
# Caso práctico
# HICloud

## Presentación del caso práctico

**HICloud** (Hospital information cloud) es un proveedor de servicios a hospitales que ofrece sus productos como SaaS (Software as a Service). Sus soluciones son multitenancy, es decir, dan soporte a varias organizaciones pero mantienen sus datos totalmente aislados. Los servicios ofrecidos se presentan como facilidades, de modo que los clientes pueden contratar las facilidades que deseen de forma independiente, aunque algunas de ellas sólo se pueden contratar si se dispone de otra, debido a que tienen alguna dependencia.

Los equipos de trabajo están organizados en función de las facilidades ofrecidas por HICloud, las 8 principales son:

1) Demográfica: gestión de usuarios (pacientes y profesionales)
2) Imágenes: gestión de imágenes clínicas (PACS)
3) Citas: gestión de citas clínicas
4) Hospitalización: gestión de altas y bajas hospitalarias
5) Farmacia: gestión de medicamentos
6) Recursos: gestión de recursos fungibles e instrumental clínico
7) Historia clínica: gestión de información clínica
8) Actividades: gestión de turnos, reparto de tareas asistenciales

Todos los desarrollos de la empresa utilizan JPA [1,2] para minimizar las dependencias de los servicios de persistencia utilizados. Todas las APIs ofrecidas y usadas por las facilidades para su interoperatividad (cuando sea necesaria), se desarrollan conforme al estándar FHIR [3,4], con la intención de facilitar la integración de las soluciones propias con las de terceros.

## Procedimiento

A cada equipo de trabajo se le asignará una de las facilidades identificadas. En su solución se utilizará, **obligatoriamente**, la API JPA. El uso de esta API debe verse claramente reflejado en el modelo, que debe especificar cómo se utiliza la misma en cada una de las entidades participantes. Así en la representación UML deben incluirse las interfaces, clases, objetos, etc… de JPA que se necesiten y deben relacionarse con el resto de componentes que se diseñen.

Previa a la toma de requisitos funcionales, se identifican una serie de requisitos no funcionales que deben considerar todos y cada uno de los equipos:

a) Todas las soluciones deben utilizar JPA para la persistencia y para la implementación deben utilizar eclipseLink [5]como proveedor. Esta reutilización debe verse claramente reflejada en el modelo presentado.

b) El diseño de su solución debe ser conforme a un cliente FHIR allí donde sea necesario utilizar algún servicio de otra facilidad. Se utilizará la implementación HAPIFHIR [6] allí donde sea posible. Esta reutilización tiene que verse claramente reflejada en el modelo presentado.

c) El diseño de su solución debe ser conforme a un servidor FHIR allí donde sea necesario ofrecer un servicio a otras facilidades. Se utilizará la implementación HAPIFHIR siempre que sea posible. Esta reutilización tiene que verse claramente reflejada en el modelo presentado.

d) Todas las soluciones deben incorporar los aspectos de vista como una GUI al menos para un rol de usuario y otro de administrador.

e) Deben plantearse los aspectos de configuración de las soluciones desarrolladas.

## Primera tarea de los equipos de trabajo

La primera tarea de cada equipo de trabajo es hacer un estudio de la situación y del estado de la técnica; revisar las soluciones existentes en el dominio del aspecto asignado. Deberá además revisar los fundamentos de JPA y FHIR, así como las respectivas implementaciones: eclipselink y hapifhir. Se recomienda hacer un reparto de tareas y una posterior puesta en común de los aspectos analizados.

Referencias

[1] https://javaee.github.io/javaee-spec/javadocs/javax/persistence/package-summary.html

[2] https://docs.oracle.com/javaee/6/tutorial/doc/bnbpy.html

[3] https://www.hl7.org/fhir/

[4] https://fhir-drills.github.io/index.html

[5] https://www.eclipse.org/eclipselink/

[6] https://hapifhir.io/