



# Reporte Técnico de Actividades Práctico-Experimentales Nro. 001

## 1. Datos de Identificación del Estudiante y la Práctica

<b>Nombre del estudiante(s)</b>	Gabriel Ricardo Cevallos Medina Ivan Alexander Fernandez Cañar David Alexander Paccha Gallegos
<b>Asignatura</b>	Desarrollo Basado en Plataformas
<b>Ciclo</b>	5 A
<b>Unidad</b>	1
<b>Resultado de aprendizaje de la unidad</b>	Discute cómo los estándares Web influyen en el desarrollo de software, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
<b>Práctica Nro.</b>	001
<b>Título de la Práctica</b>	Implementar un servicio REST con Node.js
<b>Nombre del Docente</b>	Edison Leonardo Coronel Romero
<b>Fecha</b>	Viernes 3 de octubre
<b>Horario</b>	07h30 – 10h30
<b>Lugar</b>	Laboratorio Computación aplicada Laboratorio Desarrollo de Software Laboratorio de redes y Sistemas Operativos Laboratorio Virtual EVA Aula
<b>Tiempo planificado en el Sílabo</b>	3 horas

## 2. Objetivo(s) de la Práctica

- Diseñar y desplegar un microservicio funcional que se comunique con el backend principal del proyecto.
- Implementar comunicación REST entre servicios utilizando un API Gateway o endpoint compartido.
- Documentar la arquitectura de microservicios en el modelo C4 y registrar evidencias del despliegue.

## 3. Materiales, Reactivos, Equipos y Herramientas

- Computador con acceso a Internet.
- Docker / Docker Compose.
- Repositorio del proyecto (GitHub/GitLab).



- Postman / Swagger.
- NestJS.
- Laboratorio de Desarrollo de Software o equipo personal.
- IDE: Visual Studio Code.
- Git.
- Terminal de comandos y contenedores Docker.
- Herramientas de modelado C4 (Draw.io).

## 4. Procedimiento / Metodología Ejecutada

### 4.1 Diseño del microservicio

- **Seleccionar una funcionalidad del proyecto que pueda desacoplarse:**  
Medidas Corporales del Usuario (UserMeasurements)
- **Definir su API interna y endpoints principales:**

#### API Interna

The screenshot shows a code editor interface with the following details:

- EXPLORER:** Shows the project structure under "ATHLETIA".
- routes.py:** The active file in the editor, containing Python code for a Flask application.
- Code Content:** The routes.py file contains two main routes: a GET route for listing user measurements and a POST route for creating new measurements. Both routes require a JWT token. The code uses SQLAlchemy to query a "UserMeasurements" database table. It includes validation logic to ensure the user\_id in the request matches the current user's ID, and it checks for required fields like start date and end date.

```
routes.py 1 ✘
routes.py 1 ×
user_measurements_microservice > app > routes.py > create_user_measurements
54
55     @bp.route('/', methods=['GET'])
56     @jwt_required()
57     def list_user_measurements():
58         current_user = get_jwt_identity()
59         user_id = request.args.get('user_id', type=str)
60         q = UserMeasurements.query
61         if user_id:
62             if str(user_id) != str(current_user):
63                 return jsonify({'error': 'Unauthorized access to other user statistics'}), 403
64             q = q.filter_by(user_id=user_id)
65         items = q.all()
66         return jsonify([s.to_dict() for s in items]), 200
67
68
69     @bp.route('/', methods=['POST'])
70     @jwt_required()
71     def create_user_measurements():
72         current_user = get_jwt_identity()
73         data = request.get_json() or {}
74         required = ['user_id', 'startDate', 'endDate', 'periodType']
75         for f in required:
76             if f not in data:
77                 return jsonify({'error': f'Missing field: {f}'}), 400
78         # Ensure the caller is creating user measurements only for themselves (basic ownership check)
79         if str(data.get('user_id')) != str(current_user):
80             return jsonify({'error': 'Cannot create user measurements for a different user'}), 403
81
82         if not check_user_exists(data['user_id']):
83             return jsonify({'error': 'Referenced user_id not found in user service'}), 400
84
85         sd = parse_date(data['startDate'])
86         ed = parse_date(data['endDate'])
87         if not sd or not ed:
88             return jsonify({'error': 'startDate and endDate must be ISO dates (YYYY-MM-DD)'})
```

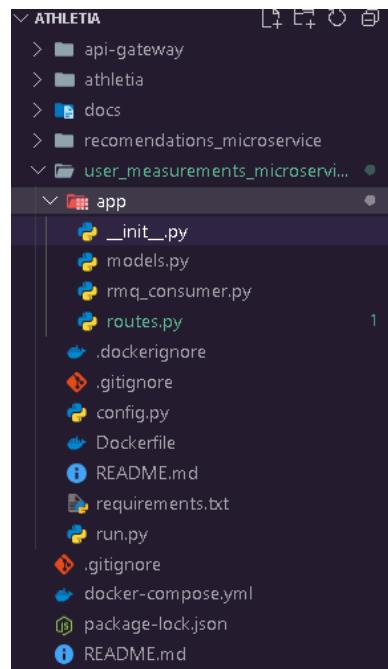


```
121 @bp.route('/<stat_id>', methods=['GET'])
122 @jwt_required()
123 def get_user_measurements(stat_id: str):
124     current_user = get_jwt_identity()
125     stat = UserMeasurements.query.get_or_404(stat_id)
126     if str(stat.user_id) != str(current_user):
127         return jsonify({'error': 'Unauthorized access to this statistics resource'}), 403
128     return jsonify(stat.to_dict(include_metrics=True)), 200
129
130
131 @bp.route('/<stat_id>', methods=['PUT'])
132 @jwt_required()
133 def update_user_measurements(stat_id: str):
134     current_user = get_jwt_identity()
135     stat = UserMeasurements.query.get_or_404(stat_id)
136     if str(stat.user_id) != str(current_user):
137         return jsonify({'error': 'Unauthorized to update this statistics resource'}), 403
138     data = request.get_json() or {}
139
140
141 @bp.route('/<stat_id>', methods=['DELETE'])
142 @jwt_required()
143 def delete_user_measurements(stat_id: str):
144     current_user = get_jwt_identity()
145     stat = UserMeasurements.query.get_or_404(stat_id)
146     if str(stat.user_id) != str(current_user):
147         return jsonify({'error': 'Unauthorized to delete this statistics resource'}), 403
148     try:
149         db.session.delete(stat)
150         db.session.commit()
151     except Exception as e:
152         db.session.rollback()
153     return jsonify({'error': 'Database error', 'details': str(e)}), 500
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
```

## 4.2 Configuración del entorno

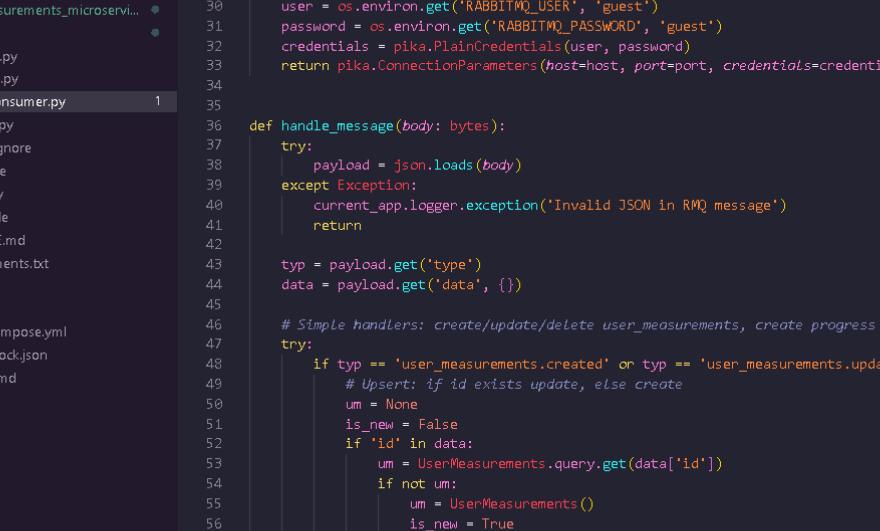
- **Crear una nueva carpeta <nombre>\_microservice y configurarlo como módulo independiente:**

**user\_measurements\_microservice**



- **Implementar la estructura base con servicios y rutas.**

# Servicio RabbitMQ



The screenshot shows the VS Code interface with the following details:

- File Explorer:** On the left, the project structure for "ATHLETIA" is shown. It includes "api-gateway", "athletia", "docs", "recommendations\_microservice", and a folder "user\_measurements\_microservice" which contains "app" (with files: \_\_init\_\_.py, models.py, rmq\_consumer.py), "routes.py", ".dockercfg", ".gitignore", "config.py", "Dockerfile", "README.md", "requirements.txt", "run.py", ".gitignore", "docker-compose.yml", "package-lock.json", and another "README.md".
- Code Editor:** The main area displays the content of "rmq\_consumer.py".

```
user_measurements_microservice > app > rmq_consumer.py > ...  
● Click to add a breakpoint  
27     def _get_rabbit_connection_params():  
28         host = os.environ.get('RABBITMQ_HOST', 'rabbitmq')  
29         port = int(os.environ.get('RABBITMQ_PORT', '5672'))  
30         user = os.environ.get('RABBITMQ_USER', 'guest')  
31         password = os.environ.get('RABBITMQ_PASSWORD', 'guest')  
32         credentials = pika.PlainCredentials(user, password)  
33         return pika.ConnectionParameters(host=host, port=port, credentials=credentials)  
34  
35  
36     def handle_message(body: bytes):  
37         try:  
38             payload = json.loads(body)  
39         except Exception:  
40             current_app.logger.exception('Invalid JSON in RMQ message')  
41             return  
42  
43         typ = payload.get('type')  
44         data = payload.get('data', {})  
45  
46         # Simple handlers: create/update/delete user_measurements, create progress metric  
47         try:  
48             if typ == 'user_measurements.created' or typ == 'user_measurements.updated':  
49                 # Upsert: if id exists update, else create  
50                 um = None  
51                 is_new = False  
52                 if 'id' in data:  
53                     um = UserMeasurements.query.get(data['id'])  
54                     if not um:  
55                         um = UserMeasurements()  
56                         is_new = True  
57                 else:  
58                     um = UserMeasurements()  
59                     is_new = True  
60  
61             if is_new:  
62                 db.session.add(um)  
63             db.session.commit()  
64             return Response(status=201)  
65         except Exception as e:  
66             current_app.logger.error(f'Error processing message: {e}')  
67             return Response(status=500)
```

- Configurar variables de entorno, dependencias y documentación básica (README).

## Variables de Entorno



**UNL**

Universidad  
Nacional  
de Loja

1859

## FEIRNNR - Carrera de Computación

### Dependencias

```

user_measurements_microservice > requirements.txt
1  blinker==1.9.0
2  certifi==2025.10.5
3  charset-normalizer==3.4.4
4  click==8.3.0
5  colorama==0.4.6
6  Flask==3.1.2
7  Flask-JWT-Extended==4.7.1
8  Flask-SQLAlchemy==3.1.1
9  greenlet==3.2.4
10 idna==3.11
11 itsdangerous==2.2.0
12 Jinja2==3.1.6
13 MarkupSafe==3.0.3
14 psycopg2-binary==2.9.11
15 PyJWT==2.10.1
16 python-dotenv==1.1.1
17 requests==2.32.5
18 SQLAlchemy==2.0.44
19 typing_extensions==4.15.0
20 urllib3==2.5.0
21 Werkzeug==3.1.3
22 pika==1.3.1
23

```

### 4.3 Comunicación entre servicios

- **Configurar API Gateway o endpoint del backend principal para enlazar el nuevo servicio.**

### API Gateway

```

api-gateway > src > gateway > gateway.controller.ts > GatewayController > proxyAthletia
7  export class GatewayController {
8    @All('user_measurements/*')
9    async proxyUserMeasurements(@Req() req: Request, @Res() res: Response) {
10      console.log('Proxying request to User Measurements service');
11      const base = process.env.USER_MEASUREMENTS_SERVICE_URL || 'http://user_measurements:5000';
12      const forwardPath = req.url.replace(/^\user_measurements/, '') || '/';
13      const target = `${base}${forwardPath}`;
14
15      console.log(`Proxied request to User Measurements: ${req.method} ${target}`);
16      try {
17        const resp = await firstValueFrom(this.http.request({
18          method: req.method as any,
19          url: target,
20          data: req.body,
21          headers: { Authorization: req.headers['authorization'] || '' },
22          params: req.query,
23          responseType: 'stream',
24        }));
25
26        console.log(`Received response from User Measurements: ${resp.status}`);
27
28        const headers = { ...resp.headers } as Record<string, any>;
29        delete headers['transfer-encoding'];
30        delete headers['connection'];
31        res.status(resp.status).set(headers);
32        if (resp.data && typeof resp.data.pipe === 'function') {
33          resp.data.pipe(res);
34        } else if (Buffer.isBuffer(resp.data)) {
35          res.end(resp.data);
36        } else if (typeof resp.data === 'string') {
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97

```



UNL

Universidad  
Nacional  
de Loja

1859

## FEIRNNR - Carrera de Computación

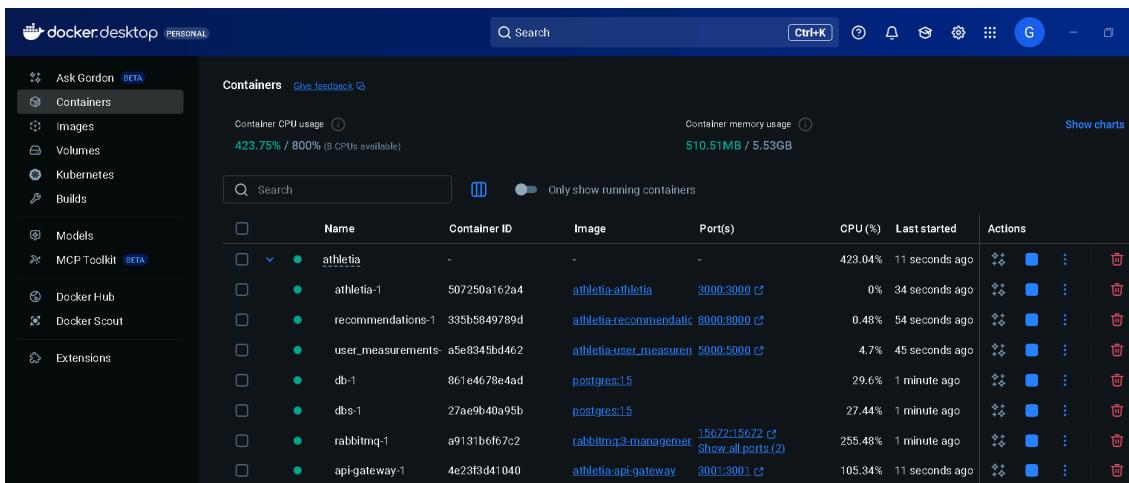
- Probar el intercambio de datos mediante peticiones HTTP.  
Peticiones HTTP mediante Postman

The image displays three separate instances of the Postman application interface, each showing a different collection of APIs:

- AthletIA Collection:** This collection contains several endpoints: POST Login, POST RegisterAccount, POST CompleteProfileSetup, PATCH ChangePassword, and POST Logout. The first screenshot shows a successful GET request to `http://localhost:3001/athletia/users`, returning a JSON response with a total count of 0 items and a success message.
- AthletIA Collection:** This collection contains the same set of endpoints. The second screenshot shows a successful GET request to `http://localhost:3001/athletia/`, returning a simple response: "Hello World, this is the first test of the project: AthletIA!"
- user\_measurements/ Collection:** This collection contains the POST Logout endpoint. The third screenshot shows a successful GET request to `http://localhost:3001/user_measurements/`, returning a JSON response with one measurement entry.

## 4.4 Despliegue y validación

- Ejecutar el microservicio en contenedor Docker o entorno local.

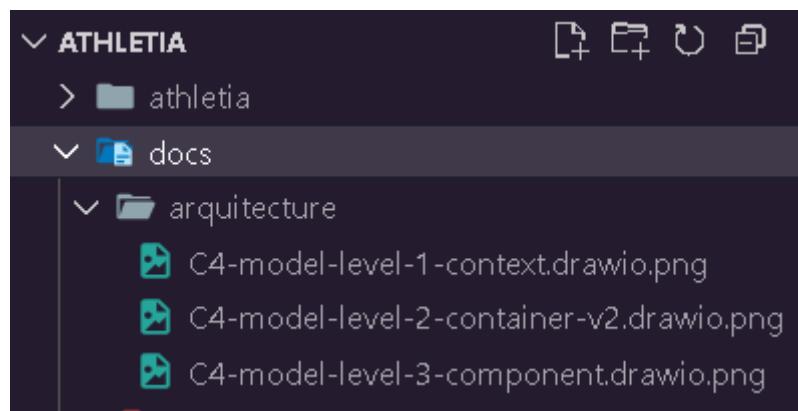


- Validar la conexión y registrar logs de comunicación.

### Core (AthletIA) conectado con servicio RabbitMQ

```
athletia:1 [Nest] 1 - 10/27/2025, 12:32:53 AM LOG [RouterExplorer] Mapped [/workout/exercises, POST] route +1ms
athletia:1 [Nest] 1 - 10/27/2025, 12:32:53 AM LOG [RouterExplorer] Mapped [/workout/exercises, GET] route +0ms
athletia:1 [Nest] 1 - 10/27/2025, 12:32:53 AM LOG [RouterExplorer] Mapped [/workout/exercises/:id, GET] route +2ms
athletia:1 [Nest] 1 - 10/27/2025, 12:32:53 AM LOG [RouterExplorer] Mapped [/workout/exercises/:id, PATCH] route +1ms
athletia:1 [Nest] 1 - 10/27/2025, 12:32:53 AM LOG [RouterExplorer] Mapped [/workout/exercises/:id, DELETE] route +1ms
athletia:1 [Nest] 1 - 10/27/2025, 12:32:53 AM LOG [RabbitmqService] Connecting to RabbitMQ rabbitmq:5672
```

- Documentar resultados en [/docs/architecture/container-diagram-v2.png](#).

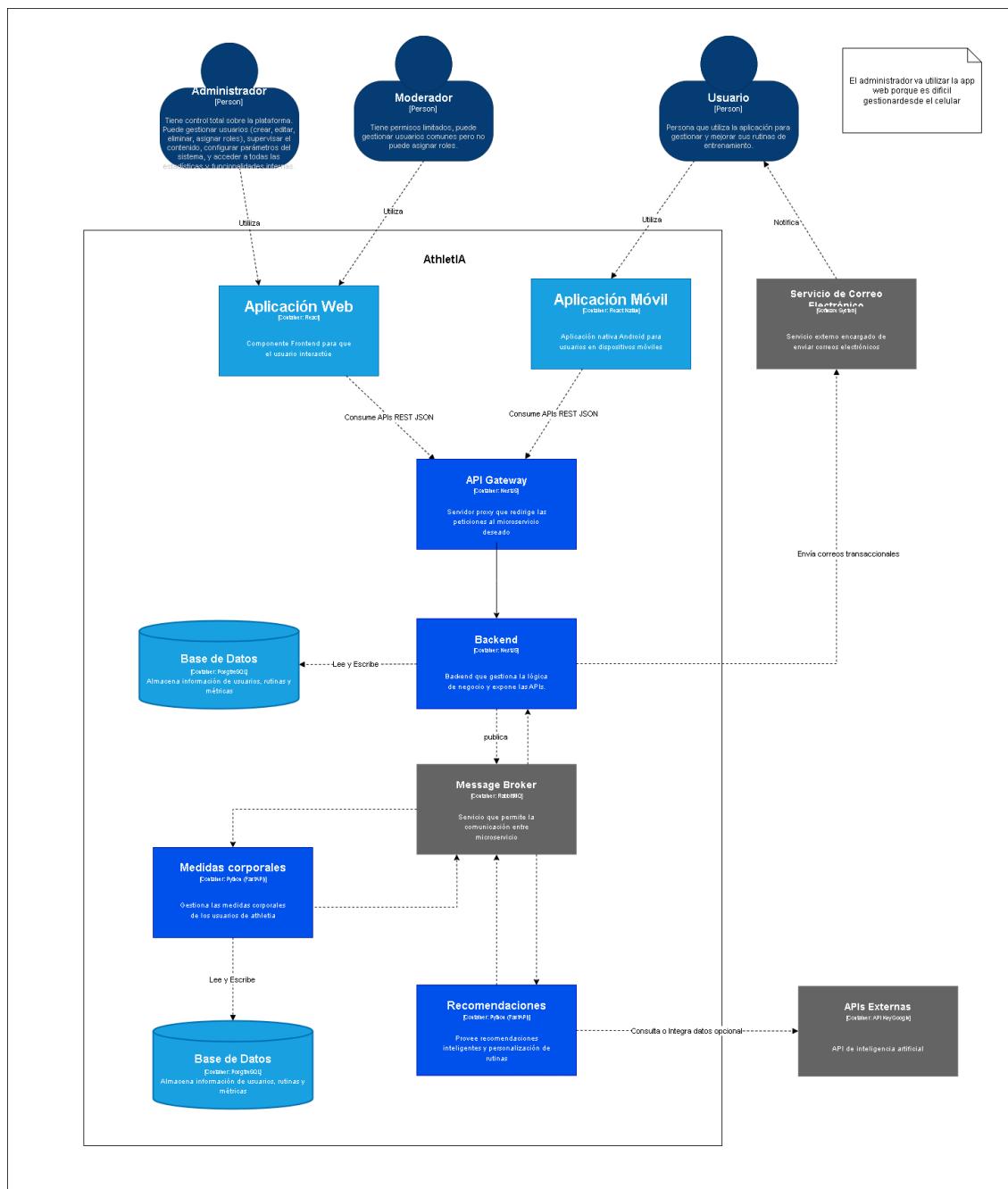


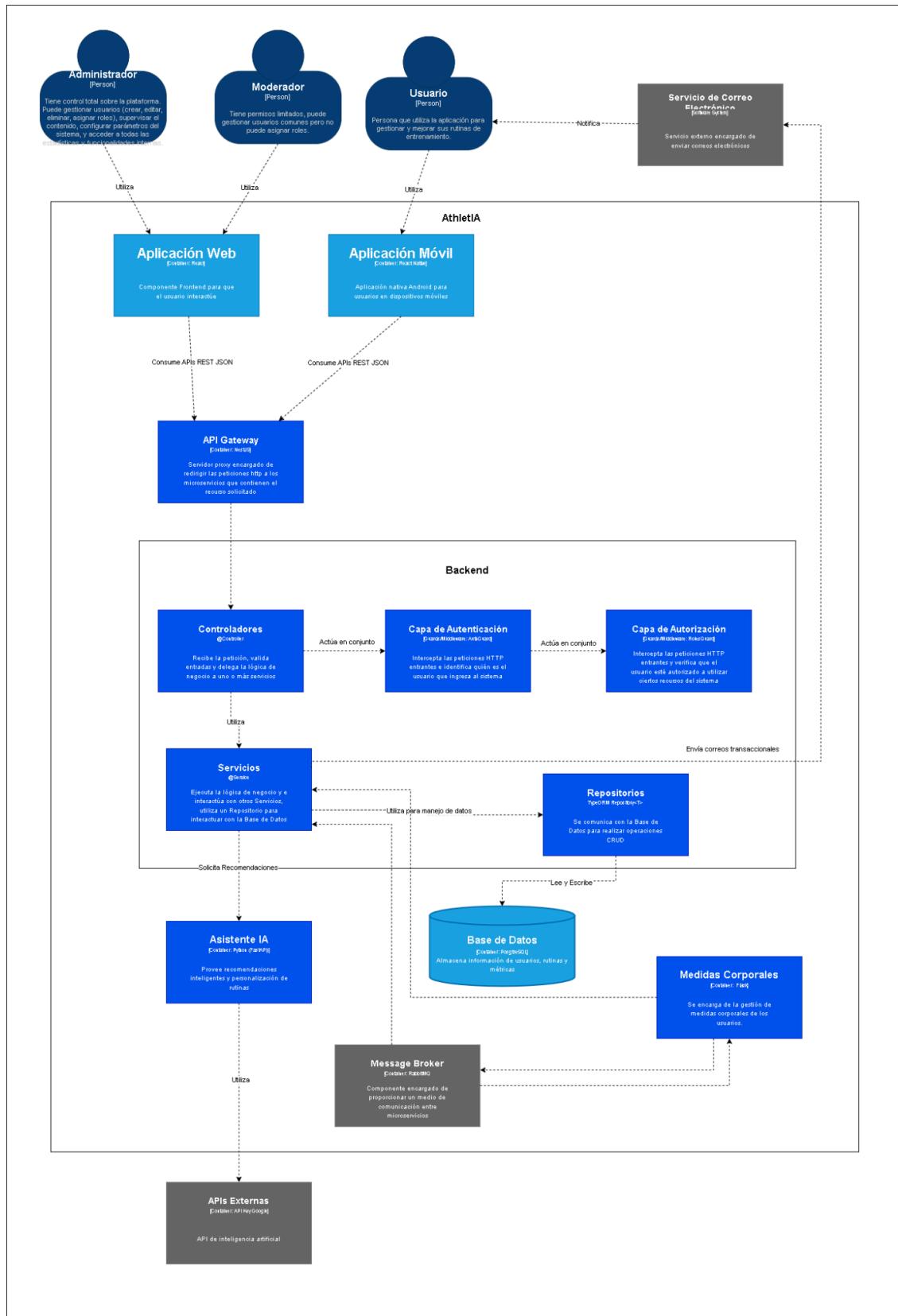


#### 4.5 Registro en modelo C4

- **Actualizar los diagramas Container y Component incorporando el microservicio y el gateway.**
- **Registrar capturas o exportar los diagramas actualizados.**

**MODELO C4 - NIVEL 2 - CONTAINER**



**MODELO C4 - NIVEL 3 - COMPONENT**



## 5. Resultados

- Microservicio desplegado y comunicándose correctamente con el backend principal.
- API Gateway o punto de integración funcional.
- Documentación técnica y diagramas C4 actualizados.
- Logs y capturas de comunicación.

## 6. Preguntas de Control

- **¿Qué diferencia principal existe entre una arquitectura monolítica y una basada en microservicios?**

Una arquitectura monolítica es un sistema todo en uno, es decir que está unido en un solo bloque de código, mientras que la arquitectura basada en microservicios se divide en servicios independientes, es decir que si uno deja de funcionar no le afecta a los demás y pueden seguir funcionando correctamente.

- **¿Qué beneficios aporta el uso de un API Gateway en la integración de servicios?**

El uso de un API Gateway ayuda a mejorar significativamente la seguridad, escalabilidad y la observabilidad, al tiempo que se desacopla los clientes de los servicios del backend, permitiendo que los desarrolladores se concentren en la lógica del negocio.

- **¿Qué riesgos se presentan al no manejar correctamente la comunicación entre microservicios?**

Esto puede ocasionar riesgos como fallas en cascada, latencia de red, inconsistencia de datos, complejidad en la depuración y monitoreo, y vulnerabilidades de seguridad.

- **¿Cómo se refleja la modularidad en los niveles Container y Component del modelo C4?**

Esto se refleja al descomponer el sistema en niveles jerárquicos. El nivel de Contenedor muestra las divisiones físicas o lógicas de alto nivel, mientras que el nivel de Componentes detalla la estructura interna de cada contenedor.

- **¿Qué consideraciones de seguridad se deben mantener al exponer endpoints entre servicios?**

Es muy importante implementar medidas de seguridad como la autenticación y autorización robustas, el cifrado de datos, el uso de APIs seguras y la implementación de una arquitectura de confianza cero. Además es crucial realizar monitoreos continuos y registrar los accesos, mantener los sistemas actualizados y parcheados, también tener un inventario de APIs para identificar los riesgos.



## 7. Conclusiones

- En esta práctica se diseñó y se implementó un microservicio para medidas corporales de los usuarios, integrado con el backend principal de la aplicación mediante API Gateway y RabbitMQ.
- Se validó la comunicación REST con herramientas como Postman y Docker, y se actualizó el documento C4 con diagramas de contenedores.

## 8. Recomendaciones

- Incorporar pruebas automatizadas para detectar errores tempranos.
- Usar monitoreo avanzado en producción para logs en tiempo real.
- Aplicar seguridad adicional para endpoint.
- extender el modelo C4 a niveles más detallados y probar despliegues en la nube para escenarios reales.