

2022/src/services/main/app.py

```
1 import os
2
3 from contactos import contactos_bp
4 from dotenv import load_dotenv
5 from flask import Flask, request
6 from mensajes import mensajes_bp
7 from usuarios import usuarios_bp
8
9 app = Flask(__name__)
10 # Define blueprints:
11 app.register_blueprint(usuarios_bp, url_prefix='/usuarios')
12 app.register_blueprint(mensajes_bp, url_prefix='/mensajes')
13 app.register_blueprint(contactos_bp, url_prefix='/usuarios/<telefono>/contactos')
14
15 @app.route('/')
16 def hello_world():
17     return 'Hello, World!'
18
19 if __name__ == '__main__':
20     load_dotenv()
21     app.run(host="0.0.0.0", port=os.getenv("SERVICE_PORT_MAIN"))
22
23 # Rutas que se deben implementar:
24
```

2022/src/services/main/usuarios.py

```
1 import json
2 import os
3
4 import pymongo
5 import requests
6 from bson import json_util
7 from bson.objectid import ObjectId
8 from dotenv import load_dotenv
9 from flask import Blueprint, current_app, jsonify, request
10
11 load_dotenv()
12 MONGODB_URL = os.getenv("MONGODB_URL")
13 usuarios_bp = Blueprint("usuarios", __name__)
14
15 client = pymongo.MongoClient(MONGODB_URL)
16 db = client.examen2022
17 usuarios = db.usuarios
18
19 #CRUD de usuarios (GET ALL, GET, POST, PUT, DELETE).
20 # Buscar un usuario de la red social a partir de su alias.
21
22
23 @usuarios_bp.route("/", methods=["GET"])
24 def get_usuarios():
25     try:
26         params = {}
27         if "alias" in request.args:
28             params["alias"] = {"$regex": request.args["alias"], "$options": "i"}
29         users = usuarios.find(params)
30         return jsonify(json.loads(json_util.dumps(users))), 200
31
32     except Exception as e:
33         return jsonify({"error": str(e)}), 400
34
35 @usuarios_bp.route("/<telefono>", methods=["GET"])
36 def get_usuario(telefono):
37     try:
38         user = usuarios.find_one({"telefono": telefono})
39         return jsonify(json.loads(json_util.dumps(user))), 200
40     except Exception as e:
41         return jsonify({"error": str(e)}), 400
42
43 @usuarios_bp.route("/", methods=["POST"])
44 def post_usuario():
45     try:
46         data = request.json
47         requestData = {}
48         if data is None:
49             return jsonify({"error": "No se encontró el body"}), 400
50         if "alias" not in data:
```

```
51         return jsonify({"error": "Alias es requerido"}), 400
52     if "telefono" not in data:
53         return jsonify({"error": "Telefono es requerido"}), 400
54
55     requestData["alias"] = str(data["alias"])
56     requestData["telefono"] = str(data["telefono"])
57
58     if usuarios.find_one({"telefono": requestData["telefono"]}):
59         return jsonify({"error": "Usuario ya existe"}), 400
60
61     usuarios.insert_one(requestData)
62     return jsonify({"message": "Usuario creado"}), 201
63 except Exception as e:
64     return jsonify({"error": str(e)}), 400
65
66 @usuarios_bp.route("/<telefono>", methods=["PUT"])
67 def put_usuario(telefono):
68     try:
69         data = request.json
70         newData = {}
71         if data is None:
72             return jsonify({"error": "No se encontró el body"}), 400
73
74         if "alias" in data:
75             newData["alias"] = str(data["alias"])
76         if "telefono" in data:
77             newData["telefono"] = str(data["telefono"])
78
79         res = usuarios.find_one_and_update({"telefono": telefono}, {"$set":
newData})
80         if res is None:
81             return jsonify({"error": "Usuario no encontrado"}), 404
82         return jsonify({"message": "Usuario actualizado"}), 200
83     except Exception as e:
84         return jsonify({"error": str(e)}), 400
85
86 @usuarios_bp.route("/<telefono>", methods=["DELETE"])
87 def delete_usuario(telefono):
88     try:
89         res = usuarios.find_one_and_delete({"telefono": telefono})
90         if res is None:
91             return jsonify({"error": "Usuario no encontrado"}), 404
92         return jsonify({"message": "Usuario eliminado"}), 200
93     except Exception as e:
94         return jsonify({"error": str(e)}), 400
95
```

2022/src/services/main/mensajes.py

```
1
2 import datetime
3 import json
4 import os
5
6 import pymongo
7 import requests
8 from bson import json_util
9 from bson.objectid import ObjectId
10 from dotenv import load_dotenv
11 from flask import Blueprint, current_app, jsonify, request
12
13 load_dotenv()
14 MONGODB_URL = os.getenv("MONGODB_URL")
15 mensajes_bp = Blueprint("mensajes", __name__)
16
17 client = pymongo.MongoClient(MONGODB_URL)
18 db = client.examen2022
19 mensajes = db.mensajes
20 contactos = db.contactos
21
22 #CRUD de mensajes (GET ALL, GET, POST, PUT, DELETE).
23 # Atributos de un mensaje: id, timestmap, origen, destino, texto (max 400 chars).
24 @mensajes_bp.route("/", methods=["GET"])
25 def get_mensajes():
26     try:
27         params = {}
28         if "origen" in request.args:
29             params["origen"] = request.args["origen"]
30         if "destino" in request.args:
31             params["destino"] = request.args["destino"]
32         msgs = mensajes.find(params)
33         if msgs is None:
34             return jsonify({"error": "No se encontraron mensajes"}), 400
35         return jsonify(json.loads(json_util.dumps(msgs))), 200
36     except Exception as e:
37         return jsonify({"error": str(e)}), 400
38
39 @mensajes_bp.route("/<telefono>", methods=["GET"])
40 def get_mensaje(telefono):
41     try:
42         params = {}
43         if "texto" in request.args:
44             params["texto"] = {"$regex": request.args["texto"], "$options": "i"}
45         msg = mensajes.find({"$or": [{"origen": telefono}, {"destino": telefono}],
46 **params})
47         if msg is None:
48             return jsonify({"error": "Mensaje no encontrado"}), 404
49         return jsonify(json.loads(json_util.dumps(msg))), 200
50     except Exception as e:
```

```
50     return jsonify({"error": str(e)}), 400
51
52 @mensajes_bp.route("/", methods=["POST"])
53 def post_mensaje():
54     try:
55         data = request.json
56         requestData = {}
57         if data is None:
58             return jsonify({"error": "No se encontró el body"}), 400
59         if "origen" not in data:
60             return jsonify({"error": "Origen es requerido"}), 400
61         if "destino" not in data:
62             return jsonify({"error": "Destino es requerido"}), 400
63         if "texto" not in data:
64             return jsonify({"error": "Texto es requerido"}), 400
65         if len(data["texto"]) > 400:
66             return jsonify({"error": "Texto no puede ser mayor a 400
caracteres"}), 400
67
68         requestData["origen"] = str(data["origen"])
69         requestData["destino"] = str(data["destino"])
70         requestData["texto"] = str(data["texto"])
71         requestData["timestamp"] = str(datetime.datetime.now())
72
73         mensajes.insert_one(requestData)
74         return jsonify({"success": "Mensaje enviado"}), 200
75     except Exception as e:
76         return jsonify({"error": str(e)}), 400
77
78 # enviar mensaje a un contacto a partir de su alias.
79 @mensajes_bp.route("<telefono>/enviar/<alias>", methods=["POST"])
80 def post_mensaje_alias(telefono, alias):
81     try:
82         contacto = contactos.find_one({"telefono": telefono, "aliasContacto":
alias})
83         if contacto is None:
84             return jsonify({"error": "Usuario no encontrado"}), 404
85         data = request.json
86         requestData = {}
87         if data is None:
88             return jsonify({"error": "No se encontró el body"}), 400
89         if "texto" not in data:
90             return jsonify({"error": "Texto es requerido"}), 400
91         if len(data["texto"]) > 400:
92             return jsonify({"error": "Texto no puede ser mayor a 400
caracteres"}), 400
93
94         requestData["origen"] = str(telefono)
95         requestData["destino"] = str(contacto["telefonoContacto"])
96         requestData["texto"] = str(data["texto"])
97         requestData["timestamp"] = str(datetime.datetime.now())
98
99         mensajes.insert_one(requestData)
```

```
100         return jsonify({"success": "Mensaje enviado"}), 200
101     except Exception as e:
102         return jsonify({"error": str(e)}), 400
103
104
105 @mensajes_bp.route("/<id>", methods=["PUT"])
106 def put_mensaje(id):
107     try:
108         data = request.json
109         requestData = {}
110         if data is None:
111             return jsonify({"error": "No se encontró el body"}), 400
112         if "texto" not in data:
113             return jsonify({"error": "Texto es requerido"}), 400
114         if len(data["texto"]) > 400:
115             return jsonify({"error": "Texto no puede ser mayor a 400
116 caracteres"}), 400
117
118         requestData["texto"] = str(data["texto"])
119         requestData["timestamp"] = str(datetime.datetime.now())
120
121         mensajes.update_one({"_id": ObjectId(id)}, {"$set": requestData})
122         return jsonify({"success": "Mensaje actualizado"}), 200
123     except Exception as e:
124         return jsonify({"error": str(e)}), 400
125
126 @mensajes_bp.route("/<id>", methods=["DELETE"])
127 def delete_mensaje(id):
128     try:
129         res = mensajes.find_one_and_delete({"_id": ObjectId(id)})
130         if res is None:
131             return jsonify({"error": "Mensaje no encontrado"}), 404
132         return jsonify({"success": "Mensaje eliminado"}), 200
133     except Exception as e:
134         return jsonify({"error": str(e)}), 400
135
136 #Obtener las conversaciones de un usuario, representadas como una lista de alias
de contactos ordenada
137 #(descendente) por el timestamp del último mensaje enviado o recibido, de forma
similar a como se muestra en la
138 #interfaz de WhatsApp o Telegram
139 @mensajes_bp.route("/<telefono>/conversaciones", methods=["GET"])
140 def get_conversaciones(telefono):
141     try:
142         messages = mensajes.find({"$or": [{"origen": telefono}, {"destino":
telefono}]}).sort("timestamp", pymongo.DESCENDING)
143         return jsonify(json.loads(json_util.dumps(messages))), 200
144     except Exception as e:
145         return jsonify({"error": str(e)}), 400
146
147 #Obtener los mensajes de una conversación con un contacto, ordenados por
timestamp.
148 @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

```
1 import json
2 import os
3
4 import pymongo
5 import requests
6 from bson import json_util
7 from bson.objectid import ObjectId
8 from dotenv import load_dotenv
9 from flask import Blueprint, current_app, jsonify, request
10
11 load_dotenv()
12 MONGODB_URL = os.getenv("MONGODB_URL")
13 contactos_bp = Blueprint("contactos", __name__)
14
15 client = pymongo.MongoClient(MONGODB_URL)
16 db = client.examen2022
17 contactos = db.contactos
18 usuarios = db.usuarios
19
20 #CRUD de contactos (GET ALL, GET, POST, PUT, DELETE).
21
22 @contactos_bp.route("/", methods=["GET"])
23 def get_contactos(telefono):
24     try:
25         if not usuarios.find_one({"telefono": telefono}):
26             return jsonify({"error": "Usuario no existe"}), 400
27
28         params = {"telefono": telefono}
29         if "alias" in request.args:
30             params["aliasContacto"] = {"$regex": request.args["alias"],
31 "$options": "i"}
32
33         contacts = contactos.find(params)
34         if contacts is None:
35             return jsonify({"error": "No se encontraron contactos"}), 400
36         return jsonify(json.loads(json_util.dumps(contacts))), 200
37     except Exception as e:
38         return jsonify({"error": str(e)}), 400
39
40 @contactos_bp.route("/<contacto>", methods=["GET"])
41 def get_contacto(telefono, contacto):
42     try:
43         if not usuarios.find_one({"telefono": telefono}):
44             return jsonify({"error": "Usuario no existe"}), 400
45         contact = contactos.find_one({"telefono": telefono, "telefonoContacto":
46 contacto})
47         if contact is None:
48             return jsonify({"error": "No se encontró el contacto"}), 400
49         return jsonify(json.loads(json_util.dumps(contact))), 200
50     except Exception as e:
51         return jsonify({"error": str(e)}), 400
```



```
50
51 @contactos_bp.route("/", methods=["POST"])
52 def post_contacto(telefono):
53     try:
54         if not usuarios.find_one({"telefono": telefono}):
55             return jsonify({"error": "Usuario no existe"}), 400
56         data = request.json
57         requestData = {}
58         if data is None:
59             return jsonify({"error": "No se encontró el body"}), 400
60         if "telefonoContacto" not in data:
61             return jsonify({"error": "Telefono de contacto es requerido"}), 400
62         if "aliasContacto" not in data:
63             return jsonify({"error": "Alias de contacto es requerido"}), 400
64
65         requestData["telefono"] = telefono
66         requestData["telefonoContacto"] = str(data["telefonoContacto"])
67         requestData["aliasContacto"] = str(data["aliasContacto"])
68
69         if contactos.find_one({"telefono": requestData["telefono"],
70 "telefonoContacto": requestData["telefonoContacto"]}):
71             return jsonify({"error": "Contacto ya existe"}), 400
72
73         if not usuarios.find_one({"telefono": requestData["telefonoContacto"],
74 "alias": requestData["aliasContacto"]}):
75             return jsonify({"error": "El contacto no existe en la lista de
76 usuarios"}), 400
77
78         contactos.insert_one(requestData)
79         return jsonify({"message": "Contacto creado exitosamente"}), 200
80     except Exception as e:
81         return jsonify({"error": str(e)}), 400
82
83 @contactos_bp.route("/<contacto>", methods=["PUT"])
84 def put_contacto(telefono, contacto):
85     try:
86         if not usuarios.find_one({"telefono": telefono}):
87             return jsonify({"error": "Usuario no existe"}), 400
88         data = request.json
89         newData = {}
90         if data is None:
91             return jsonify({"error": "No se encontró el body"}), 400
92         for key in data:
93             if key not in ["aliasContacto", "telefonoContacto"]:
94                 return jsonify({"error": "Campo no permitido"}), 400
95             newData[key] = str(data[key])
96
97         contact = contactos.find_one_and_update({"telefono": telefono,
98 "telefonoContacto": contacto}, {"$set": newData})
99         if contact is None:
100             return jsonify({"error": "No se encontró el contacto"}), 400
101         return jsonify({"message": "Contacto actualizado exitosamente"}), 200
102     except Exception as e:
```

```
99         return jsonify({"error": str(e)}), 400
100
101 @contactos_bp.route("/<contacto>", methods=["DELETE"])
102 def delete_contacto(telefono, contacto):
103     try:
104         if not usuarios.find_one({"telefono": telefono}):
105             return jsonify({"error": "Usuario no existe"}), 400
106         res = contactos.find_one_and_delete({"telefono": telefono,
107 "telefonoContacto": contacto})
108         if res is None:
109             return jsonify({"error": "Contacto no encontrado"}), 404
110         return jsonify({"message": "Contacto eliminado exitosamente"}), 200
111     except Exception as e:
112         return jsonify({"error": str(e)}), 400
```

2023/src/services/main/app.py

```
1 import os
2
3 from contactos import contactos_bp
4 from dotenv import load_dotenv
5 from eventos import eventos_bp
6 from flask import Flask, request
7 from usuarios import usuarios_bp
8
9 app = Flask(__name__)
10 # Define blueprints:
11 app.register_blueprint(usuarios_bp, url_prefix='/usuarios')
12 app.register_blueprint(contactos_bp, url_prefix='/usuarios/<email>/contactos')
13 app.register_blueprint(eventos_bp, url_prefix='/eventos')
14
15 @app.route('/')
16 def hello_world():
17     return "Hello, World!"
18
19 if __name__ == '__main__':
20     load_dotenv()
21     app.run(host="0.0.0.0", port=os.getenv("SERVICE_PORT_MAIN"))
22
```

2023/src/services/main/usuarios.py

```
1 import json
2 import os
3
4 import pymongo
5 import requests
6 from bson import json_util
7 from bson.objectid import ObjectId
8 from dotenv import load_dotenv
9 from flask import Blueprint, current_app, jsonify, request
10
11 load_dotenv()
12 MONGODB_URL = os.getenv("MONGODB_URL")
13 usuarios_bp = Blueprint("usuarios", __name__)
14
15 client = pymongo.MongoClient(MONGODB_URL)
16 db = client.examen2023
17 usuarios = db.usuarios
18 eventos = db.eventos
19
20 @usuarios_bp.route("/", methods=["GET"])
21 def get_usuarios():
22     try:
23         query = {}
24         if request.args:
25             if "email" in request.args:
26                 query["email"] = request.args["email"]
27             if "nombre" in request.args:
28                 query["nombre"] = request.args["nombre"]
29
30         usuarios_list = usuarios.find(query)
31         return jsonify(json.loads(json_util.dumps(usuarios_list))), 200
32     except Exception as e:
33         return jsonify({"error": str(e)}), 500
34
35 @usuarios_bp.route("/<email>", methods=["GET"])
36 def get_usuario(email):
37     try:
38         usuario = usuarios.find({"email": email})
39         return jsonify(json.loads(json_util.dumps(usuario))), 200
40     except Exception as e:
41         return jsonify({"error": str(e)}), 500
42
43 @usuarios_bp.route("/", methods=["POST"])
44 def create_usuario():
45     try:
46         data = request.json
47         if not data:
48             return jsonify({"error": "No se envió la información requerida"}), 400
49         if "email" not in data:
50             return jsonify({"error": "El email es requerido"}), 400
```

```
51     if "nombre" not in data:
52         return jsonify({"error": "El nombre es requerido"}), 400
53
54     if usuarios.find_one({"email": data["email"]}):
55         return jsonify({"error": "El email ya está registrado"}), 400
56
57     usuario = usuarios.insert_one(data)
58     return jsonify({"mensaje": "Usuario creado exitosamente", "id":
59 str(usuario.inserted_id)}), 201
60     except Exception as e:
61         return jsonify({"error": str(e)}), 500
62
63 @usuarios_bp.route("/<email>", methods=["PUT"])
64 def update_usuario(email):
65     try:
66         data = request.json
67         if not data:
68             return jsonify({"error": "No se envió la información requerida"}), 400
69         query = {}
70         if "nombre" in data:
71             query["nombre"] = data["nombre"]
72         if "email" in data:
73             query["email"] = data["email"]
74
75         if usuarios.find_one({"email": email}):
76             usuarios.update_one({"email": email}, {"$set": query})
77             return jsonify({"mensaje": "Usuario actualizado exitosamente"}), 200
78         else:
79             return jsonify({"error": "El usuario no existe"}), 404
80     except Exception as e:
81         return jsonify({"error": str(e)}), 500
82
83 @usuarios_bp.route("/<email>", methods=["DELETE"])
84 def delete_usuario(email):
85     try:
86         if usuarios.find_one_and_delete({"email": email}):
87             return jsonify({"mensaje": "Usuario eliminado exitosamente"}), 200
88         else:
89             return jsonify({"error": "El usuario no existe"}), 404
90     except Exception as e:
91         return jsonify({"error": str(e)}), 500
92
93 @usuarios_bp.route("/<email>/eventos", methods=["GET"])
94 def get_eventos(email):
95     try:
96         if not usuarios.find_one({"email": email}):
97             return jsonify({"error": "El email del usuario no existe"}), 400
98
99         eventos_list = eventos.find({"$or": [{"anfitrión": email},
100 {"invitados.email": email}]}).sort("inicio", pymongo.ASCENDING)
101         return jsonify(json.loads(json_util.dumps(eventos_list))), 200
102     except Exception as e:
103         return jsonify({"error": str(e)}), 500
```


2023/src/services/main/eventos.py

```
1 import datetime
2 import json
3 import os
4
5 import pymongo
6 import requests
7 from bson import json_util
8 from bson.objectid import ObjectId
9 from dotenv import load_dotenv
10 from flask import Blueprint, current_app, jsonify, request
11
12 load_dotenv()
13 MONGODB_URL = os.getenv("MONGODB_URL")
14 eventos_bp = Blueprint("eventos", __name__)
15
16 client = pymongo.MongoClient(MONGODB_URL)
17 db = client.examen2023
18 usuarios = db.usuarios
19 eventos = db.eventos
20 contactos = db.contactos
21
22 # Eventos. Representa los eventos de agenda, con los siguientes atributos:
23 # o identificador. Clave o identificador del evento, propia de la base de datos
24 # elegida.
25 # o anfitrión. Dirección de email del usuario anfitrión del evento (el que crea el
26 # evento).
27 # o descripción. Título o descripción breve (hasta 50 caracteres) del evento.
28 # o inicio. Timestamp con la fecha y hora de inicio del evento (en tramos de 15
29 # minutos).
30 # o duración. Duración del evento (en tramos de 15 minutos).
31 # o invitados. Lista de invitados al evento, representado cada uno por su email y
32 # el estado de la invitación
33 # (aceptada/pendiente).
34
35 # • Invitar a un evento a un contacto de un usuario, identificado por su email. El
36 # contacto invitado se incluirá en la
37 # lista de invitados del evento, con la invitación en estado pendiente.
38 # • Aceptar una invitación a un evento. El estado de la invitación pasará a
39 # aceptada.
40 # • Reprogramar un evento ya pasado, indicando cuánto tiempo se desplaza (un
41 # número de días determinado, una
42 # semana, un mes, o un año). Se creará un nuevo evento, con la nueva fecha y el
43 # resto de valores iguales a los del
44 # evento reprogramado.
45 # • Obtener la agenda de un usuario, representada por una lista de eventos, tanto
46 # propios como invitados, por orden
47 # ascendente de inicio
48 # Se valorará que en todas estas operaciones se realicen los controles de errores
49 # necesarios para evitar corromper la
50 # información de la base de datos (fechas/horas inexistentes o incorrectas,
51 # invitar a un evento a alguien que no sea
```

```
41 # contacto de su anfitrión, aceptar un evento al que no se ha sido invitado, etc.)
42
43 @eventos_bp.route("/", methods=["GET"])
44 def get_eventos():
45     try:
46         query = {}
47         if request.args:
48             if "anfitrión" in request.args:
49                 query["anfitrión"] = request.args["anfitrión"]
50             if "descripcion" in request.args:
51                 query["descripcion"] = request.args["descripcion"]
52
53         eventos_list = eventos.find(query)
54         return jsonify(json.loads(json_util.dumps(eventos_list))), 200
55     except Exception as e:
56         return jsonify({"error": str(e)}), 500
57
58 @eventos_bp.route("/<id>", methods=["GET"])
59 def get_evento(id):
60     try:
61         evento = eventos.find({"_id": ObjectId(id)})
62         return jsonify(json.loads(json_util.dumps(evento))), 200
63     except Exception as e:
64         return jsonify({"error": str(e)}), 500
65
66 @eventos_bp.route("/", methods=["POST"]) # lista de invitados [email, estado]
67 def create_evento():
68     try:
69         data = request.json
70         if not data:
71             return jsonify({"error": "No se envió la información requerida"}), 400
72         if "anfitrión" not in data:
73             return jsonify({"error": "El anfitrión es requerido"}), 400
74         if "descripcion" not in data:
75             return jsonify({"error": "La descripción es requerida"}), 400
76         if "inicio" not in data:
77             return jsonify({"error": "El inicio es requerido"}), 400
78         if "duración" not in data:
79             return jsonify({"error": "La duración es requerida"}), 400
80
81         if len(data["descripcion"]) > 50:
82             return jsonify({"error": "La descripción debe tener menos de 50
83 caracteres"}), 400
84
85         if not data["invitados"]:
86             return jsonify({"error": "La lista de invitados es requerida"}), 400
87
88         for invitado in data["invitados"]:
89             if "email" not in invitado:
90                 return jsonify({"error": "El email del invitado es requerido"}),
91                 400
92             if "estado" not in invitado:
```



```
91         return jsonify({"error": "El estado del invitado es requerido"}),
400
92         if not contactos.find_one({"email": data["anfitrión"],
"emailContacto": invitado["email"]}):
93             return jsonify({"error": "El invitado no es un contacto del
anfitrión"}), 400
94         if invitado["estado"] not in ["aceptada", "pendiente"]:
95             return jsonify({"error": "El estado del invitado debe ser aceptada
o pendiente"}), 400
96
97         # comprobar tramos
98         inicio = data["inicio"]
99         inicioDatetime = datetime.datetime.strptime(inicio, "%Y-%m-%dT%H:%M:%S.
%fZ")
100         duracion = int(data["duración"])
101         #inicio es un timestamp en string formato ISO "2023-06-01T00:00:00.000Z"
102         if inicioDatetime.minute % 15 != 0:
103             return jsonify({"error": "El inicio debe ser un tramo de 15
minutos"}), 400
104         if duracion % 15 != 0:
105             return jsonify({"error": "La duración debe ser un tramo de 15
minutos"}), 400
106
107         if not usuarios.find_one({"email": data["anfitrión"]}):
108             return jsonify({"error": "El anfitrión no existe"}), 400
109
110         evento = eventos.insert_one(data)
111         return jsonify({"mensaje": "Evento creado exitosamente", "id":
str(evento.inserted_id)}), 201
112     except Exception as e:
113         return jsonify({"error": str(e)}), 400
114
115 @eventos_bp.route("/<id>", methods=["PUT"])
116 def update_evento(id):
117     try:
118         data = request.json
119         if not data:
120             return jsonify({"error": "No se envió la información requerida"}), 400
121
122         if data["descripcion"]:
123             if len(data["descripcion"]) > 50:
124                 return jsonify({"error": "La descripción debe tener menos de 50
caracteres"}), 400
125
126         if data["invitados"]:
127             for invitado in data["invitados"]:
128                 if "email" not in invitado:
129                     return jsonify({"error": "El email del invitado es
requerido"}), 400
130                 if "estado" not in invitado:
131                     return jsonify({"error": "El estado del invitado es
requerido"}), 400
132                 if not contactos.find_one({"email": data["anfitrión"],
"emailContacto": invitado["email"]}):
```

```
133         return jsonify({"error": "El invitado no es un contacto del
anfitrión"}), 400
134         if invitado["estado"] not in ["aceptada", "pendiente"]:
135             return jsonify({"error": "El estado del invitado debe ser
aceptada o pendiente"}), 400
136
137     # comprobar tramos
138     if data["inicio"]:
139         inicio = data["inicio"]
140         inicioDatetime = datetime.datetime.strptime(inicio, "%Y-%m-%dT%H:%M:
%S.%fZ")
141         if inicioDatetime.minute % 15 != 0:
142             return jsonify({"error": "El inicio debe ser un tramo de 15
minutos"}), 400
143         if data["duracion"]:
144             duracion = int(data["duracion"])
145             if duracion % 15 != 0:
146                 return jsonify({"error": "La duración debe ser un tramo de 15
minutos"}), 400
147
148         if data["anfitrión"]:
149             if not usuarios.find_one({"email": data["anfitrión"]}):
150                 return jsonify({"error": "El anfitrión no existe"}), 400
151
152         if eventos.find_one({"_id": ObjectId(id)}):
153             eventos.update_one({"_id": ObjectId(id)}, {"$set": data})
154             return jsonify({"mensaje": "Evento actualizado exitosamente"}), 200
155         else:
156             return jsonify({"error": "El evento no existe"}), 404
157     except Exception as e:
158         return jsonify({"error": str(e)}), 500
159
160 @eventos_bp.route("/<id>", methods=["DELETE"])
161 def delete_evento(id):
162     try:
163         res = eventos.find_one_and_delete({"_id": ObjectId(id)})
164         if not res:
165             return jsonify({"error": "El evento no existe"}), 400
166         return jsonify({"mensaje": "Evento eliminado exitosamente"}), 200
167     except Exception as e:
168         return jsonify({"error": str(e)}), 500
169
170 @eventos_bp.route("/<id>/invitar", methods=["PUT"])
171 def invitar_evento(id):
172     try:
173         evento = eventos.find_one({"_id": ObjectId(id)})
174         if not evento:
175             return jsonify({"error": "El evento no existe"}), 404
176         anfitrión = evento["anfitrión"]
177         invitados = []
178         data = request.json # data tiene que tener formato [{},{},...]
179         if not data:
180             return jsonify({"error": "No se envió la información requerida"}), 400
```

```
181         if not data["invitados"]:
182             return jsonify({"error": "La lista de invitados es requerida"}), 400
183
184         for invitado in data["invitados"]:
185             if "email" not in invitado:
186                 return jsonify({"error": "El email del invitado es requerido"}),
400
187
188             if not contactos.find_one({"email": anfitrión, "emailContacto":
invitado["email"]})):
189                 return jsonify({"error": "El invitado no es un contacto del
anfitrión"}), 400
190             if not eventos.find_one({"_id": ObjectId(id), "invitados.email":
invitado["email"]})):
191                 invitados.append({"email": invitado["email"], "estado":
"pendiente"})
192
193
194             if not eventos.update_one({"_id": ObjectId(id)}, {"$push": {"invitados":
{"$each": invitados}}}):
195                 return jsonify({"error": "El evento no existe"}), 404
196
197             return jsonify({"mensaje": "Invitación enviada exitosamente"}), 200
198     except Exception as e:
199         return jsonify({"error": str(e)}), 400
200
201 @eventos_bp.route("/<id>/aceptar", methods=["PUT"])
202 def aceptar_evento(id):
203     try:
204         data = request.json
205         if not data:
206             return jsonify({"error": "No se envió la información requerida"}), 400
207         if "email" not in data:
208             return jsonify({"error": "El email es requerido"}), 400
209
210         evento = eventos.find_one({"_id": ObjectId(id)})
211         if not evento:
212             return jsonify({"error": "El evento no existe"}), 404
213
214         if not eventos.find_one({"_id": ObjectId(id), "invitados.email":
data["email"]})):
215             return jsonify({"error": "No se ha sido invitado a este evento"}), 400
216
217         if not eventos.update_one({"_id": ObjectId(id), "invitados.email":
data["email"]}, {"$set": {"invitados.$estado": "aceptada"}}):
218             return jsonify({"error": "El evento no existe"}), 404
219
220         return jsonify({"mensaje": "Invitación aceptada exitosamente"}), 200
221     except Exception as e:
222         return jsonify({"error": str(e)}), 400
223
224 @eventos_bp.route("/<id>/reprogramar", methods=["POST"]) # crea una copia del
evento con la nueva fecha
225 def reprogramar_evento(id):
```

```
226     try:
227         data = request.json
228         if not data:
229             return jsonify({"error": "No se envió la información requerida"}), 400
230         if "dias" not in data:
231             return jsonify({"error": "Los días son requeridos"}), 400
232
233         evento = eventos.find_one({"_id": ObjectId(id)})
234         if not evento:
235             return jsonify({"error": "El evento no existe"}), 404
236         del evento["_id"]
237         if not evento:
238             return jsonify({"error": "El evento no existe"}), 404
239
240         inicio = evento["inicio"]
241         inicioDatetime = datetime.datetime.strptime(inicio, "%Y-%m-%dT%H:%M:%S.%fZ")
242         dias = int(data["dias"])
243         inicioDatetime = inicioDatetime + datetime.timedelta(days=int(dias))
244         inicio = inicioDatetime.strftime("%Y-%m-%dT%H:%M:%S.%fZ")
245         evento["inicio"] = inicio
246
247         if not eventos.insert_one(evento):
248             return jsonify({"error": "Error al reprogramar el evento"}), 400
249
250         return jsonify({"mensaje": "Evento reprogramado exitosamente"}), 201
251     except Exception as e:
252         return jsonify({"error": str(e)}), 400
253
```

2023/src/services/main/contactos.py

```
1 import json
2 import os
3
4 import pymongo
5 import requests
6 from bson import json_util
7 from bson.objectid import ObjectId
8 from dotenv import load_dotenv
9 from flask import Blueprint, current_app, jsonify, request
10
11 load_dotenv()
12 MONGODB_URL = os.getenv("MONGODB_URL")
13 contactos_bp = Blueprint("contactos", __name__)
14
15 client = pymongo.MongoClient(MONGODB_URL)
16 db = client.examen2023
17 usuarios = db.usuarios
18 contactos = db.contactos
19
20 @contactos_bp.route("/", methods=["GET"])
21 def get_contactos(email):
22     try:
23         query = {"email": email}
24         if request.args:
25             if "nombreContacto" in request.args:
26                 query["nombreContacto"] = {"$regex":
request.args["nombreContacto"], "$options": "i"}
27
28         contactos_list = contactos.find(query)
29         return jsonify(json.loads(json_util.dumps(contactos_list))), 200
30     except Exception as e:
31         return jsonify({"error": str(e)}), 500
32
33 @contactos_bp.route("/<emailContacto>", methods=["GET"])
34 def get_contacto(email, emailContacto):
35     try:
36         if not usuarios.find_one({"email": email}):
37             return jsonify({"error": "El email del usuario no existe"}), 400
38         if not usuarios.find_one({"email": emailContacto}):
39             return jsonify({"error": "El email del contacto no existe"}), 400
40
41         contacto = contactos.find({"email": email, "emailContacto": emailContacto})
42         return jsonify(json.loads(json_util.dumps(contacto))), 200
43     except Exception as e:
44         return jsonify({"error": str(e)}), 500
45
46 @contactos_bp.route("/", methods=["POST"])
47 def create_contacto(email):
48     try:
49         query = {"email": email}
```

```
50     data = request.json
51     if not data:
52         return jsonify({"error": "No se envió la información requerida"}), 400
53     if "emailContacto" not in data:
54         return jsonify({"error": "El email del contacto es requerido"}), 400
55     if "nombreContacto" not in data:
56         return jsonify({"error": "El nombre del contacto es requerido"}), 400
57
58     if not usuarios.find_one({"email": email}):
59         return jsonify({"error": "El email del usuario no existe"}), 400
60     if not usuarios.find_one({"email": data["emailContacto"]}):
61         return jsonify({"error": "El email del contacto no existe"}), 400
62
63     query["emailContacto"] = data["emailContacto"]
64     query["nombreContacto"] = data["nombreContacto"]
65
66     if contactos.find_one({"email": email, "emailContacto":
67 query["emailContacto"]}):
68         return jsonify({"error": "El contacto ya está registrado"}), 400
69
70     contacto = contactos.insert_one(query)
71     return jsonify({"mensaje": "Contacto creado exitosamente", "id":
72 str(contacto.inserted_id)}), 201
73
74     except Exception as e:
75         return jsonify({"Los datos no son correctos": str(e)}), 400
76
77 @contactos_bp.route("/<emailContacto>", methods=["DELETE"])
78 def delete_contacto(email, emailContacto):
79     try:
80         query = {"email": email, "emailContacto": emailContacto}
81         res = contactos.find_one_and_delete(query)
82         if not res:
83             return jsonify({"error": "El contacto no existe"}), 400
84             return jsonify({"mensaje": "Contacto eliminado exitosamente"}), 200
85     except Exception as e:
86         return jsonify({"Los datos no son correctos": str(e)}), 400
```