LENGUAJE DE PROGRAMACIÓN JAVA – COMPLEMENTO 2016

Documentación de “GG Rosario”

Alberto Daniel Ojeda
Legajo: 43824

Índice de contenidos

Índice de imágenes (capturas de pantalla)

Herramientas y frameworks utilizados

Hibernate

Con el fin de manipular objetos puros y abstraer la capa de datos decidí utilizar el ORM Hibernate, configurado de la siguiente forma:

```
package com.herokuapp.ggrosario.util;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

/**
* @author Ojeda Alberto Daniel
*/
public class HibernateUtil {

private static SessionFactory sessionFactory;

public static SessionFactory getSessionFactory() {
      if (sessionFactory == null) {
            // loads configuration and mappings
            Configuration configuration = new
Configuration().configure();
            ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder().applySettings(configuration.getPropert
ies()).build();

            // builds a session factory from the service registry
            sessionFactory =
configuration.buildSessionFactory(serviceRegistry);
      }

      return sessionFactory;
}

/**
 * Guarda un objeto en la base de datos
 * @param unObjeto Objeto que se desea guardar
 */
public static void guardar(Object unObjeto) {
      Session session = getSessionFactory().openSession();
      Transaction transaction = session.beginTransaction();
      session.persist(unObjeto);
      transaction.commit();
      session.close();
}

/**
 * Actualiza un objeto de la base de datos
 * @param unObjeto Objeto que se desea actualizar
 */
public static void actualizar(Object unObjeto) throws
HibernateException {
      Session session = getSessionFactory().openSession();
      Transaction transaction = session.beginTransaction();
      try{
            session.merge(unObjeto);
      }catch (HibernateException he){
            System.out.println(he.getMessage());
      }
      transaction.commit();
      session.close();

}

public static void eliminar(Object unObjeto) throws HibernateException
{
      Session session = getSessionFactory().openSession();
      Transaction transaction = session.beginTransaction();
```

```
		session.delete(unObjeto);
		transaction.commit();
		session.close();
	}

	/**
	 * Recupera un objeto de la base de datos
	 * @param id Identificador que el objeto tiene en la BD
	 * @param clase Nombre de la clase de la que es instancia el objeto a
recuperar
	 * @return Object Un objeto. Debe ser casteado al tipo de objeto que se
quiso recuperar (<code>clase</code>)
	 *
	 */
	public static Object obtener (String id, String clase) throws
HibernateException {
		Session session = getSessionFactory().openSession();
		Transaction transaction = session.beginTransaction();
		Object unObjeto;
		unObjeto = session.get("com.herokuapp.ggrosario.modelo."+clase,
id);
		transaction.commit();
		session.close();
		return unObjeto;
	}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</prope
rty>
        <property
namge="hibernate.connection.driver_class">org.postgresql.Driver</proper
ty>

        <!-- Local settings [F O R   L O C A L   D E V E L O P I N G] -
->
        <property
name="hibernate.connection.url">jdbc:postgresql://localhost:5432/ggrosa
rio</property>
        <property
name="hibernate.connection.username">postgres</property>
        <property
name="hibernate.connection.password">38131125</property>

        <!-- Remote settings -->
        <property
name="hibernate.connection.url">jdbc:postgresql://ec2-54-163-239-
63.compute-
1.amazonaws.com:5432/dekiv1ush42g4n?sslmode=require</property>
        <property
name="hibernate.connection.username">pbegunawlvcxlb</property>
        <property
name="hibernate.connection.password">odnFxqunuNuck27htAdxNqcPzR</proper
ty>

        <property name="hibernate.show_sql">false</property>
        <property name="hibernate.format_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property
name="hibernate.enable_lazy_load_no_trans">true</property>

        <!-- Classes to be mapped -->
        <mapping class="com.herokuapp.ggrosario.modelo.Tienda"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Usuario"/>
        <mapping class="com.herokuapp.ggrosario.modelo.ListaDeseos"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Rol"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Permisos"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Requisito"/>
        <mapping
class="com.herokuapp.ggrosario.modelo.RequisitoMinimo"/>
        <mapping
class="com.herokuapp.ggrosario.modelo.RequisitoRecomendado"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Catalogo"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Categoria"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Juego"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Comentario"/>
        <mapping
class="com.herokuapp.ggrosario.modelo.ListaDeseosJuegos"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Reserva"/>
        <mapping
class="com.herokuapp.ggrosario.modelo.UsuarioComentario"/>
        <mapping
class="com.herokuapp.ggrosario.modelo.JuegoComentario"/>
        <mapping class="com.herokuapp.ggrosario.modelo.JuegoReserva"/>
        <mapping class="com.herokuapp.ggrosario.modelo.Stock"/>
        <mapping class="com.herokuapp.ggrosario.modelo.ABMRol"/>
    </session-factory>
</hibernate-configuration>
```

API de Cloudinary

Debido a dificultades para subir imágenes una vez realizado el despliegue de la aplicación en el servidor (Heroku), se ha optado por utilizar un servicio de almacenamiento de imágenes en la nube que cuente con una API que sea sencilla de manejar en Java. Sólo se suben imágenes de las carátulas o covers de los juegos, para que sean sencillos de identificar.

La configuración utilizada para poder subir imágenes a una cuenta en Cloudinary (la cual primero debe crearse) es la siguiente:

```
/* Datos de acceso a la cuenta de Cloudinary */
            String apiKey = "594979417922161";
            String apiSecret = "kW0lFSLADk8vp8ma_QgX6dFFMjE";
            String cloudName = "ggrosario";

            /* Instanciamos un servicio de Cloudinary */
            Cloudinary cloudinary = new
Cloudinary(ObjectUtils.asMap(
                    "cloud_name", cloudName,
                    "api_key", apiKey,
                    "api_secret", apiSecret));
```

Ojeda, Alberto Daniel
Legajo: 43824

Diagrama de casos de uso general

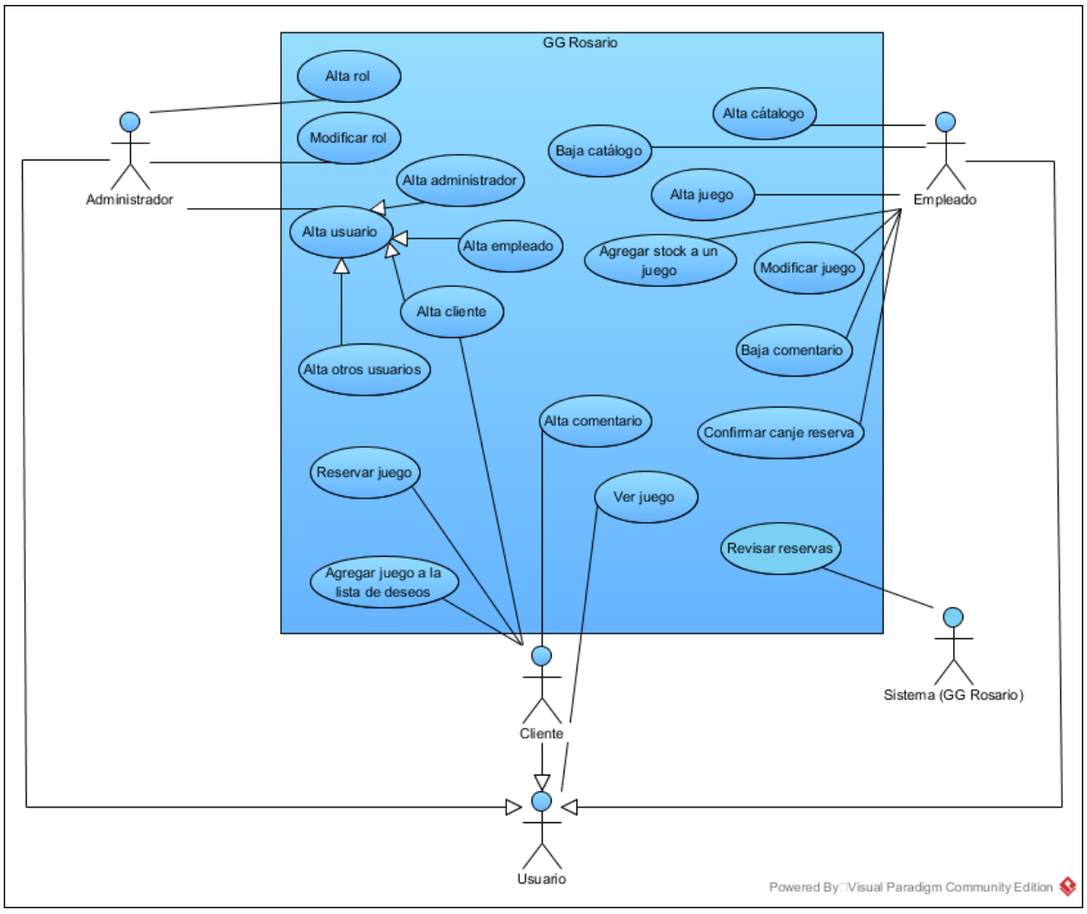
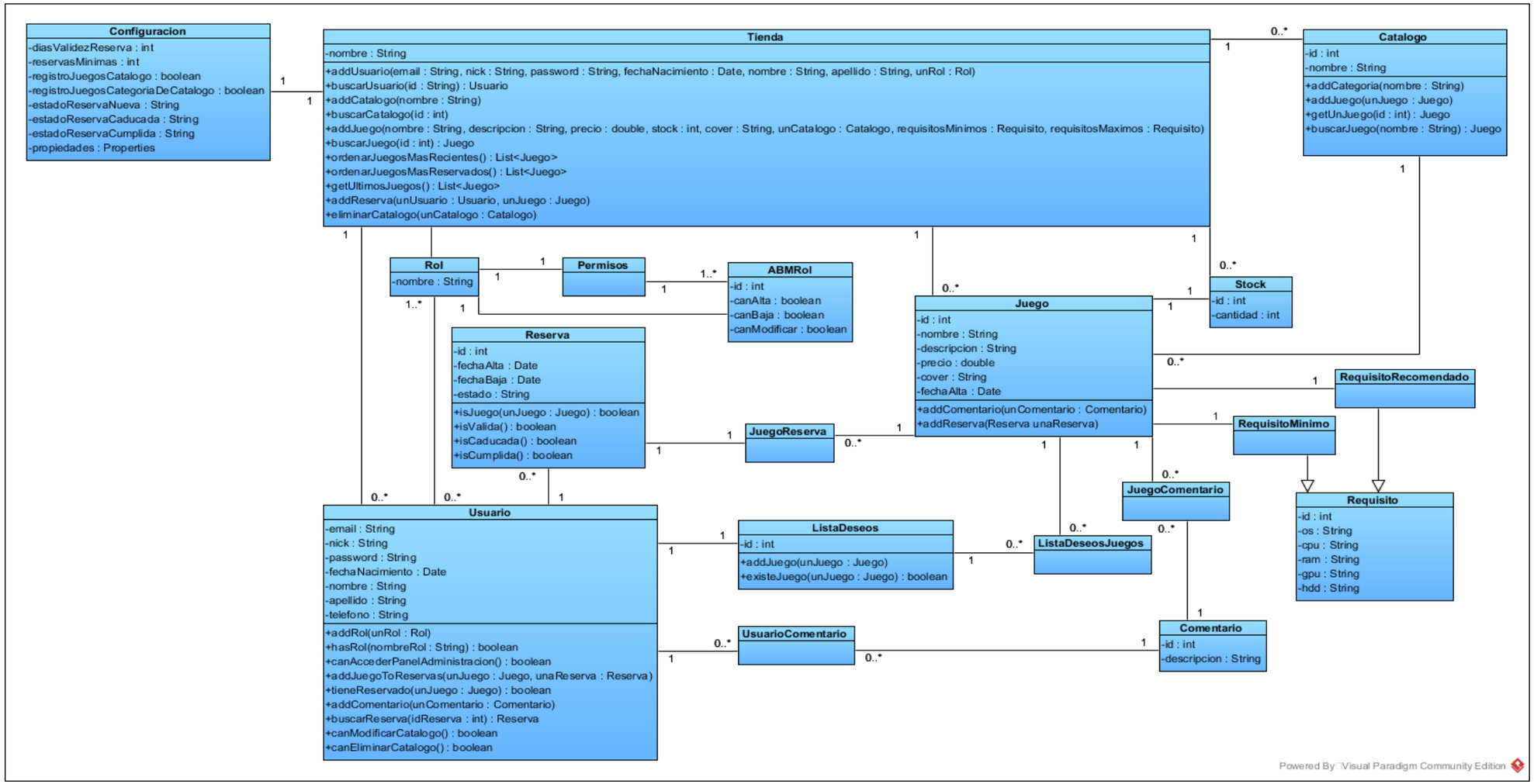
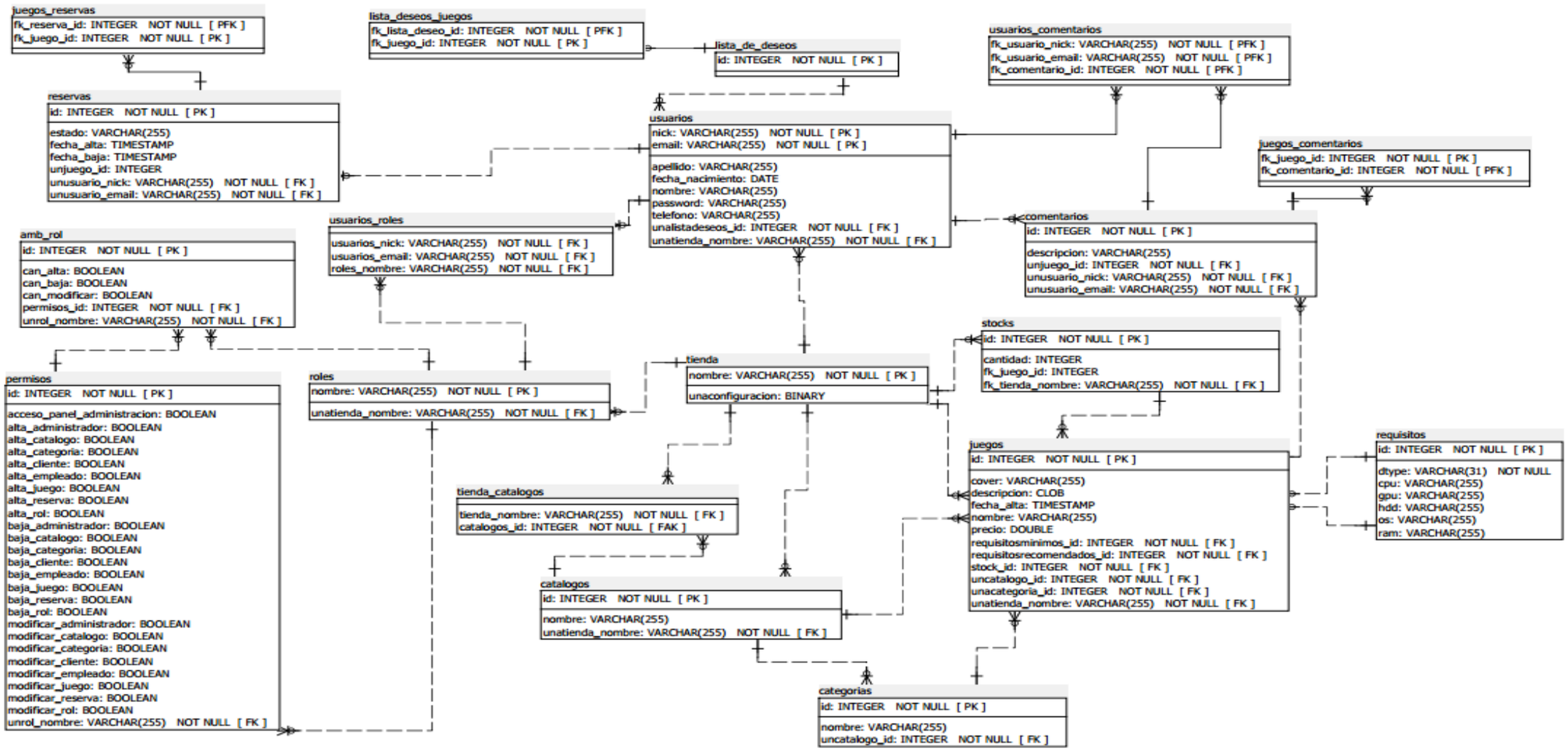


Diagrama de clases

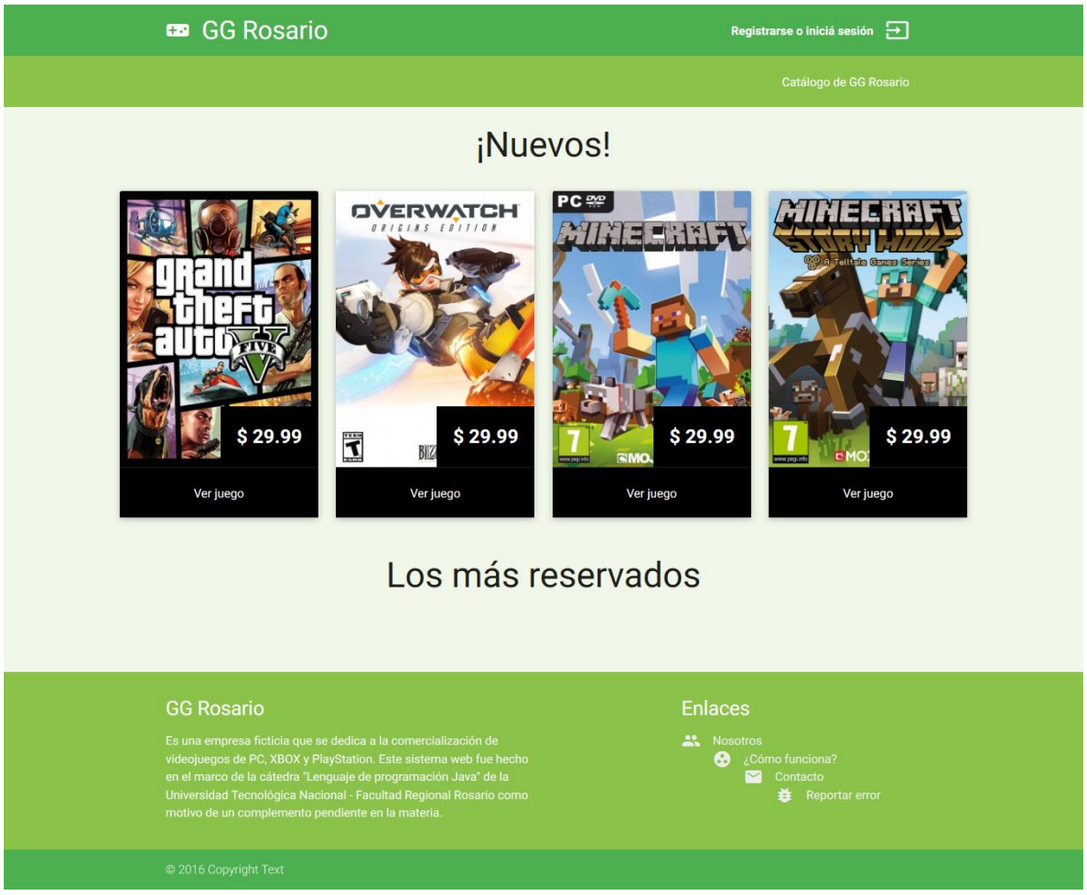


Modelo de datos - Modelo relacional



Ojeda, Alberto Daniel
Legajo: 43824

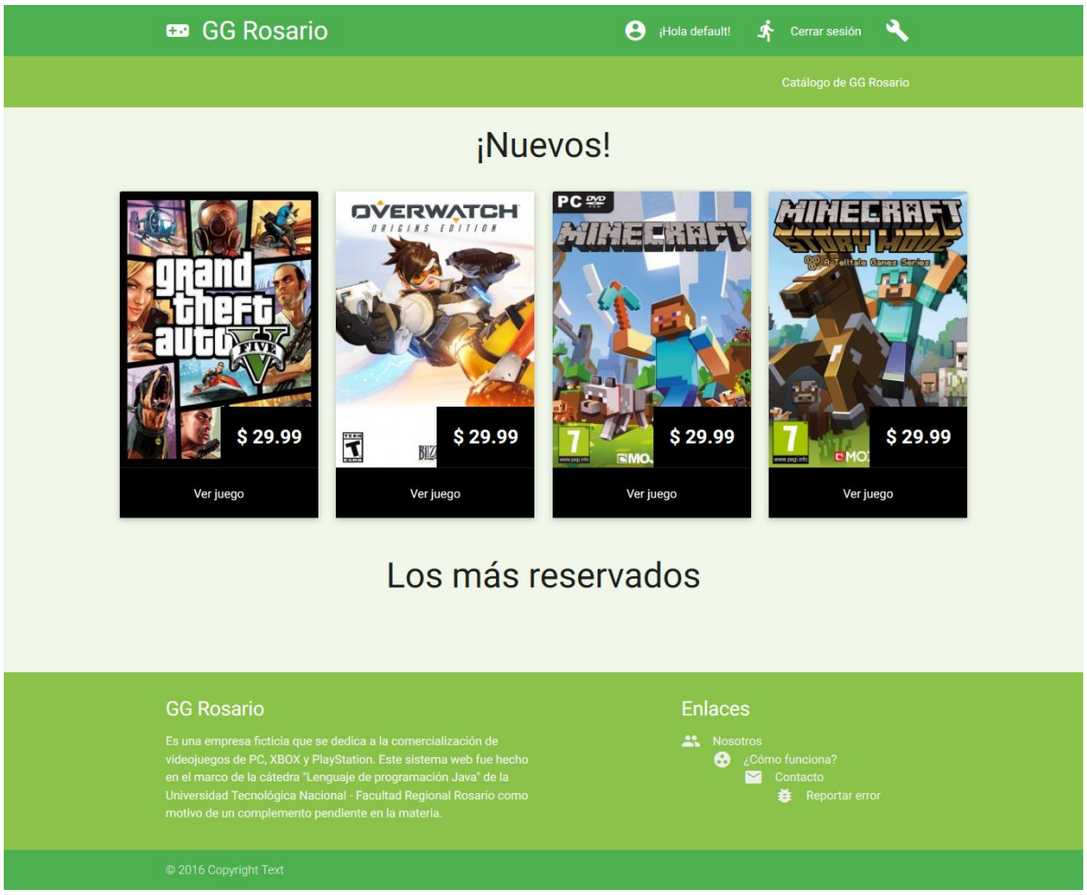
Capturas de pantalla



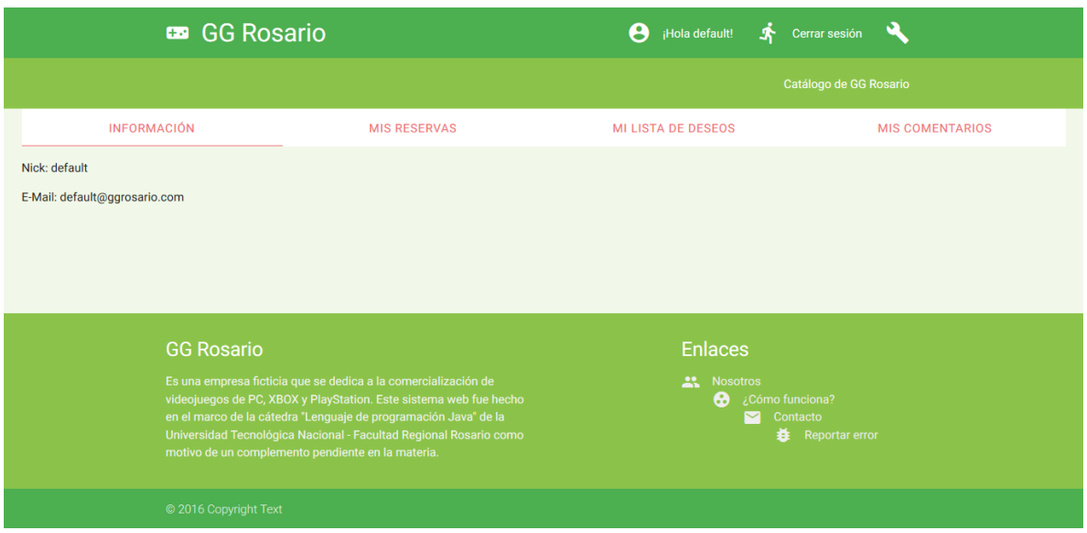
Pantalla 1 - Página de inicio sin loguearse

Pantalla 2 - Página para registrarse o para iniciar sesión

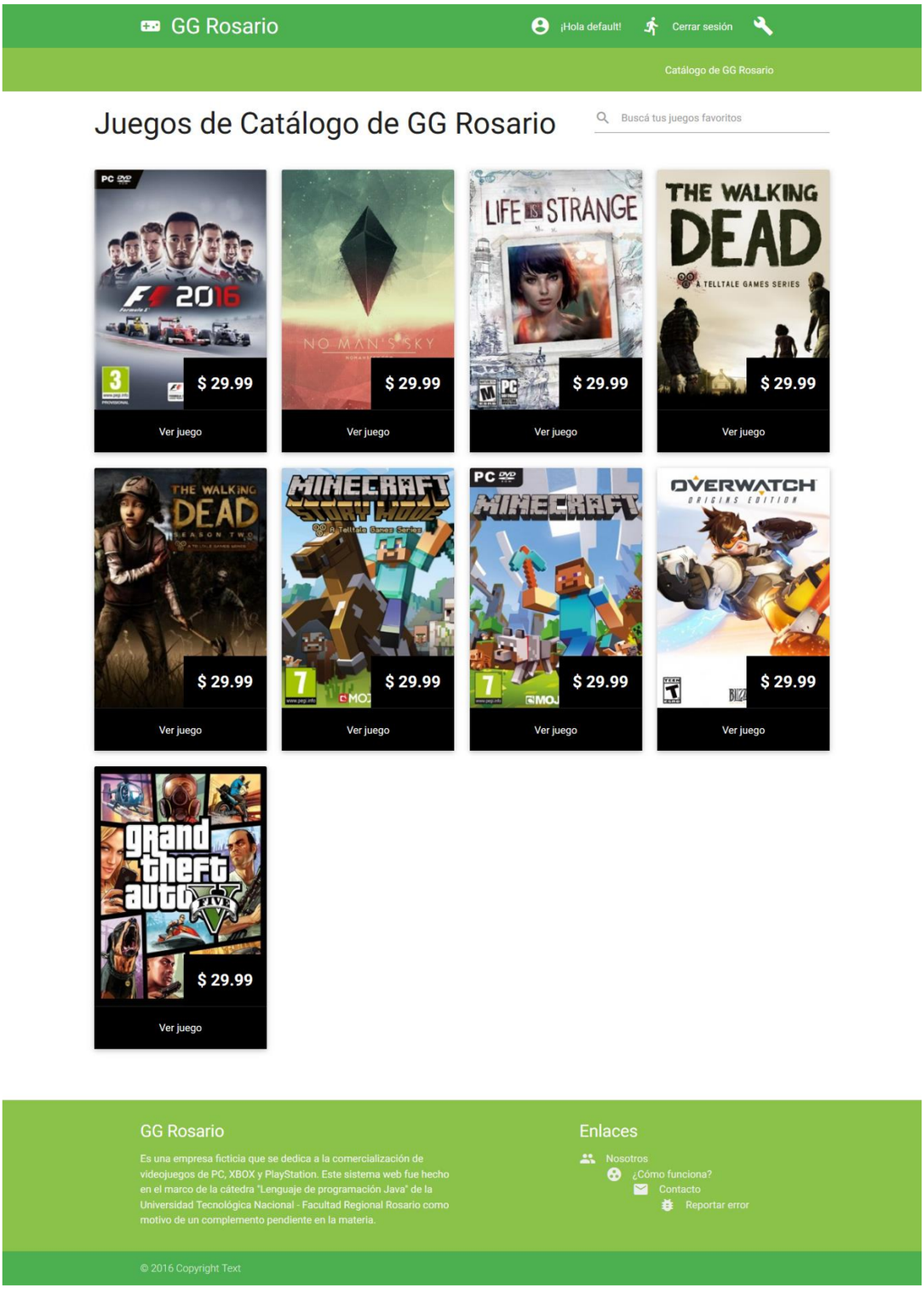
Ojeda, Alberto Daniel
Legajo: 43824



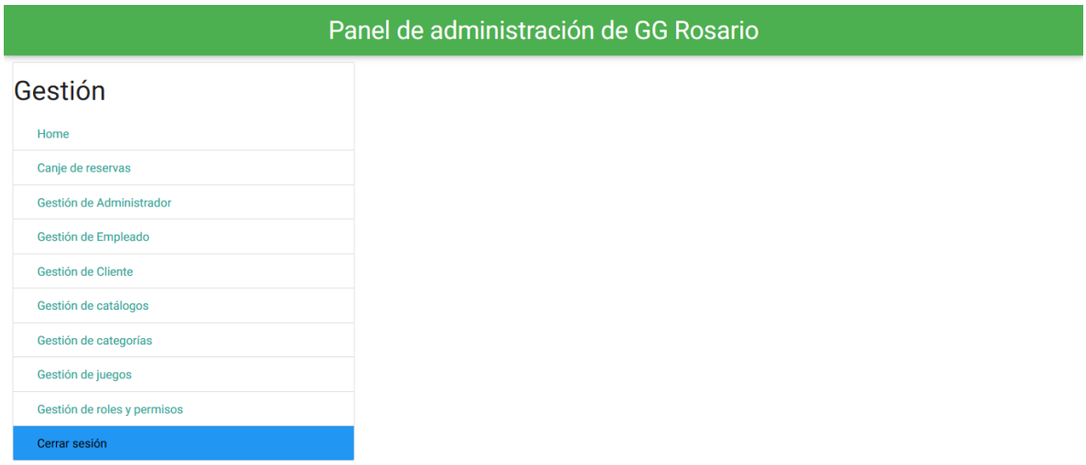
Pantalla 3 - Pantalla de inicio con usuario logueado



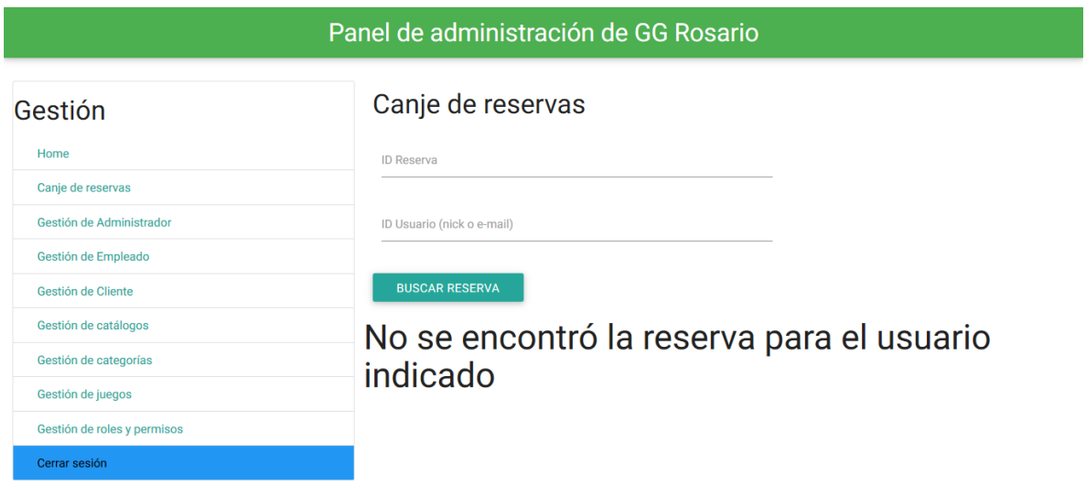
Pantalla 4 - Perfil del usuario



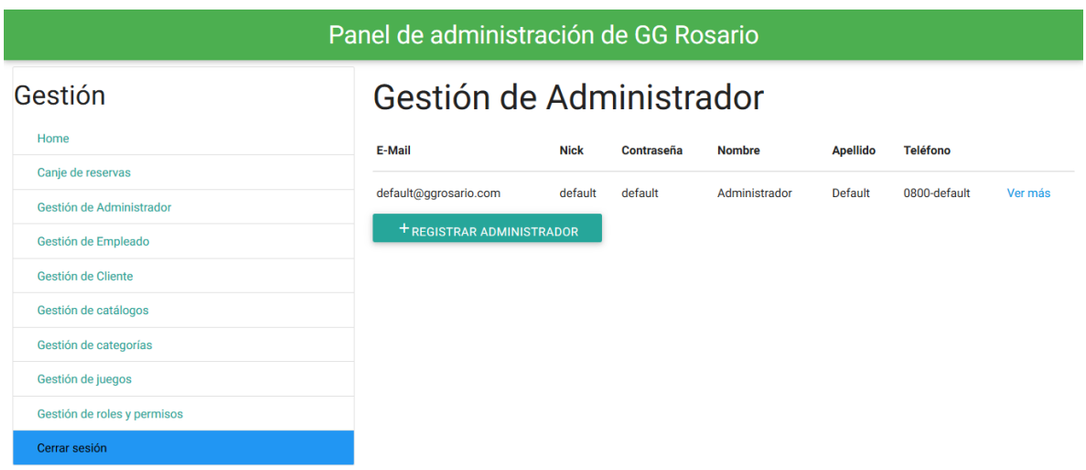
Pantalla 5 - Página de un catálogo con sus juegos y un buscador de juegos



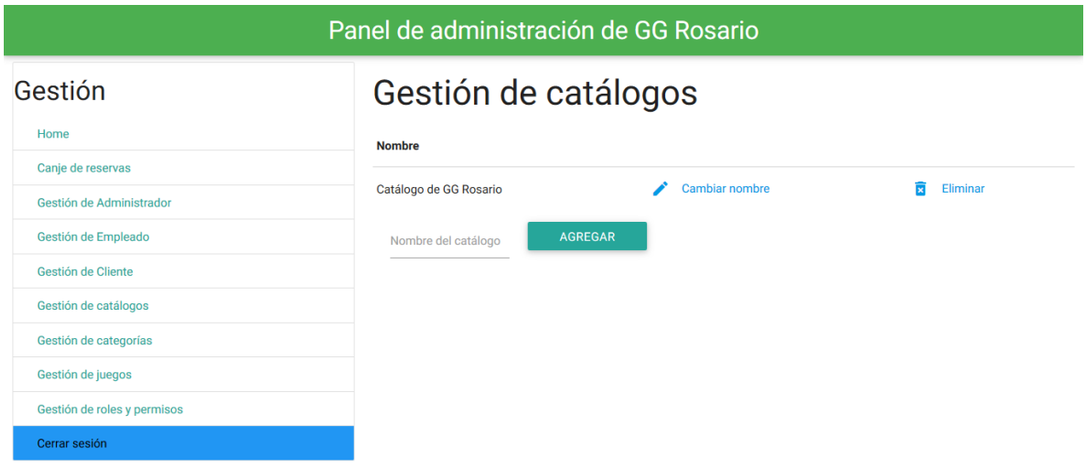
Pantalla 6 - Panel de administración



Pantalla 7 - Página para confirmar un canje de reserva

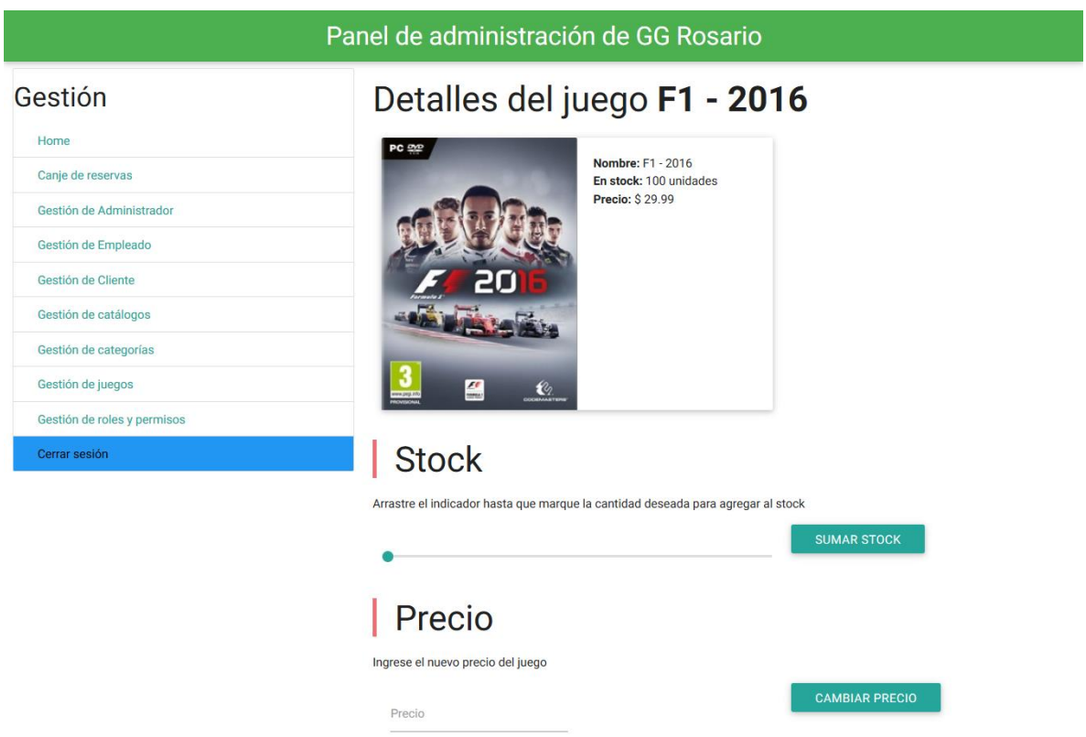


Pantalla 8 - Página para gestionar usuarios (es la misma para todos, sólo cambia los tipos de usuarios que son mostrados)

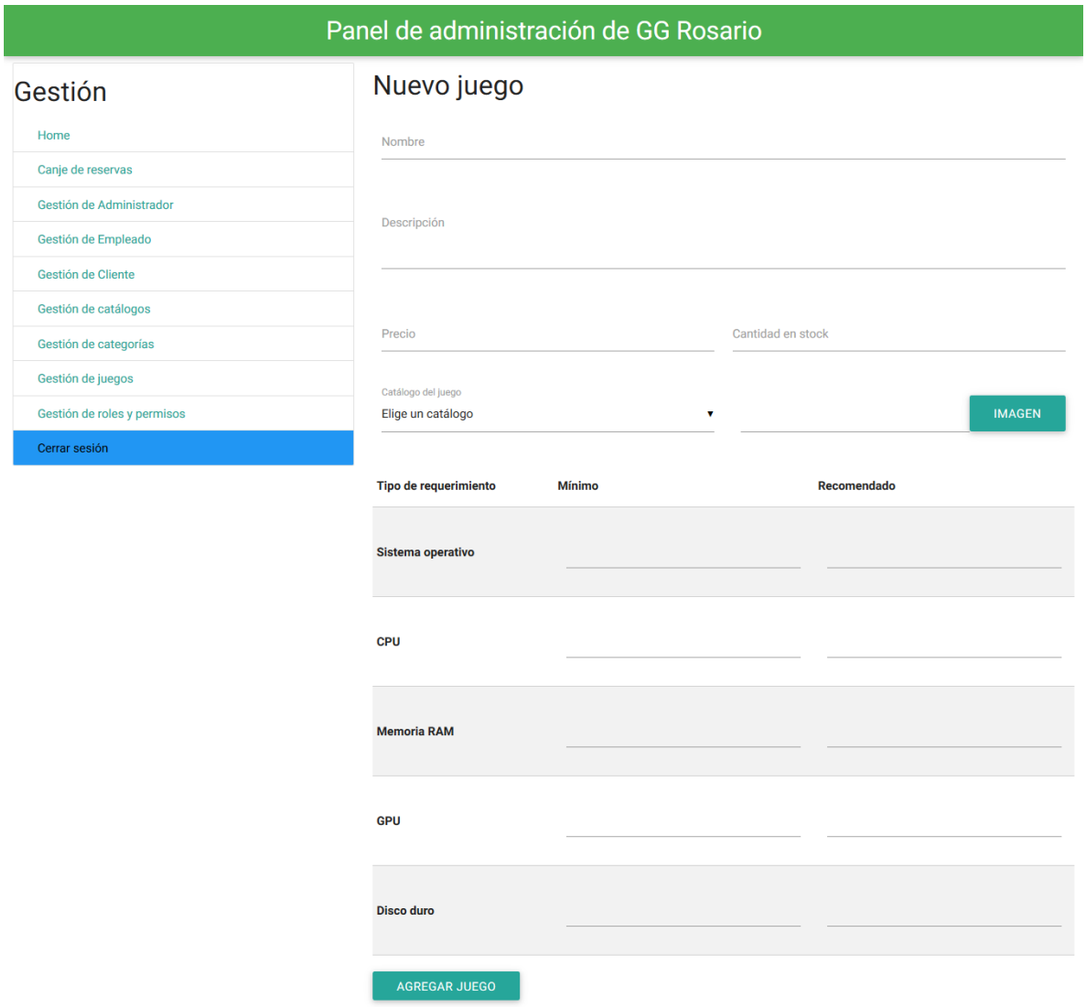


Pantalla 9 - Página para agregar o eliminar catálogos

Pantalla 10 - Página para visualizar todos los juegos



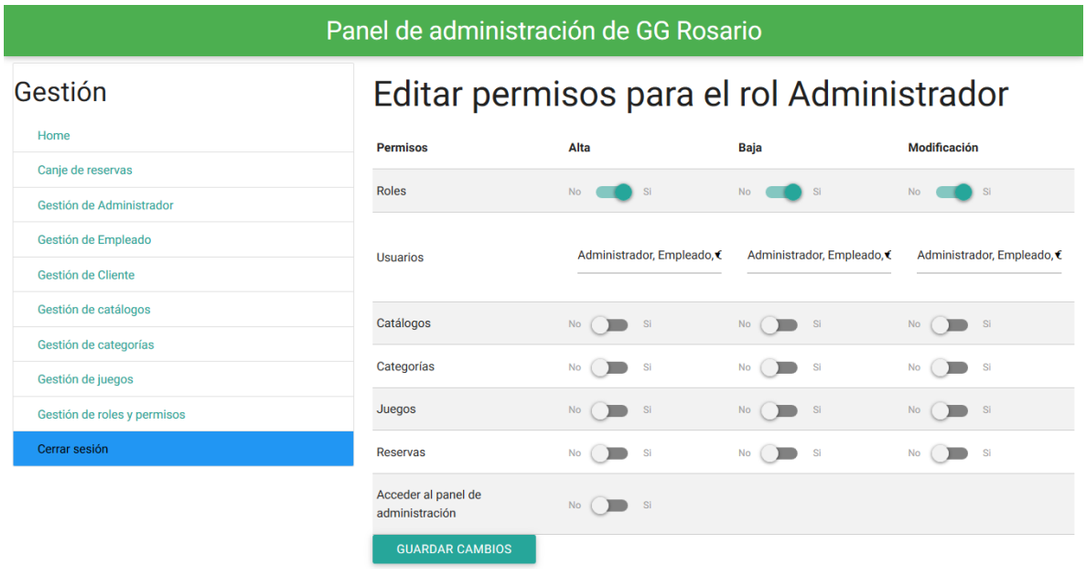
Pantalla 11 - Página para ver los detalles de un juego y actualizar su stock y su precio



Pantalla 12 - Página para dar de alta un nuevo juego

Pantalla 13 - Página para la gestión de roles

Ojeda, Alberto Daniel
Legajo: 43824



Pantalla 14 - Página para modificar los permisos de un rol

Descripción de los casos de uso

Caso de uso: “Agregar juego a la lista de deseos”

Caso de uso: “Agregar juego a la lista de deseos”
Objetivo: “Este caso de uso consiste en permitirle a un usuario agregar un juego a su lista de deseos”
Actores: Usuario cliente y demás usuarios registrados del sistema
Pre-condiciones: El usuario debe estar autenticado en el sistema y debe tener permisos para ejecutar el caso de uso. El juego debe estar dado de alta en el sistema.
Pos-condiciones: El juego se ha añadido a la lista de deseos del usuario.

Curso típico de eventos

Acción del actor	Acción del sistema
1.- El caso de uso comienza cuando un usuario utiliza la opción “Añadir juego a la lista de deseos”	2.- El sistema verifica que el juego no esté en la lista de deseos del usuario
	3.- El sistema añade el juego en la lista de deseos del usuario

Curso alternativo:

Acción del actor	Acción del sistema
	2.- Si el juego está en la lista de deseos del usuario, el sistema lo informará y termina el caso de uso

Llamada al servlet AgregarJuegoListaDeseosServlet:

```
@Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response);
        String idJuego = request.getParameter("idJuego");
        String idUsuario = request.getParameter("idUsuario");
        Usuario miUsuario = (Usuario)
request.getSession().getAttribute("miUsuario");
        boolean exito;
        if (miUsuario.getNick().equals(idUsuario)) {
            try {
                Tienda unaTienda = Tienda.getInstance();
                miUsuario.getUnaListaDeseos().
                  addJuego(unaTienda.buscarJuego(Integer.valueOf(idJuego)));
                exito = true;
            } catch (Exception ex) {
                exito = false;
            }

        } else {
            exito = false;
        }
        request.getSession().setAttribute("exitoFavorito", exito);
        response.sendRedirect("info-juego?idJuego=" + idJuego);
    }
```

Llamada al método Usuario#addJuego

```
/**
 * Agrega un juego a esta lista de deseos
 *
 * @param unJuego Instancia de la clase Juego que se agregará a esta lista
 * de deseos
 * @throws JuegoException Cuando el juego ya existe en la lista de deseos
 */
public void addJuego(Juego unJuego) throws JuegoException {
	if (existeJuego(unJuego)) {
		throw new JuegoException("El juego ya existe en la lista de
deseos");
	}
	ListaDeseosJuegos listaDeseosJuegos = new ListaDeseosJuegos(this,
unJuego);
	this.unaListaDeseosJuegos.add(listaDeseosJuegos);
	HibernateUtil.actualizar(this);
}
```

Caso de uso: “Reservar juego”

Caso de uso: “Reservar juego”
Objetivo: “Este caso de uso consiste en permitirle a un usuario reservar un juego para poder adquirirlo”
Actores: Usuario cliente y los usuarios registrados en el sistema que tengan los permisos apropiados para ejecutar el caso de uso
Pre-condiciones: El usuario debe estar autenticado en el sistema y debe tener permisos para ejecutar el caso de uso. El juego debe estar dado de alta en el sistema.
Pos-condiciones: El usuario ha reservado el juego.

Curso típico de eventos

Acción del actor	Acción del sistema
1.- El caso de uso comienza cuando un usuario utiliza la opción “Reservar juego”	2.- El sistema verifica que el juego no haya sido reservado por el usuario
	3.- El sistema añade el juego en la lista de juegos reservados del usuario

Curso alternativo:

Acción del actor	Acción del sistema
	2.- Si el juego ya ha sido reservado por el usuario, el sistema lo informará y termina el caso de uso

Llamada al servlet ReservarJuegoServlet

```
/**
 * Handles the HTTP <code>GET</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
      processRequest(request, response);
      String idJuego = request.getParameter("idJuego");
      String idUsuario = request.getParameter("idUsuario");
      Usuario miUsuario = (Usuario)
request.getSession().getAttribute("miUsuario");
      boolean exito;

      if (miUsuario.getNick().equals(idUsuario)) {

            try {
                  Juego unJuego = (Juego)
request.getSession().getAttribute("unJuego");
                  Tienda.getInstance().addReserva(miUsuario, unJuego);
                  exito = true;
            } catch (Exception ex) {
                  exito = false;
            }

      } else {
            exito = false;
      }
      request.getSession().setAttribute("exitoReserva", exito);
      response.sendRedirect("info-juego?idJuego=" + idJuego);
}
```

Llamada al método Tienda#addReserva

```
/**
 * Añade una reserva
 *
 * @param unUsuario Usuario que quiere hacer la reserva
 * @param unJuego Juego que el usario quiere reservar
 * @throws JuegoException Si el juego ya está reservado por el usuario
 */
public void addReserva(Usuario unUsuario, Juego unJuego) throws JuegoException
{
      if (unUsuario.tieneReservado(unJuego)) {
            throw new JuegoException("El juego ya está reservado por el
usuario");
      }
      Reserva unaReserva = new Reserva(unUsuario, unJuego);
      unUsuario.addJuegoToReservas(unaReserva);
      unJuego.addReserva(unaReserva);
}
```

Caso de uso: “Comentar juego”

Caso de uso: “Comentar juego”
Objetivo: “Este caso de uso consiste en permitirle a un usuario realizar un comentario en un juego”
Actores: Usuario cliente y demás usuarios registrados en el sistema
Pre-condiciones: El usuario debe estar autenticado en el sistema y debe tener permisos para ejecutar el caso de uso. El juego debe estar dado de alta en el sistema.
Pos-condiciones: Un nuevo comentario ha sido añadido al juego. El comentario ha sido añadido a la lista de comentarios del usuario.

Curso típico de eventos

Acción del actor	Acción del sistema
1.- El caso de uso desea dejar un comentario en un juego	
2.- El usuario escribe un texto que será el comentario	
3.- El usuario utiliza la opción “Enviar comentario”	4.- El sistema verifica que se haya escrito un texto
	5.- El sistema añade el comentario al juego

Curso alternativo:

Acción del actor	Acción del sistema
	4.- Si el sistema no encuentra un texto para añadir como comentario, se notificará de esto al usuario y finalizará el caso de uso

Validación del lado del cliente

```
$('#formComentar').on('submit', function(e){
	if ($('#cajaComentario').val().length === 0){
		e.preventDefault();
		Materialize.toast('Ingrese un comentario', 4000);
	}
});
```

Llamada al servlet AgregarComentarioServlet

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response)
		throws ServletException, IOException {
	processRequest(request, response);

	String comentarioEnviado = request.getParameter("comentario-enviado");
	Usuario miUsuario = (Usuario)
request.getSession().getAttribute("miUsuario");
	Juego unJuego = (Juego) request.getSession().getAttribute("unJuego");
	String errorComentario = null;
	try {
		if (comentarioEnviado.length() == 0) {
			throw new Exception("El comentario no puede estar vacío");
		} else {
			miUsuario.addComentario(new Comentario(comentarioEnviado,
miUsuario, unJuego));
		}
	} catch (Exception e) {
		errorComentario = e.getMessage();
		request.getSession().setAttribute("errorComentario",
errorComentario);
	}
	response.sendRedirect("info-juego?idJuego=" + unJuego.getId());
}
```

Llamada al método Usuario#addComentario

```
public void addComentario(Comentario unComentario) {
	UsuarioComentario usuarioComentario = new UsuarioComentario(this,
unComentario);
	this.comentarios.add(usuarioComentario);
	unComentario.getUnJuego().addComentario(unComentario);
	HibernateUtil.actualizar(this);
}
```

Caso de uso: “Confirmar canje de la reserva”

Caso de uso: “Confirmar canje de la reserva”
Objetivo: “Este caso de uso consiste en cambiar el estado de una reserva a ‘Cumplida’”
Actores: Usuario empleado
Pre-condiciones: El usuario debe estar autenticado en el sistema y debe tener permisos para ejecutar el caso de uso.
Pos-condiciones: El estado de una reserva fue modificado.

Curso típico de eventos

Acción del actor	Acción del sistema
1.- El caso de uso comienza cuando un usuario desea canjear una reserva	
2.- El usuario ingresa el número de la reserva y el nick del usuario que está solicitando el canje	
3.- El usuario utiliza la opción “Confirmar canje”	4.- El sistema verifica que el usuario indicado tenga una reserva con el número ingresado
	5.- El sistema confirma la reserva cambiando su estado a “Cumplida”

Curso alternativo:

Acción del actor	Acción del sistema
	4.- Si el sistema no encuentra el número de reserva entre las reservas del usuario indicado, se notificará de esta situación y finalizará el caso de uso

Llamada al servlet BuscarReservaServlet

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
		throws ServletException, IOException {
	processRequest(request, response);
	String idUsuario = request.getParameter("idUsuario");
	String idReserva = request.getParameter("idReserva");
	Tienda unaTienda = Tienda.getInstance();
	String errorBuscarReserva = null;
	try {
		Reserva reserva =
unaTienda.buscarUsuario(idUsuario).buscarReserva(Integer.parseInt(idReserva));
		if (reserva != null) {
			request.getSession().setAttribute("reservaEncontrada",
reserva);
		}
	} catch (NumberFormatException e) {
		errorBuscarReserva = "El número de la reserva debe ser un entero
positivo";
		request.getSession().setAttribute("errorBuscarReserva",
errorBuscarReserva);
	}
	response.sendRedirect("canje-reservas");
}
```

JavaScript que llama al siguiente servlet

```
/* Confirmar canje de una reserva */
$(this).on("click", "a.confirmar-canje", function () {
	var parametrosReserva = {
		idR: $('p.res').attr('id'),
		idU: $('p.user').attr('id')
	};
	$(this).parent().html("<p>Confirmando canje...<div class='progress'>
<div class='indeterminate'></div> </div></p>");
	$.post("confirmar-canje-reserva", $.param(parametrosReserva), function
(respuesta) {
		Materialize.toast(respuesta, 3000);
	});
});
```

Llamada al servlet ConfirmarCanjeReservaServlet (llamado por el JavaScript anterior)

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response)
		throws ServletException, IOException {
	processRequest(request, response);
	String idReserva = request.getParameter("idR");
	String idUsuario = request.getParameter("idU");

	Tienda unaTienda = Tienda.getInstance();
	String errorBuscarReserva = null;
	try {
		Reserva reserva = unaTienda.buscarUsuario(idUsuario)
			.buscarReserva(Integer.parseInt(idReserva));

		if (reserva != null) {
			if (reserva.getUnJuego().getStock().getCantidad() > 0)
{

	request.getSession().setAttribute("reservaEncontrada", reserva);

	reserva.setEstado(unaTienda.getUnaConfiguracion().getEstadoReserv
aCumplida());
				reserva.getUnJuego().getStock()

	.setCantidad(reserva.getUnJuego().getStock().getCantidad() - 1);
			}else{
				throw new JuegoException("No hay unidades de
juego disponible para canjear");
			}
		}
	} catch (NumberFormatException e) {
		errorBuscarReserva = "El número de la reserva debe ser un
entero positivo";
		request.getSession().setAttribute("errorBuscarReserva",
errorBuscarReserva);
	} catch (JuegoException ex) {
		errorBuscarReserva = ex.getMessage();
		request.getSession().setAttribute("errorBuscarReserva",
errorBuscarReserva);
	}

	Gson gson = new Gson();
	String mensaje = "¡Canje realizado con éxito!";
	mensaje = gson.toJson(mensaje);
	response.setContentType("application/json");
	response.setCharacterEncoding("UTF-8");
	response.getWriter().write(mensaje);
}
```

Caso de uso: “Agregar stock a un juego”

Caso de uso: “Agregar stock a un juego”
Objetivo: “Este caso de uso consiste en cambiar la cantidad de unidades en el stock de un juego”
Actores: Usuario empleado y demás usuarios registrados en el sistema que tenga permisos para modificar los datos asociados de un juego
Pre-condiciones: El usuario debe estar autenticado en el sistema y debe estar viendo los detalles del juego. El usuario también debe tener permisos para ejecutar el caso de uso.
Pos-condiciones: La cantidad de unidades en el stock de un juego fue modificada.

Curso típico de eventos

Acción del actor	Acción del sistema
1.- El caso de uso comienza cuando un usuario desea agregar stock en un juego	
2.- El usuario ingresa el número de unidades que quiere agregar al stock del juego	
3.- El usuario utiliza la opción “Sumar stock”	4.- El sistema verifica que el número de unidades a aumentar sea positivo
	5.- El sistema cambia la cantidad de unidades disponibles en el stock del juego

Curso alternativo:

Acción del actor	Acción del sistema
	4.- Si el sistema verifica que se ha ingresado un número negativo, se informa de esto al usuario y finalizará el caso de uso

Validación del lado del cliente durante ingreso de datos al intentar modificar el stock

```
/* Controla valor positivo del stock a agregar en un juego */
$('#formAumentarStock').on('submit', function(e){
	if ($('#editarStock').val() <= 0){
		e.preventDefault();
		Materialize.toast("Se debe ingresar un número positivo para poder modificar el stock", 4000);
	}
});
```

Llamada al servlet AumentarStockServlet

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
		throws ServletException, IOException {
	processRequest(request, response);

	Usuario usuario = (Usuario) request.getSession().getAttribute("miUsuario");
	Juego unJuego = (Juego) request.getSession().getAttribute("unJuego");
	String stock = request.getParameter("stock");
	String mensajeEdicionJuego = null;
	try {
		if (Integer.parseInt(stock) > 0) {

	unJuego.getStock().setCantidad(unJuego.getStock().getCantidad() +
				Integer.parseInt(stock));
			mensajeEdicionJuego = "Datos del juego modificados con éxito!";
		}else{
			throw new NumberFormatException
```

```
                        ("Se debe ingresar un número positivo para poder
modificar el stock");
            }
      } catch (NumberFormatException e) {
            mensajeEdicionJuego = "Se debe ingresar un número positivo
para poder modificar el stock";
      }
      request.getSession().setAttribute("mensajeEdicionJuego",
mensajeEdicionJuego);
      response.sendRedirect("../admin/verDetallesJuego?idJuego="+unJueg
o.getId());

}
```

Caso de uso: “Revisar reservas”

Caso de uso: “Revisar reservas”
Objetivo: “Este caso de uso consiste en revisar las reservas para determinar si aún son válidas o no”
Actores: El mismo sistema
Pre-condiciones:
Pos-condiciones:

Curso típico de eventos

Acción del actor	Acción del sistema
	1.- El caso de uso comienza cuando empieza un nuevo día según la hora del servidor.
	2.- Recolecta todas las reservas hechas por todos los usuarios
	3.- Revisa cada reserva: Si una reserva es válida o ya caducó o ya fue cumplida/confirmada no se hará nada. Pero si una reserva no es válida y no está cumplida, se cambiará su estado a “Caducada”.

Curso alternativo:

Acción del actor	Acción del sistema

Revisión de reservas automática (Servidor GlassFish)

```
package com.herokuapp.ggrosario.background.glassfish;

import com.herokuapp.ggrosario.modelo.Reserva;
import com.herokuapp.ggrosario.modelo.Tienda;
import com.herokuapp.ggrosario.modelo.Usuario;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.Schedule;
import javax.ejb.Singleton;

/**
 *
 * @author Ojeda Alberto Daniel
 */
@Singleton
public class ReservasChecker {

    @Schedule(hour = "0", minute = "0", second = "0", persistent =
false)
    public void actualizarEstadoReservas() {
        Tienda unaTienda = Tienda.getInstance();

        List<Reserva> reservas = new ArrayList<>();

        for (Usuario usuario : unaTienda.getUsuarios()) {
            for (Reserva reserva : usuario.getReservas()) {
                reservas.add(reserva);
            }
        }

        for (Reserva unaReserva : reservas) {
            if (unaReserva.isValida()) {
                System.out.println("La reserva #" + unaReserva.getId()
+ " aún es válida");
            } else if (!unaReserva.isCumplida()) {

unaReserva.setEstado(unaTienda.getUnaConfiguracion().getEstadoReservaCa
ducada());
```

```
                System.out.println("La reserva #" + unaReserva.getId()
+ " ha caducado");
            }
            if (unaReserva.isCaducada()) {
                System.out.println("La reserva #" + unaReserva.getId()
+ " ya ha caducado");
            }
            if (unaReserva.isCumplida()) {
                System.out.println("La reserva #" + unaReserva.getId()
+ " ha sido cumplida");
            }
        }
    }

}
```

Revisión de reservas automáticas (Servidor Tomcat)

```
package com.herokuapp.ggrosario.background.tomcat;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class BackgroundTasksManager implements ServletContextListener{

    private ScheduledExecutorService planificador;

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        this.planificador =
Executors.newSingleThreadScheduledExecutor();
        this.planificador.scheduleAtFixedRate(new ReservasChecker(), 0,
1, TimeUnit.DAYS);
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        this.planificador.shutdownNow();
    }

}
```

```
package com.herokuapp.ggrosario.background.tomcat;
import com.herokuapp.ggrosario.modelo.Reserva;
import com.herokuapp.ggrosario.modelo.Tienda;
import com.herokuapp.ggrosario.modelo.Usuario;
import java.util.ArrayList;
import java.util.List;

public class ReservasChecker implements Runnable {

    @Override
    public void run() {
        Tienda unaTienda = Tienda.getInstance();

        List<Reserva> reservas = new ArrayList<>();

        for (Usuario usuario : unaTienda.getUsuarios()) {
            for (Reserva reserva : usuario.getReservas()) {
                reservas.add(reserva);
            }
        }

        for (Reserva unaReserva : reservas) {
            if (unaReserva.isValida()){
                System.out.println("La reserva #" + unaReserva.getId()
+ " aún es válida");
            } else if (!unaReserva.isCumplida()) {
```

```
unaReserva.setEstado(unaTienda.getUnaConfiguracion().getEstadoReservaCa
ducada());
                System.out.println("La reserva #" + unaReserva.getId()
+ " ha caducado");
            }
            if (unaReserva.isCaducada()){
                System.out.println("La reserva #" + unaReserva.getId()
+ " ya ha caducado");
            }
            if (unaReserva.isCumplida()){
                System.out.println("La reserva #" + unaReserva.getId()
+ " ha sido cumplida");
            }
        }
    }

}
```