Documentación del Proyecto: Reservas Viajes API

1. Nombre del Proyecto

Reservas Viajes API

Reservas

2. Idea del Proyecto

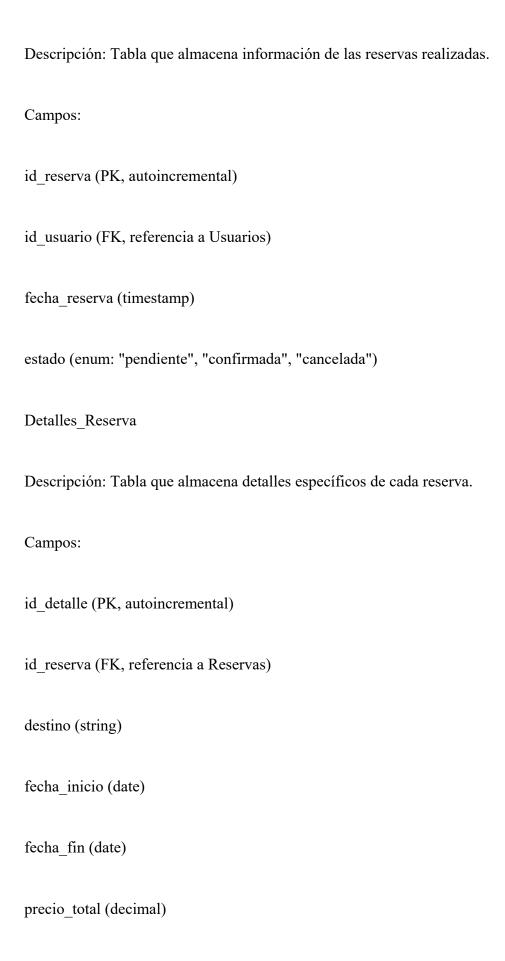
El proyecto consiste en desarrollar una API REST para gestionar reservas de viajes, incluyendo la administración de usuarios, la creación de reservas y la gestión de los detalles de cada reserva. La API debe ser segura, eficiente y capaz de manejar las operaciones requeridas por una plataforma de reservas de viajes.

3. Justificación del Proyecto

La digitalización de las reservas de viaje permite a las agencias y clientes optimizar procesos, reducir errores y mejorar la experiencia del usuario. Esta API servirá como base para una plataforma escalable y segura que centraliza la gestión de reservas y datos relacionados.

4. Descripción Detallada de las Tablas

Usuarios
Descripción: Tabla que almacena información de los usuarios registrados.
Campos:
id_usuario (PK, autoincremental)
nombre (string)
correo (string, único)
contraseña (string, encriptada)
fecha_creacion (timestamp)



5. Endpoints de la API

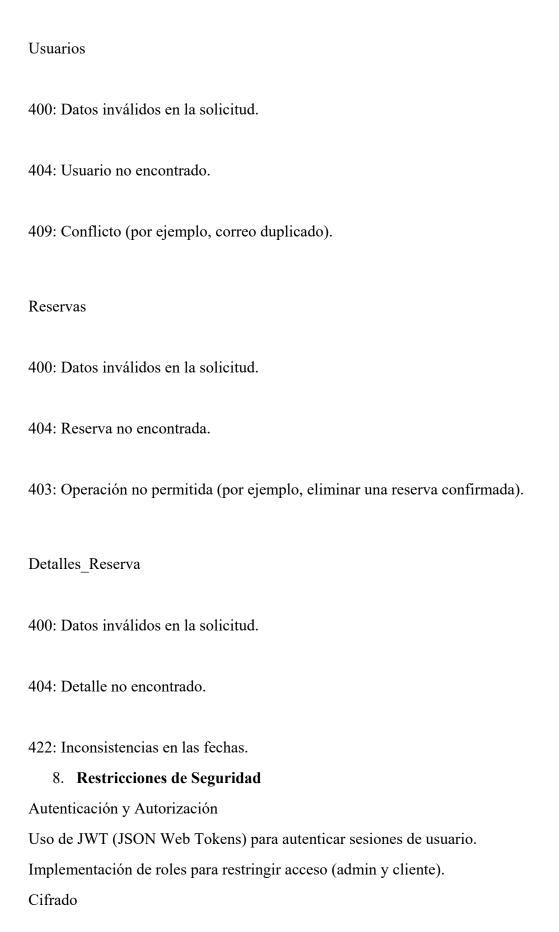
a. Endpoints a Desarrollar para Cada Tabla
Usuarios
POST /usuarios
GET /usuarios/{id}
PUT /usuarios/{id}
DELETE /usuarios/{id}
Reservas
POST /reservas
GET /reservas/{id}
GET /reservas
PUT /reservas/{id}
DELETE /reservas/{id}
Detalles_Reserva
POST /detalles

GET /detalles/{id}

PUT /detalles/{id}
DELETE /detalles/{id} b. Descripción de los Endpoints
Usuarios
POST /usuarios: Crea un nuevo usuario. Recibe datos como nombre, correo y contraseña.
GET /usuarios/{id}: Obtiene información de un usuario específico por su ID.
PUT /usuarios/{id}: Actualiza la información de un usuario.
DELETE /usuarios/{id}: Elimina un usuario del sistema.
Reservas
POST /reservas: Crea una nueva reserva asociada a un usuario.
GET /reservas/{id}: Obtiene detalles de una reserva específica.
GET /reservas: Lista todas las reservas existentes.
PUT /reservas/{id}: Actualiza el estado de una reserva (por ejemplo, confirmar o cancelar).
DELETE /reservas/{id}: Cancela una reserva pendiente.

Detalles_Reserva
POST /detalles: Agrega detalles a una reserva existente.
GET /detalles/{id}: Obtiene los detalles específicos de una reserva.
PUT /detalles/{id}: Modifica los detalles de una reserva.
DELETE /detalles/{id}: Elimina detalles específicos de una reserva. 6. Lógica de Negocio
Usuarios
Validación de datos al crear o actualizar usuarios.
Encriptación de contraseñas para mayor seguridad.
Reservas
Una reserva solo puede tener detalles si está en estado "confirmada".
No se permite eliminar reservas en estado "confirmada".
Detalles_Reserva
Validar que las fechas de inicio y fin sean coherentes.
Calcular el precio total basado en la duración y tarifas.

7. Excepciones y Códigos de Estado



Cifrado de contraseñas utilizando berypt.

Uso de HTTPS para la comunicación entre cliente y servidor.

Validación de Solicitudes

Sanitización de entradas para prevenir inyecciones SQL y ataques XSS.

Límites de tasa de solicitudes para evitar ataques de fuerza bruta.

9. Pruebas de la API:

a. Herramientas de Pruebas:

 Se utilizaron herramientas como Postman para realizar pruebas de los endpoints, verificando que las respuestas son correctas según los códigos de estado HTTP y los datos devueltos.

b. Pruebas Realizadas:

Pruebas de Funcionalidad

1. Crear Usuario

```
• Ruta: POST /usuarios
```

```
• Body:
```

```
Conjar códi
```

json

```
Copiar código
{
    "nombre": "Juan Pérez",
    "correo": "juan.perez@example.com",
    "contraseña": "123456"
```

2. Login Usuario

• Ruta: POST /usuarios/login

```
• Body:
```

json

```
Copiar código
```

```
{
"correo": "juan.perez@example.com",
```

"contraseña": "123456"

```
}
     Respuesta esperada:
json
Copiar código
 "token": "GENERATED_JWT_TOKEN"
3. Obtener Usuario
      Ruta: GET /usuarios/{id}
      Header:
            Authorization: Bearer {GENERATED_JWT_TOKEN}
4. Crear Reserva
      Ruta: POST /reservas
      Body:
json
Copiar código
 "usuario": {
  "id": 1
 },
 "estado": "PENDIENTE"
   • Header:
         o Authorization: Bearer {GENERATED_JWT_TOKEN}
5. Crear Detalle Reserva
   • Ruta: POST /detalles
   • Body:
json
Copiar código
```

```
"destino": "Cancún",

"fechaInicio": "2024-12-15",

"fechaFin": "2024-12-20",

"reserva": {

"id": 1

}
```

- Header:
 - o Authorization: Bearer {GENERATED JWT TOKEN}

6. Operaciones ADMIN

- Para actualizar y eliminar reservas o detalles:
 - o Asegúrate de que el usuario tenga el rol ROLE_ADMIN.
 - o Usa el JWT del usuario con permisos de administrador.

10. Conclusión del Proyecto:

a. Tecnologías Utilizadas:

- Framework: Spring Boot para desarrollar la API REST de forma eficiente.
- Spring Security para manejar la autenticación y autorización.
- JWT para la autenticación basada en token.
- **RSA (Asymmetric Encryption)** para la seguridad de la aplicación utilizando claves públicas y privadas.
- JPA (Java Persistence API) para manejar la base de datos relacional y persistir los datos de las entidades.
- Postman para probar los diferentes endpoints de la API.

b. ¿Qué es una API REST?

- **API REST**: Es una interfaz de comunicación basada en el protocolo HTTP que permite a los sistemas intercambiar información de manera sencilla y eficiente utilizando métodos como GET, POST, PUT, DELETE.
- Principios de una API REST:
 - 1. **Stateless**: El servidor no guarda información de estado entre las solicitudes.
 - 2. Uniform Interface: La API tiene una interfaz uniforme.

- 3. Cacheable: Las respuestas pueden ser cacheadas para mejorar el rendimiento.
- 4. **Layered System**: El cliente no sabe si está interactuando con el servidor final o un servidor intermedio.

c. Ventajas de la Separación de Responsabilidades Cliente-Servidor:

- Escalabilidad: Los sistemas pueden evolucionar independientemente.
- Mantenimiento más sencillo: El cliente y el servidor pueden modificarse sin afectar el otro.
- **Seguridad**: El servidor controla los datos y el acceso, mientras que el cliente solo muestra la interfaz.