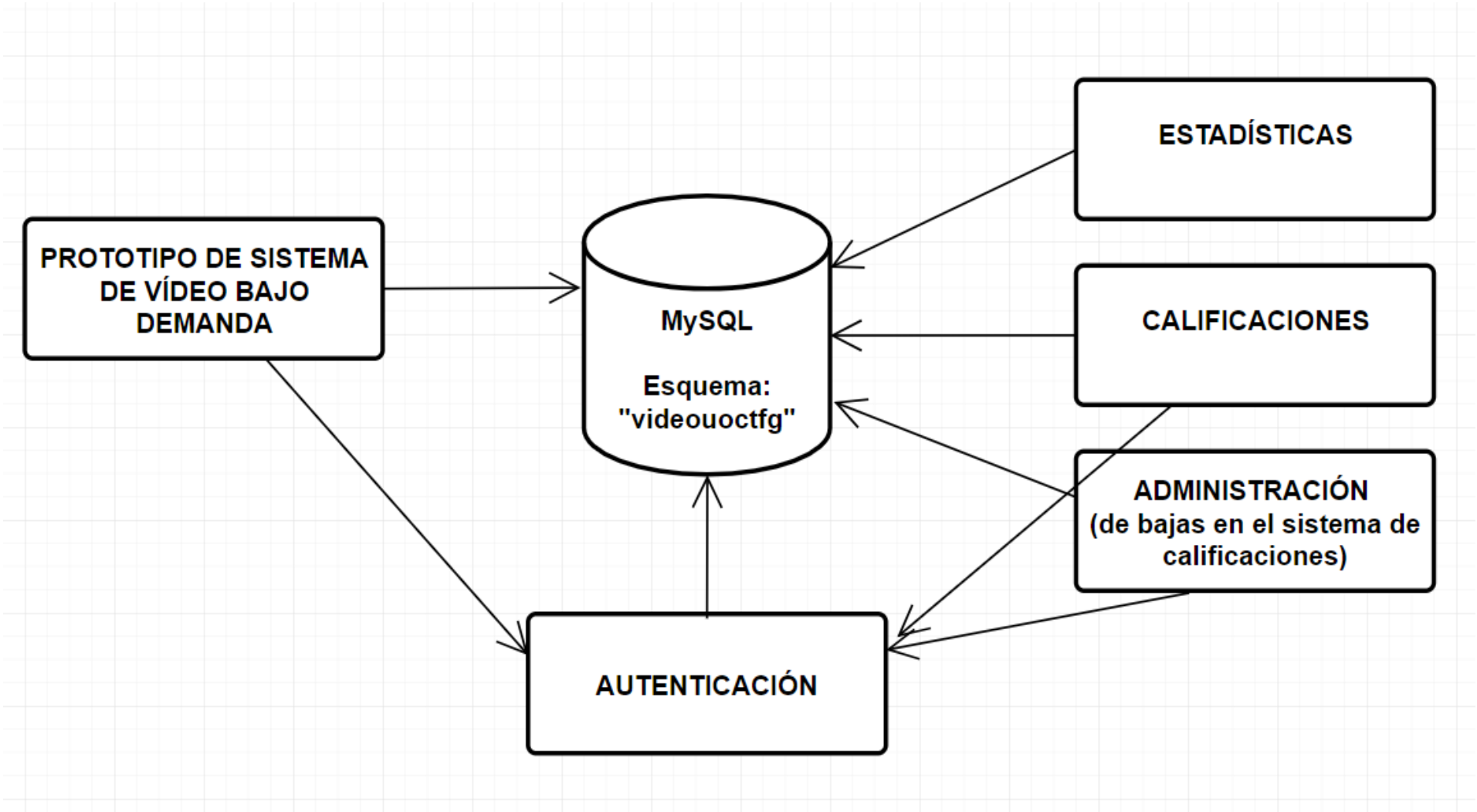
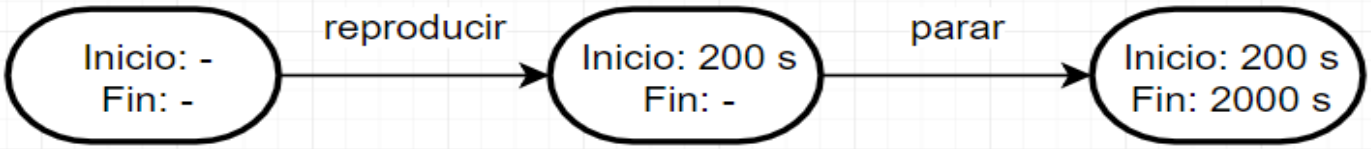
VÍDEOUOC - MÓDULOS



Prototipo

Duración de capítulo: 7200 s

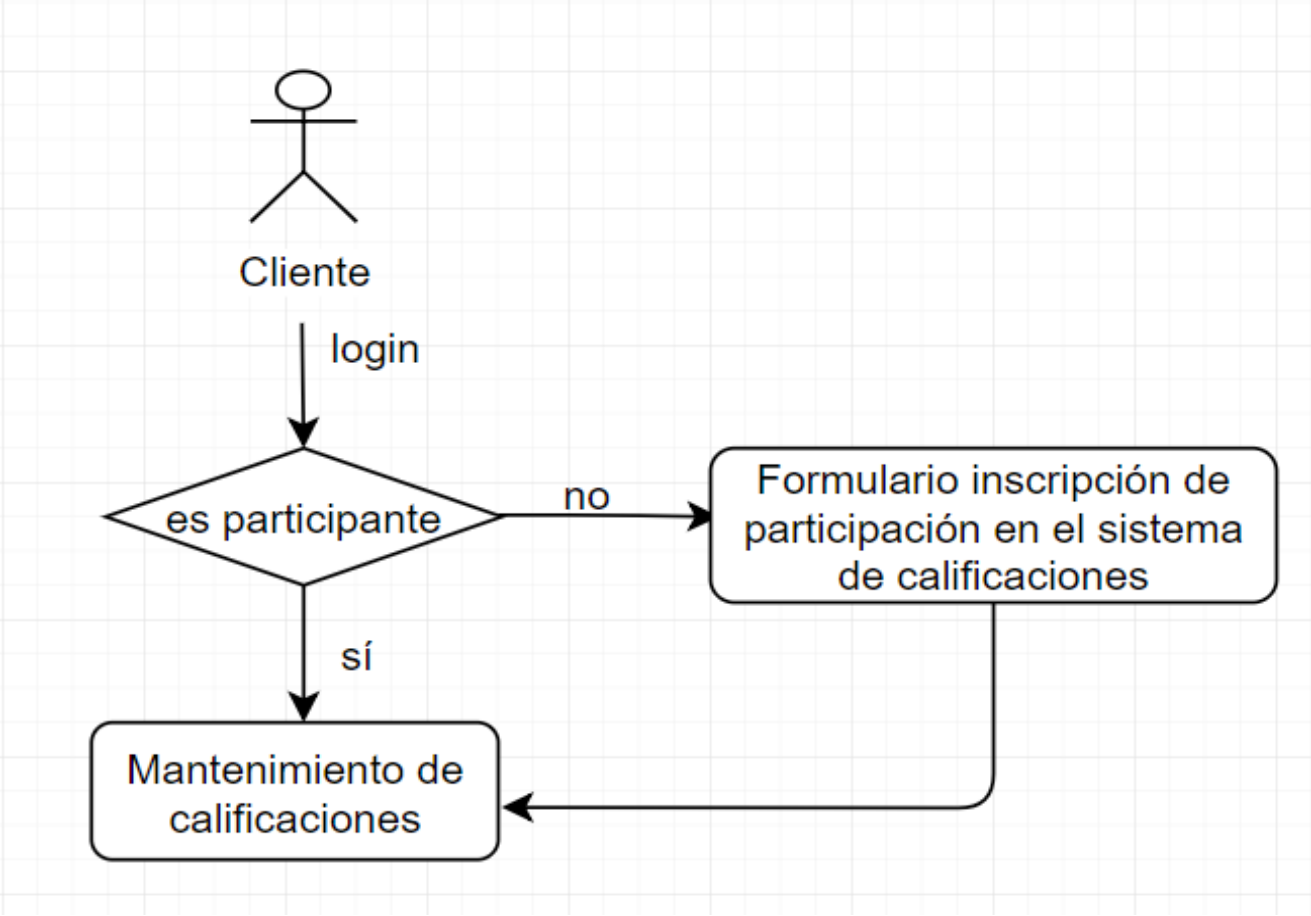
Secuencia reproducción: { usuario, capítulo, instante_inicio, instante_fin }



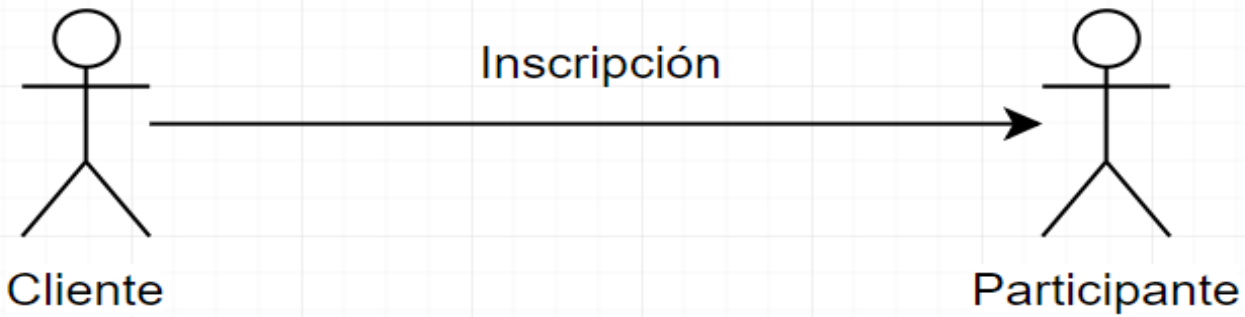

```
si (unión de secuencias de reproducción para un usuario y un capítulo >=  duración de capítulo / 4
    && no existe visionado del capítulo para usuario)
{

        añadir_visionado_de_capítulo_para usuario();
}
```

El registro de visionado se inserta sin puntuación

Calificaciones



Calificaciones (2)



· **Un participante puede puntuar sus visionados de capítulos que aún no haya puntuado.**

· **La interfaz para puntuar visionados recibe el nombre de "Mantenimiento de calificaciones"**

Calificaciones (3)

Un cliente del sistema de vídeo bajo demanda, cuando se inscribe como **participante** en el sistema de calificaciones **registra**:

- Su **sexo**.

- Su **edad**. El sistema registra la edad en una de las siguientes **categorías**:

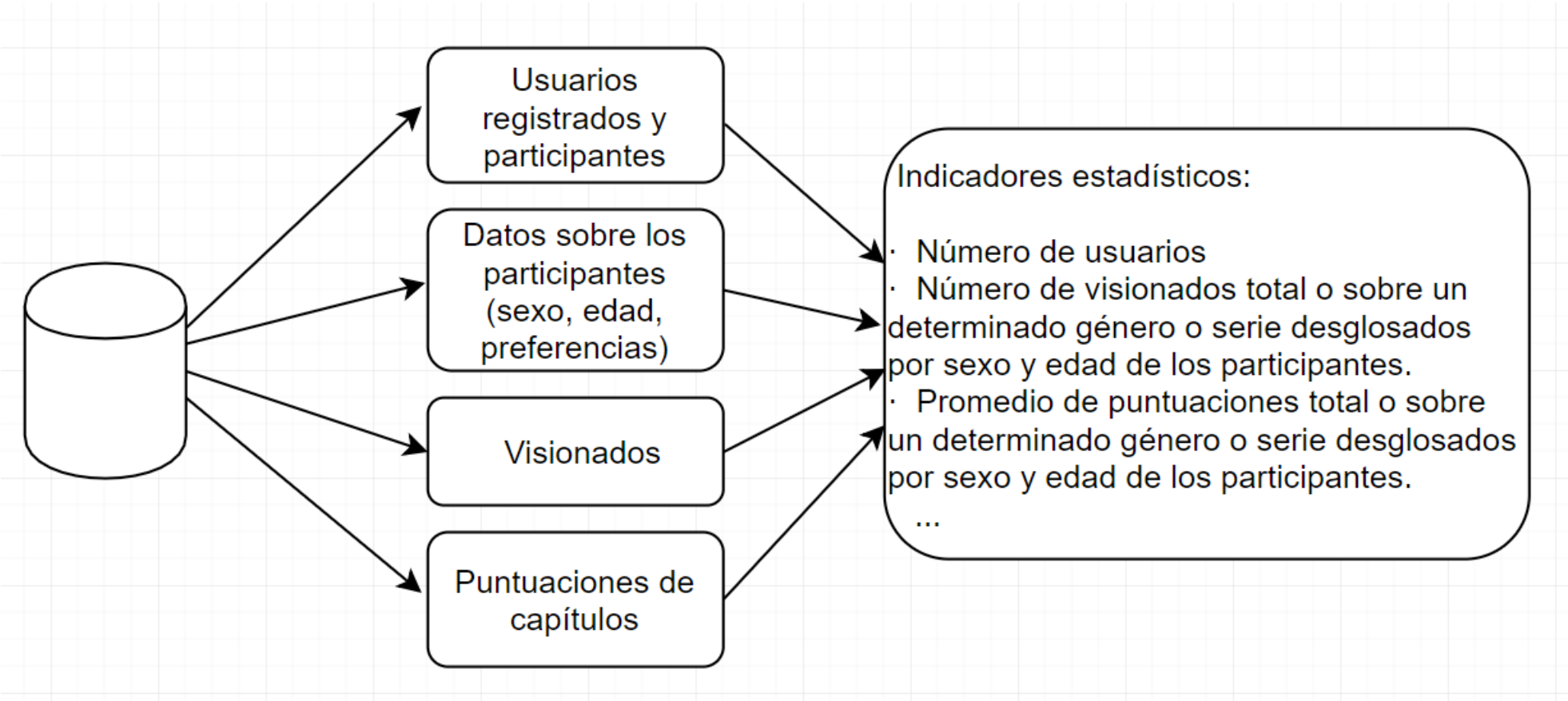
  · **Entre 18 y 35 años.**
  · **Entre 36 y 59 años.**
  · **Desde 60 años.**

- Sus **dispositivos favoritos** (al menos 1).

- Sus **géneros de series favoritos** (al menos 1).

- Sus **series favoritas** (al menos 1).

Estadísticas



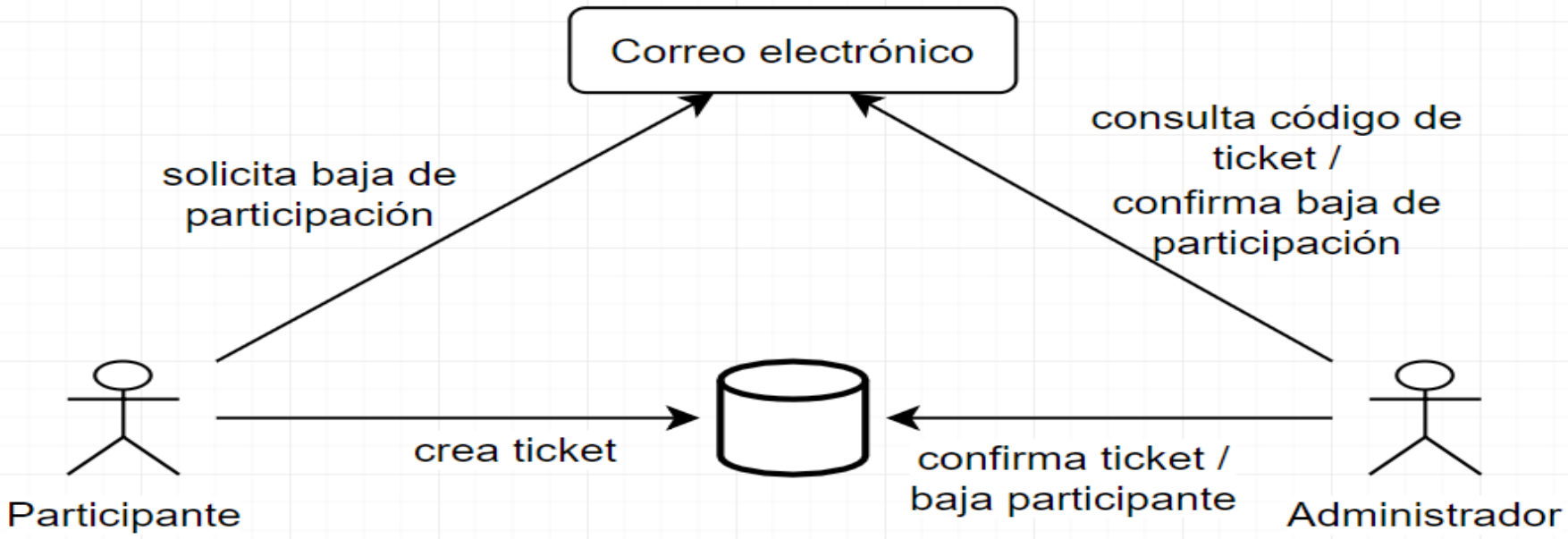
Estadísticas (2)

Resultados:

- **Globales** (sobre todos los géneros y series).
- Sobre un determinado **género**.
- Sobre una determinada **serie**.

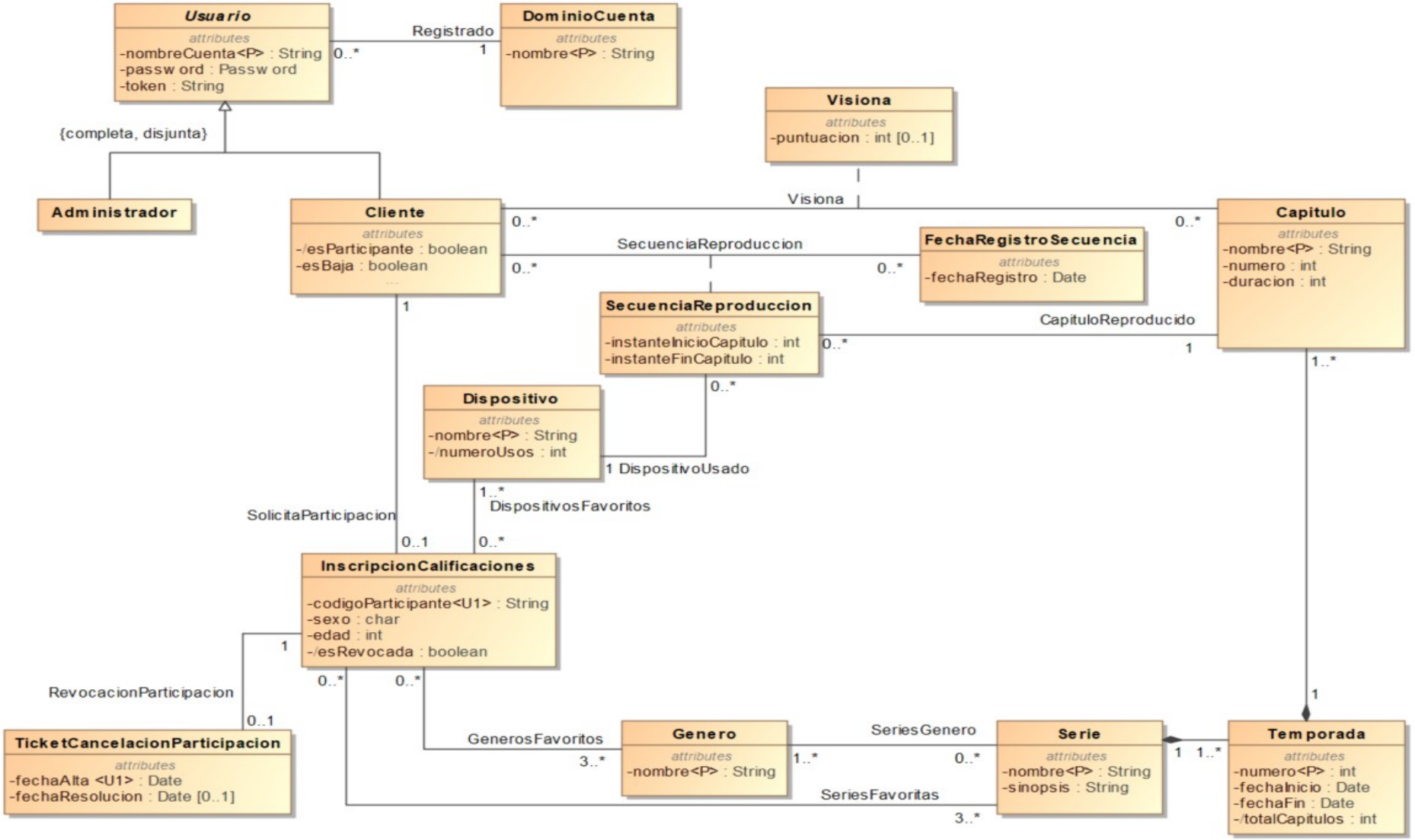
El **acceso** a los resultados estadísticos es **público** (no requiere inicio de sesión).

Gestor de calificaciones (administración)

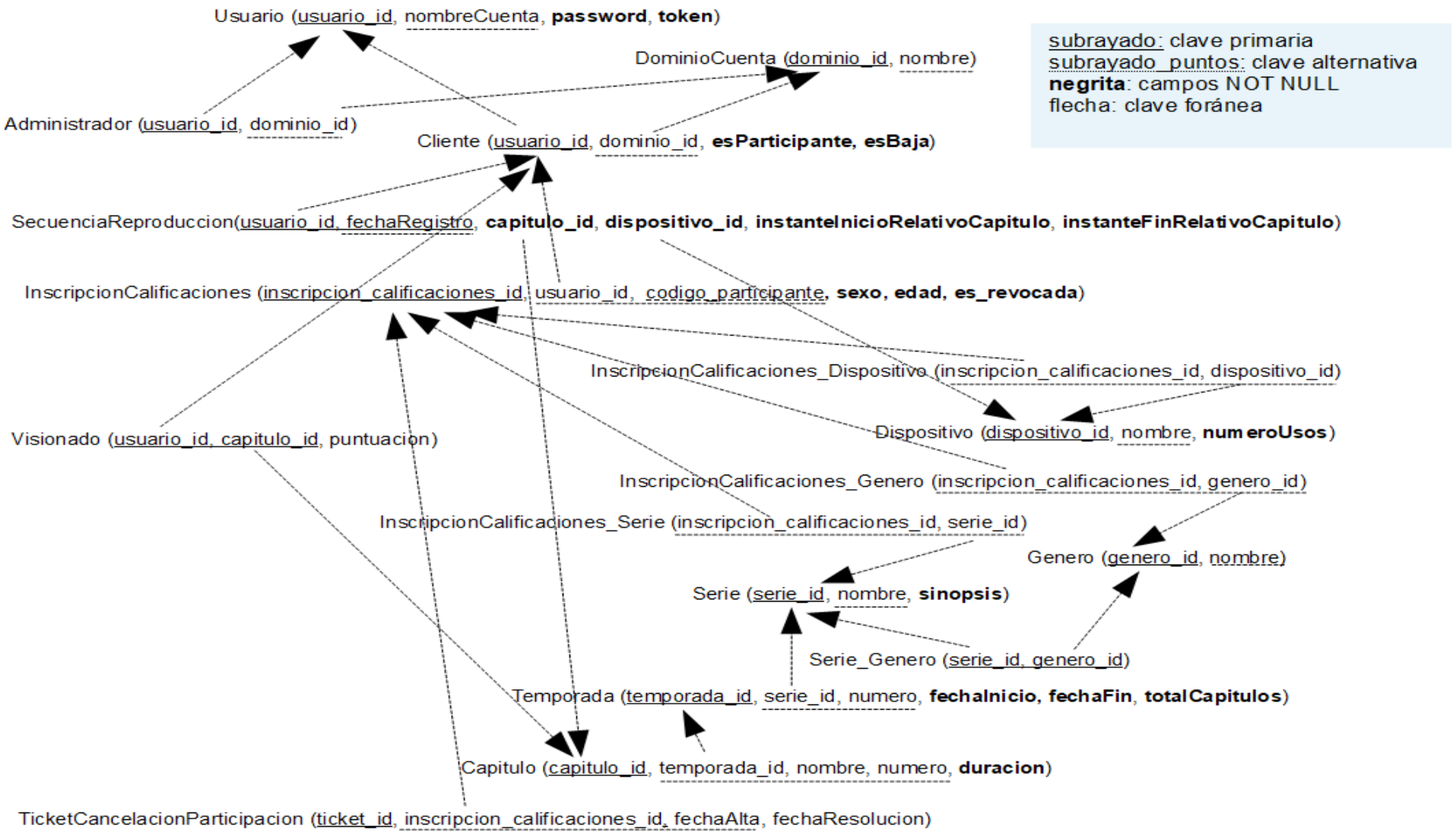


- El usuario es dado de **baja** en el **sistema de calificaciones** pero sigue registrado en la plataforma de vídeo bajo demanda.
- La baja se formaliza por medio de un **ticket** creado por el demandante y aprobado por el administrador.
- Los datos del usuario son dados de **baja lógica** (no física).

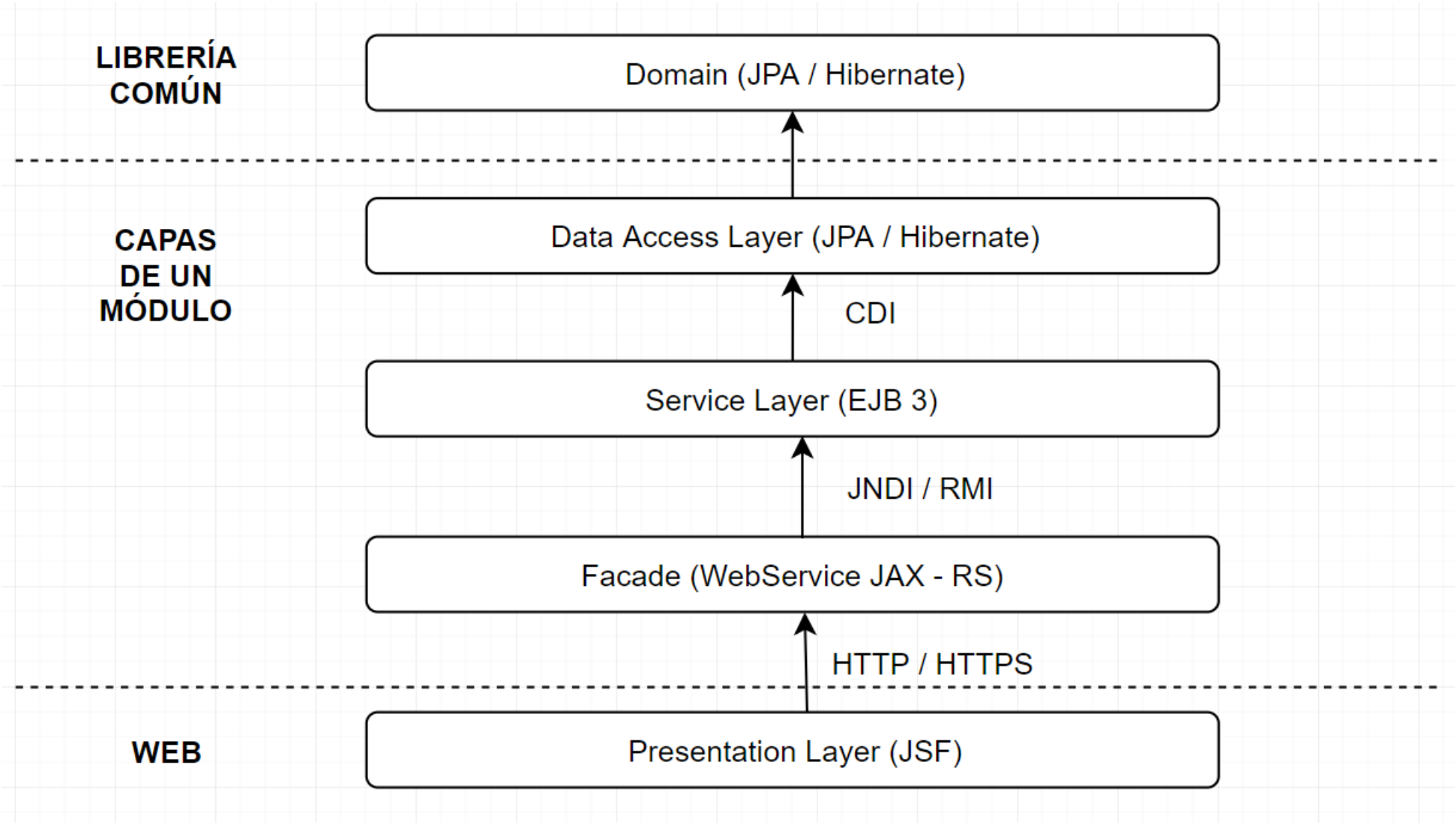
VÍDEOUOC – DIAGRAMA E/R



VÍDEOUOC – ESQUEMA RELACIONAL



VÍDEOUOC - ARQUITECTURA



VÍDEOUOC – ARQUITECTURA (2)

· Los datos se **transfieren entre capas** mediante **objetos de transporte (Data Transfer Objects)**.

· Cada **objeto de transporte** es una instancia de una clase Java **serializable**, **con atributos y métodos GET/SET** y **no dependiente de ningún framework (Plain Old Java Objects)**.

· Los datos se **transmiten por red entre la capa de presentación y el servicio web** serializándolos en **formato JSON (JavaScript Object Notation)**.

· Para **serializar y deserializar** los datos de los objetos de transporte se recurre a la **libería Jackson**.

Vinculación JPA-DTO

```
@Stateless
@CatalogoGeneros
public class ConsultaCatalogoGenerosImplDao implements ConsultaCatalogoDao{

    @PersistenceContext(unitName = "VideoUoc_PU")
    private EntityManager em;


    @SuppressWarnings("unchecked")
    @Override
    public CatalogoDto consultar() {

        CatalogoDto catalogo = new CatalogoDto();

        Query query = em.createQuery("SELECT new edu.ilopezalmtfg.recursos.dto.ElementoCatalogoDto(g.id, g.nombre) "
                + "FROM Genero g ORDER BY g.nombre", ElementoCatalogoDto.class);

        List<ElementoCatalogoDto> elementosCatalogo = query.getResultList();
        catalogo.setCatalogo(elementosCatalogo);

        return catalogo;
    }


}
```

Objeto DTO

Entidad JPA

Patrón Repository

- **Reutilización de interfaces de acceso a datos**: interfaces de uso común que implementan diferentes clases de acceso a los datos.

- **Evitar saturación de parámetros de entrada en los métodos de las interfaces.**

- **Uso de tipos genéricos** en el diseño de **interfaces**.

```java
package edu.ilopezalmtfg.dao.mantenimiento;

import java.util.List;


public interface MantenimientoDao<E> {

    public void insertar(E entity) throws Exception;

    public void eliminar(E entity) throws Exception;

    public void actualizar(E entity) throws Exception;

    public List<E> consultar(E entity) throws Exception;
}
```

EJB y servicio de transacciones

- Uso de **Enterprise Java Beans de sesión sin estado**.

- **El contenedor EJB del servidor de aplicaciones gestiona las transacciones (usando Java Transaction API)**. El desarrollador se despreocupa de la gestión de las transacciones y pasan a gestionarse por el servidor de aplicaciones sobre la capa de servicios.

- El **ciclo de vida de una transacción** transcurre sobre **una petición al servidor** lo que implica que, al finalizar la operación EJB y la petición todos los cambios son consumados siempre y cuando las operaciones hacia la base de datos se procesen sin incidencias.

Seguridad

· **Autenticación y autorización** a recursos del servicio web a través de la **API JAX-RS** mediante anotación @Secured.

· **Inicio de sesión:**

1) **Valida usuario y contraseña.**
2) **Registra token de sesión.**

· **Autenticación** (mediante filtro de autenticación):

Comprobar que el **token** enviado por el cliente hacia el servidor para cada petición durante la sesión corresponda al token registrado en su inicio de sesión.

· **Autorización** (mediante filtro de autorización):

Comprobar que el **rol** del usuario autenticado tenga **privilegios** de acceso sobre el recurso solicitado.

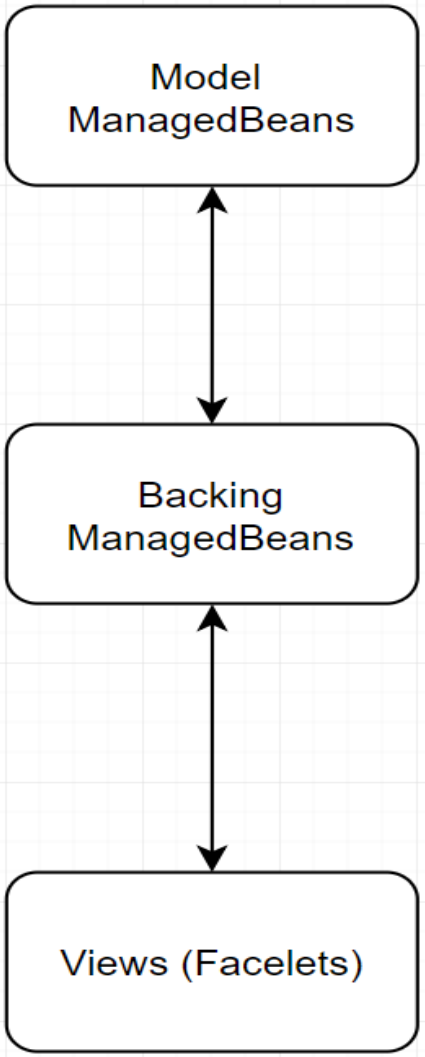
Seguridad (2)

```java
@Secured
@Provider
@Priority(Priorities.AUTHENTICATION)
public class AutenticacionFiltro implements ContainerRequestFilter {

    @EJB
    private AutenticacionBeanRemote autenticacion;

    private static final String AUTHENTICATION_SCHEME = "Uoc";


    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {

        // Obtener la cabecera HTTP estándar para la autenticación: "Authorization"
        // (el nombre estándar no es apropiado porque no corresponde al proceso de autorización, sino al de autenticación)
        String cabeceraAutenticacion = requestContext.getHeaderString(HttpHeaders.AUTHORIZATION);

        // Validar el patrón de la cabecera HTTP según la referencia "AUTHENTICATION_SCHEME" ("Uoc")
        if (!esAutenticacionBasadaEnToken(cabeceraAutenticacion)) {
            // Si el esquema de cabecera no coincide con el patrón de referencia
            abortarConNoAutorizado();


        } else {

            // Extraer el token de la cabecera HTTP
            String token = cabeceraAutenticacion.substring(AUTHENTICATION_SCHEME.length()).trim();

            if (token == null || token == "") {
                // Si el token no existe
                abortarConNoAutorizado();

            } else {

                try {

                    // Validar el token en la base de datos y obtener los datos del usuario (nombre de cuenta y rol)
                    CredencialesDto dto = validarToken(token);

                    // Asociar los datos del usuario al contexto
                    final SecurityContext currentSecurityContext = (SecurityContext) requestContext.getSecurityContext();
                    requestContext.setSecurityContext(new SecurityContext() {

                        private String rolAsignado = dto.getRol().toString();

                        @Override
                        public Principal getUserPrincipal() {
                            return () -> dto.getNombreCuenta();
```

Seguridad (3)

```java
@Secured
@Provider
@Priority(Priorities.AUTHORIZATION)
public class AutorizacionFiltro implements ContainerRequestFilter {

    // Obtiene por inyección de dependencias la información sobre el recurso solicitado (indicada por anotación "Secured")
    @Context
    private ResourceInfo resourceInfo;

    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {

        // Obtener la clase objeto de petición del cliente y sus roles asociados
        Class<?> claseRecurso = resourceInfo.getResourceClass();
        List<Rol> rolesDeClase = extraerRolesAnotacionesSecured(claseRecurso);

        // Obtener el método objeto de petición del cliente y sus roles asociados
        Method metodoRecurso = resourceInfo.getResourceMethod();
        List<Rol> rolesDeMetodo = extraerRolesAnotacionesSecured(metodoRecurso);

        try {

            // Comprobar si el cliente tiene permisos para ejecutar el método:
            // Primero comprueba los roles de método, que tienen preferencia sobre los de clase y
            // , si no existen roles de método para el recurso solicitado, comprueba los de la clase de este
            if (rolesDeMetodo.isEmpty()) {
                comprobarPermisos(rolesDeClase, requestContext);
            } else {
                comprobarPermisos(rolesDeMetodo, requestContext);
            }
```

Presentación (JSF)



· Representan los datos a los que acceden las vistas indirectamente a través de los *backing managed beans*.

· Alcance de petición o de sesión (*@RequestScoped, @SessionScoped*).

· Representan el contenido las vistas.

· Referencian los *model managed beans* y los hacen visibles a las vistas.

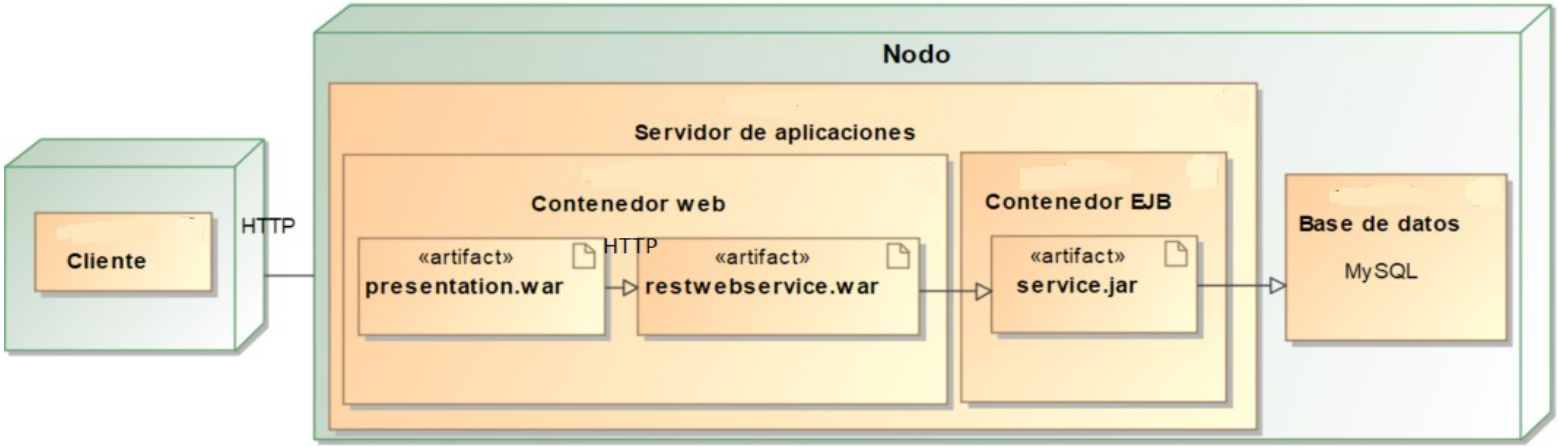
· Contienen acciones para la navegación y llamadas al servicio web por HTTP.

· Alcance de petición (*@RequestScoped*).

· Vistas con componentes reutilizables (facelets).

· Muestran los atributos y las acciones de los backing managed beans mediante Expression Language (EL).

Despliegue



En diagramas de arquitectura: cada componente se representa con un nombre correspondiente a su módulo, incluídos servicios web.

En la implementación: por simplificar se han creado dos artefactos, que corresponden a una web y a un servicio web. El nombre de la web es "CalificacionesWeb" y el nombre del servicio web es "MonitorEstadisticas", aunque este último incluya, además, recursos sobre otros módulos que no son el de estadísticas.

Agradecimientos

Gracias por la amistad y atención recibida por Jordi Ascensión Pestonit, gerente de:

Enterprise Consulting & Training