🧪 Actividad de Laboratorio – UniHelp (RestClient + Gateway)

> **Contexto**: trabajaremos sobre el dominio **UniHelp** (asignaturas, comisiones, horarios) en **tres etapas**. El objetivo es comprender y practicar **comunicación entre microservicios** usando **Spring RestClient** y **exposición unificada** mediante **Spring Cloud Gateway**.

---

1) Etapa I – Microservicio base: `ms-materias`

🎯 Objetivo `ms-materias`

Diseñar y construir el servicio **fuente de verdad** de datos académicos: **asignaturas**, **comisiones** y **slots (horarios)**. Debe ofrecer endpoints para consulta y administración en memoria.

🧠 Fundamentación conceptual `ms-materias`

- **Bounded Context** (DDD): "Materias" define su propio modelo (Asignatura, Comision, Slot) y reglas (nomenclatura de comisiones, turnos M/T/N, slots).
- **Resource-first** (REST): los recursos primarios son **asignaturas**; **comisiones** son sub-recursos.
- **Contratos claros**: el consumidor (ms-horarios) se acopla al **contrato HTTP/JSON**, no a la implementación.

📐 Requerimientos `ms-materias`

1. **Modelo**
   - `Asignatura { codigo, nombre, creditos, cuatrimestre, comisiones[] }`
   - `Comision { id, codigoAsignatura, turno(M/T/N), slots[] }`
   - `Slot { dia(LU..VI), desde(HH:mm), hasta(HH:mm), aula }`
2. **Inicialización**
   - Cargar dataset en memoria con **todas las asignaturas** del plan y **comisiones reales** por nivel (1K*, 2K*, …) mapeadas a turnos.
   - **Slots**: coherentes con M/T/N (horas realistas). Permitimos ajustar luego.
3. **Endpoints (HTTP/JSON)**
   - `GET /api/asignaturas?q=&cuat=&turno=` → lista completa (incluye comisiones)
   - `GET /api/asignaturas/resumen` → lista sin comisiones (solo metadatos)
   - `GET /api/asignaturas/{codigo}` → detalle de una asignatura
   - `POST /api/asignaturas` y `PUT /api/asignaturas/{codigo}` → alta/actualización (in-memory)
   - `DELETE /api/asignaturas/{codigo}`
   - `GET /api/asignaturas/{codigo}/comisiones` y `POST /api/asignaturas/{codigo}/comisiones`
   - `GET /api/comisiones/{id}` (detalle)

4. **Validaciones**
    - Campos obligatorios (`codigo`, `nombre` …), formatos de hora `HH:mm`, `turno ∈ {M,T,N}`.
    - Respuestas `404` para recursos inexistentes.

🧩 Pseudocódigo – inicialización de datos (in-memory)

```
onApplicationStart():
  materias = Map<String, Asignatura>()
  for each nivel in [1..5]:
    materiasNivel = listaAsignaturas(nivel)  // nombre, codigoCorto,
creditos
    comisionesNivel = listaComisionesReales(nivel) // p.ej. [1K1,1K2,...]
    buckets = splitM_T_N(comisionesNivel)   // distribuye en 3 grupos

    for each (nombre,codCorto,cred) in materiasNivel:
      asig = Asignatura(
        codigo = "ISI-" + nivel + codCorto,
        nombre = nombre,
        creditos = cred,
        cuatrimestre = nivel,
        comisiones = []
      )
      asig.comisiones += crearComisiones(buckets.M, turno='M',
slots=plantillaM())
      asig.comisiones += crearComisiones(buckets.T, turno='T',
slots=plantillaT())
      asig.comisiones += crearComisiones(buckets.N, turno='N',
slots=plantillaN())
      materias[asig.codigo] = asig
```

🔬 Pruebas esperadas `ms-materias`

- **Listar**: `GET /api/asignaturas?cuat=1` devuelve asignaturas de nivel 1 (con comisiones).
- **Resumen**: `GET /api/asignaturas/resumen` devuelve solo metadatos (sin `comisiones`).
- **Detalle**: `GET /api/asignaturas/ISI-1AMI` devuelve la asignatura.
- **Alta/Modificación**: crear/actualizar recursos y re-listar para verificar persistencia en memoria.
- **Errores**: `GET /api/asignaturas/XYZ` → `404`.

---

2) Etapa II – Microservicio consumidor: `ms-horarios`

🎯 Objetivo `ms-horarios`

Construir un servicio que **consuma** `ms-materias` vía **RestClient** y **genere un pre-horario sin choques** en base a un conjunto de asignaturas y preferencias de turno.

🧠 Fundamentación conceptual `ms-horarios`

- **Cliente HTTP tipado**: `RestClient` centraliza base URL, manejo de errores, serialización JSON → DTOs.
- **Separación de responsabilidades**: lógica de **planificación** independiente del cliente HTTP (testable).
- **Criterios de elección**: priorizar turno preferido; minimizar días nuevos; evitar solapamientos.

📐 Requerimientos `ms-horarios`

1. **Configuración**
   - `RestClient` con `baseUrl` apuntando a `ms-materias`.
   - Manejo de errores 4xx/5xx (mapear a respuestas claras del API).
2. **DTOs espejo** (solo campos necesarios) de Asignatura, Comision, Slot.
3. **Endpoint principal**
   - `POST /api/horarios/sugerir`
   - Body:

```
{
  "codigos": ["ISI-1AMI","ISI-1AGA"],
  "preferencias": { "turnoPreferido": "M", "maxDias": 4 }
}
```

   - Respuesta: lista de items (materia, comisión, día, desde/hasta), días totales, y observaciones.
4. **Heurística**
   - Evitar choques por día/hora.
   - Favorecer turno preferido.
   - Minimizar apertura de días nuevos.

🧩 Pseudocódigo – consumo + planificación

```
POST /api/horarios/sugerir (codigos[], preferencias):
  materias = []
  for c in codigos:
    a = RestClient.GET("/api/asignaturas/{c}")
    materias.add(a)

  resultado = planificar(materias, preferencias)
  return resultado

planificar(materias, pref):
  elegidos = []
  diasOcupados = Set()
  noAsignadas = []
```

```
    for a in materias:
      candidatas = ordenar(a.comisiones,
        porTurnoPreferido = (c.turno == pref.turnoPreferido ? 0 : 1),
        porCostoDiasNuevos = cantidadDeDiasNuevos(c.slots, diasOcupados))

      asignada = false
      for c in candidatas:
        if not hayChoque(elegidos, c.slots):
          elegidos += mapSlots(a.codigo, a.nombre, c.id, c.slots)
          diasOcupados += dias(c.slots)
          asignada = true
          break
      if not asignada:
        noAsignadas += a.codigo

    return {
      items: elegidos,
      choques: noAsignadas,
      diasTotales: countDistinctDia(elegidos),
      nota: noAsignadas.empty ? "OK" : "Sin combinación para: ..."
    }

  hayChoque(items, slotsC):
    for s in slotsC:
      for it in items:
        if s.dia == it.dia and (s.desde < it.hasta) and (it.desde <
  s.hasta):
          return true
    return false
```

🔬 Pruebas esperadas `ms-horarios`

- **Básica**: con 2–3 asignaturas de 1er nivel y `turnoPreferido=M`, la respuesta **no debe** tener solapamientos y debe preferir comisiones de mañana.
- **Edge**: forzar colisiones (e.g., mismas franjas) y verificar que **al menos una** quede en `choques`.
- **Robustez**: intentar códigos inexistentes ⇒ respuesta clara (400/404) sin stacktrace.

---

3) Etapa III – API Gateway: `api-gateway`

🎯 Objetivo `api-gateway`

Exponer una **fachada única** que enrute peticiones a `ms-materias` y `ms-horarios`, agregando **CORS**, **X-Request-Id**, **logging** y **rate-limit**.

🧠 Fundamentación conceptual `api-gateway`

- **Gateway pattern**: simplifica al cliente (un único host), concentra cross-cutting concerns.
- **Observabilidad**: `X-Request-Id` + logs permiten correlación de trazas entre servicios.
- **Resiliencia**: rate-limit básico protege recursos sensibles (e.g., planificación).

📐 Requerimientos `api-gateway`

1. **Rutas**
   - `/api/asignaturas/**` → `ms-materias`
   - `/api/horarios/**` → `ms-horarios`
2. **Filtros globales** (concepto y finalidad)
   - `RequestIdFilter`: si no hay `X-Request-Id`, generarlo y propagar a la respuesta.
   - `LoggingFilter`: log de entrada/salida con el request id.
   - `InMemoryRateLimitFilter`: ventana 60s, límite N req/min para `/api/horarios/**`.
3. **CORS global**: orígenes locales de desarrollo.
4. **Compatibilidad**: Spring Boot y Spring Cloud versiones compatibles.

🧩 Pseudocódigo – filtros

```
GlobalFilter RequestIdFilter(exchange, chain):
  reqId = exchange.request.headers["X-Request-Id"] ?: uuid()
  exchange.request.headers["X-Request-Id"] = reqId
  exchange.response.headers["X-Request-Id"] = reqId  // antes de chain
  return chain.filter(exchange)

GlobalFilter LoggingFilter(exchange, chain):
  id = exchange.request.headers["X-Request-Id"]
  log("-->", id, method, uri)
  return chain.filter(exchange)
    .then( log("<--", id, response.status, uri) )

GlobalFilter InMemoryRateLimitFilter(exchange, chain):
  if path startsWith /api/horarios/:
    ip = clientIp(exchange)
    (count, start) = bucket[ip]
    if now - start > 60s: reset bucket
    if count+1 > LIMIT: return 429
    else count++
  return chain.filter(exchange)
```

🔬 Pruebas esperadas `api-gateway`

- **Ruteo**: `GET /api/asignaturas/resumen` vía gateway devuelve 200.
- **Trazabilidad**: respuestas contienen `X-Request-Id` (si no se envía, debe generarse).
- **Rate limit**: superar el umbral en `/api/horarios/**` ⇒ `429 Too Many Requests`.
- **CORS**: petición desde `http://localhost:5173` permitida.

✅ Criterios de logro

- **Etapa I**: API de materias estable, dataset cargado, contratos claros.

- **Etapa II**: Consumo con RestClient, heurística sin choques funcionando, manejo de errores.
- **Etapa III**: Gateway enruta y aplica filtros; trazabilidad y límites operativos.

🔧 Tips técnicos

- **Diseño**: separar **modelo** (dominio) de **DTOs** (contrato HTTP). Evita acoplamientos accidentales.
- **RestClient**: centralizar `baseUrl`, `defaultHeaders`, `onStatus` para errores.
- **Validación**: anotar DTOs de request con `@Valid` y reportar errores con `ProblemDetail`.
- **Datos**: comenzar con slots plantillas M/T/N y luego permitir override por JSON.
- **Testing manual**: `curl`/Bruno/RestClient + Swagger UI de cada servicio y a través del gateway.

📦 Entregable del laboratorio

- No se solicita entrega formal. Recomendamos **guardar el repo** con los tres servicios.
- Guardar el ejemplo para utilizarlo en el despliegue a contenedores cuando llegue el momento.