Trabajo Práctico Integrador

Objetivo

Construir una **solución de software** con **API REST**, **un flujo asincrónico** (productor→broker→consumidor), **una integración** (Webhook o gRPC o WebSocket), **seguridad** (OAuth2+JWT), **contenedores** (Docker + Compose), **observabilidad** y **pruebas** (Postman). La **documentación** será un único **README.md** en el repositorio.

Dominio

Elegir **uno** del Anexo (5 dominios con igual complejidad).

Entregables por etapa (y modalidad)

ETAPA 1 — Diseño & Arquitectura

Entrega asincrónica (por repositorio) • Límite: **Dom 26-Oct-2025**
Revisión presencial (defensa corta): Lun 27-Oct-2025

Qué entregar (repo):

- **Diagramas C4**: Context, Container, Component.
- **ADRs**: decisiones clave (REST/gRPC, tipo DB, seguridad).
- **Contrato de API**: **OpenAPI 3.1** (YAML/JSON) + ejemplo de request/response.
- **Modelo de datos** (SQL o NoSQL) + estrategia de **migraciones/seed**.
- **Esqueleto de ejecución**: `docker-compose.yml` inicial con servicios "hello/placeholder" (API, DB, broker) y **README.md** con pasos de ejecución local mínimos.
- **Tag de release** `v1.0.0` y última **commit hash** en el README.
- **Descargar del repositorio el archivo .zip y subir a la UV, en entrega Etapa 1**

ETAPA 2 — Desarrollo, Observabilidad, Pruebas y README

Entrega asincrónica (por repositorio) • Límite: **Miér 26-Nov-2025**
Demo/defensa presencial: Jueves 27-Nov-2025 (15 min/equipo: 10 demo + 5 preguntas)
Recuperatorio: Lun 4-Dic-2025

Qué entregar (repo):

- **Código** de todos los componentes.
- **API REST** operativa (CRUD de 2 entidades + **1 transacción** multi-paso).
- **Seguridad**: **OAuth2 + JWT** (expiración/validación; scopes o roles básicos).
- **Asincronía**: **productor→broker→consumidor** con **RabbitMQ/Kafka/SQS/EventBridge** y un efecto visible (p. ej., notificación/conciliación/proceso diferido).
- **Integración (elegir 1)**:
    - **Webhook** (callback con firma o secreto compartido), **o**
    - **gRPC** (proto + stub; HTTP/2), **o**
    - **WebSocket** (suscripción/stream en tiempo real).
- **Contenerización**: **Docker** por servicio + **Compose** para levantar todo local (API, DB, broker, y—si corresponde—visualizadores).
- **DB**: SQL o NoSQL con **migraciones/seed** reproducibles.
- **Observabilidad**: logs estructurados (JSON) y **dashboard** (Elastic/Kibana **o** OpenTelemetry+Jaeger/Grafana).
- **Pruebas**:
    - **Postman collection** (con variables/ambientes) + ejemplos
- **README.md** (obligatorio) en la raíz del repo (ver checklist abajo).
- **Tag de release** `v1.0.0` y última **commit hash** en el README.
- **Descargar del repositorio el archivo .zip y subir a la UV, , en entrega Etapa 2.**

Consignas (requisitos)

Obligatorios

1. **API REST** sobre HTTP con contrato **OpenAPI 3.1** (paths, schemas, responses, ejemplos).

2. **Seguridad**: **OAuth2 + JWT** (emisión/verificación, expiración; scopes o roles básicos).

3. **Asincronía**: **productor→broker→consumidor** con **RabbitMQ/Kafka/SQS/EventBridge** y caso visible en la demo.

4. **Integración (1)**: **Webhook** (callback firmado o secreto) **o gRPC** (proto/stub) **o WebSocket** (stream/suscripción).

5. **Datos**: **SQL o NoSQL** con **migraciones/seed** + validación de entrada + manejo de errores.

6. **Contenedores**: **Docker** por servicio y **Docker Compose** funcional para levantar todo local.

7. **Observabilidad**: logs JSON y **dashboard** con **latencia p95**, **throughput** y **error rate**.

8. **Pruebas**: **Postman collection** + **una prueba de carga** (JMeter o Postman) con reporte.

9. **README.md** completo y reproducible (ver checklist).

10. **Demo presencial** con defensa técnica.

Optativos (bono)

A) **Despliegue** en nube (ECS/Fargate o Serverless).
B) **BFF/GraphQL** además de REST.
C) **Métricas+trazas** con OpenTelemetry (export a Jaeger/Grafana).
D) **CI** con ejecución automática de colección/linters/tests.

Criterios de aceptación (5) — escala 1..10 (Aprobado ≥ 6)

1. **Funcionalidad End-to-End (2.5 pts)**
    - CRUD de **dos entidades** y **una transacción** multi-paso funcionando.
    - Flujo **asincrónico** operativo (**productor→broker→consumidor**).
    - **Una integración** implementada (**Webhook** con firma/secreto **o gRPC** con proto/stub **o WebSocket** con stream).
2. **Diseño & Arquitectura (2.5 pts)**
    - **C4 (Context/Container/Component) + Despliegue** consistentes con el código.
    - **ADRs** claras (contexto, decisión, consecuencias) para broker, DB, seguridad y estilo de API.
3. **API & Seguridad (1.5 pts)**
    - **OpenAPI 3.1** completo y versionado en el repo.
    - **OAuth2 + JWT** correcto (expiración, validación de firma, scopes/roles aplicados).
4. **Observabilidad & Pruebas (1.5 pts)**
    - Dashboard con **latencia p95**, **throughput** y **error rate**; logs correlacionables.
    - **Postman collection** ejecutable y **prueba de carga** (JMeter/Postman) con resultados adjuntos.
5. **README & Reproducibilidad (2.0 pts)**
    - **README.md** exhaustivo (pasos, variables, seed, cómo probar, cómo observar, cómo demostrar).
    - `docker-compose` levanta todo de punta a punta; tag de release y commit hash incluidos.

Checklist obligatorio para el README.md

- **Proyecto y dominio elegido** (del anexo).
- **Arquitectura en 1 vistazo** (imagen C4 + link a carpeta `/docs`).
- **Requisitos previos** (versiones de Docker/Compose, RAM mínima).
- **Variables de entorno** (`.env.example`) y **secretos** (cómo configurarlos).
- **Cómo levantar local**: comandos `docker compose up` y orden de servicios.
- **Usuarios/credenciales de prueba** y **tokens** (si aplica).
- **Cómo ejecutar pruebas**: colección Postman y prueba de carga (comando + dónde ver el reporte).
- **Cómo observar**: URL o puerto del dashboard (Kibana/Jaeger/Grafana) + qué gráficos mirar (p95, throughput, error rate).
- **Flujo asincrónico**: cómo dispararlo y dónde ver el efecto.
- **Integración**: cómo simular **Webhook/gRPC/WebSocket** en local.
- **Limitaciones y mejoras futuras** (breve).
- **Tag y commit** de la entrega (`v1.0.0`, hash).

Anexo — Dominios propuestos (misma complejidad)

1. **Reserva de Turnos de Salud Ambulatoria**
    - Entidades: **Paciente**, **Profesional** (agenda).
    - Transacción: **Reservar turno** (disponibilidad, bloqueo, confirmación).

- Asincronía: **Recordatorio** + reintentos.
- Integración: **Webhook** de confirmación/cancelación **o WebSocket** tablero en vivo.

2. **Alquiler de Vehículos Urbanos**
   - Entidades: **Vehículo**, **Reserva**.
   - Transacción: **Confirmar reserva** (solapamientos, bloqueo, contrato).
   - Asincronía: **Verificación diferida** (licencia/crédito simulado) y notificación.
   - Integración: **gRPC** a “inventario” externo **o Webhook** de reserva.
3. **Biblioteca Digital (Préstamo de e-books)**
   - Entidades: **Usuario**, **Título**.
   - Transacción: **Préstamo** (cupos, devolución, listas de espera).
   - Asincronía: **Vencimientos/multas**.
   - Integración: **WebSocket** para turnos de espera **o Webhook** a mailing.
4. **Pedidos en Restaurante con Cocina**
   - Entidades: **Pedido**, **Producto**.
   - Transacción: **Confirmar pedido** (stock, total, estados).
   - Asincronía: **Avances de cocina** y notificaciones al cliente.
   - Integración: **WebSocket** tablero de cocina **o gRPC** al servicio de stock.
5. **Reserva de Salas en Co-working**
   - Entidades: **Sala**, **Reserva**.
   - Transacción: **Programar reserva** (conflictos, capacidad, extras).
   - Asincronía: **Recordatorios** y **liberación por no-show**.
   - Integración: **Webhook** con calendario externo **o WebSocket** ocupación en tiempo real.