2022/src/services/main/app.py

```python
import os

from contactos import contactos_bp
from dotenv import load_dotenv
from flask import Flask, request
from mensajes import mensajes_bp
from usuarios import usuarios_bp

app = Flask(__name__)
# Define blueprints:
app.register_blueprint(usuarios_bp, url_prefix='/usuarios')
app.register_blueprint(mensajes_bp, url_prefix='/mensajes')
app.register_blueprint(contactos_bp, url_prefix='/usuarios/<telefono>/contactos')

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    load_dotenv()
    app.run(host="0.0.0.0", port=os.getenv("SERVICE_PORT_MAIN"))

# Rutas que se deben implementar:

```

2022/src/services/main/usuarios.py

```python
import json
import os

import pymongo
import requests
from bson import json_util
from bson.objectid import ObjectId
from dotenv import load_dotenv
from flask import Blueprint, current_app, jsonify, request

load_dotenv()
MONGODB_URL = os.getenv("MONGODB_URL")
usuarios_bp = Blueprint("usuarios", __name__)

client = pymongo.MongoClient(MONGODB_URL)
db = client.examen2022
usuarios = db.usuarios

#CRUD de usuarios (GET ALL, GET, POST, PUT, DELETE).
# Buscar un usuario de la red social a partir de su alias.


@usuarios_bp.route("/", methods=["GET"])
def get_usuarios():
    try:
        params = {}
        if "alias" in request.args:
            params["alias"] = {"$regex": request.args["alias"], "$options": "i"}
        users = usuarios.find(params)
        return jsonify(json.loads(json_util.dumps(users))), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 400

@usuarios_bp.route("/<telefono>", methods=["GET"])
def get_usuario(telefono):
    try:
        user = usuarios.find_one({"telefono": telefono})
        return jsonify(json.loads(json_util.dumps(user))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@usuarios_bp.route("/", methods=["POST"])
def post_usuario():
    try:
        data = request.json
        requestData = {}
        if data is None:
            return jsonify({"error": "No se encontró el body"}), 400
        if "alias" not in data:
```

```
            return jsonify({"error": "Alias es requerido"}), 400
        if "telefono" not in data:
            return jsonify({"error": "Telefono es requerido"}), 400

        requestData["alias"] = str(data["alias"])
        requestData["telefono"] = str(data["telefono"])

        if usuarios.find_one({"telefono": requestData["telefono"]}):
            return jsonify({"error": "Usuario ya existe"}), 400

        usuarios.insert_one(requestData)
        return jsonify({"message": "Usuario creado"}), 201
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@usuarios_bp.route("/<telefono>", methods=["PUT"])
def put_usuario(telefono):
    try:
        data = request.json
        newData = {}
        if data is None:
            return jsonify({"error": "No se encontró el body"}), 400

        if "alias" in data:
            newData["alias"] = str(data["alias"])
        if "telefono" in data:
            newData["telefono"] = str(data["telefono"])

        res = usuarios.find_one_and_update({"telefono": telefono}, {"$set":
newData})
        if res is None:
            return jsonify({"error": "Usuario no encontrado"}), 404
        return jsonify({"message": "Usuario actualizado"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@usuarios_bp.route("/<telefono>", methods=["DELETE"])
def delete_usuario(telefono):
    try:
        res = usuarios.find_one_and_delete({"telefono": telefono})
        if res is None:
            return jsonify({"error": "Usuario no encontrado"}), 404
        return jsonify({"message": "Usuario eliminado"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

```

2022/src/services/main/mensajes.py

```python

import datetime
import json
import os

import pymongo
import requests
from bson import json_util
from bson.objectid import ObjectId
from dotenv import load_dotenv
from flask import Blueprint, current_app, jsonify, request

load_dotenv()
MONGODB_URL = os.getenv("MONGODB_URL")
mensajes_bp = Blueprint("mensajes", __name__)

client = pymongo.MongoClient(MONGODB_URL)
db = client.examen2022
mensajes = db.mensajes
contactos = db.contactos

#CRUD de mensajes (GET ALL, GET, POST, PUT, DELETE).
# Atributos de un mensaje: id, timestmap, origen, destino, texto (max 400 chars).
@mensajes_bp.route("/", methods=["GET"])
def get_mensajes():
    try:
        params = {}
        if "origen" in request.args:
            params["origen"] = request.args["origen"]
        if "destino" in request.args:
            params["destino"] = request.args["destino"]
        msgs = mensajes.find(params)
        if msgs is None:
            return jsonify({"error": "No se encontraron mensajes"}), 400
        return jsonify(json.loads(json_util.dumps(msgs))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@mensajes_bp.route("/<telefono>", methods=["GET"])
def get_mensaje(telefono):
    try:
        params = {}
        if "texto" in request.args:
            params["texto"] = {"$regex": request.args["texto"], "$options": "i"}
        msg = mensajes.find({"$or": [{"origen": telefono}, {"destino": telefono}],
**params})
        if msg is None:
            return jsonify({"error": "Mensaje no encontrado"}), 404
        return jsonify(json.loads(json_util.dumps(msg))), 200
    except Exception as e:
```

```
        return jsonify({"error": str(e)}), 400

@mensajes_bp.route("/", methods=["POST"])
def post_mensaje():
    try:
        data = request.json
        requestData = {}
        if data is None:
            return jsonify({"error": "No se encontró el body"}), 400
        if "origen" not in data:
            return jsonify({"error": "Origen es requerido"}), 400
        if "destino" not in data:
            return jsonify({"error": "Destino es requerido"}), 400
        if "texto" not in data:
            return jsonify({"error": "Texto es requerido"}), 400
        if len(data["texto"]) > 400:
            return jsonify({"error": "Texto no puede ser mayor a 400
caracteres"}), 400

        requestData["origen"] = str(data["origen"])
        requestData["destino"] = str(data["destino"])
        requestData["texto"] = str(data["texto"])
        requestData["timestamp"] = str(datetime.datetime.now())

        mensajes.insert_one(requestData)
        return jsonify({"success": "Mensaje enviado"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

# enviar mensaje a un contacto a partir de su alias.
@mensajes_bp.route("<telefono>/enviar/<alias>", methods=["POST"])
def post_mensaje_alias(telefono, alias):
    try:
        contacto = contactos.find_one({"telefono": telefono, "aliasContacto":
alias})
        if contacto is None:
            return jsonify({"error": "Usuario no encontrado"}), 404
        data = request.json
        requestData = {}
        if data is None:
            return jsonify({"error": "No se encontró el body"}), 400
        if "texto" not in data:
            return jsonify({"error": "Texto es requerido"}), 400
        if len(data["texto"]) > 400:
            return jsonify({"error": "Texto no puede ser mayor a 400
caracteres"}), 400

        requestData["origen"] = str(telefono)
        requestData["destino"] = str(contacto["telefonoContacto"])
        requestData["texto"] = str(data["texto"])
        requestData["timestamp"] = str(datetime.datetime.now())

        mensajes.insert_one(requestData)
```

```
        return jsonify({"success": "Mensaje enviado"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400


@mensajes_bp.route("/<id>", methods=["PUT"])
def put_mensaje(id):
    try:
        data = request.json
        requestData = {}
        if data is None:
            return jsonify({"error": "No se encontró el body"}), 400
        if "texto" not in data:
            return jsonify({"error": "Texto es requerido"}), 400
        if len(data["texto"]) > 400:
            return jsonify({"error": "Texto no puede ser mayor a 400 caracteres"}), 400

        requestData["texto"] = str(data["texto"])
        requestData["timestamp"] = str(datetime.datetime.now())

        mensajes.update_one({"_id": ObjectId(id)}, {"$set": requestData})
        return jsonify({"success": "Mensaje actualizado"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@mensajes_bp.route("/<id>", methods=["DELETE"])
def delete_mensaje(id):
    try:
        res = mensajes.find_one_and_delete({"_id": ObjectId(id)})
        if res is None:
            return jsonify({"error": "Mensaje no encontrado"}), 404
        return jsonify({"success": "Mensaje eliminado"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

#Obtener las conversaciones de un usuario, representadas como una lista de alias de contactos ordenada
#(descendente) por el timestamp del último mensaje enviado o recibido, de forma similar a como se muestra en la
#interfaz de WhatsApp o Telegram
@mensajes_bp.route("/<telefono>/conversaciones", methods=["GET"])
def get_conversaciones(telefono):
    try:
        messages = mensajes.find({"$or": [{"origen": telefono}, {"destino": telefono}]}).sort("timestamp", pymongo.DESCENDING)
        return jsonify(json.loads(json_util.dumps(messages))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

#Obtener los mensajes de una conversación con un contacto, ordenados por timestamp.
@mensajes_bp.route("/<telefono>/conversaciones/<contacto>", methods=["GET"])
```

```
148 def get_conversacion(telefono, contacto):
149     try:
150         messages = mensajes.find({"$or": [{"origen": telefono, "destino":
    contacto}, {"origen": contacto, "destino": telefono}]}).sort("timestamp",
    pymongo.DESCENDING)
151         return jsonify(json.loads(json_util.dumps(messages))), 200
152     except Exception as e:
153         return jsonify({"error": str(e)}), 400
154
```

2022/src/services/main/contactos.py

```python
import json
import os

import pymongo
import requests
from bson import json_util
from bson.objectid import ObjectId
from dotenv import load_dotenv
from flask import Blueprint, current_app, jsonify, request

load_dotenv()
MONGODB_URL = os.getenv("MONGODB_URL")
contactos_bp = Blueprint("contactos", __name__)

client = pymongo.MongoClient(MONGODB_URL)
db = client.examen2022
contactos = db.contactos
usuarios = db.usuarios

#CRUD de contactos (GET ALL, GET, POST, PUT, DELETE).

@contactos_bp.route("/", methods=["GET"])
def get_contactos(telefono):
    try:
        if not usuarios.find_one({"telefono": telefono}):
            return jsonify({"error": "Usuario no existe"}), 400

        params = {"telefono": telefono}
        if "alias" in request.args:
            params["aliasContacto"] = {"$regex": request.args["alias"],
"$options": "i"}

        contacts = contactos.find(params)
        if contacts is None:
            return jsonify({"error": "No se encontraron contactos"}), 400
        return jsonify(json.loads(json_util.dumps(contacts))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@contactos_bp.route("/<contacto>", methods=["GET"])
def get_contacto(telefono, contacto):
    try:
        if not usuarios.find_one({"telefono": telefono}):
            return jsonify({"error": "Usuario no existe"}), 400
        contact = contactos.find_one({"telefono": telefono, "telefonoContacto":
contacto})
        if contact is None:
            return jsonify({"error": "No se encontró el contacto"}), 400
        return jsonify(json.loads(json_util.dumps(contact))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400
```

```python

@contactos_bp.route("/", methods=["POST"])
def post_contacto(telefono):
    try:
        if not usuarios.find_one({"telefono": telefono}):
            return jsonify({"error": "Usuario no existe"}), 400
        data = request.json
        requestData = {}
        if data is None:
            return jsonify({"error": "No se encontró el body"}), 400
        if "telefonoContacto" not in data:
            return jsonify({"error": "Telefono de contacto es requerido"}), 400
        if "aliasContacto" not in data:
            return jsonify({"error": "Alias de contacto es requerido"}), 400

        requestData["telefono"] = telefono
        requestData["telefonoContacto"] = str(data["telefonoContacto"])
        requestData["aliasContacto"] = str(data["aliasContacto"])

        if contactos.find_one({"telefono": requestData["telefono"], "telefonoContacto": requestData["telefonoContacto"]}):
            return jsonify({"error": "Contacto ya existe"}), 400

        if not usuarios.find_one({"telefono": requestData["telefonoContacto"], "alias": requestData["aliasContacto"]}):
            return jsonify({"error": "El contacto no existe en la lista de usuarios"}), 400

        contactos.insert_one(requestData)
        return jsonify({"message": "Contacto creado exitosamente"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@contactos_bp.route("/<contacto>", methods=["PUT"])
def put_contacto(telefono, contacto):
    try:
        if not usuarios.find_one({"telefono": telefono}):
            return jsonify({"error": "Usuario no existe"}), 400
        data = request.json
        newData = {}
        if data is None:
            return jsonify({"error": "No se encontró el body"}), 400
        for key in data:
            if key not in ["aliasContacto", "telefonoContacto"]:
                return jsonify({"error": "Campo no permitido"}), 400
            newData[key] = str(data[key])

        contact = contactos.find_one_and_update({"telefono": telefono, "telefonoContacto": contacto}, {"$set": newData})
        if contact is None:
            return jsonify({"error": "No se encontró el contacto"}), 400
        return jsonify({"message": "Contacto actualizado exitosamente"}), 200
    except Exception as e:
```

```
        return jsonify({"error": str(e)}), 400

@contactos_bp.route("/<contacto>", methods=["DELETE"])
def delete_contacto(telefono, contacto):
    try:
        if not usuarios.find_one({"telefono": telefono}):
            return jsonify({"error": "Usuario no existe"}), 400
        res = contactos.find_one_and_delete({"telefono": telefono, "telefonoContacto": contacto})
        if res is None:
            return jsonify({"error": "Contacto no encontrado"}), 404
        return jsonify({"message": "Contacto eliminado exitosamente"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

```

2023/src/services/main/app.py

```python
import os

from contactos import contactos_bp
from dotenv import load_dotenv
from eventos import eventos_bp
from flask import Flask, request
from usuarios import usuarios_bp

app = Flask(__name__)
# Define blueprints:
app.register_blueprint(usuarios_bp, url_prefix='/usuarios')
app.register_blueprint(contactos_bp, url_prefix='/usuarios/<email>/contactos')
app.register_blueprint(eventos_bp, url_prefix='/eventos')

@app.route('/')
def hello_world():
    return "Hello, World!"

if __name__ == '__main__':
    load_dotenv()
    app.run(host="0.0.0.0", port=os.getenv("SERVICE_PORT_MAIN"))

```

2023/src/services/main/usuarios.py

```python
import json
import os

import pymongo
import requests
from bson import json_util
from bson.objectid import ObjectId
from dotenv import load_dotenv
from flask import Blueprint, current_app, jsonify, request

load_dotenv()
MONGODB_URL = os.getenv("MONGODB_URL")
usuarios_bp = Blueprint("usuarios", __name__)

client = pymongo.MongoClient(MONGODB_URL)
db = client.examen2023
usuarios = db.usuarios
eventos = db.eventos

@usuarios_bp.route("/", methods=["GET"])
def get_usuarios():
    try:
        query = {}
        if request.args:
            if "email" in request.args:
                query["email"] = request.args["email"]
            if "nombre" in request.args:
                query["nombre"] = request.args["nombre"]

        usuarios_list = usuarios.find(query)
        return jsonify(json.loads(json_util.dumps(usuarios_list))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@usuarios_bp.route("/<email>", methods=["GET"])
def get_usuario(email):
    try:
        usuario = usuarios.find({"email": email})
        return jsonify(json.loads(json_util.dumps(usuario))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@usuarios_bp.route("/", methods=["POST"])
def create_usuario():
    try:
        data = request.json
        if not data:
            return jsonify({"error": "No se envió la información requerida"}), 400
        if "email" not in data:
            return jsonify({"error": "El email es requerido"}), 400
```

```
        if "nombre" not in data:
            return jsonify({"error": "El nombre es requerido"}), 400

        if usuarios.find_one({"email": data["email"]}):
            return jsonify({"error": "El email ya está registrado"}), 400

        usuario = usuarios.insert_one(data)
        return jsonify({"mensaje": "Usuario creado exitosamente", "id":
str(usuario.inserted_id)}), 201
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@usuarios_bp.route("/<email>", methods=["PUT"])
def update_usuario(email):
    try:
        data = request.json
        if not data:
            return jsonify({"error": "No se envió la información requerida"}), 400
        query = {}
        if "nombre" in data:
            query["nombre"] = data["nombre"]
        if "email" in data:
            query["email"] = data["email"]

        if usuarios.find_one({"email": email}):
            usuarios.update_one({"email": email}, {"$set": query})
            return jsonify({"mensaje": "Usuario actualizado exitosamente"}), 200
        else:
            return jsonify({"error": "El usuario no existe"}), 404
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@usuarios_bp.route("/<email>", methods=["DELETE"])
def delete_usuario(email):
    try:
        if usuarios.find_one_and_delete({"email": email}):
            return jsonify({"mensaje": "Usuario eliminado exitosamente"}), 200
        else:
            return jsonify({"error": "El usuario no existe"}), 404
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@usuarios_bp.route("/<email>/eventos", methods=["GET"])
def get_eventos(email):
    try:
        if not usuarios.find({"email": email}):
            return jsonify({"error": "El email del usuario no existe"}), 400

        eventos_list = eventos.find({"$or": [{"anfitrion": email},
{"invitados.email": email}]}).sort("inicio", pymongo.ASCENDING)
        return jsonify(json.loads(json_util.dumps(eventos_list))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

```
102
```

2023/src/services/main/eventos.py

```python
import datetime
import json
import os

import pymongo
import requests
from bson import json_util
from bson.objectid import ObjectId
from dotenv import load_dotenv
from flask import Blueprint, current_app, jsonify, request

load_dotenv()
MONGODB_URL = os.getenv("MONGODB_URL")
eventos_bp = Blueprint("eventos", __name__)

client = pymongo.MongoClient(MONGODB_URL)
db = client.examen2023
usuarios = db.usuarios
eventos = db.eventos
contactos = db.contactos

# Eventos. Representa los eventos de agenda, con los siguientes atributos:
# o identificador. Clave o identificador del evento, propia de la base de datos elegida.
# o anfitrión. Dirección de email del usuario anfitrión del evento (el que crea el evento).
# o descripción. Título o descripción breve (hasta 50 caracteres) del evento.
# o inicio. Timestamp con la fecha y hora de inicio del evento (en tramos de 15 minutos).
# o duración. Duración del evento (en tramos de 15 minutos).
# o invitados. Lista de invitados al evento, representado cada uno por su email y el estado de la invitación
# (aceptada/pendiente).

# • Invitar a un evento a un contacto de un usuario, identificado por su email. El contacto invitado se incluirá en la
# lista de invitados del evento, con la invitación en estado pendiente.
# • Aceptar una invitación a un evento. El estado de la invitación pasará a aceptada.
# • Reprogramar un evento ya pasado, indicando cuánto tiempo se desplaza (un número de días determinado, una
# semana, un mes, o un año). Se creará un nuevo evento, con la nueva fecha y el resto de valores iguales a los del
# evento reprogramado.
# • Obtener la agenda de un usuario, representada por una lista de eventos, tanto propios como invitados, por orden
# ascendente de inicio
# Se valorará que en todas estas operaciones se realicen los controles de errores necesarios para evitar corromper la
# información de la base de datos (fechas/horas inexistentes o incorrectas, invitar a un evento a alguien que no sea
```

```python
# contacto de su anfitrión, aceptar un evento al que no se ha sido invitado, etc.)

@eventos_bp.route("/", methods=["GET"])
def get_eventos():
    try:
        query = {}
        if request.args:
            if "anfitrion" in request.args:
                query["anfitrion"] = request.args["anfitrion"]
            if "descripcion" in request.args:
                query["descripcion"] = request.args["descripcion"]

        eventos_list = eventos.find(query)
        return jsonify(json.loads(json_util.dumps(eventos_list))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@eventos_bp.route("/<id>", methods=["GET"])
def get_evento(id):
    try:
        evento = eventos.find({"_id": ObjectId(id)})
        return jsonify(json.loads(json_util.dumps(evento))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@eventos_bp.route("/", methods=["POST"]) # lista de invitados [email, estado]
def create_evento():
    try:
        data = request.json
        if not data:
            return jsonify({"error": "No se envió la información requerida"}), 400
        if "anfitrion" not in data:
            return jsonify({"error": "El anfitrión es requerido"}), 400
        if "descripcion" not in data:
            return jsonify({"error": "La descripción es requerida"}), 400
        if "inicio" not in data:
            return jsonify({"error": "El inicio es requerido"}), 400
        if "duracion" not in data:
            return jsonify({"error": "La duración es requerida"}), 400

        if len(data["descripcion"]) > 50:
            return jsonify({"error": "La descripción debe tener menos de 50
caracteres"}), 400

        if not data["invitados"]:
            return jsonify({"error": "La lista de invitados es requerida"}), 400

        for invitado in data["invitados"]:
            if "email" not in invitado:
                return jsonify({"error": "El email del invitado es requerido"}),
400
            if "estado" not in invitado:
```

```
                return jsonify({"error": "El estado del invitado es requerido"}), 400
            if not contactos.find_one({"email": data["anfitrion"], "emailContacto": invitado["email"]}):
                return jsonify({"error": "El invitado no es un contacto del anfitrión"}), 400
            if invitado["estado"] not in ["aceptada", "pendiente"]:
                return jsonify({"error": "El estado del invitado debe ser aceptada o pendiente"}), 400

        # comprobar tramos
        inicio = data["inicio"]
        inicioDatetime = datetime.datetime.strptime(inicio, "%Y-%m-%dT%H:%M:%S.%fZ")
        duracion = int(data["duracion"])
        #inicio es un timestamp en string formato ISO "2023-06-01T00:00:00.000Z"
        if inicioDatetime.minute % 15 != 0:
            return jsonify({"error": "El inicio debe ser un tramo de 15 minutos"}), 400
        if duracion % 15 != 0:
            return jsonify({"error": "La duración debe ser un tramo de 15 minutos"}), 400

        if not usuarios.find_one({"email": data["anfitrion"]}):
            return jsonify({"error": "El anfitrión no existe"}), 400

        evento = eventos.insert_one(data)
        return jsonify({"mensaje": "Evento creado exitosamente", "id": str(evento.inserted_id)}), 201
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@eventos_bp.route("/<id>", methods=["PUT"])
def update_evento(id):
    try:
        data = request.json
        if not data:
            return jsonify({"error": "No se envió la información requerida"}), 400

        if data["descripcion"]:
            if len(data["descripcion"]) > 50:
                return jsonify({"error": "La descripción debe tener menos de 50 caracteres"}), 400

        if data["invitados"]:
            for invitado in data["invitados"]:
                if "email" not in invitado:
                    return jsonify({"error": "El email del invitado es requerido"}), 400
                if "estado" not in invitado:
                    return jsonify({"error": "El estado del invitado es requerido"}), 400
                if not contactos.find_one({"email": data["anfitrion"], "emailContacto": invitado["email"]}):
```

```
                    return jsonify({"error": "El invitado no es un contacto del anfitrión"}), 400
                if invitado["estado"] not in ["aceptada", "pendiente"]:
                    return jsonify({"error": "El estado del invitado debe ser aceptada o pendiente"}), 400

        # comprobar tramos
        if data["inicio"]:
            inicio = data["inicio"]
            inicioDatetime = datetime.datetime.strptime(inicio, "%Y-%m-%dT%H:%M:%S.%fZ")
            if inicioDatetime.minute % 15 != 0:
                return jsonify({"error": "El inicio debe ser un tramo de 15 minutos"}), 400
        if data["duracion"]:
            duracion = int(data["duracion"])
            if duracion % 15 != 0:
                return jsonify({"error": "La duración debe ser un tramo de 15 minutos"}), 400

        if data["anfitrion"]:
            if not usuarios.find_one({"email": data["anfitrion"]}):
                return jsonify({"error": "El anfitrión no existe"}), 400

        if eventos.find_one({"_id": ObjectId(id)}):
            eventos.update_one({"_id": ObjectId(id)}, {"$set": data})
            return jsonify({"mensaje": "Evento actualizado exitosamente"}), 200
        else:
            return jsonify({"error": "El evento no existe"}), 404
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@eventos_bp.route("/<id>", methods=["DELETE"])
def delete_evento(id):
    try:
        res = eventos.find_one_and_delete({"_id": ObjectId(id)})
        if not res:
            return jsonify({"error": "El evento no existe"}), 400
        return jsonify({"mensaje": "Evento eliminado exitosamente"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@eventos_bp.route("/<id>/invitar", methods=["PUT"])
def invitar_evento(id):
    try:
        evento = eventos.find_one({"_id": ObjectId(id)})
        if not evento:
            return jsonify({"error": "El evento no existe"}), 404
        anfitrion = evento["anfitrion"]
        invitados = []
        data = request.json # data tiene que tener formato [{},{},...]
        if not data:
            return jsonify({"error": "No se envió la información requerida"}), 400
```

```
        if not data["invitados"]:
            return jsonify({"error": "La lista de invitados es requerida"}), 400

        for invitado in data["invitados"]:
            if "email" not in invitado:
                return jsonify({"error": "El email del invitado es requerido"}),
400

            if not contactos.find_one({"email": anfitrion, "emailContacto":
invitado["email"]}):
                return jsonify({"error": "El invitado no es un contacto del
anfitrión"}), 400
            if not eventos.find_one({"_id": ObjectId(id), "invitados.email":
invitado["email"]}):
                invitados.append({"email": invitado["email"], "estado":
"pendiente"})


        if not eventos.update_one({"_id": ObjectId(id)}, {"$push": {"invitados":
{"$each": invitados}}}):
            return jsonify({"error": "El evento no existe"}), 404

        return jsonify({"mensaje": "Invitación enviada exitosamente"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@eventos_bp.route("/<id>/aceptar", methods=["PUT"])
def aceptar_evento(id):
    try:
        data = request.json
        if not data:
            return jsonify({"error": "No se envió la información requerida"}), 400
        if "email" not in data:
            return jsonify({"error": "El email es requerido"}), 400

        evento = eventos.find_one({"_id": ObjectId(id)})
        if not evento:
            return jsonify({"error": "El evento no existe"}), 404

        if not eventos.find_one({"_id": ObjectId(id), "invitados.email":
data["email"]}):
            return jsonify({"error": "No se ha sido invitado a este evento"}), 400

        if not eventos.update_one({"_id": ObjectId(id), "invitados.email":
data["email"]}, {"$set": {"invitados.$.estado": "aceptada"}}):
            return jsonify({"error": "El evento no existe"}), 404

        return jsonify({"mensaje": "Invitación aceptada exitosamente"}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 400

@eventos_bp.route("/<id>/reprogramar", methods=["POST"]) # crea una copia del
evento con la nueva fecha
def reprogramar_evento(id):
```

```
    try:
        data = request.json
        if not data:
            return jsonify({"error": "No se envió la información requerida"}), 400
        if "dias" not in data:
            return jsonify({"error": "Los días son requeridos"}), 400

        evento = eventos.find_one({"_id": ObjectId(id)})
        if not evento:
            return jsonify({"error": "El evento no existe"}), 404
        del evento["_id"]
        if not evento:
            return jsonify({"error": "El evento no existe"}), 404

        inicio = evento["inicio"]
        inicioDatetime = datetime.datetime.strptime(inicio, "%Y-%m-%dT%H:%M:%S.
%fZ")
        dias = int(data["dias"])
        inicioDatetime = inicioDatetime + datetime.timedelta(days=int(dias))
        inicio = inicioDatetime.strftime("%Y-%m-%dT%H:%M:%S.%fZ")
        evento["inicio"] = inicio

        if not eventos.insert_one(evento):
            return jsonify({"error": "Error al reprogramar el evento"}), 400

        return jsonify({"mensaje": "Evento reprogramado exitosamente"}), 201
    except Exception as e:
        return jsonify({"error": str(e)}), 400

```

2023/src/services/main/contactos.py

```python
import json
import os

import pymongo
import requests
from bson import json_util
from bson.objectid import ObjectId
from dotenv import load_dotenv
from flask import Blueprint, current_app, jsonify, request

load_dotenv()
MONGODB_URL = os.getenv("MONGODB_URL")
contactos_bp = Blueprint("contactos", __name__)

client = pymongo.MongoClient(MONGODB_URL)
db = client.examen2023
usuarios = db.usuarios
contactos = db.contactos

@contactos_bp.route("/", methods=["GET"])
def get_contactos(email):
    try:
        query = {"email": email}
        if request.args:
            if "nombreContacto" in request.args:
                query["nombreContacto"] = {"$regex":
request.args["nombreContacto"], "$options": "i"}

        contactos_list = contactos.find(query)
        return jsonify(json.loads(json_util.dumps(contactos_list))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@contactos_bp.route("/<emailContacto>", methods=["GET"])
def get_contacto(email, emailContacto):
    try:
        if not usuarios.find_one({"email": email}):
            return jsonify({"error": "El email del usuario no existe"}), 400
        if not usuarios.find_one({"email": emailContacto}):
            return jsonify({"error": "El email del contacto no existe"}), 400

        contacto = contactos.find({"email": email, "emailContacto": emailContacto})
        return jsonify(json.loads(json_util.dumps(contacto))), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@contactos_bp.route("/", methods=["POST"])
def create_contacto(email):
    try:
        query = {"email": email}
```

```
        data = request.json
        if not data:
            return jsonify({"error": "No se envió la información requerida"}), 400
        if "emailContacto" not in data:
            return jsonify({"error": "El email del contacto es requerido"}), 400
        if "nombreContacto" not in data:
            return jsonify({"error": "El nombre del contacto es requerido"}), 400

        if not usuarios.find_one({"email": email}):
            return jsonify({"error": "El email del usuario no existe"}), 400
        if not usuarios.find_one({"email": data["emailContacto"]}):
            return jsonify({"error": "El email del contacto no existe"}), 400

        query["emailContacto"] = data["emailContacto"]
        query["nombreContacto"] = data["nombreContacto"]

        if contactos.find_one({"email": email, "emailContacto": query["emailContacto"]}):
            return jsonify({"error": "El contacto ya está registrado"}), 400

        contacto = contactos.insert_one(query)
        return jsonify({"mensaje": "Contacto creado exitosamente", "id": str(contacto.inserted_id)}), 201
    except Exception as e:
        return jsonify({"Los datos no son correctos": str(e)}), 400

@contactos_bp.route("/<emailContacto>", methods=["DELETE"])
def delete_contacto(email, emailContacto):
    try:
        query = {"email": email, "emailContacto": emailContacto}
        res = contactos.find_one_and_delete(query)
        if not res:
            return jsonify({"error": "El contacto no existe"}), 400
        return jsonify({"mensaje": "Contacto eliminado exitosamente"}), 200
    except Exception as e:
        return jsonify({"Los datos no son correctos": str(e)}), 400

```