**PRUEBAS UNITARIAS Clase GesSchoolDAOTest.java**

**“testGetAllAlumnos()”**

Esta prueba unitaria, testGetAllAlumnos, evalúa el método getAllAlumnos de la clase DaoAlumno. Para simular los datos de prueba, se crea una lista de objetos Alumno llamada listaAlumnosEsperada. En este caso, se agrega un único alumno con algunos atributos como idUsu, dniUsu, y nomCompUsu. Esta lista representa el resultado esperado que se debería obtener al llamar al método getAllAlumnos.

**testGetAlumnoById()**

Esta prueba verifica si el método getAlumnoById de la clase DaoAlumno devuelve el alumno esperado cuando se le pasa un ID específico como argumento. Si los atributos del alumno obtenido coinciden con los del alumno esperado, la prueba se considera exitosa.

**testGetAllAlumnosByEstudio()**

Esta prueba unitaria garantiza que el método getAllAlumnosByEstudio de la clase DaoAlumno recupere correctamente los datos de los alumnos por estudio y que estos datos coincidan con los esperados.

**testGetAllMatriculaciones ()**

Esta prueba unitaria garantiza que el método getAllMatriculaciones de la clase DaoAlumno recupere correctamente todas las matriculaciones y que estos datos coincidan con los esperados.

**testBorrarMatricula()**

Esta prueba unitaria garantiza que el método borrarMatricula de la clase DaoAlumno sea capaz de borrar una matrícula existente de la base de datos correctamente.

**testBorrarMatricula()**

Esta prueba unitaria quiere verificar si el método getEstudioById de la clase DaoEstudio retorna correctamente un objeto de tipo Estudio cuando se le proporciona un ID de estudio existente. Se espera que el método retorne un objeto de tipo Estudio con la información del estudio correspondiente al ID proporcionado.

**testGetAllEstudios ()**

En este caso, la prueba unitaria se asegura de que la lista de estudios no sea nula, que contenga al menos un estudio y que los atributos de al menos un estudio sean no nulos. Esto garantiza que el método getAllEstudios está recuperando datos de la base de datos y que esos datos son coherentes con la estructura esperada de la clase Estudio.

**testGetEstudioByMateria()**

En este caso, la prueba unitaria garantiza que el método getEstudioByMateria devuelve un estudio asociado a una materia específica. Se verifica que el estudio devuelto no sea nulo y que sus atributos sean coherentes con la estructura esperada de la clase Estudio.

**testGetAllAsignacionesMateriaEstudios()**

Esta prueba unitaria verifica que el método getAllAsignacionesMateriaEstudios devuelve una lista no nula y no vacía de asignaciones de materias a estudios. Además, verifica que cada asignación tenga los atributos esperados.

**testGetAsignacionMateriaYEstudio()**

Esta prueba unitaria verifica que el método getAsignacionMateriaYEstudio devuelve una asignación de materia y estudio no nula, y que sus atributos coinciden con los valores esperados.

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.junit.Assert.*assertFalse*;

**import** **static** org.junit.Assert.*assertNotEquals*;

**import** **static** org.junit.Assert.*assertNotNull*;

**import** **static** org.junit.Assert.*assertTrue*;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.junit.Test;

**import** com.dao.DaoAlumno;

**import** com.dao.DaoEstudio;

**import** com.logica.Alumno;

**import** com.logica.Estudio;

**import** com.logica.MateriaEnEstudio;

**import** com.logica.Matricula;

**public** **class** GesSchoolDAOTest {

@Test

**public** **void** testGetAllAlumnos() **throws** Exception {

// Simular datos de prueba

List<Alumno> listaAlumnosEsperada = **new** ArrayList<>();

Alumno alumnoEsperado = **new** Alumno();

alumnoEsperado.setIdUsu(1);

alumnoEsperado.setDniUsu("12345678A");

alumnoEsperado.setNomCompUsu("Juan Pérez");

// Agregar más atributos según sea necesario

listaAlumnosEsperada.add(alumnoEsperado);

// Crear instancia de DaoAlumno

DaoAlumno daoAlumno = **new** DaoAlumno();

// Llamar al método bajo prueba

List<Alumno> listaAlumnosObtenida = daoAlumno.getAllAlumnos();

// Verificar que se devuelva una lista con al menos un alumno

*assertEquals*(listaAlumnosEsperada.size(), listaAlumnosObtenida.size());

// Verificar que los datos del primer alumno sean los esperados

**if** (!listaAlumnosEsperada.isEmpty() && !listaAlumnosObtenida.isEmpty()) {

Alumno alumnoObtenido = listaAlumnosObtenida.get(0);

*assertEquals*(alumnoEsperado.getIdUsu(), alumnoObtenido.getIdUsu());

*assertEquals*(alumnoEsperado.getDniUsu(), alumnoObtenido.getDniUsu());

*assertEquals*(alumnoEsperado.getNomCompUsu(), alumnoObtenido.getNomCompUsu());

}

}

@Test

**public** **void** testGetAlumnoById() **throws** Exception {

// Simular datos de prueba

Alumno alumnoEsperado = **new** Alumno();

alumnoEsperado.setIdUsu(1);

alumnoEsperado.setDniUsu("12345678A");

alumnoEsperado.setNomCompUsu("Juan Pérez");

// Agregar más atributos según sea necesario

// Crear instancia de DaoAlumno

DaoAlumno daoAlumno = **new** DaoAlumno();

// Llamar al método bajo prueba

Alumno alumnoObtenido = daoAlumno.getAlumnoById(1); // Supongamos que queremos obtener el alumno con id 1

// Verificar que el alumno obtenido sea el esperado

*assertEquals*(alumnoEsperado.getIdUsu(), alumnoObtenido.getIdUsu());

*assertEquals*(alumnoEsperado.getDniUsu(), alumnoObtenido.getDniUsu());

*assertEquals*(alumnoEsperado.getNomCompUsu(), alumnoObtenido.getNomCompUsu());

}

@Test

**public** **void** testGetAllAlumnosByEstudio() **throws** Exception {

// Simular datos de prueba

**int** idAlumno = 1; // ID de un alumno ficticio

List<Alumno> listaAlumnosEsperada = **new** ArrayList<>();

Alumno alumnoEsperado = **new** Alumno();

alumnoEsperado.setIdUsu(1);

alumnoEsperado.setNomCompUsu("Juan Pérez");

alumnoEsperado.setEmailUsu("juan@example.com");

alumnoEsperado.setNomEst("Estudio 1");

// Agregar más atributos según sea necesario

listaAlumnosEsperada.add(alumnoEsperado);

// Crear instancia de DaoAlumno

DaoAlumno daoAlumno = **new** DaoAlumno();

// Llamar al método bajo prueba

List<Alumno> listaAlumnosObtenida = daoAlumno.getAllAlumnosByEstudio(idAlumno);

// Verificar que se devuelva una lista con al menos un alumno

*assertEquals*(listaAlumnosEsperada.size(), listaAlumnosObtenida.size());

// Verificar que los datos del primer alumno sean los esperados

**if** (!listaAlumnosEsperada.isEmpty() && !listaAlumnosObtenida.isEmpty()) {

Alumno alumnoObtenido = listaAlumnosObtenida.get(0);

*assertEquals*(alumnoEsperado.getIdUsu(), alumnoObtenido.getIdUsu());

*assertEquals*(alumnoEsperado.getNomCompUsu(), alumnoObtenido.getNomCompUsu());

*assertEquals*(alumnoEsperado.getEmailUsu(), alumnoObtenido.getEmailUsu());

*assertEquals*(alumnoEsperado.getNomEst(), alumnoObtenido.getNomEst());

}

}

@Test

**public** **void** testGetAllMatriculaciones() **throws** Exception {

// Simular datos de prueba

List<Matricula> matriculasEsperadas = **new** ArrayList<>();

Matricula matriculaEsperada = **new** Matricula();

matriculaEsperada.setIdAlu(1);

matriculaEsperada.setNomCompAlu("Juan Pérez");

matriculaEsperada.setIdEst(1);

matriculaEsperada.setNomEst("Estudio A");

// Agregar más atributos según sea necesario

matriculasEsperadas.add(matriculaEsperada);

// Crear instancia de DaoAlumno

DaoAlumno daoAlumno = **new** DaoAlumno();

// Llamar al método bajo prueba

List<Matricula> matriculasObtenidas = daoAlumno.getAllMatriculaciones();

// Verificar que la lista de matriculaciones obtenida no esté vacía

*assertFalse*(matriculasObtenidas.isEmpty());

// Verificar que los datos de la primera matrícula sean los esperados

**if** (!matriculasEsperadas.isEmpty() && !matriculasObtenidas.isEmpty()) {

Matricula matriculaObtenida = matriculasObtenidas.get(0);

*assertEquals*(matriculaEsperada.getIdAlu(), matriculaObtenida.getIdAlu());

*assertEquals*(matriculaEsperada.getNomCompAlu(), matriculaObtenida.getNomCompAlu());

*assertEquals*(matriculaEsperada.getIdEst(), matriculaObtenida.getIdEst());

*assertEquals*(matriculaEsperada.getNomEst(), matriculaObtenida.getNomEst());

}

}

@Test

**public** **void** testBorrarMatricula() **throws** Exception {

// Simular un id de matrícula existente en la base de datos

**int** idMatriculaExistente = 1;

// Crear instancia de DaoAlumno

DaoAlumno daoAlumno = **new** DaoAlumno();

// Llamar al método bajo prueba para borrar una matrícula existente

**boolean** borradoExitoso = daoAlumno.borrarMatricula(idMatriculaExistente);

// Verificar que el borrado haya sido exitoso

*assertTrue*(borradoExitoso);

}

@Test

**public** **void** testGetEstudioById() **throws** Exception {

// Simular un ID de estudio existente en la base de datos

**int** idEstudioExistente = 1;

// Crear instancia de DaoAlumno

DaoEstudio daoEstudio = **new** DaoEstudio();

// Llamar al método bajo prueba para obtener el estudio con el ID especificado

Estudio estudioObtenido = daoEstudio.getEstudioById(idEstudioExistente);

// Verificar que se obtenga un objeto de tipo Estudio

*assertNotNull*(estudioObtenido);

// Verificar que el ID del estudio obtenido sea el esperado

*assertEquals*(idEstudioExistente, estudioObtenido.getIdEst());

}

@Test

**public** **void** testGetAllEstudios() {

DaoEstudio daoEstudio = **new** DaoEstudio();

// Llamar al método para obtener la lista de estudios

List<Estudio> listaEstudios = daoEstudio.getAllEstudios();

// Verificar que la lista no sea nula

*assertNotNull*(listaEstudios);

// Verificar que la lista contenga al menos un estudio

*assertNotEquals*(0, listaEstudios.size());

// Verificar que los atributos de al menos un estudio sean los esperados

**for** (Estudio estudio : listaEstudios) {

*assertNotNull*(estudio.getIdEst());

*assertNotNull*(estudio.getNomEst());

*assertNotNull*(estudio.getEspeEst());

*assertNotNull*(estudio.getHoraEst());

*assertNotNull*(estudio.getObsEst());

}

}

@Test

**public** **void** testGetEstudioByMateria() {

DaoEstudio daoEstudio = **new** DaoEstudio();

// Definir una materia existente en la base de datos de prueba

String idMateria = "ID\_DE\_LA\_MATERIA"; // Reemplazar con el ID de una materia existente

// Llamar al método para obtener el estudio asociado a la materia

Estudio estudio = daoEstudio.getEstudioByMateria(idMateria);

// Verificar que el estudio no sea nulo

*assertNotNull*(estudio);

// Verificar que los atributos del estudio sean los esperados

*assertNotNull*(estudio.getIdEst());

*assertNotNull*(estudio.getNomEst());

}

@Test

**public** **void** testGetAllAsignacionesMateriaEstudios() {

DaoEstudio daoEstudio = **new** DaoEstudio();

// Llamar al método para obtener todas las asignaciones de materia a estudios

List<MateriaEnEstudio> asignaciones = daoEstudio.getAllAsignacionesMateriaEstudios();

// Verificar que la lista no sea nula

*assertNotNull*(asignaciones);

// Verificar que la lista no esté vacía

*assertFalse*(asignaciones.isEmpty());

// Verificar que cada asignación tenga los atributos esperados

**for** (MateriaEnEstudio asignacion : asignaciones) {

*assertNotNull*(asignacion.getIdEst());

*assertNotNull*(asignacion.getNomEst());

*assertNotNull*(asignacion.getIdMat());

*assertNotNull*(asignacion.getNomMat());

*assertNotNull*(asignacion.getObsMatEst());

}

}

}

**PRUEBAS DE INTEGRACIÓN Clase GesSchoolIntegrationTest.java**

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.junit.Assert.*assertNotNull*;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**import** org.junit.AfterClass;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

**import** com.dao.DaoAlumno;

**import** com.dao.DaoEstudio;

**import** com.logica.MateriaEnEstudio;

**public** **class** GesSchoolIntegrationTest {

**private** **static** **final** String ***URL*** = "jdbc:mysql://localhost:3306/db\_schoolges\_v1";

**private** **static** **final** String ***USER*** = "root";

**private** **static** **final** String ***PASSWORD*** = "123456";

**private** **static** Connection *connection*;

@BeforeClass

**public** **static** **void** setUp() **throws** SQLException {

// Establecer la conexión con la base de datos antes de ejecutar las pruebas

*connection* = DriverManager.*getConnection*(***URL***, ***USER***, ***PASSWORD***);

**new** DaoAlumno(*connection*);

}

@AfterClass

**public** **static** **void** tearDown() **throws** SQLException {

// Cerrar la conexión después de ejecutar todas las pruebas

**if** (*connection* != **null**) {

*connection*.close();

}

}

@Test

**public** **void** testGetAsignacionMateriaYEstudio() **throws** SQLException {

DaoEstudio daoEstudio = **new** DaoEstudio();

// Definir los IDs de estudio y materia para buscar la asignación

**int** idEst = 1;

**int** idMat = 1;

// Llamar al método para obtener la asignación de materia y estudio

MateriaEnEstudio asignacion = daoEstudio.getAsignacionMateriaYEstudio(idEst, idMat);

// Verificar que la asignación no sea nula

*assertNotNull*(asignacion);

// Verificar que los atributos de la asignación coincidan con los valores esperados

*assertEquals*(idEst, asignacion.getIdEst());

*assertEquals*(idMat, asignacion.getIdMat());

}

}

PREGUNTAS ACTIVIDAD 4. TAREA 3.

1. **¿Has podido cumplir la planificación? ¿Qué problemas han surgido?**

En general sí he podido cumplir la planificación, aunque hay cosas que me hubiera gustado mejorar e incluir más funcionalidades.

Me han surgido numerosos problemas entre ellos el usar servlet y con el servidor Tomcat en Java, ya que a la hora de ejecutar aparecen errores, que me han entorpecido el trabajo y tener que dedicar muchas horas a solventar errores e incluso el tener errores que ya no eran errores y a base de actualizar y volver a limpiar y construir desaparecían.

Por otro lado, no he utilizado ningún framework para el lado del backend y he usado Java puro lo cual me ha generado mayor dificultad a la hora de utilizar sentencias **SQL** en los métodos de clases DAO que se implementan en la aplicación para mostrar, actualizar, borrar y modificar tablas relacionadas entre sí.

1. **¿Su proyecto necesita algún tipo de permiso o autorización administrativa?**

No, solo necesita conocer el email y contraseña del usuario con el que se quiere conectar, hay 3 roles, admin, estudiante y docente para cada uno de ellos se muestran unas opciones de menú diferentes. En el manual de usuario se incluye una credencial de cada uno de ellos para que se pueda probar el código.

1. **¿Ha establecido algún documento de prevención de riesgos laborales?**

En el punto 5.1. de la planificación del proyecto, se presenta un detalle de aquellos riesgos que pueden existir o surgir a lo largo del proceso de desarrollo del proyecto, así como su nivel de impacto. A continuación se vuelven a detallar.

|  |  |
| --- | --- |
| **Riesgo** | **R01** |
| Nombre | **Incumplimiento de plazos** |
| Descripción | Proyecto no entregado a tiemplo, motivado por retrasos en la planificación y desarrollo. |
| Probabilidad | Baja. |
| Impacto | Alto |

|  |  |
| --- | --- |
| **Riesgo** | **R02** |
| Nombre | **Modificación de los términos y condiciones de la parte técnica** |
| Descripción | Aplicación de nuevas condiciones económicas en las herramientas tecnológicas utilizadas, donde se establezca un sistema de pago obligatorio. |
| Probabilidad | Baja. |
| Impacto | Medio |

|  |  |
| --- | --- |
| **Riesgo** | **R03** |
| Nombre | **Compatibilidad** |
| Descripción | Distintos comportamientos de la página web según navegador y/o sistema operativo. |
| Probabilidad | Baja. |
| Impacto | Medio |

|  |  |
| --- | --- |
| **Riesgo** | **R04** |
| Nombre | **Falta de acogida por parte de los usuarios finales** |
| Descripción | Dificultades en la aceptación del aplicativo web por no ser suficientemente intuitiva o de fácil acceso |
| Probabilidad | Media. |
| Impacto | Alto |

|  |  |
| --- | --- |
| **Riesgo** | **R05** |
| Nombre | **Seguridad en la página web** |
| Descripción | Deficiencias en la protección y/o encriptación de datos de usuarios, tratándose éstos de datos sensibles. |
| Probabilidad | Media. |
| Impacto | Alto |

|  |  |
| --- | --- |
| **Riesgo** | **R06** |
| Nombre | **Seguridad en la parte técnica** |
| Descripción | Detección de vulnerabilidades en herramientas de desarrollo utilizadas. |
| Probabilidad | Media. |
| Impacto | Medio |

|  |  |
| --- | --- |
| **Riesgo** | **R07** |
| Nombre | **Problemas técnicos durante la ejecución del proyecto** |
| Descripción | Surgimiento de incidencias durante la ejecución del proyecto con el hardware del equipo utilizado. |
| Probabilidad | Bajo. |
| Impacto | Medio |

1. **Realiza una valoración económica respecto a su ejecución.**

En el punto 5.2. Presupuesto de la planificación del proyecto se incluye una estimación de los costes, a continuación, se vuelve a incluir.

**Costes directos**

Los costes directos del proyecto se relacionan inevitablemente con los gastos del personal encargado del desarrollo del mismo incluyendo, por tanto, los salarios según el tiempo de dedicación. A continuación, en las tablas se presentan los costes según las tareas realizadas y el coste total. Se debe tener en cuenta que el desarrollador dedicará 1 o 2 horas al día a este proyecto y que el coste está calculado en base a 20€/hora.

|  |  |  |
| --- | --- | --- |
| Tareas | **HORAS** | **COSTE/HORAS** |
| Configuración de entornos | 5 | 60€ |
| Estructuración lógica del proyecto | 8 | 160€ |
| Diseño de formularios e interfaces | 20 | 400€ |
| Configuración de la base de datos | 4 | 80€ |
| Creaciones de consultas de creación e inserción | 5 | 100€ |
| Implementación lógica de negocio y algoritmos | 30 | 600€ |
| Pruebas en entorno de desarrollo | 5 | 100€ |
| Corrección de errores y bugs | 5 | 100€ |
| Puesta en producción | 1 | 20€ |
| **Total Presupuesto** | **81** | **1620** |

**Costes indirectos**

Hacen referencia a aquellos costes del proyecto que son imprescindibles para completar la ejecución, pero que no tienen que ver con los costes de personal.

En este caso no hay costes indirectos ya que las licencias de uso de software son gratuitas. El equipo hardware que se ha usado ha sido el que había disponible y no va a suponer un desembolso económico.

**Otros costes**

En este apartado entrarían aquellos gastos que deben ser tenidos en cuenta pero que no se relacionan de forma directa con el desarrollo de aplicaciones, como son los gastos de internet y luz. Respecto al internet, en este momento se dispone de tarifa plana y bono social para el consumo de luz, por lo que no se ha considerado valorar este aspecto.

Financiación

La financiación será acorde a la magnitud del proyecto y las tecnologías, personal y características del mismo. Al ser un proyecto que está dirigido por la propia alumna, por tanto, solo habrá esfuerzo personal y el factor humano no va a suponer ningún coste económico, pero en caso de que hubiera una persona al cargo de la aplicación, el presupuesto sería el indicado en el apartado Costes. En cuanto a las tecnologías usadas, todas son de libre uso por lo cual tampoco se necesitará un desembolso económico. Si en un futuro se decidiera mejorar dicho proyecto se podría solicitar algún tipo de subvención o ayuda para emprendedores digitales, como por ejemplo **MINECO**, que financia programas con el fin de impulsar el emprendimiento digital femenino y así reducir la brecha de género en este sector. También existe el programa de ayudas **kit digital** que forma parte del programa **Acelera Pyme**.