INTEGRACIÓN DE LA API

UNIVERSIDAD
Blas Pascal

Abril David, Checa Sol, Gómez Martina

FECHA: 05/09/20 Profesor: Ing. Gustavo Funes

Contenido

1. Introducción

Este informe detalla los requisitos, endpoints propuestos y los procesos clave para la sincronización y manejo de vuelos entre bases de datos locales y una base de datos global en la nube. El sistema permite la creación y gestión de vuelos en cada base de datos local, con la capacidad de sincronizarlos con la base de datos global, facilitando la consulta y reserva de vuelos entre múltiples agencias. Además, se incluye un mecanismo de verificación de disponibilidad entre las bases de datos antes de confirmar las reservas, asegurando que no haya conflictos o doble reserva.

2. Revisión de Requerimientos

Requerimiento 1: No centralizado y sincronización entre bases de datos locales y global

- Cada base de datos local debe poder crear, gestionar y subir vuelos a la base de datos global.
- La base de datos global debe permitir la consulta de vuelos locales y de otras agencias.
- Se deben crear endpoints para:
    - Crear vuelos localmente y enviarlos a la base de datos global.
    - Consultar vuelos disponibles, tanto en la base de datos local como en la global.
    - Actualizar vuelos en ambas bases de datos.

Requerimiento 2: Verificación de disponibilidad antes de la reserva

- Implementar un mecanismo que permita la verificación de disponibilidad tanto en la base de datos local como en la global antes de confirmar una reserva.
- La base de datos global y la local deben "preguntarse" mutuamente sobre la disponibilidad del vuelo, y solo después de una respuesta afirmativa se debe confirmar la reserva.

3. Endpoints Propuestos

Sistema Local

- **POST /vuelos/local**: Crear un nuevo vuelo en la base de datos local.
- **POST /vuelos/sync**: Sincronizar el vuelo local con la base de datos global.
- **GET /vuelos/local**: Consultar los vuelos disponibles en la base de datos local.
- **GET /vuelos/global**: Consultar los vuelos disponibles en la base de datos global.

Sistema en la Nube

- **POST /vuelos**: Agregar un vuelo desde una base de datos local a la base de datos en la nube.
- **GET /vuelos**: Consultar vuelos de la base de datos global.
- **POST /reservas/confirmar**: Confirmar una reserva en la base de datos global (luego de validar en la local).

Sincronización y Reserva con Validación

- **GET /vuelos/disponibilidad**: Verificar disponibilidad de vuelos entre la base de datos local y global.
    - Este endpoint debe consultar ambas bases de datos para confirmar que el vuelo esté disponible en ambas antes de proceder con la reserva.

4. Proceso Detallado

4.1 Crear Vuelos Locales y Sincronizarlos con la Nube

- **Crear Vuelo Local**: El vuelo se crea en la base de datos local mediante el endpoint POST /vuelos/local.
- **Sincronización**: Luego de crear el vuelo local, se llama al endpoint POST /vuelos/sync para subir el vuelo a la base de datos global. Si la sincronización falla, se puede reintentar.

4.2 Verificación de Disponibilidad en Ambas Bases de Datos

1. **Solicitar Disponibilidad**: Antes de confirmar una reserva, el sistema local debe llamar al endpoint GET /vuelos/disponibilidad en la base de datos global.
    - La nube verifica que el vuelo esté disponible y que no haya sido reservado en otro sistema.
    - La base de datos local también verifica si el vuelo sigue disponible.
2. **Confirmar Reserva:** Si ambos sistemas responden que el vuelo está disponible, se procede a confirmar la reserva llamando a POST /reservas/confirmar tanto en la base de datos local como en la global.

5. Manejo de Conflictos

- **Exclusión Mutua**: En caso de que un vuelo esté siendo reservado en paralelo desde otro sistema, la respuesta de disponibilidad será negativa. Así, el vuelo será marcado como no disponible antes de completar la reserva en cualquiera de las bases de datos.

6. Mecanismo para Evitar Exclusión Mutua

6.1 Bloqueo Optimista

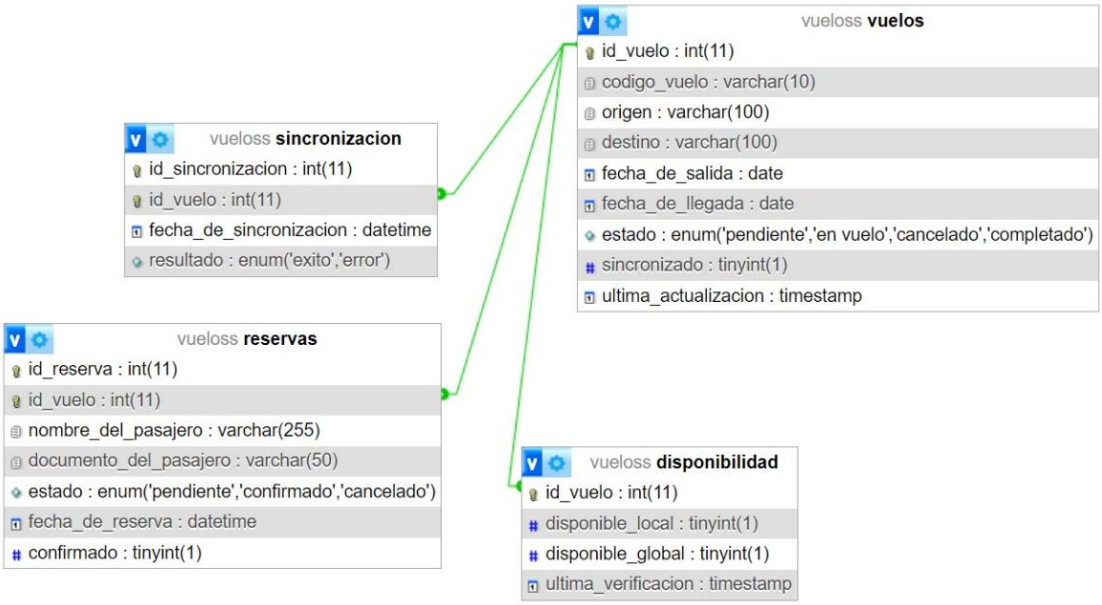
- Cada vuelo tendrá un estado de "disponibilidad" tanto en la base de datos local como en la global. Antes de confirmar una reserva, ambos sistemas deben cambiar el estado del vuelo a "en revisión" y verificar la disponibilidad en la otra base de datos.

6.2 Timeout

- Si uno de los sistemas no responde en un tiempo determinado (por ejemplo, 5 segundos), el vuelo se marca como no disponible y la reserva no se realiza.

7. Estructura y Entidades de las bases de datos

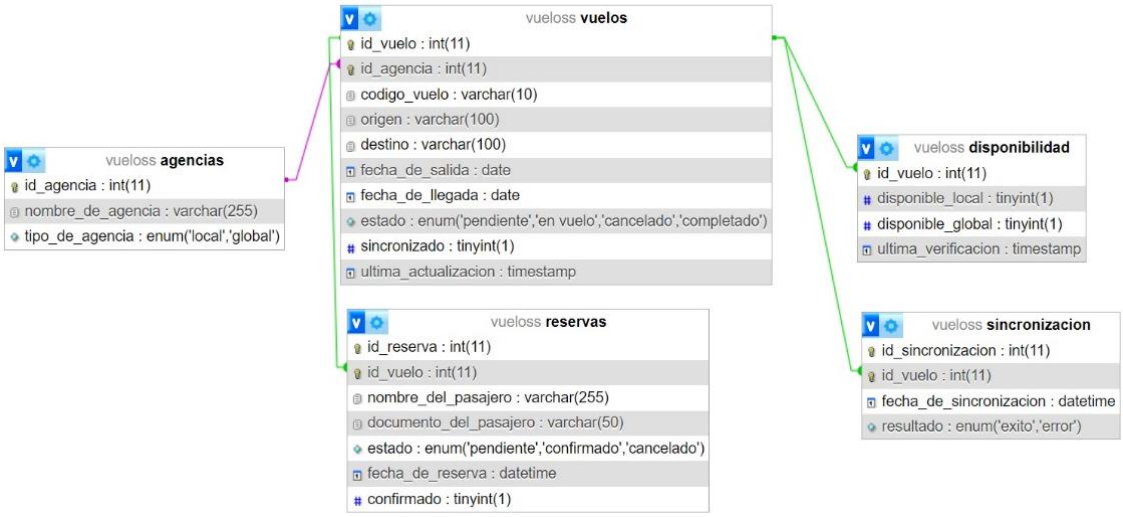
Base de Datos Local



Relaciones:

- Vuelos(Tabla de Origen) a Disponibilidad(Tabla de Destino) de 1 a 1.
  - Un vuelo tiene una única disponibilidad asociada
- Vuelos(Tabla de Origen) a Reservas(Tabla de Destino) de 1 a muchos.
  - Un vuelo tiene varias reservas, una reserva pertenece a un vuelo.
- Vuelos(Tabla de Origen) a HistorialSincronizacion(Tabla de Destino) de 1 a muchos.
  - Un vuelo tiene múltiples registros de sincronización.

Base de Datos Global



Relaciones:

- Se le agrega a la misma estructura de la base de datos local una tabla de Agencias(cada agencia tiene su id) y la relación:
  - Agencias (Tabla de Origen) a Vuelos (Tabla de Destino) de 1 a muchos.

- Una agencia gestiona múltiples vuelos.

8. Estructura de API

Manejo de Vuelos:

Endpoints:

- **GET /vuelos**: Obtener la lista de vuelos disponibles.
  - **Query Parameters**:
    - `origen`: Filtra los vuelos por ciudad de origen.
    - `destino`: Filtra los vuelos por ciudad de destino.
    - `fecha`: Filtra los vuelos por fecha de salida.
  - **Response (ejemplo)**:

```
[
    {
      "id": "123",
      "origen": "EZE",
      "destino": "JFK",
      "hora_salida": "2024-09-10T14:00:00Z",
      "hora_llegada": "2024-09-10T18:00:00Z",
      "disponibilidad_asientos": 5,
      "precio": 500.00
    }
  ]
```

- **POST /vuelos/sincronizar**: Sincronizar los vuelos creados localmente con la base de datos en la nube.
  - **Request Body**:

```
    {
        "id": "123",
        "origen": "EZE",
        "destino": "JFK",
        "hora_salida": "2024-09-10T14:00:00Z",
        "hora_llegada": "2024-09-10T18:00:00Z",
        "disponibilidad_asientos": 5,
        "precio": 500.00
      }
```

- **Response**:

```
{
    "status": "success",
    "message": "Sincronización completada"
}
```

- **POST /vuelos/verificar-disponibilidad**: Verificar si el vuelo está disponible en ambas bases de datos (local y en la nube) antes de confirmar la reserva.
  - **Request Body**:

```
{
```

```
    "id_vuelo": "123",
    "origen": "EZE",
    "destino": "JFK"
}
```

- Response:

```
{
    "status": "available",
    "mensaje": "El vuelo está disponible en ambas bases de datos"
}
```

- PUT /vuelos/{id_vuelo}/actualizar-disponibilidad: Actualizar la disponibilidad de asientos de un vuelo en ambas bases de datos.
  - Request Body:

```
{
    "disponibilidad_asientos": 3
}
```

- Response:

```
{
"status": "success",
"message": "Disponibilidad actualizada en ambas bases de datos"
}
```

Manejo de Reservas:

Endpoints:

- POST /reservas: Crear una nueva reserva de vuelo después de verificar la disponibilidad en ambas bases de datos.
  - Request Body:

```
{
    "id_vuelo": "123",
    "nombre_pasajero": "David Perez",
    "detalles_pago": {
      "tarjeta": "4111-1111-1111-1111",
      "expiracion": "12/24",
      "cvv": "123"
    }
}
```

- Response:

```
{
    "id_reserva": "456",
    "estado": "confirmada",
    "mensaje": "Reserva realizada con éxito"
```

```
}
```

- GET /reservas/{id_reserva}: Obtener detalles de una reserva.
    - Response:

```
{
    "id_reserva": "456",
    "id_vuelo": "123",
    "nombre_pasajero": "David Perez",
    "estado": "confirmada",
    "detalles_pago": {
      "metodo": "tarjeta",
      "estado": "aprobado"
    }
}
```

1. PUT /reservas/{id_reserva}: Actualizar el estado de una reserva (por ejemplo, cancelación).
    - Request Body:

```
{
    "estado": "cancelada"
}
```

    - Response:

```
{
    "id_reserva": "456",
    "estado": "cancelada",
    "mensaje": "Reserva cancelada con éxito"
}
```

Manejo de Destinos Preferidos:

Endpoints:

1. **GET /destinos-preferidos**: Obtener la lista de destinos preferidos del usuario.
    - Response:

```
[
    {
      "id": "789",
      "nombre_destino": "New York",
      "preferencia": "alta"
    }
]
```

2. **POST /destinos-preferidos**: Agregar un destino preferido.
    - Request Body:

```
{
```

```
    "nombre_destino": "New York",
    "preferencia": "alta"
}
```

- Response:

```
{
    "status": "success",
    "message": "Destino agregado"
}
```

3. **DELETE /destinos-preferidos/{id}**: Eliminar un destino preferido.
   - Response:

```
{
    "status": "success",
    "message": "Destino eliminado"
}
```

9. Conclusión: Integración del Sistema de Manejo de Vuelos y Reservas

Este sistema distribuye la gestión de vuelos, reservas y destinos preferidos entre una base de datos local y una en la nube. La local optimiza la velocidad de respuesta, mientras que la base centralizada asegura la disponibilidad y coherencia de la información para todos los usuarios.

- Manejo de Vuelos: Los vuelos se crean en la base de datos local y se sincronizan en la nube para ser accesibles desde cualquier agencia.
- Manejo de Reservas: Las reservas se hacen luego de validar la disponibilidad en ambas bases de datos para evitar conflictos de asientos.
- Manejo de Destinos Preferidos: Los usuarios pueden gestionar sus destinos preferidos desde cualquier dispositivo, con sincronización continua entre el sistema local y la nube.

La API facilita la integración y sincronización entre los dos sistemas, proporcionando una experiencia sin fricciones tanto para los usuarios locales como para los globales.