# Resumen Implementación Filtro de Pedidos por Cliente (idCliente Integer)

## 1. Envío del filtro desde HTML

Se cambió el formulario para enviar el idCliente (de tipo Integer) en lugar de enviar todo el objeto cliente.  
El <select> del cliente envía solo el id del cliente seleccionado.

<select name="idCliente" id="idCliente">  
 <option value="">--Seleccionar Cliente--</option>  
 <option th:each="c : ${clientes}" th:value="${c.id}" th:text="${c.nombre}"></option>  
</select>

## 2. Controlador (PedidoController)

Se recibe el parámetro idCliente como Integer en el método que filtra los pedidos.  
Se mantienen otros filtros opcionales como estado o rango de fechas.

@GetMapping("/api/pedidos/buscar")  
public ResponseEntity<?> buscarPedidos(  
 @RequestParam(required = false) Integer idCliente,  
 @RequestParam(required = false) String estado,  
 @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate fechaInicio,  
 @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate fechaFin  
) {  
 List<Pedido> pedidos = pedidoService.buscarPedidosPorFiltros(idCliente, estado, fechaInicio, fechaFin);  
 return ResponseEntity.ok(Map.of("data", pedidos));  
}

## 3. Servicio (PedidoService)

Se delega al repositorio el filtrado dinámico con los parámetros recibidos.

@Override  
public List<Pedido> buscarPedidosPorFiltros(Integer idCliente, String estado, LocalDate fechaInicio, LocalDate fechaFin) {  
 return pedidoRepository.buscarPorFiltros(idCliente, estado, fechaInicio, fechaFin);  
}

## 4. Repositorio con Criteria API y Join

Para filtrar por cliente se hace un join entre Pedido y Cliente.  
Se compara el id del cliente con el parámetro recibido.  
También se aplican filtros para estado y fecha, si se proporcionan.

@Override  
public List<Pedido> buscarPorFiltros(Integer idCliente, String estado, LocalDate fechaInicio, LocalDate fechaFin) {  
 CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
 CriteriaQuery<Pedido> cq = cb.createQuery(Pedido.class);  
 Root<Pedido> pedido = cq.from(Pedido.class);  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (idCliente != null) {  
 Join<Pedido, Cliente> clienteJoin = pedido.join("cliente");  
 predicates.add(cb.equal(clienteJoin.get("id"), idCliente));  
 }  
  
 if (estado != null && !estado.isEmpty()) {  
 predicates.add(cb.equal(pedido.get("estado"), estado));  
 }  
  
 if (fechaInicio != null && fechaFin != null) {  
 predicates.add(cb.between(pedido.get("fecha"), fechaInicio, fechaFin));  
 }  
  
 cq.where(predicates.toArray(new Predicate[0]));  
 return entityManager.createQuery(cq).getResultList();  
}

## Conclusión

Con este enfoque:  
- Se mejora el rendimiento al filtrar por un simple id en lugar de enviar y comparar objetos completos.  
- La búsqueda es más flexible y permite combinar filtros (cliente, estado, fecha).  
- Se utiliza el CriteriaBuilder para construir consultas dinámicas, manteniendo el código limpio y mantenible.