**1. Introducción a la arquitectura por capas**

1. **Domain**: las entidades JPA (Cliente, Producto, Pedido, DetallePedido) y el DTO de filtro (PedidoFiltro), que modelan el dominio.
2. **Repository**: las interfaces que heredan de JpaRepository (y JpaSpecificationExecutor), donde declaramos las consultas.
3. **Service**: la lógica de negocio, orquesta llamadas a repositorios y traduce parámetros en filtros dinámicos.
4. **Controller**: exposición vía REST de los servicios, recibe parámetros HTTP y devuelve JSON.

*Objetivo de la sesión:* que los alumnos entiendan dónde vive cada parte de la lógica de acceso a datos y cómo fluye una petición de extremo a extremo.

---

**2. Consultas Avanzadas con JPQL y @Query**

- **Dónde**: en la capa **repository**, dentro de PedidoRepository.
- **Qué ver**:
    - Las anotaciones @Query("SELECT p FROM Pedido p WHERE p.fecha >= :haceUnMes") para JPQL orientado a entidad.
    - El uso de nativeQuery = true si quisieras SQL puro.
    - Ventaja de definir proyecciones parciales (por ejemplo, solo IDs y fechas).
- **Ejercicio**:

1. Levantar el proyecto y desde Postman invocar GET /api/pedidos/ultimos.

2. Ver en consola SQL generado y en pgAdmin hacer un EXPLAIN ANALYZE de la misma consulta.

---

**3. Métodos derivados de Spring Data**

- **Dónde**: en ProductoRepository.
- **Qué ver**:

- Métodos como findByNombreContaining(...) o findByPrecioGreaterThanOrderByPrecioDesc(...).
- Cómo Spring infiere la consulta a partir del nombre.

- **Ejercicio**:

1. Insertar varios productos en la BD (p. ej. con un data.sql).

2. Probar GET /api/productos?nombre=camisa (si creas un endpoint) o invocar directamente el repositorio en un test.

---

**4. Criteria API para consultas dinámicas**

- **Dónde**: en la capa **service**, en PedidoServiceImpl.buscarDinamico(...).
- **Qué ver**:
  - Cómo se arma un CriteriaBuilder y un CriteriaQuery paso a paso.
  - Uso de Predicate para condicionar la inclusión de filtros (fecha, total, cliente, etc.).
- **Ejercicio**:

1. En el controller llamar POST /api/pedidos/buscar-dinamico con JSON:

2. { "fechaDesde": "2025-05-01", "totalMin": 100.0 }

3. Analizar en consola la query generada.

4. Añadir otro filtro (por ejemplo clienteId), modificar PedidoFiltro y observar cómo basta con agregar un if en el service.

---

**5. Paginación y ordenamiento**

- **Dónde**: en la parte de topProductos(Pageable pageable) de PedidoRepository y en el endpoint /api/pedidos/top-productos.
- **Qué ver**:
  - Cómo construir un PageRequest.of(pagina, tamaño, Sort.by(...)).
  - Cómo leer Page.getTotalPages(), getContent().
- **Ejercicio**:

1. Probar GET /api/pedidos/top-productos?n=3 y luego n=10.
2. Con pgAdmin verificar el límite (LIMIT/OFFSET) en la SQL.

---

**6. Análisis y optimización en PostgreSQL**

- **Dónde**: fuera del código, en la herramienta (pgAdmin o psql).
- **Qué ver**:
    - Creación de índices con CREATE INDEX sobre columnas de filtro (fecha, producto_id).
    - Uso de EXPLAIN ANALYZE SELECT ... para interpretar costos y tiempos.
- **Ejercicio**:

1. Ejecutar la consulta de "últimos pedidos" con y sin índices.
2. Medir la mejora y discutir cuándo vale la pena añadir un índice.

---

**7. Recorrido completo de una petición**

1. **Cliente HTTP** llama al endpoint (por ejemplo /api/pedidos/ultimos).
2. **Controller** delega a PedidoService.
3. **Service** invoca PedidoRepository.ultimosPedidos(...).
4. **Spring Data JPA** traduce JPQL a SQL y ejecuta sobre PostgreSQL.
5. **PostgreSQL** ejecuta la consulta (potencialmente usando índices) y devuelve resultados.
6. **Spring** serializa entidades a JSON y las envía al cliente.

Con este flujo, los estudiantes verán claramente en qué capa residen las consultas avanzadas (repository y service) y cómo impactan en la base de datos.