

# 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 comuniquen 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).

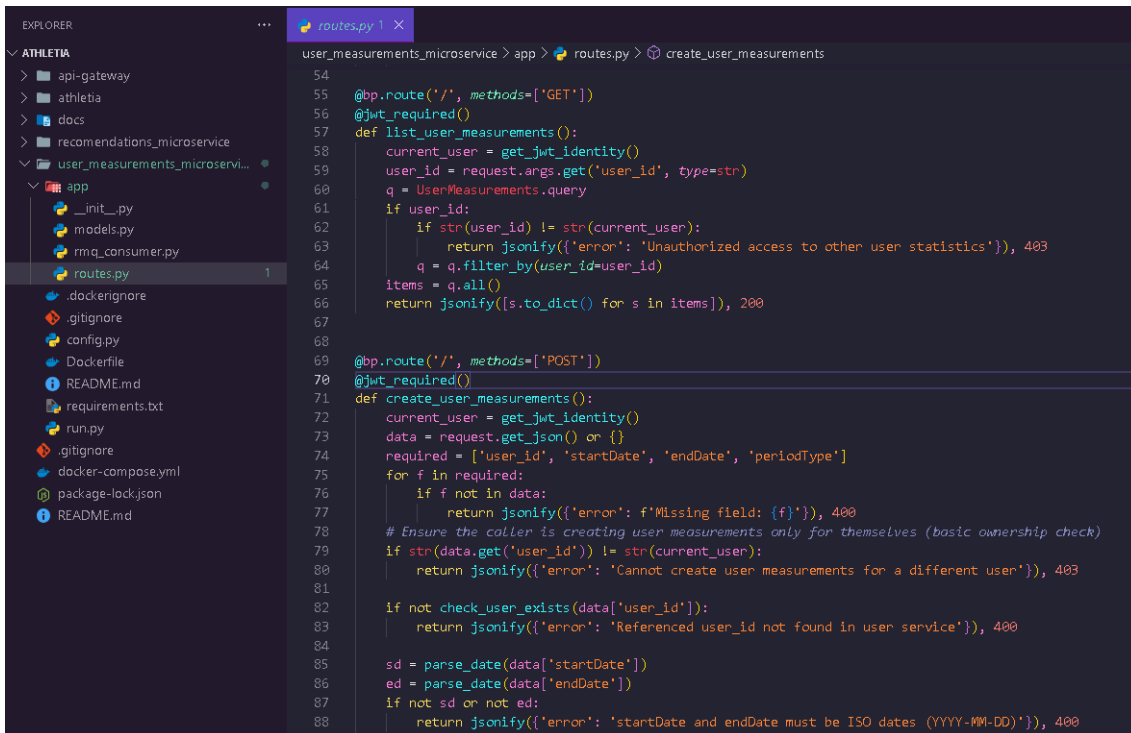
- Postman / Swagger.
- NestJS.
- Laboratorio de Desarrollo de Software o equipo personal.
- IDE: Visual Studio Code.
- Git.
- Terminal de comandos y contenedores Docker.
- Herramientas de modelado C4 (Draw.io).

## 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



```
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)'}), 400
```

```

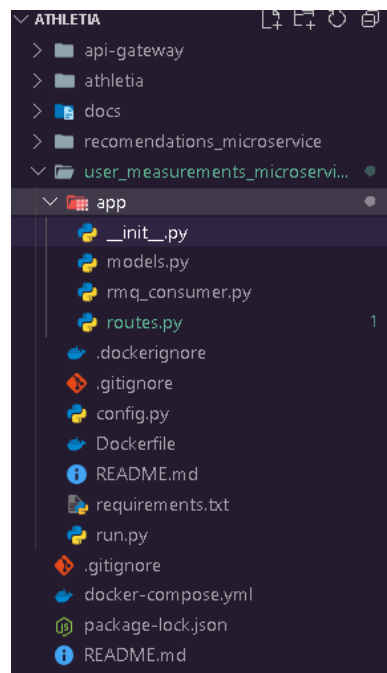
121 @bp.route('/<stat_id>', methods=['GET'])
122 @jwt_required()
123 def get_user_measurements(stat_id: str):
124     current_user = get_jwt_identity()
125     stat = UserMeasurements.query.get_or_404(stat_id)
126     if str(stat.user_id) != str(current_user):
127         return jsonify({'error': 'Unauthorized 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
142 @bp.route('/<stat_id>', methods=['DELETE'])
143 @jwt_required()
144 def delete_user_measurements(stat_id: str):
145     current_user = get_jwt_identity()
146     stat = UserMeasurements.query.get_or_404(stat_id)
147     if str(stat.user_id) != str(current_user):
148         return jsonify({'error': 'Unauthorized to delete this statistics resource'}), 403
149     try:
150         db.session.delete(stat)
151         db.session.commit()
152     except Exception as e:
153         db.session.rollback()
154         return jsonify({'error': 'Database error', 'details': str(e)}), 500
155     return jsonify({'deleted': stat_id}), 200
156

```

## 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 displays a code editor interface. On the left, a file explorer shows the project structure for 'ATHLETHA'. The 'user\_measurements\_microservice' folder is expanded, showing an 'app' subfolder where 'rmq\_consumer.py' is selected. On the right, the code for 'rmq\_consumer.py' is visible. It includes a function to get RabbitMQ connection parameters and a message handler function.

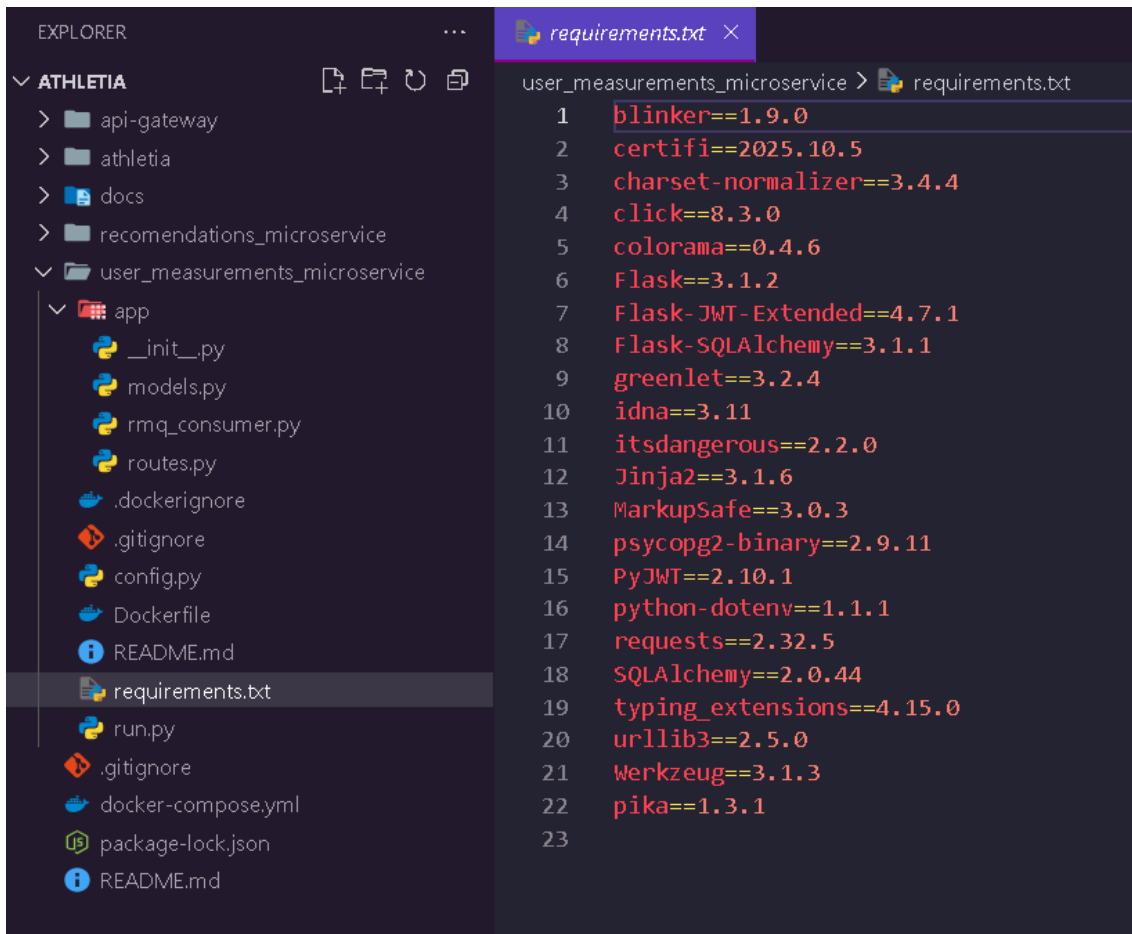
```
27 def _get_rabbit_connection_params():
28     host = os.environ.get('RABBITMQ_HOST', 'rabbitmq')
29     port = int(os.environ.get('RABBITMQ_PORT', '5672'))
30     user = os.environ.get('RABBITMQ_USER', 'guest')
31     password = os.environ.get('RABBITMQ_PASSWORD', 'guest')
32     credentials = pika.PlainCredentials(user, 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
```

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

## Variables de Entorno

[illegible]

## Dependencias



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure under 'ATHLETIA'. The 'user\_measurements\_microservice' folder is expanded, showing an 'app' subfolder with files like \_\_init\_\_.py, models.py, rmq\_consumer.py, routes.py, .dockerignore, .gitignore, config.py, Dockerfile, README.md, requirements.txt (selected), and run.py. On the right, the 'requirements.txt' file is open, listing 23 dependencies:

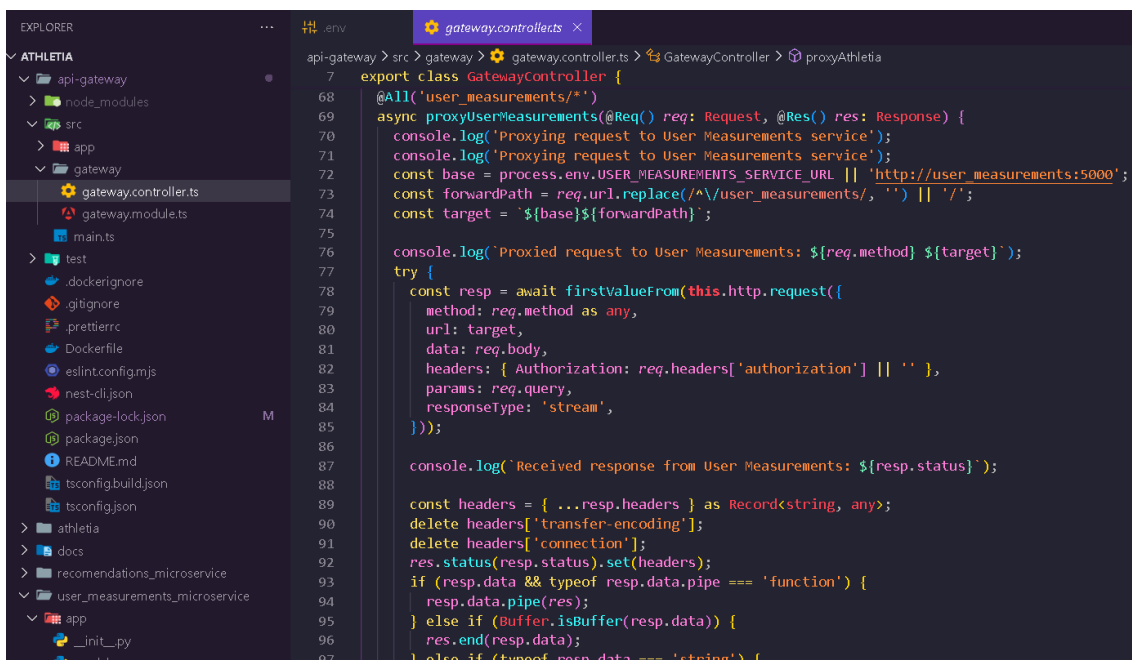
```

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 psycpg2-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



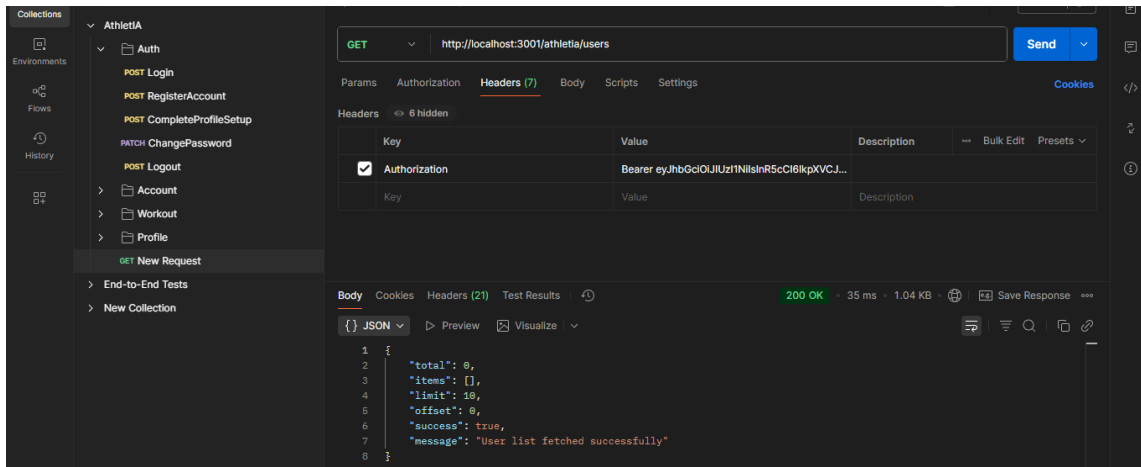
The screenshot shows the VS Code interface with the 'gateway.controller.ts' file open. The file is located in the 'api-gateway' folder under the 'src' directory. The code defines a 'GatewayController' class with a 'proxyAthletia' method. The method uses the '@All' decorator to handle all HTTP methods and proxies requests to the 'user\_measurements' service. It sets the base URL to 'http://user\_measurements:5000' and replaces the path to the 'user\_measurements' endpoint. The response is then proxied back to the client.

```

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     console.log('Proxying request to User Measurements service');
12     const base = process.env.USER_MEASUREMENTS_SERVICE_URL || 'http://user_measurements:5000';
13     const forwardPath = req.url.replace(/^\/user_measurements/, '') || '/';
14     const target = `${base}${forwardPath}`;
15
16     console.log('Proxied request to User Measurements: ${req.method} ${target}');
17     try {
18       const resp = await firstValueFrom(this.http.request({
19         method: req.method as any,
20         url: target,
21         data: req.body,
22         headers: { Authorization: req.headers['authorization'] || '' },
23         params: req.query,
24         responseType: 'stream',
25       }));
26
27       console.log('Received response from User Measurements: ${resp.status}');
28
29       const headers = { ...resp.headers } as Record<string, any>;
30       delete headers['transfer-encoding'];
31       delete headers['connection'];
32       res.status(resp.status).set(headers);
33       if (resp.data && typeof resp.data.pipe === 'function') {
34         resp.data.pipe(res);
35       } else if (Buffer.isBuffer(resp.data)) {
36         res.end(resp.data);
37       } else if (typeof resp.data === 'string') {
38
  
```

- Probar el intercambio de datos mediante peticiones HTTP.

## Peticiones HTTP mediante Postman



GET <http://localhost:3001/athletia/users> Send

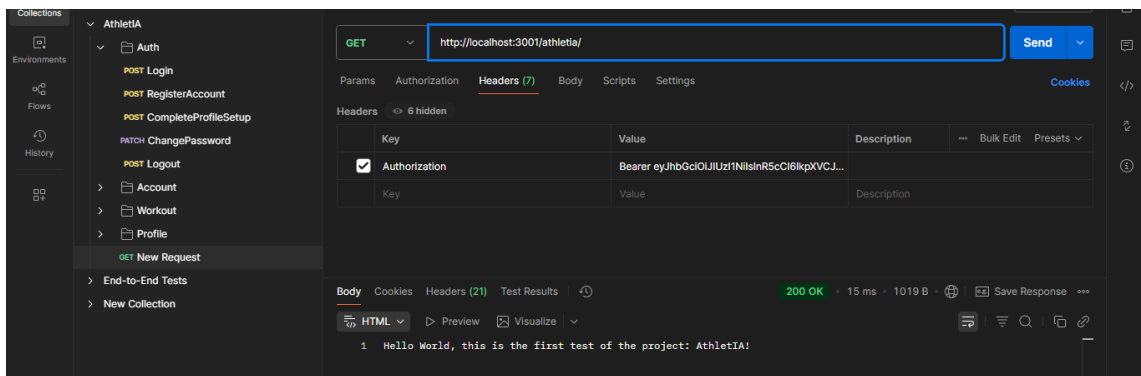
Params Authorization Headers (7) Body Scripts Settings Cookies

Headers 6 hidden

Key	Value	Description
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...	

Body Cookies Headers (21) Test Results 200 OK · 35 ms · 1.04 KB Save Response

```
1 {
2   "total": 0,
3   "items": [],
4   "limit": 10,
5   "offset": 0,
6   "success": true,
7   "message": "User list fetched successfully"
8 }
```



GET <http://localhost:3001/athletia/> Send

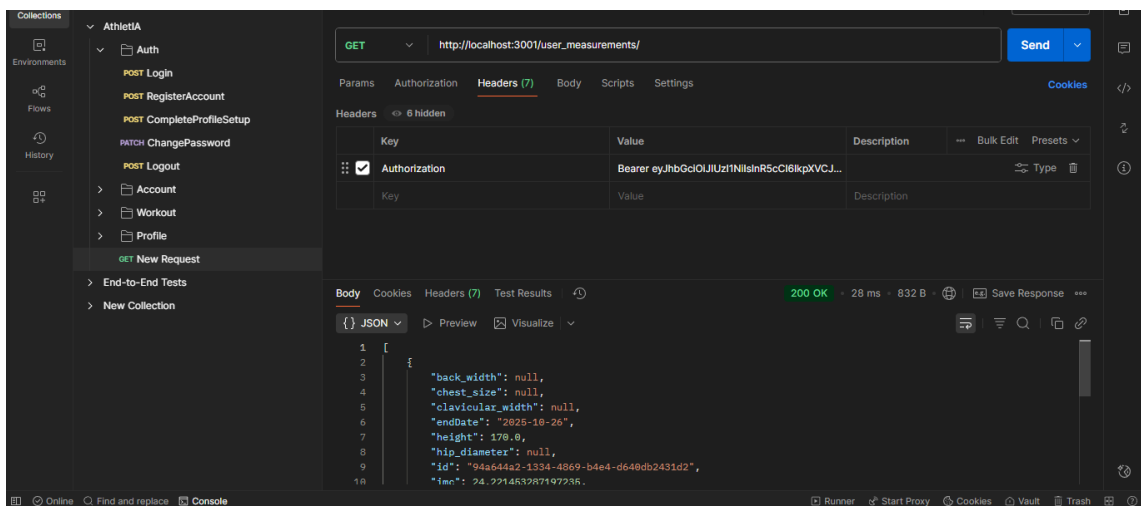
Params Authorization Headers (7) Body Scripts Settings Cookies

Headers 6 hidden

Key	Value	Description
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...	

Body Cookies Headers (21) Test Results 200 OK · 15 ms · 1019 B Save Response

```
1 <html>
2   <body>
3     <div>Hello World, this is the first test of the project: AthletiA!</div>
4   </body>
5 </html>
```



GET <http://localhost:3001/user-measurements/> Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Headers 6 hidden

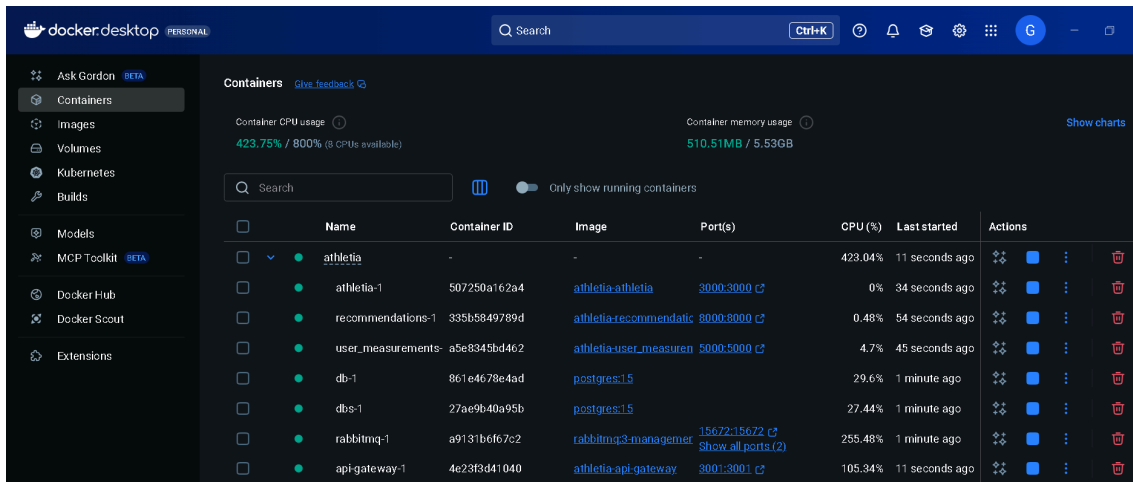
Key	Value	Description
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...	

Body Cookies Headers (7) Test Results 200 OK · 28 ms · 832 B Save Response

```
1 [
2   {
3     "back_width": null,
4     "chest_size": null,
5     "clavicular_width": null,
6     "endDate": "2025-10-26",
7     "height": 178.0,
8     "hip_diameter": null,
9     "id": "94a644a2-1334-4869-b4e4-d640db2431d2",
10    "fmc": "24.221453287197236"
11  }
12 ]
```

## 4.4 Despliegue y validación

- Ejecutar el microservicio en contenedor Docker o entorno local.



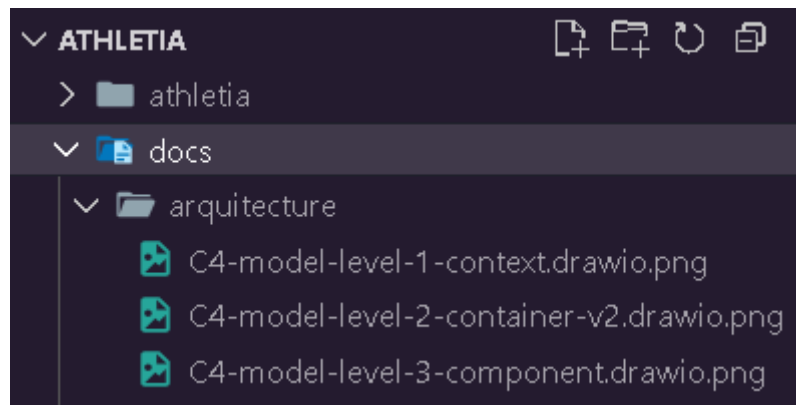
	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	athletia	-	-	-	423.04%	11 seconds ago	
<input type="checkbox"/>	athletia-1	507250a162a4	athletia-athletia	3000:3000	0%	34 seconds ago	
<input type="checkbox"/>	recommendations-1	335b5849789d	athletia-recommendatic	8000:8000	0.48%	54 seconds ago	
<input type="checkbox"/>	user_measurements-	a5e8345bd462	athletia-user-measuremen	5000:5000	4.7%	45 seconds ago	
<input type="checkbox"/>	db-1	861e4678e4ad	postgres:15		29.6%	1 minute ago	
<input type="checkbox"/>	db-1	27ae9b40a95b	postgres:15		27.44%	1 minute ago	
<input type="checkbox"/>	rabbitmq-1	a9131b6f67c2	rabbitmq3-management	15672:15672	255.48%	1 minute ago	
<input type="checkbox"/>	api-gateway-1	4e23f3d41040	athletia-api-gateway	3001:3001	105.34%	11 seconds ago	

- 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
```

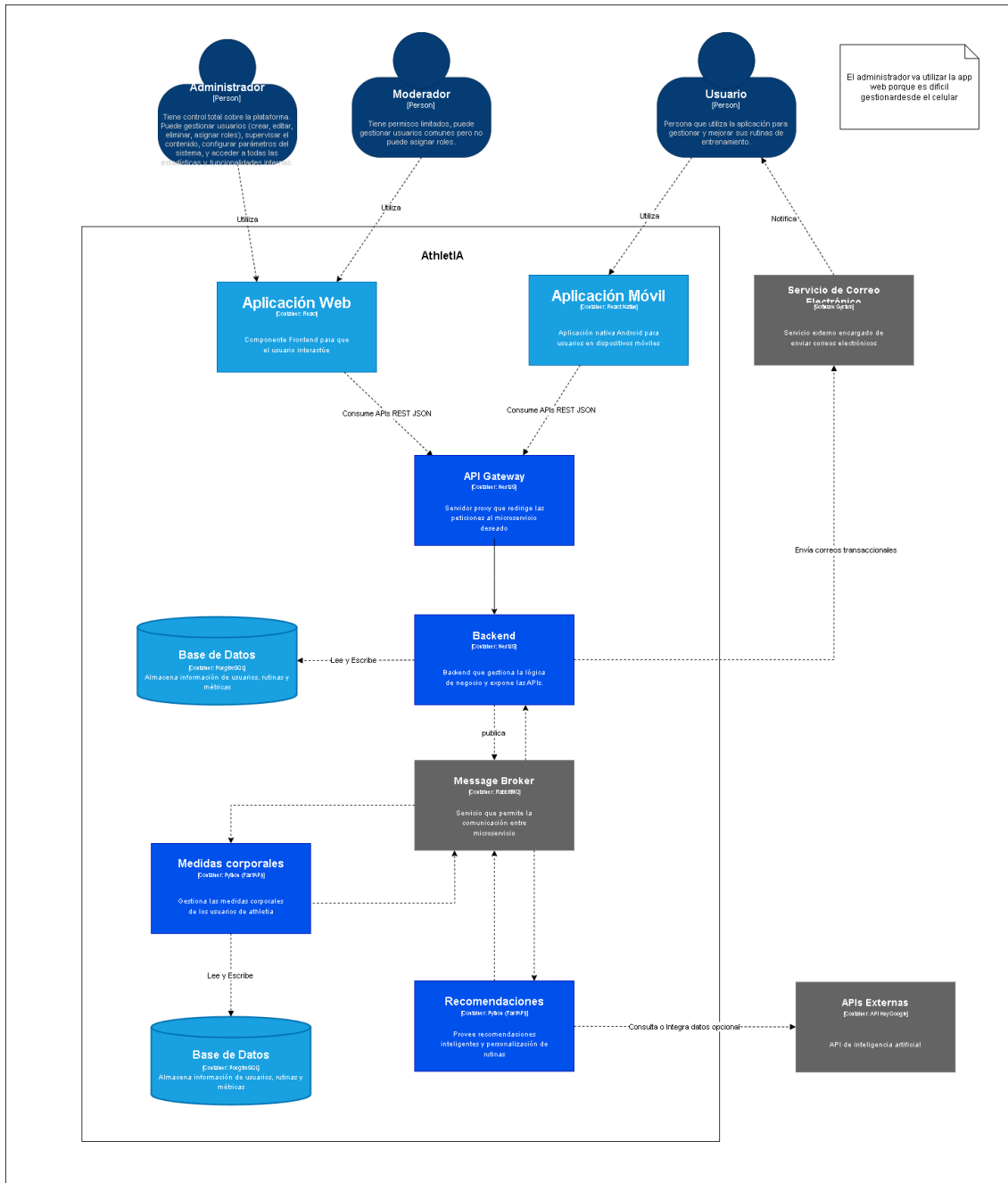
- Documentar resultados en `/docs/architecture/container-diagram-v2.png`.



#### 4.5 Registro en modelo C4

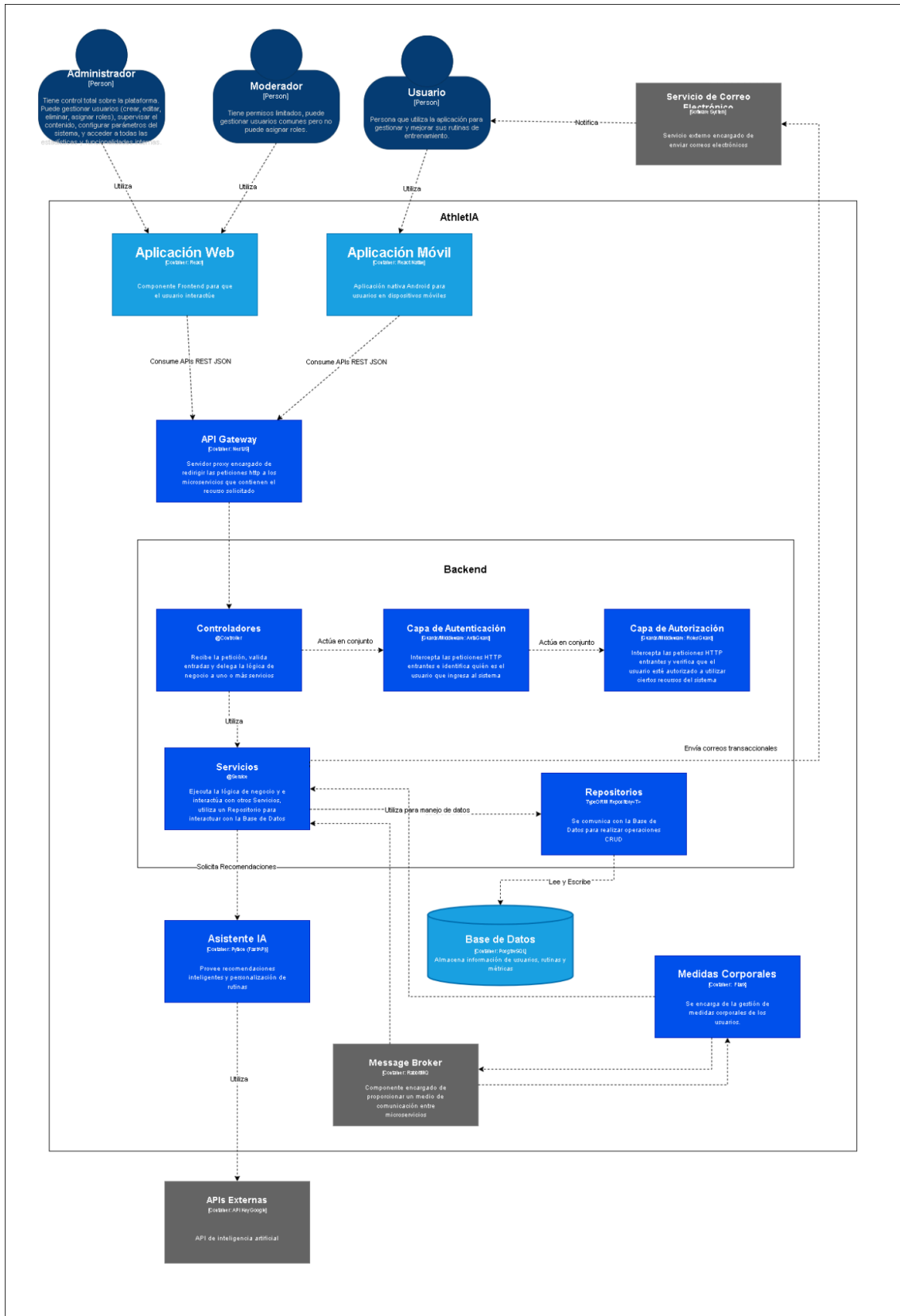
- Actualizar los diagramas Container y Component incorporando el microservicio y el gateway.
- Registrar capturas o exportar los diagramas actualizados.

#### 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.