir) el cambio de esquema.
    3. **Revisión y Versionado:** El script de migración es revisado y versionado en Git junto con el código fuente del microservicio. Esto asegura que

cada cambio de esquema esté asociado a una versión específica del código.

4. **Aplicación en Entornos:**
    - **Entornos de Desarrollo/Staging:** Al desplegar una nueva versión, el pipeline de CI/CD (o un paso manual en desarrollo) ejecuta automáticamente los scripts de migración pendientes.
    - **Entorno de Producción:** Las migraciones se aplican de forma controlada y planificada antes o durante el despliegue de la nueva versión del microservicio. Se prefiere aplicar migraciones que no causen un bloqueo (non-blocking migrations) si es posible, para minimizar el tiempo de inactividad.

- **Consideraciones Clave:**
    - **Migraciones "hacia adelante" y "hacia atrás" (Up/Down):** Cada migración debe poder aplicarse y revertirse de forma segura.
    - **Atomicidad:** Las operaciones dentro de una migración deben ser atómicas para asegurar que el esquema esté siempre en un estado consistente.
    - **Auditoría:** La herramienta de migración mantiene un registro de qué migraciones se han aplicado a cada base de datos, facilitando el seguimiento y la depuración.
    - **Migraciones de Datos:** En caso de que se necesiten transformar o migrar datos existentes como parte de un cambio de esquema, esto se incluirá dentro del script de migración o en un script de una sola ejecución posterior, dependiendo de la complejidad y el volumen.