|  |
| --- |
| Universidad veracruzana |
| Documento de plan de pruebas. |
| Desarrollo de Software. |
|  |
| **Víctor Javier García Mascareñas – Mario Hurtado López.** |
| **18/04/2018** |

|  |
| --- |
| El presente documento muestra el plan de pruebas para el proyecto de desarrollo de software, Ared espacio. |

Contenido

[1. Introducción. 2](#_Toc511809214)

[1.1. Objetivo general. 2](#_Toc511809215)

[1.2. Objetivos específicos 2](#_Toc511809216)

[2. Plantillas de pruebas. 3](#_Toc511809217)

[2.1. Modulo egresos. 3](#_Toc511809218)

[2.1.1. Clase EgresoDAOSql. 3](#_Toc511809219)

[2.2. Modulo sesiones. 6](#_Toc511809220)

[2.2.1. Clase UsuarioDAOSql. 6](#_Toc511809221)

[2.3. Modulo pagos. 8](#_Toc511809222)

[2.3.1. Clase PagoProfesorDAOSql. 8](#_Toc511809223)

[2.3.2. Clase promocionDAOSql. 10](#_Toc511809224)

[2.3.3. Clase PagoAlumnoDAOSql. 13](#_Toc511809225)

[2.4. Módulo catálogos. 15](#_Toc511809226)

[2.4.1. Clase AlumnoDAOSql. 15](#_Toc511809227)

[2.4.2. Clase clienteDAOSql. 18](#_Toc511809228)

[2.4.3. Clase ProfesorDAOSql 20](#_Toc511809229)

[2.5. Modulo grupos. 24](#_Toc511809230)

[2.5.1. Clase DiaDAOSql. 24](#_Toc511809231)

[2.5.2. Clase GrupoDAOSql. 26](#_Toc511809232)

# Introducción.

El proyecto de la asignatura desarrollo de Software para el que se realizó el presente plan de pruebas consiste en un centro de control para la escuela de danza Ared espacio, en esta existen diversas actividades como registro de estudiantes, egresos, pagos y grupos, y en conjunto la administración de todos estos (modificar y visualizar datos), dentro de este solo existen dos tipos de usuario, profesor y director, donde este último es un profesor con mayores privilegios de administración ¿ en el sistema

## Objetivo general.

El objetivo general del presente documento es validar el correcto funcionamiento de los módulos desarrollados en el proyecto.

## Objetivos específicos

1. Mostrar el correcto funcionamiento del sistema en diversas situaciones que podrían presentarse el flujo normal de la aplicación.
2. Mostrar como reaccionará el sistema cuando una situación irregular pudiera suceder.

# Plantillas de pruebas.

## Modulo egresos.

### Clase EgresoDAOSql.

#### Función RegistrarEgresos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarEgreso: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * Monto: 1200 * Descripción:   Una escoba.   * Fecha: 2012-12-12 | Registro valido. true | N/A. | Registro valido de un nuevo egreso. | Registro valido. true |
| * Monto: monto invalido * Descripción:   Una escoba.   * Fecha: 2012-12-12 | Registro no valido. false | N/A. | Notificación de un registro invalido. | Registro no valido. false |

#### Función EditarEgreso.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | editarEgreso: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * IdCosto: 3 * Monto: 1200 * Descripción:   Una escoba.   * Fecha: 2012-12-12 | Edición valida: true | N/A. | Edición correcta. | Edición valida: true |
| * IdCosto: 500 * Monto: monto * Descripción:   Una escoba.   * Fecha: 2012-12-12 | Edición no valida: false | N/A. | Edición incorrecta. | Edición no valida: false |

#### Función ObtenerEgresos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerEgresos: list | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
|  | List<egreso> | N/A. | Una lista con valores correctos de egresos. | List<Egreso> |

#### Función editarGasto.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | editarGasto: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * Descripción: módulo de prueba. Fecha inicio:2012-12-12, * fecha fin: 2012-12-12 * monto: "1200" * URL: "www.uv.mx" | Edición de gasto valido | N/A. | Una edición de gasto valida. | Edición de gasto valido |
| * New Gasto(); | Gasto invalido | N/A | Confirmación de que no puede registrarse el gasto. | gasto invalido. |

#### Función ObtenerGastos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerGastos : list | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
|  | List<Gasto> | N/A. | Una lista con valores correctos de gastos. | List<Gasto> |

#### Función RegistrarGasto.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarGasto: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * Descripción: módulo de prueba. Fecha inicio:2012-12-12, * fecha fin: 2012-12-12 * monto: "1200" * URL: "www.uv.mx" | Gasto valido. true | N/A. | Un registro de un profesor existente en el sistema. | Gasto valido. True |
| Gasto invalido | Gasto invalido false | N/A | Confirmación de que no pueden registrarse gastos sin datos. | Gasto invalido false |

## Modulo sesiones.

### Clase UsuarioDAOSql.

#### Función buscarUsuarioSesion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | buscarUsuarioSsion: Usuario | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * Nombre: jose * Contraseña: snape | Usuario valido. | N/A. | Usuario existente en el sistema con todos los atributos correspondientes. | Usuario valido. |
| * Nombre: Mario. * Contraseña: 123 | Usuario no valido. | N/A. | Un usuario nulo o confirmación de usuario no existente. | Usuario no valido. |
| * Nombre: jose. * Contraseña: snape | Identificador de usuario valido. | N/A | Un id de un usuario existente en el sistema. | Identificador de usuario valido. |
| * Nombre Gabriela * Contraseña: * Nemesis | Id de usuarios diferente. | N/A | Dos usuarios existentes no iguales. | Id de usuarios diferente. |

#### Función BuscarUsuario.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | buscarUsuario: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * Nombre: mario | False. | N/A. | Un usuario nulo o confirmación de usuario no existente. | False. |
| * Nombre: jose. | True. | N/A. | Un registro de un usuario existente. | True. |

#### Función CrearUsuario.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | crearUsuario: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * contraseña: 1234 * nombre: Antonio * idTipoUsuario: 1 * idUsuario: 1 * tipoUsuario: 1 | true | N/A. | Un registro exitoso de un nuevo usuario. | true. |

#### Función ObtenerProfesor.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerProfesor: Profesor | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * contraseña: nemesis * nombre: Gabriela | Profesor valido. | N/A. | Un registro de un profesor existente en el sistema. | Profesor valido. |
| * idTipoUsuario: -1 | Profesor invalido | N/A | Confirmación de que no existen los registros solicitados. | Profesor invalido. |

## Modulo pagos.

### Clase PagoProfesorDAOSql.

#### Función ObtenerPagos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerPagos(int idProfesor): List<PagoProfesor> | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| idProfesor: 1. | Ninguna excepción relativa a la persistencia. | No hay excepciones. | * El profesor con el id ‘1’ está registrado. * Existen o no pagos del profesor registrados. | Lista válida. |
| idProfesor: -1. | NullPointer-  Exception | NullPointer-  Exception | * El profesor con el id ‘-1’ no existe. | Lista vacía. |

#### Función RegistrarPago.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarPago(PagoProfesor pago, int idProfesor): boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| PagoProfesor:   * tipoPago: true. * monto: 900.5.   idProfesor: 1. | True. | N/A. | * PagoProfesor con todas sus propiedades establecidas correctamente. * El profesor con el id ‘1’ está registrado. | True. |
| PagoProfesor:   * tipoPago: true. * monto: 3t.   idProfesor: 1. | False. | N/A. | * PagoProfesor con un monto no válido. * El profesor con el id ´1´ está registrado. | False. |
| PagoProfesor:   * tipoPago: true. * monto: 900.5.   idProfesor: -1. | False. | N/A. | * El profesor con el id ‘-1’ no existe. | False. |

### Clase promocionDAOSql.

#### Función EditarPromocion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | editarPromocion(Promocion promocion): boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Promocion:   * Descripción: promoción de prueba. * idPromocion: 1 * idProfesor: 1 * Nombre: promoción de decuento editada. * Porcentaje: 400 | True. | N/A. | Se espera la adicción de una promoción existente. | True. |
| Promocion:   * Descripción: promoción de prueba. * idPromocion: 1 * idProfesor: 0 * Nombre: promoción de decuento editada.   Porcentaje: 20 | False. | N/A. | Se espera que no se cree un registro de pago con el id de un profesor inexistente. | False. |

#### Función RegistrarPromocion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarPromocion(Promocion promocion): boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * Nombre: promoción de prueba * idPromocion: 0 * idProfesor:   1   * Descripción: promoción de descuento. * Porcentaje: 20 | True. | N/A. | Se espera el registro correcto de una promoción con un id de profesor existente. | True. |
| * Nombre: promoción de prueba * idPromocion: 0 * idProfesor:   - 1   * Descripción: promoción de descuento.   Porcentaje: 20 | False. | N/A. | Se espera que no se cree un registro de una promoción con un id de un profesor no existente. | False. |

#### Función ObtenerPromociones.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | ObtenerPromociones(int idProfesor): list<promocion> | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * Id profesor: 1 | List <Promocion> | N/A. | Se espera una lista de promociones registradas por un profesor. | List <Promocion> |
| * Id profesor: 0 | NullPointer Exception. | N/A. | Se espera que se lance una excepción sobre un id de profesor que no existe. | NullPointer Exception. |

### Clase PagoAlumnoDAOSql.

#### Función RegistrarPagoAlumno.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarPago(PagoAlumno pago, idProfesor, int idAlumno, int idPromocion: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * Pago alumno: fecha: new Date(). idPago: 0. Monto: “500”   idTipoPago: 1   * idAlumno: 1 * idPromocion: 1 | True. | N/A. | Se espera un registro de un nuevo pago de alumno con todos los atributos pertenecientes. | True. |
| * Pago alumno: fecha: new Date(). idPago: 0. Monto: “no es un pago”   idTipoPago: 1   * idAlumno: 1 * idPromocion: 1 | False. | N/A. | Se espera que no se registre un pago de alumno con una cantidad no valida o con incoherencia de datos. | False. |
| * Pago alumno: fecha: new Date(). idPago: 0. Monto: “500”   idTipoPago: 1   * idAlumno: 1 * TidPromocion: 0 | True. | N/A. | Se espera el registro de un pago de alumno sin la propiedad de promoción. | True. |

#### Función RegistrarPago.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarPago(PagoAlumno pago, idProfesor, int idAlumno, int idPromocion: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * idAlumno. 1 | List <PagoAlumno> | N/A. | Se espera una lista de pagos de un alumno. | List <PagoAlumno> |
| * idAlumno: 0 | NullPointer Exception. | N/A. | Se espera que el proceso arroje una excepción de tipo null debido a la no existencia del alumno. | NullPointer Exception. |

#### Función ObtenerPromocion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerPromocion(int idPromocion): Promocion | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * idPromocion. 1 | Promocion. | N/A. | Para que esta prueba funcione debe existir en base de datos una promoción con ese id. | Promocion |
| * idPromocion: 0 | NullPointer Exception. | N/A. | Para que esta prueba funcione no debe existir en base de datos una promoción con ese id. | NullPointer Exception. |

## Módulo catálogos.

### Clase AlumnoDAOSql.

#### Función registrarAlumno.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarAlumno: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Alumno:   * **Nombre**: Alumno de prueba. * **Correo**: [alumno@prueba .clase](mailto:alumno@prueba.clase) * **Teléfono**: 2281917762. * **Dirección**: Dirección de prueba. * **Estado**: true. | N/A. | N/A. | Alumno con todas sus propiedades establecidas correctamente. | True. |
| Alumno:   * **Nombre**: Alumno de prueba. * **Correo**: @com.gmail * **Teléfono**: 2281917762. * **Dirección**: Dirección de prueba. * **Estado**: true. | N/A. | N/A. | Alumno con todas sus propiedades establecidas correctamente, pero un correo no permitido. | False. |
| Alumno:   * **Nombre**: Alumno de prueba. * **Correo**: [alumno@prueba .clase](mailto:alumno@prueba.clase) * **Teléfono**: 228262z221. * **Dirección**: Dirección de prueba. * **Estado**: true. | N/A. | N/A. | Alumno con todas sus propiedades establecidas correctamente, pero un teléfono no permitido. | False. |

#### Función editarAlumno.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | editarAlumno: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Alumno existente en base. | N/A. | N/A. | Alumno ya registrado en base. | True. |
| Alumno existente en base:   * Id: 0. | N/A. | N/A. | Alumno ya registrado en base, pero con un id inexistente. | False. |
| Alumno existente en base.   * Correo: kjh. | N/A. | N/A. | Alumno ya registrado en base, pero con un correo no válido. | False. |
| Alumno existente en base.   * Teléfono: 2281716s. | N/A. | N/A. | Alumno ya registrado en base, pero con un teléfono no válido. | False. |

#### Función obtenerAlumno.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerAlumno: List<Alumno> | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| N/A. | Ninguna excepción relativa a la persistencia. | No hay excepciones. | Existen o no alumnos registrados. | Lista válida. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerAlumno: List<Alumno> | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Nombre: victor. | Ninguna excepción relativa a la persistencia. | No hay excepciones. | Cadena de texto para búsqueda por nombre. Existen o no alumnos registrados. | Lista válida. |

#### Función obtenerAlumnos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | List<Alumno> obtenerAlumnos(int idGrupo) | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| idGrupo: 18. | Ninguna excepción relativa a la persistencia. | No hay excepciones. | El id del grupo puede hacer referencia a un grupo que existe o no. | Lista válida. |

### Clase clienteDAOSql.

#### Función registrarCliente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarCliente: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Cliente:   * **Nombre**: Cliente de prueba. * **Correo**: [cliente@prueba .clase](mailto:alumno@prueba.clase) * **Teléfono**: 2289365356. * **Dirección**: Dirección de prueba. | N/A. | N/A. | Cliente con todas sus propiedades establecidas correctamente. | True. |
| Cliente:   * **Nombre**: Cliente de prueba. * **Correo**: @com.gmail * **Teléfono**: 2289365356. * **Dirección**: Dirección de prueba. | N/A. | N/A. | Cliente con todas sus propiedades establecidas correctamente, pero un correo no permitido. | False. |
| Cliente:   * **Nombre**: Alumno de prueba. * **Correo**: [cliente@prueba .clase](mailto:alumno@prueba.clase) * **Teléfono**: 228262z221. * **Dirección**: Dirección de prueba. | N/A. | N/A. | Cliente con todas sus propiedades establecidas correctamente, pero un teléfono no permitido. | False. |

#### Función editarCliente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | editarCliente: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Cliente existente en base. | N/A. | N/A. | Cliente ya registrado en base. | True. |
| Cliente existente en base:   * Id: 0. | N/A. | N/A. | Cliente ya registrado en base, pero con un id inexistente. | False. |
| Cliente existente en base.   * Correo: vija.com. | N/A. | N/A. | Cliente ya registrado en base, pero con un correo no válido. | False. |
| Cliente existente en base.   * Teléfono: 1. | N/A. | N/A. | Cliente ya registrado en base, pero con un teléfono no válido. | False. |

#### Función obtenerClientes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerCientes: List<Cliente> | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| N/A. | Ninguna excepción relativa a la persistencia. | No hay excepciones. | Existen o no clientes registrados. | Lista válida. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerCientes: List<Cliente> | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Nombre: cli. | Ninguna excepción relativa a la persistencia. | No hay excepciones. | Cadena de texto para búsqueda por nombre. Existen o no clientes registrados. | Lista válida. |

### Clase ProfesorDAOSql

#### Función registrarProfesor.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | registrarProfesor: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Profesor:   * **Nombre**: Profesor de prueba. * **Correo**: profe@sor.clase * **Teléfono**: 2281916654. * **Dirección**: Dirección de prueba. * **Monto**: 1500. | N/A. | N/A. | Profesor con todas sus propiedades establecidas correctamente. | True. |
| Profesor:   * **Nombre**: Profesor de prueba. * **Correo**: lkjhlkj * **Teléfono**: 2281916654. * **Dirección**: Dirección de prueba. * **Monto**: 1500. | N/A. | N/A. | Cliente con todas sus propiedades establecidas correctamente, pero un correo no permitido. | False. |
| Cliente:   * **Nombre**: Profesor de prueba. * **Correo**: profe@sor.clase * **Teléfono**: lkjhlkj. * **Dirección**: Dirección de prueba. * **Monto**: 1500. | N/A. | N/A. | Cliente con todas sus propiedades establecidas correctamente, pero un teléfono no permitido. | False. |
| Profesor:   * **Nombre**: Profesor de prueba. * **Correo**: profe@sor.clase * **Teléfono**: 2281916654. * **Dirección**: Dirección de prueba. * **Monto**: 1iuy. | N/A. | N/A. | Profesor con todas sus propiedades establecidas correctamente, pero un monto no permitido. | False. |

#### Función editarProfesor.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | editarProfesor: boolean | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Profesor existente en base. | N/A. | N/A. | Profesor ya registrado en base. | True. |
| Profesor existente en base:   * Id: 0. | N/A. | N/A. | Profesor ya registrado en base, pero con un id inexistente. | False. |
| Profesor existente en base.   * Correo: s. | N/A. | N/A. | Profesor ya registrado en base, pero con un correo no válido. | False. |
| Profesor existente en base.   * Teléfono: 876. | N/A. | N/A. | Profesor ya registrado en base, pero con un teléfono no válido. | False. |
| Profesor existente en base.   * Monto: 1. | N/A. | N/A. | Profesor ya registrado en base, pero con un monto no válido. | False. |

#### Función obtenerProfesores.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerProfesores: List<Profesor> | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| N/A. | Ninguna excepción relativa a la persistencia. | No hay excepciones. | Existen o no profesores registrados. | Lista válida. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | obtenerProfesores: List<profesor> | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Nombre: angel. | Ninguna excepción relativa a la persistencia. | No hay excepciones. | Cadena de texto para búsqueda por nombre. Existen o no profesores registrados. | Lista válida. |

## Modulo grupos.

### Clase DiaDAOSql.

#### Función obtenerDias.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | List<Dia> obtenerDias(int idGrupo) | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| idGrupo: 18. | Ninguna excepción relativa a la persistencia, | Ninguna excepción. | El id del grupo corresponde a un grupo que existe o no. | Lista válida. |

#### Función agregarDia.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | boolean agregarDia(Dia dia) | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Día:   * Día: Lunes. * HoraFIn: 20:00. * HoraInicio: 19:00. * Id: 0. * idTipo: 5. * Salón: Y. * Tipo: true. | True. | True. | El día tienes establecidos todos sus atributos correctamente. | True. |

#### Función editarDia.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | boolean editarDia(Dia dia) | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Día:   * Día: Viernes. * HoraFIn: 09:30. * HoraInicio: 09:00. * Id: 35. * idTipo: 18. * Salón: X. * Tipo: true. | True. | True. | El día existe en base de datos y tiene sus atributos establecidos correctamente. | True. |
| Día:   * Día: Viernes. * HoraFIn: 09:30. * HoraInicio: 09:00. * Id: 0. * idTipo: 18. * Salón: X. * Tipo: true. | False. | False. | El día con el id 0 no existe. | False. |

### Clase GrupoDAOSql.

#### Función obtenerGrupos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | List<Grupo> obtenerGrupos() | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Ninguna. | Ninguna excepción relativa a la persistencia. | Ninguna excepción. | Existen o no grupos registrados. | Lista válida. |

#### Función registrarGrupo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | boolean registrarGrupo(Grupo grupo) | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Grupo:   * Nombre: Soy un grupo. * Danza: Un tipo de danza. * Id: 0. * Profesor: id: 1. * Dia:   + Dia: Miércoles.   + Id: 0.   + Inicio: 09:30.   + Fin: 11:00.   + Salón: X.   + Tipo: true.   + idTipo: 0. | True. | True. | * El grupo tiene sus atributos asignados correctamente * El día marcado tiene sus atributos correctos y no choca con otro horario en la base de datos. * El profesor con id 1 existe en base de datos. | True. |
| Grupo:   * Nombre: Soy un grupo. * Danza: Un tipo de danza. * Id: 0. * Profesor: id: 0. * Dia:   + Dia: Miércoles.   + Id: 0.   + Inicio: 09:30.   + Fin: 11:00.   + Salón: X.   + Tipo: true.   + idTipo: 0. | False. | False. | * El grupo tiene sus atributos asignados correctamente * El día marcado tiene sus atributos correctos y no choca con otro horario en la base de datos. * El profesor con id 0 no existe en base de datos. | False |
| Grupo:   * Nombre: Soy un grupo. * Danza: Un tipo de danza. * Id: 0. * Profesor: id: 0. * Dia:   + Dia: Lunes.   + Id: 0.   + Inicio: 04:00.   + Fin: 15:00.   + Salón: X.   + Tipo: true.   + idTipo: 0. | Horario Exception. | Horario Exception. | * El grupo tiene sus atributos asignados correctamente * El día marcado choca con otro horario en la base de datos. * El profesor con id 1 existe en base de datos. | Horario Exception. |

#### Función obtenerGruposProfesor.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | List<Grupo> obtenerGruposProfesor(int idProfesor) | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| * idProfesor: 1. | Ninguna excepción relativa a la persistencia. | Ninguna excepción. | * El profesor con id 1 existe en base de datos. | Lista válida. |
| * idProfesor: 0. | Null Pointer Exception. | Null Pointer Exception. | * El profesor con id 0 no existe. | Null Pointer Exception. |

#### Función obtenerGrupos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Función** | List<Grupo> obtenerGrupos() | | | |
| **Entradas** | **Salidas esperadas** | **Salidas obtenidas** | **Condiciones de salida** | **Resultado** |
| Ninguna. | Ninguna excepción relativa a la persistencia. | Ninguna excepción. | Existen o no grupos registrados. | Lista válida. |