Informe Técnico

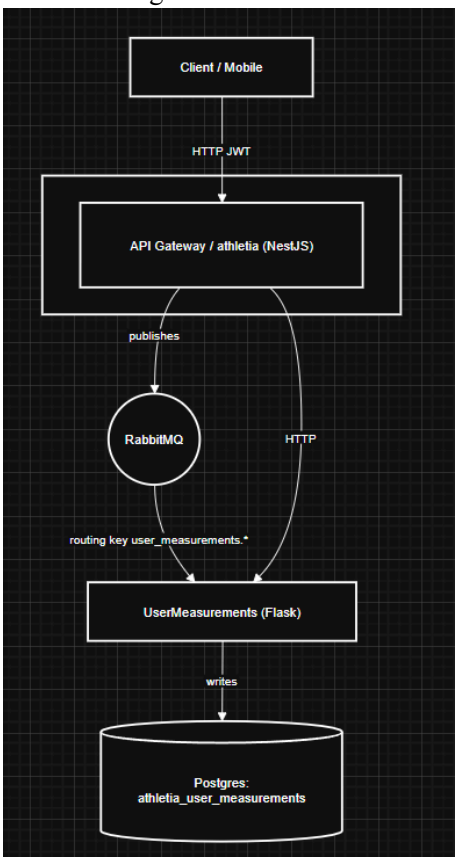
Integrantes: Gabriel Cevallos, David Paccha, Ivan Fernandez

Documentación del desarrollo de microservicios y su integración mediante herramientas colaborativas y APIs.

4.1 Resumen del microservicio creado

- **Nombre**: user_measurements (microservicio)
- **Propósito**: almacenar y exponer mediciones corporales por usuario (peso, altura, medidas de extremidades, etc.) y métricas de progreso; calcular dinámicamente el IMC; recibir eventos desde el servicio principal (athletia) vía RabbitMQ para crear/eliminar recursos por usuario automáticamente.
- **Relación con el sistema** principal: servicio independiente (Flask) que se comunica por:
  - RabbitMQ que consume eventos: user_measurements.created, user_measurements.deleted.
  - HTTP (REST) — API protegida con JWT para CRUD de UserMeasurements y progress_metrics
- Diagrama simple de su estructura (puede usar Mermaid, Draw.io o PlantUML).

Diagrama en Draw.io



4.2 Integración mediante APIs

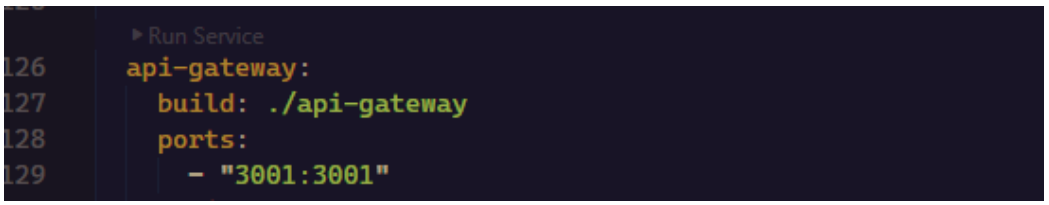
- Descripción del método de comunicación (REST, API Gateway o endpoints compartidos).

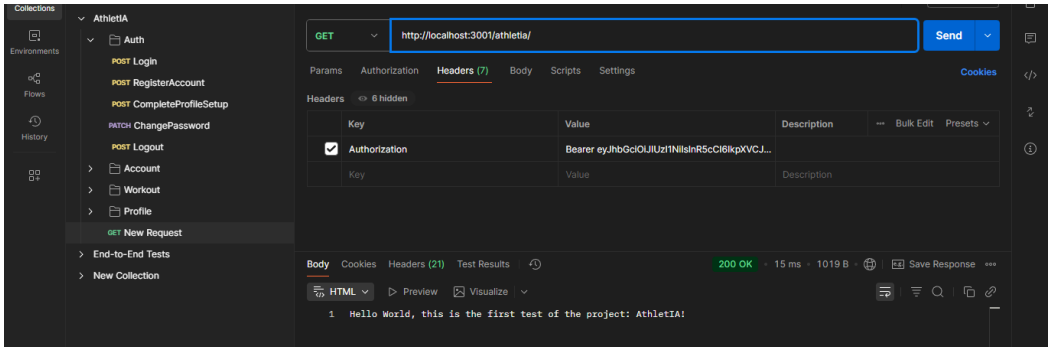
  Método de comunicación:
  - Sincronía: REST HTTP JSON entre clientes, api-gateway, core (AthletIA) y user_measurements, mediante endpoints protegidos con JWT.
  - Asíncrona: RabbitMQ para eventos de dominio, como la creación y eliminación de usuarios; gestión de UserMeasurements, etc.

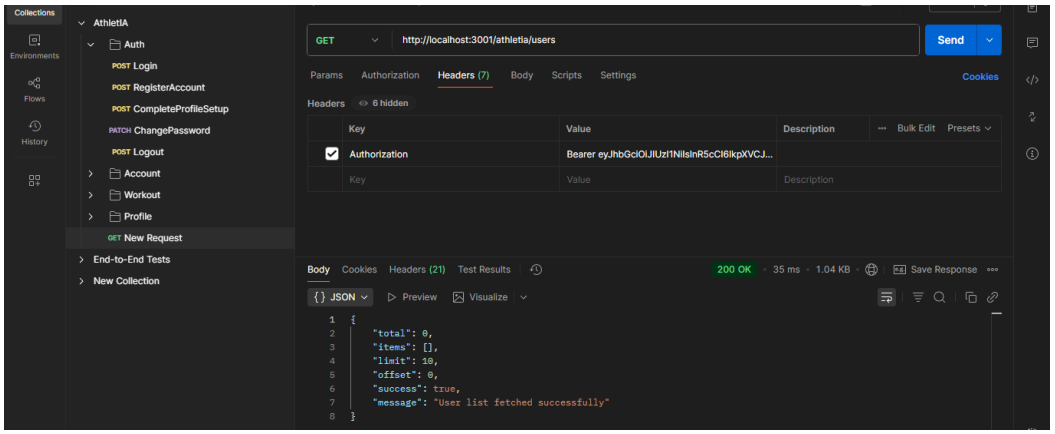
Endpoints principales (user_measurements_microservice/app/routes.py):

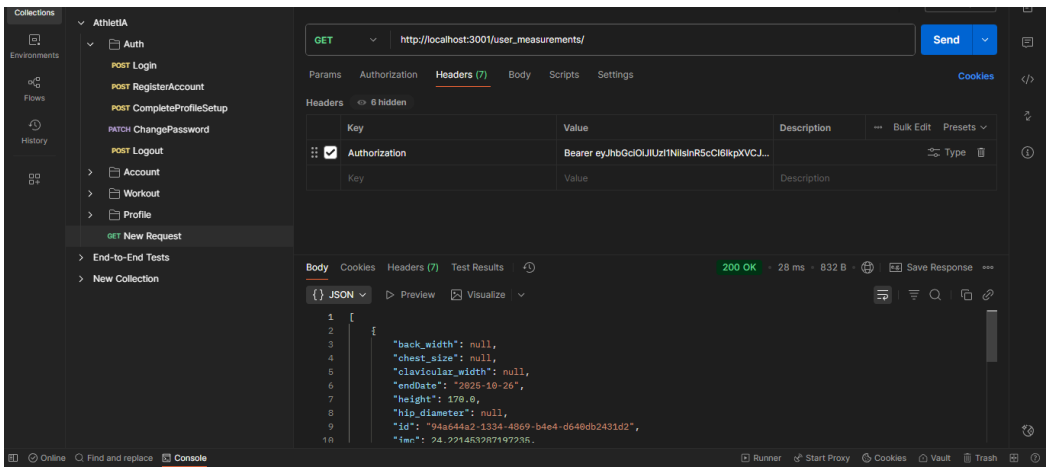
- GET /user_measurements?user_id=<id>: Retorna la lista de mediciones del usuario (requiere JWT).
- POST /user_measurements: Crea una clase UserMeasurements (Body con startDate, endDate, periodType y medidas opcionales).
- GET/PUT/DELETE /user_measurements/<id>: CRUD individual.
- GET/POST /user_measurements/<id>/progress_metrics: Métricas asociadas.

- Ejemplos de llamadas y respuestas (fragmentos Postman o Swagger).









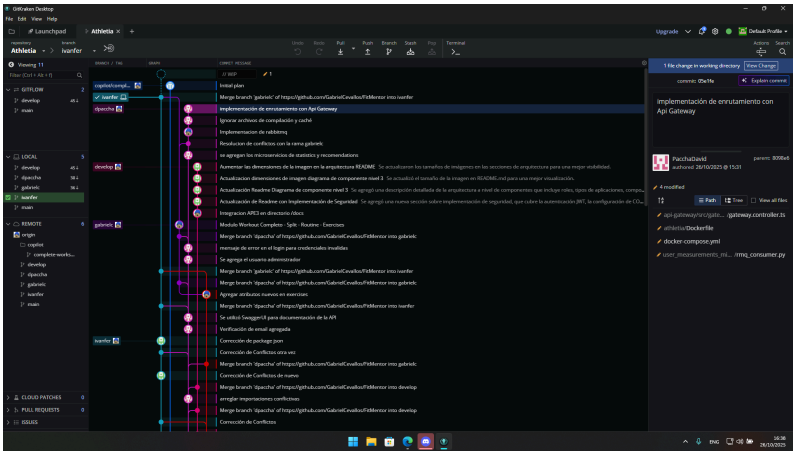
4.3 Herramientas colaborativas utilizadas

- Describir el uso de Trello, Taiga, GitHub Projects, Discord u otras herramientas para la coordinación del equipo.

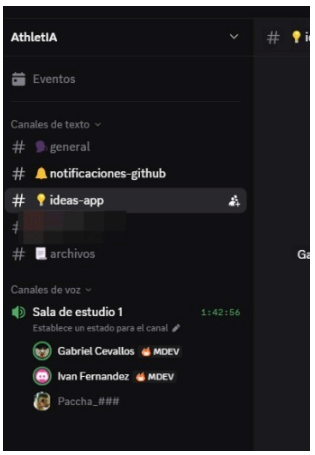
  Se usó Github con Git Flow para el manejo de ramas dentro del repositorio. También se uso Discord para la comunicación y coordinación entre los integrantes del grupo

- Evidenciar cómo estas herramientas contribuyeron a la planificación y seguimiento de tareas.

Uso de Github con gitflow



Uso de Discord



4.4 Buenas prácticas y aprendizajes

- Identificar dos buenas prácticas aplicadas durante el desarrollo.
  1. Separación de responsabilidades: microservicio dedicado a mediciones (UserMeasurements) aislado del core (athletia), que facilita despliegues y escalado independiente.
  2. Comunicación híbrida (REST + mensajes): uso de RabbitMQ para eventos de dominio (crear / borrar recursos asociados) evita llamadas síncronas entre servicios y reduce acoplamiento temporal.

- Reflexión personal sobre cómo la modularización mejora la escalabilidad y mantenimiento del proyecto.

  La modularización permite evolucionar cada componente con la tecnología apropiada (NestJS en core, Flask/Python en user_measurements), pruebas y despliegue individual, facilitando el debugging. Además, la modularización permite escalabilidad selectiva y menor riesgo en despliegues.

5. Preguntas de Control:

5.1 ¿Qué ventajas observas en la integración mediante APIs REST respecto a un monolito tradicional?

- **Desacoplamiento y modularidad**: cada servicio expone contratos claros (recursos, verbos HTTP), facilitando la evolución independiente sin recompilar todo el sistema.
- **Escalabilidad selectiva**: se puede escalar solo los endpoints más demandados en lugar de escalar un binario monolítico completo.
- **Tecnología heterogénea**: cada servicio puede elegir stack óptimo (NestJS para core, Python/Flask para IA), manteniendo interoperabilidad vía HTTP/JSON.
- **Entregas más rápidas:** pipelines CI/CD por servicio reducen el “blast radius” y el tiempo de ciclo.
- **Observabilidad por dominio**: métricas y trazas por API permiten detectar cuellos de botella específicos.

5.2 ¿Cómo aportan las herramientas colaborativas a la gestión técnica de los microservicios?

- **GitHub:** facilita la revisión de código, control de versiones, revisión colaborativa, CI/CD automatizado y despliegue coordinado.
- **Discord:** facilita la comunicación activa entre los miembros del equipo, permitiendo un trabajo síncrono para acciones y eventos importantes, como los cambios en el repositorio.

5.3 ¿Qué riesgos existen al distribuir un sistema en varios microservicios y cómo pueden mitigarse?

- **Complejidad operativa**
- **Latencia y fallos en red**
- **Consistencia de datos**
- **Observabilidad fragmentada**
- **Gobernanza de contratos**
- **Seguridad expandida**

Se mitigan con automatización, gestión de eventos y colas para consistencia, logs centralizados, versionado semántico y contract testing para APIs, además de autenticación segura.

6. Conclusiones:

- El desarrollo del microservicio y su integración con herramientas colaborativas y APIs ha demostrado ser una estrategia robusta para la construcción escalable y mantenible del sistema.
- La implementación de "user_measurements" permite una clara separación de responsabilidades, por ejemplo gestión de mediciones corporales, cálculo dinámico del IMC, etc. Este enfoque modular facilita el desarrollo, despliegue y escalabilidad de los componentes.
- La comunicación entre microservicios se realiza mediante el uso de rabbitmq, lo que proporciona desacoplamiento y resiliencia ante posibles fallos de llamadas a servicios independientes.
- La adopción de discord como herramienta colaborativa ha sido fundamental para la gestión técnica del proyecto, mejorando la comunicación, la gestión del repositorio y pull requests.

7. Recomendaciones:

- Verificar que en docker-compose el servicio rabbitmq está declarado y que el microservicio tenga habilitado rabbitmq.
- Asegurar JWT válido al probar endpoints del microservicio (GET/POST/PUT/DELETE requieren Authorization).
- Configurar los volúmenes en el archivo docker-compose.yaml para poder ejecutar los cambios realizados en el código fuente de una manera inmediata.
- Separar las funcionalidades del sistema en componentes claramente definidos.
- Seleccionar herramientas colaborativas según las necesidades del proyecto.
- Realizar pruebas de componentes, es decir, probar cada microservicios de forma aislada, pero incluyendo sus dependencias directas.