**Customer Search & Management**

1. **Customer search** by criteria (name, customer id, PhN, subscription id), the result must be sortable, and can be filtered.

**Imagine:** that there is a search panel I front end like this:

|  |  |
| --- | --- |
| Search customer |  |
| Name: | Customer id: |
| Subscription id: | Phone number: |

One possible solution: implement a SQL query that search in client table using these 4 criteria fields.

May be like this:

SELECT \* FROM CLIENT

WHERE id like “%:CustomerID%”

AND name like “%:name%”

AND subscription\_id like “%:SubscriptionID%”

AND phone\_number like “%:phn%”

Order by “:orderColumn”

The result will be a list of clients that matches these criteria

But this solution: is not professional, instead of writing SQL queries for dynamic search (which already will have threats like SQL injection) you can use **JPA Specification** interface.

**Specification<T> interface:**

is part of the Criteria API abstraction. It allows you to:

* Dynamically build complex SQL WHERE clauses
* Combine filters at runtime
* Make your query logic reusable and modular
* Avoid hardcoded method names like findByNameAndPhoneNumber...

It’s designed for advanced filtering/searching use cases — perfect when your query parameters are optional or dynamic.

**When Should I Use It?**

Use Specification<T> when:

* You have optional search fields
* You need composable filtering logic
* You want to clean up large repositories with many findBy... methods
* You want to support complex filtering, like ranges or nested joins

**Basic Structure:**

public interface Specification<T> {

Predicate toPredicate(Root<T> root, CriteriaQuery<?> query, CriteriaBuilder criteriaBuilder);

}

**Components:**

**Root:** to access entity fields and joins

**Query:** to manipulate query behavior (e.g., fetch joins, distinct)

**CriteriaBuilder:** to build predicates (where, like, equal)

**advanced example** that showcases how to use all components of a JPA. You will find it in this path:

Desktop/brain storming/practice/AdvancedSearchDemo

1. **Customer Management**

Apply CRUD operations on the customer entity + record all the changes (audit): who (user) did what, at the time (when)

How?

These actions will impact on 2 tables or more; 1- client,

And audit\_actions and its related tables like audit\_calls and audit\_action\_desc.

Let’s imagine the flow:

The user needs to do an action (add & update client, add/update beneficiary) each action of these actions must have a code or (id) and description, and already this is exists in the audit\_action\_desc.

For each call: the user can do many actions, so there is a table named calls, where we store or save each call logs, this table has one-to-many relationship with the audit\_actions table, where we store the actions related to this call. Till now all things make sense. The client starts the call 🡪 the customer interacts with him 🡪 starts a session (call instance) on the system 🡪 each action the user does, must be stored in this call instance.

And performing this action.

There are some details I wanna know about how the call will be stored in the DB? How is it done manually by the user? Or

Automatically by the system, if automatically, how? how can we calculate the call duration, how can we calculate the exact call-start-time, and the mobile number?

Maybe all these cases will be handled via the IVR system (if it exists) where the user can start a call using a button (start call) 🡪 enters the client phn and id, once the call ends the user hit (end call) button and the system automatically can calculate the call duration.

Another problem: if the client wants to do more than one action in one call, example: update information, and delete beneficiary? these are 2 different services will be handled using 2 different methods. How can we store these actions in the same call log? by making the call global. But how ?

One possible solution: Create one Service Class called (ClientManagementService) and add all the functions that related to this domain (client management) in this class, also add start and stop call functions.

In startCall() function, instantiate a call object and use it in all functions, and in endCall() delete this instance.

Using this approach you will be sure that all the actions will be stored at the same call for each call reuest.

Another solution: Creatin session-scoped bean for the call:

@Component

@Scope(value = WebApplicationContext.SCOPE\_SESSION)

public class CallContext {

private Long currentCallId;

public void start(Long callId) { this.currentCallId = callId; }

public void stop() { this.currentCallId = null; }

public Long getCurrentCallId() { return currentCallId; }

}

Each call-related method reads from this context holder.

How to implement the client search?

Using JPA specification.