Informe práctica 2

i02abhak, i12argoi, i02fujuj, i02topap

Introducción

En esta práctica se ha desarrollado una aplicación web siguiendo el patrón MVC, utilizando **Java** como lenguaje principal e integrando una base de datos **MySQL** para la gestión persistente de datos. Se ha empleado *Eclipse IDE for Java Web Development* como entorno de desarrollo, aprovechando su compatibilidad con tecnologías web como **JSP** y **Servlets**, así como con herramientas específicas para la gestión de bases de datos.

El proyecto ha sido diseñado y probado en sistemas **Debian Bookworm**, **macOS** y **Windows 10/11**, garantizando su ejecución multiplataforma. Además, se ha empleado *Javadoc* para la generación automática de documentación del código fuente, y se ha configurado la conexión a la base de datos mediante *phpMyAdmin* y el controlador *JDBC*. Para ello, se utilizó la *VPN de la universidad* para establecer la conexión con el servidor de base de datos remoto.

En el desarrollo de la aplicación, se han empleado los siguientes componentes tecnológicos:

- **Vistas:** Ficheros *JSP* que combinan *HTML* para la estructura de las páginas con estilos definidos en *CSS*. Se han añadido validaciones del lado del cliente mediante *JavaScript*.
- **Controladores:** Para las funciones de inicio de sesión, registro de usuarios y modificación de datos de usuario se han utilizado controladores implementados en *Java*, sin necesidad de usar *Servlets* en estos casos específicos.
- **Servlets:** Encargados de gestionar otras partes de la lógica de la aplicación, asegurando una comunicación fluida entre las vistas y la capa de datos.
- **Modelo:** *JavaBeans* para encapsular los datos y facilitar su manipulación.

El **control de versiones** y el **seguimiento del proyecto** se han llevado a cabo a través de un repositorio en GitHub: `https://github.com/Kabhad/GM1_i02abhak_P3.git`.

El **mapa de navegación** puede encontrarse aquí fragmentado y comentado en base a las correspondientes partes del enunciado del proyecto, este mapa puede encontrarse completo tanto en el repositorio GitHub como en el .zip de la entrega.

Instrucciones de acceso:

Es posible acceder a la webapp a través del siguiente enlace: `http://localhost:8080/GM1_i02abhak_P3/`.

Para probar la aplicación, se han creado los siguientes usuarios con credenciales predefinidas:

- **Administrador:**
  Correo: `jperez@example.com`
  Contraseña: `Abc3e*`
- **Cliente con descuento de antigüedad:**
  Correo: `mgomez@example.com`
  Contraseña: `M@ar1A`
- **Cliente sin descuento de antigüedad:**
  Correo: `amartinez@example.com`
  Contraseña: `user1004`

Esquema relacional y DAO de la capa de datos

El esquema relacional de la base de datos utilizada en la aplicación se compone de las siguientes tablas y relaciones principales:

- **Jugador:** `(idJugador PK, nombreApellidos, fechaNacimiento, fechaInscripcion, correo, cuentaActiva, tipoUsuario, numeroReservasCompletadas, contrasena)`.
- **Pista:** `(idPista PK, nombre, disponible, exterior, tamanoPista, maxJugadores)`.
- **Material:** `(idMaterial PK, tipo, usoExterior, estado, idPista FK)` **FK:** `Material.idPista ->Pista.idPista`.
- **Bono:** `(idBono PK, idJugador, numeroSesion, fechaCaducidad)` **FK:** `Bono.idJugador ->Jugador.idJugador NOT NULL`.
- **Reserva:** `(idReserva PK, idJugador, idPista, fechaHora, duracionMin, precio, descuento, idBono FK)`
  FK: `Reserva.idJugador ->Jugador.idJugador NOT NULL`;
  FK: `Reserva.idPista ->Pista.idPista NOT NULL`;
  FK: `Reserva.idBono ->Bono.idBono`.
- **ReservaAdulto:** `(idReserva PK, numAdultos)`
  FK: `ReservaAdulto.idReserva ->Reserva.idReserva NOT NULL`.
- **ReservaFamiliar:** `(idReserva PK, numAdultos, numNinos)` **FK:** `ReservaFamiliar.idReserva ->Reserva.idReserva NOT NULL`.
- **ReservaInfantil:** `(idReserva PK, numNinos)`
  FK: `ReservaInfantil.idReserva ->Reserva.idReserva NOT NULL`.

Para interactuar con estas tablas, se implementaron los siguientes **DAO (Data Access Objects)**, que encapsulan

las operaciones de la base de datos:

JugadoresDAO

El `JugadoresDAO` gestiona las operaciones relacionadas con la tabla `Jugador`. Entre sus métodos principales destacan:

- `altaJugador(JugadorDTO nuevoJugador)`: Registra un nuevo jugador o reactiva uno existente si el correo ya está en uso.
- `bajaJugador(String correoElectronico)`: Desactiva la cuenta de un jugador.
- `autenticarJugador(String correo, String contrasena)`: Verifica las credenciales del jugador para permitir el acceso.
- `calcularReservasCompletadas(int idJugador)`: Calcula el número de reservas completadas por un jugador.
- `modificarJugador(String correoElectronico, String nuevoNombre, String nuevaContrasena)`: Permite actualizar el nombre y la contraseña de un jugador.
- `listarJugadores()`: Lista todos los jugadores activos en el sistema.

PistasDAO

El `PistasDAO` gestiona las operaciones relacionadas con las tablas `Pista` y `Material`. Métodos destacados:

- `crearPista(String nombre, boolean disponible, boolean exterior, TamanoPista pista, int maxJugadores)`: Añade una nueva pista a la base de datos.
- `modificarEstadoPista(int idPista, boolean disponible)`: Actualiza la disponibilidad de una pista.
- `buscarPistasPorTipoYFecha(TamanoPista tamano, Boolean exterior, Date fechaHora, int duracionMin)`: Encuentra pistas disponibles según criterios específicos.
- `asociarMaterialAPista(String nombrePista, int idMaterial)`: Asocia un material a una pista, verificando compatibilidad y condiciones.
- `listarPistas()`: Devuelve una lista completa de pistas con sus detalles.
- `obtenerTodosLosMateriales()`: Recupera todos los materiales registrados en la base de datos.
- `buscarMaterialPorId(int idMaterial)`: Encuentra un material por su identificador.

ReservasDAO

El `ReservasDAO` gestiona las operaciones relacionadas con las tablas `Reserva`, `ReservaAdulto`, `ReservaFamiliar`, `ReservaInfantil` y `Bono`. Entre sus métodos principales destacan:

- `insertarReserva(ReservaDTO reservaDTO)`: Inserta una nueva reserva en la base de datos.
- `actualizarReserva(int idReserva, Date nuevaFechaHora, int nuevaDuracionMinutos, float nuevoPrecio, float nuevoDescuento, int nuevaIdPista, Integer numeroAdultos, Integer numeroNinos)`: Actualiza los detalles de una reserva existente.
- `eliminarReserva(int idReserva)`: Elimina una reserva de la tabla principal y las tablas específicas asociadas.
- `consultarReservasFuturas()`: Recupera todas las reservas cuya fecha y hora están en el futuro.
- `obtenerReservaCompleta(int idReserva)`: Obtiene los detalles completos de una reserva por su ID.
- `hacerReservaIndividual(JugadorDTO jugadorDTO, Date fechaHora, int duracionMinutos, PistaDTO pistaDTO, int numeroAdultos, int numeroNinos)`: Realiza una nueva reserva individual.
- `hacerReservaBono(JugadorDTO jugadorDTO, Date fechaHora, int duracionMinutos, PistaDTO pistaDTO, int numeroAdultos, int numeroNinos)`: Realiza una nueva reserva utilizando un bono.

Ejercicio 1: Sistema de Autenticación (JSP)

Este ejercicio implementa las funcionalidades de registro y acceso de usuarios, siguiendo el patrón MVC simplificado y utilizando exclusivamente JSP tanto para los controladores como para las vistas. El flujo principal se detalla a continuación, apoyado en el fragmento del mapa de navegación correspondiente al sistema de autenticación:

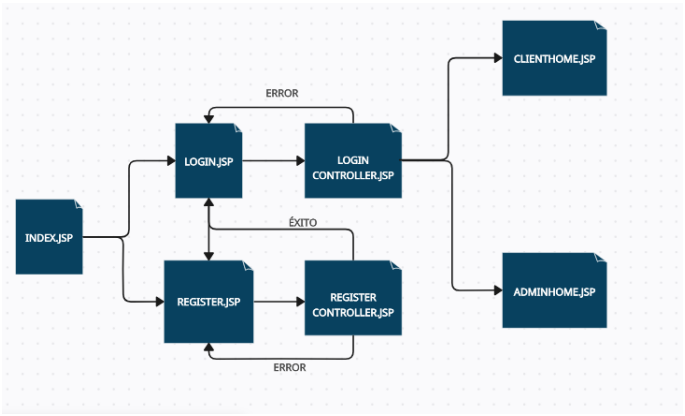

Figura 1: Mapa de navegación del sistema de autenticación.

Flujo de Navegación

- **Inicio** (`index.jsp`)**:** Al iniciar la aplicación, se presentan dos opciones principales: **Registrarse** y **Acceder**.
- **Registro** (`register.jsp` **y** `RegisterController.jsp`)**:**
  - Se muestra un formulario donde puede introducir sus datos.
  - El controlador `RegisterController.jsp` verifica que el correo electrónico proporcionado no exista en la base de datos.

- En caso de éxito, se crea el nuevo usuario en la base de datos con un tipo (`cliente` o `administrador`) y se redirige al inicio de sesión. Si ocurre algún error (p. ej., correo duplicado), se muestra un mensaje de error en `register.jsp`.

- **Acceso (`login.jsp` y `LoginController.jsp`):**
  - Se muestra un formulario donde introduce su correo y contraseña.
  - El controlador `LoginController.jsp` verifica las credenciales contra la base de datos.
  - En caso de éxito:
    - Si el usuario es un **cliente**, se redirige a `ClientHome.jsp`, donde se muestra:
      - Un mensaje de bienvenida personalizado.
      - Fecha de la primera reserva realizada (fecha de inscripción).
      - Fecha de la próxima reserva (si existe).
    - Si el usuario es un **administrador**, se redirige a `AdminHome.jsp`, donde se muestra:
      - Un listado con los clientes registrados, incluyendo sus fechas de inscripción y número de reservas completadas.
  - En caso de error, se redirige nuevamente a `login.jsp` mostrando un mensaje de error.

Opciones en las Páginas Principales

Tanto `ClientHome.jsp` como `AdminHome.jsp` incluyen las siguientes opciones:

- **Modificar Datos:** Permite modificar los datos del usuario excepto el correo (identificador único) y el tipo de usuario.
- **Desconectar:** Cierra la sesión del usuario actual y redirige a `index.jsp`.

Impacto en el Diseño de la Base de Datos

La implementación de este ejercicio requirió una actualización en el esquema relacional de la base de datos. En particular, se añadieron dos atributos clave a la tabla `Jugador`:

- **tipoUsuario:** Identifica si el usuario es `cliente` o `administrador`.
- **contrasena:** Almacena la contraseña cifrada del usuario para la validación durante el inicio de sesión.

Ejercicio 2

Ejercicio 2: Gestión de Pistas y Reservas (Servlets)

Este ejercicio abarca la implementación de todas las funcionalidades relacionadas con la gestión de pistas de baloncesto y las reservas, empleando exclusivamente *Servlets* para los controladores. Dependiendo del tipo de usuario (`administrador` o `cliente`), se otorgan permisos específicos mediante un sistema de control de acceso basado en ACL (Access Control List). Los flujos más complejos se presentan y explican a continuación, con ejemplos representativos tomados de los mapas de navegación correspondientes a cada rol.

Mapa de Navegación para Usuarios Cliente

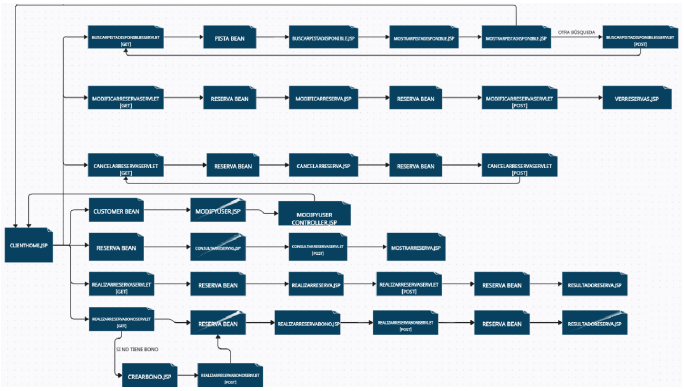

Figura 2: Mapa de navegación para usuarios cliente.

Flujo Representativo: Realizar Reserva con Bono

Uno de los flujos más complejos para los usuarios **cliente** es el de realizar una reserva utilizando un bono:

1. El usuario accede a `RealizarReservaBonoServlet` desde `ClientHome.jsp`.
2. Si no dispone de un bono válido, es redirigido a `CrearBono.jsp`, donde puede adquirir un bono y vincularlo a su cuenta.
3. En caso de tener un bono, puede seleccionar una pista compatible desde `MostrarReserva.jsp`, donde se muestran las opciones filtradas por tipo de pista, número de jugadores y disponibilidad.
4. Finalmente, la reserva se confirma mediante una llamada `POST` al `RealizarReservaBonoServlet`, que almacena la información en la base de datos y redirige a `ResultadoReserva.jsp` con un resumen de la operación.

Resumen de Funcionalidades Implementadas

- **Consultar reservas:** Implementado mediante un *formulario* en `ConsultarReservas.jsp` que permite especificar un rango de fechas. El servlet filtra las reservas según su estado (futuras o finalizadas) y muestra los resultados en `MostrarReservas.jsp`.
- **Cancelar reservas:** Mediante una tabla con las reservas del usuario en `VerReservas.jsp`, que incluye botones para cancelar reservas futuras. Esto llama al `CancelarReservaServlet`, que verifica las condiciones y actualiza la base de datos.
- **Modificar reservas:** Similar a la cancelación, pero accediendo a `ModificarReserva.jsp`, donde el usuario puede cambiar atributos como duración, número de jugadores y fecha.

- **Buscar pistas disponibles:** El cliente puede filtrar pistas activas según el tipo de pista (interior/exterior) y fecha, utilizando el `BuscarPistaDisponibleServlet`.
- **Adquirir bonos:** Los clientes pueden gestionar bonos desde `CrearBono.jsp`, asegurándose de cumplir las restricciones asociadas a sesiones y caducidad.

Mapa de Navegación para Usuarios Administradores

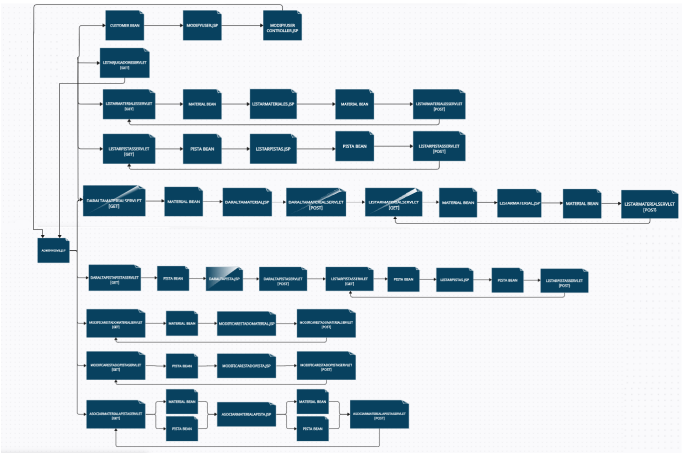

Figura 3: Mapa de navegación para usuarios administradores.

Flujo Representativo: Asociación de Materiales a Pistas

Entre las funcionalidades disponibles para los administradores, la asociación de materiales a pistas es una de las más detalladas:

1. Desde `AdminHome.jsp`, el administrador selecciona la opción `AsociarMaterialAPistaServlet`.
2. Se cargan los materiales disponibles mediante `MaterialBean` y se muestran en `AsociarMaterialAPista.jsp`.
3. El administrador selecciona un material y una pista, verificando las restricciones predefinidas (como el tipo de material y el estado de la pista).
4. Tras confirmar la asociación, el servlet actualiza las tablas de la base de datos y redirige a una página de confirmación.

Resumen de Funcionalidades Implementadas

- **Alta de materiales y pistas:** Gestionado mediante `DarAltaMaterialServlet` y `DarAltaPistaServlet`, que presentan formularios con validación de los datos requeridos.
- **Modificar estado:** Desde las vistas `ModificarMaterial.jsp` y `ModificarPista.jsp`, los administradores pueden actualizar el estado de los recursos mediante los servlets `ModificarMaterialServlet` y `ModificarPistaServlet`.
- **Eliminar reservas:** En `VerReservas.jsp`, el administrador puede eliminar reservas futuras llamando a `EliminarReservaServlet`, que libera los recursos asociados.
- **Listar materiales y pistas:** Utilizando `ListarMaterialesServlet` y `ListarPistasServlet`, los administradores pueden visualizar los recursos registrados y sus estados actuales.
- **Asociar materiales a pistas:** Implementado en `AsociarMaterialAPistaServlet`, respetando las restricciones sobre tipos y cantidades.

Consideraciones de Diseño e Implementación

- Los sistemas de consulta, alta y modificación siguen un flujo consistente, con formularios en vistas *JSP* y validaciones tanto del lado cliente como en los *Servlets*.
- Los servlets realizan operaciones directas sobre la base de datos a través de *DAO* especializados, asegurando un diseño modular y escalable.
- En lugar de un ACL básico, se desarrolló un **Access Control Framework (ACF)**. Este controla tanto el acceso según el rol del usuario como la autenticación en las diferentes partes del sistema:
  - Verifica la existencia de una sesión activa antes de permitir el acceso a cualquier recurso privado.
  - Gestiona rutas públicas (como `index.jsp` y controladores de autenticación) para garantizar accesibilidad sin sesión.
  - Diferencia permisos según el rol del usuario (`CLIENTE` o `ADMINISTRADOR`), restringiendo la navegación a vistas y funcionalidades específicas.
  - Proporciona redirecciones claras y mensajes de error personalizados en caso de accesos no autorizados.

Dificultades durante la realización de la práctica

Durante el desarrollo de la práctica, se enfrentaron las siguientes dificultades:

- **Fallo en validaciones para reservas con bono:** En `realizarReservaBono.jsp` faltaba incluir el script `realizarReservaValidation.js`. Además, en `ReservasDAO.java`, no se validaba correctamente el número máximo de jugadores al hacer una reserva con bono. La solución fue incluir el script faltante en la vista y añadir la validación de jugadores en el método del DAO.
- **Errores al realizar reservas con bono con una única sesión restante:** La vista `realizarReservaBono.jsp` no mostraba correctamente la información cuando un bono tenía solo una sesión restante. La solución fue corregir la lógica de la vista y añadir comentarios explicativos en el archivo JSP para mejorar la claridad del código.
- **Problemas al listar pistas debido a inconsistencias en la base de datos:** La consulta SQL

hacía referencia a una tabla `Pistas` que en la base de datos estaba nombrada como `Pista`. Se corrigió el archivo `sql.properties` para usar el nombre correcto de la tabla.

- **Error al buscar pistas disponibles con filtros específicos:** El filtro para buscar pistas de adultos y no exteriores devolvía resultados incorrectos. Se ajustaron las consultas en el DAO y se probó el filtro para garantizar resultados consistentes.
- **Problemas al cargar la lista de clientes en `adminHome.jsp`:** La lista de clientes no se cargaba correctamente al acceder al menú principal desde una página de error. Inicialmente, se intentó solucionar creando un servlet específico, pero esta solución fue revertida. Finalmente, se resolvió llamando al servlet `listarJugadores` antes de redirigir a `adminHome.jsp`, asegurando que los datos se cargaran correctamente.
- **Errores al modificar reservas con bonos:** Las reservas realizadas con bonos solo permitían modificar el número de participantes (adultos/niños) pero no otros campos. Se ajustaron las funcionalidades de modificación para permitir cambios en todos los campos relevantes y se actualizó la tabla para mostrar correctamente los datos.
- **Problemas con las validaciones en formularios:** Algunos formularios permitían realizar reservas sin completar datos obligatorios, como el número de niños o adultos. Se añadieron validaciones adicionales en los formularios y scripts JavaScript para prevenir datos incompletos.
- **Problemas al cancelar reservas:** La funcionalidad para cancelar reservas permitía eliminar reservas de otros usuarios. Se añadió validación para que solo el usuario autenticado pudiera cancelar sus propias reservas.
- **Falta de consistencia en el control de acceso (ACF):** Usuarios con roles de administrador podían acceder a vistas de clientes y viceversa. Se implementó un sistema de control de acceso que redirigía a la página de error si un usuario intentaba acceder a vistas que no correspondían a su rol.
- **Comentarios en vistas JSP y falta de Javadoc en algunas clases Java:** Aunque las vistas JSP incluyeron comentarios explicativos, algunas clases Java carecían de documentación con `Javadoc`. Se añadieron comentarios con `Javadoc` a las clases Java en las carpetas `es.uco.pw.business`, `es.uco.pw.data`, y `es.uco.pw.servlet`.
- **Errores con los estilos y navegación:** Algunas vistas no tenían estilos consistentes ni botones para regresar al menú principal. Se añadieron archivos CSS comunes y botones de navegación para mejorar la experiencia de usuario.
- **Problemas al exportar el proyecto a `.war`:** La exportación a un archivo `.war` fallaba debido a la falta de configuración adecuada de los servlets en el archivo `web.xml`. Se usaban anotaciones `@WebFilter` en lugar de declararlos explícitamente en el descriptor de despliegue. La solución fue modificar el archivo `web.xml` para incluir manualmente las etiquetas de los servlets y filtros necesarios.

Referencias

[1] MDN Web Docs. Javascript documentation, 2023. Accessed: 2024-12-20.

[2] Apache Software Foundation. Apache tomcat documentation, 2023. Accessed: 2024-12-18.

[3] Oracle. Httpservletrequest interface documentation, 2023. Accessed: 2024-12-01.

[4] Oracle. Httpsession interface documentation, 2023. Accessed: 2024-12-03.

[5] Oracle. Java servlet api documentation, 2023. Accessed: 2024-12-10.

[6] Oracle. Javabeans specification, 2023. Accessed: 2024-11-23.

[7] Oracle. javax.servlet.annotation package documentation, 2023. Accessed: 2024-12-06.

[8] Oracle. Servlet specification, 2023. Accessed: 2024-12-12.

[9] W3C. Css specifications, 2023. Accessed: 2024-12-16.

[10] W3C. Html5 specification, 2023. Accessed: 2024-11-29.

[2] [8] [5] [7] [3] [4] [6] [10] [9] [1]