Plan de Pruebas

El plan de pruebas se va a realizar sobre los siguientes métodos y querys, indicando la persona que originalmente se encargaba de realizar los test correspondientes a dicho método, la que finalmente la implementa, la clase donde se encuentra junto con la línea (para su fácil localización) y por último la firma del mismo.

Encargado inicial	Encargado final	Ubicación - línea	Firma
Adrián	Adrián	UsuarioRepository	UsuarioRepository en su totalidad
Lara	Lara	ExcepcionRepository - 16	public Optional<Excepcion> buscarExistenciaTerminadaExcepcion(Asignatura Asig, Alumno alumno)
Jorge	Jorge	ExcepcionRepository - 20	public Optional<Excepcion> buscarExistenciaTerminadaExcepcionExc(Asignatura Asig, Alumno alumno)
Adrián	Adrián	ExcepcionRepository - 23	public Optional<Excepcion> buscarExistenciaTerminadaExcepcionConv(Asignatura Asig, Alumno alumno)
Jorge	Jorge	AsignaturaServicio -106	public List<Asignatura> asignaturasPorAlumno(Alumno alumno)
Lara	Lara y Jorge	HorarioServicio - 295	public List<Horario> cargarListadoTest(String nombre)
Adrián	Adrián	HorarioSerivicio - 134	public List<Horario> horariosPorAlumno(Alumno alumno, List<Ampliacion> listaAmpliaciones)
Jorge	Jorge	HorarioSerivicio - 196	public List<Horario> ordenar (List<Horario> lista)
Juan	Juan	HorarioServicio - 99	public boolean solapaHora(Horario horario)
Juan	Juan	HorarioServicio - 184	public List<Horario> listaTramo(List<Horario> lista, int dia)

Antes de comenzar describiendo los métodos y sus respectivos test, hay que mencionar dos cambios que se han realizado con respecto al proyecto original.

El primero es en ExcepcionRepository, donde se han cambiado las consultas buscarExistenciaTerminadaExcepcionExc y buscarExistenciaTerminadaExcepcionConv. La razón de su modificación es que antes se les pasaba el tipo de situación que era, dejando a las dos consultas exactamente idénticas, por lo que eran redundantes. El cambio consiste en que el tipo de situación se encuentra dentro de la consulta ,necesitando solo cómo parámetros a un alumno y una asignatura.

El segundo cambio es el método de la carga de datos a través de un CSV. En este caso se ha realizado una copia del método original, primero haciendo que devuelva la lista de entidades en vez de void (donde se recorría la lista para guardarla en el repositorio correspondiente).

El segundo cambio consiste en insertar un dato como primer parámetro en la línea de datos a tratar en el CSV y una vez recogidos en un array, ignorar ese dato. Por ejemplo, si inicialmente teníamos la siguiente estructura de datos en el csv (nombre_asignatura;nombre_curso;dia;tramo_horario) y en el método se encuentra el constructor de la clase horario (new Horario(Integer.parseInt(values[3]), Integer.parseInt(values[0]),asignaturaServicio.findByNameCurs(values[1], values[2]), true)), ahora pasa a ser respectivamente : (X;nombre_asignatura;nombre_curso;dia;tramo_horario) y (new Horario(Integer.parseInt(values[4]), Integer.parseInt(values[1]),asignaturaServicio.findByNameCurs(values[2], values[3]), true)).

Este cambio es debido a que el primer elemento de la lista del CSV, a nivel de información, por ejemplo el String “hola” , se muestra de forma correcta pero a la hora de hacer comparaciones, a nivel de byte, no son 4, los que corresponderían a la h, o, l y a, sino que inserta 3 más que hace que el método equals, a pesar de mostrar el mismo string, no sean iguales a nivel de bytes y no los reconoce como equivalentes.

ordenar - HorarioSerivicio - Jorge

El método consiste en ordenar una serie de horarios por día, del 1 al 5, donde tienen todos el mismo tramo horario. En el caso de que haya algún salto por falta del horario de un día en concreto, se creará un horario dummy con una asignatura con nombre X para su correcta visualización en un futuro.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
Ordena un horario completo	ordenarListaCompleta	Lista de horarios de 5 días desordenados	Lista de horarios de 5 días ordenados	Horario de 5 días ordenados	
Ordenar un horario al que le falta 1 día	ordenarListaparcial	Lista de horarios de 4 días desordenados	Lista de horarios de 4 días ordenados con el horario vacío donde corresponda	Horario de 4 días ordenados con el horario vacío donde corresponda	
Ordenar un horario al que le falta 3 días	ordenarListaparcial2	Lista de horariosde 2 días desordenados	Lista de horarios de los 2 días ordenados con los horario vacío donde corresponda	Horario de los 2 días ordenados con los horario vacío donde corresponda	
Ordenar un horario al que le falta los 5 días	ordenarListaVacia	Lista de horarios vacía	Lista de horarios compuesta por horarios vacíos	Todas las horas están compuestas por horarios vacíos	

Se obtiene una cobertura del 100% del código del método.

asignaturasPorAlumno - AsignaturaServicio - Jorge

El método consiste en obtener la lista de asignaturas de un alumno por curso. Para ello obtenemos primero todas las asignaturas del curso y luego le quitamos las que tenga aprobadas del curso anterior y que tenga las solicitudes de convalidación o exención aprobadas por el jefe de estudios.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
El alumno no tiene ninguna asignatura convalidada ni tiene una convalidación o exención	listaSinExcSinApro	Alumno	Lista de 4 asignaturas	Lista de 4 asignaturas	
El alumno tiene una convalidación o exención y ninguna aprobada del curso anterior	listaExcSinApro	Alumno	Lista de 3 asignaturas	Lista de 3 asignaturas	
El alumno una asignatura aprobada del curso anterior pero ninguna convalidación o exención	listaSinExcApro	Alumno	Lista de 3 asignaturas	Lista de 3 asignaturas	
Tiene una convalidación o exención y una asignatura aprobada del curso anterior	listaExcApro	Alumno	Lista de 2 asignaturas	Lista de 2 asignaturas	
Tiene varias convalidaciones o exenciones y ninguna aprobada del curso anterior	listaVarExc	Alumno	Lista de 2 asignaturas	Lista de 2 asignaturas	
Tiene varias convalidaciones o exenciones y una aprobada del curso anterior	listaVarExcApro	Alumno	Lista de 1 asignatura	Lista de 1 asignatura	

Se obtiene una cobertura del 100% del código del método.

buscarExistenciaTerminadaExcepcionExc - ExcepcionRepository - Jorge

La consulta consiste en buscar en todas las excepciones guardadas en el repositorio correspondiente las que pertenezcan al alumno y asignatura indicados y que el estado de la misma sea aceptado y que el tipo sea exención.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
Comprobar una lista vacía	comprobarVacio	Un alumno y asignatura	null	null	No se va a guardar ninguna excepción
Que haya una situación excepcional de tipo exención, aceptada, para el alumno y asignatura buscados	comprobarIndividual	Un alumno y asignatura	La excepción a buscar	La excepción a buscar	
Que haya varias y una sea situación excepcional de tipo exención y aceptada para el alumno y asignatura buscados	comprobarIndividualMult	Un alumno y asignatura	La excepción a buscar	La excepción a buscar	
Que haya varias y una sea situación excepcional de tipo exención y aceptada para el alumno, pero no para la asignatura buscada	comprobarNoValidAsign	Un alumno y asignatura	null	null	
Que haya varias y una sea situación excepcional aceptada para el alumno y	comprobarNoValidTipo	Un alumno y asignatura	null	null	

asignatura buscados pero el tipo es convalidación					
Que haya varias y una sea situación excepcional para el alumno y asignatura buscados pero el tipo está pendiente	comprobarVariasNoAcep	Un alumno y asignatura	null	null	
Que haya varias y más de sea situación excepcional de tipo exención y aceptada para el alumno y asignatura buscados	comprobarValidVar	Un alumno y varias asignatura	Las varias excepciones	Las varias excepciones	

****

buscarExistenciaTerminadaExcepcion - ExcepcionRepository - Lara

Como comento anteriormente Jorge, este método para buscar la Query ha sido retocado, aquí lo que deseamos es recoger todos los alumnos que tengan una excepción, independientemente del estado en la que se encuentre.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
En este caso comprobaremos que solo tenga 1 miembro	comprobarSoloUno	Un alumno y una asignatura	el alumno introducido	el alumno introducido	
En esta ocasión la lista estará vacía	comprobarListaVacia	Un alumno y una asignatura	null	null	
Introducimos varios alumnos y obtener solo los aceptados	ComprobarIndividualMult	1 alumno y varias asignaturas	la asignatura aceptada	asignatura aceptada	
Introducimos	comprobaEnc	un alumno y	null	null	

varias asignaturas y un alumno para comprobar si no la encuentra	uentraAsignatura	varias asignaturas			
comprobarVariasNoAcep	comprobarVariasNoAcerp	varias no asignaturas no aceptadas 1 solo alumno	null	null	

cargarListadoTest - HorarioServicio - Lara

En este método vamos a comprobar a cagar un listado de CSV, para asegurar que obtenemos los datos deseados hemos creado 5 CSV uno para cada prueba con distinto números de atributos en los miembros del CSV.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
vamos a introducir 1 miembro con un solo atributo	CSV1	miembro con 1 solo atributo	excepción que controle la casuística	excepción	
vamos a introducir 1 miembro con dos atributos	CSV2	miembro con 2 solo atributo	excepción que controle la casuística	excepción	
vamos a introducir 1 miembro con tres atributos	CSV3	miembro con 3 solo atributo	excepción que controle la casuística	excepción	
vamos a introducir 1 miembro con cuatro atributo	CSV4	miembro con 4 solo atributo	carga el CSV	Carga CSV	
vamos a introducir 1 miembro con cinco atributos	CSV5	miembro con 5 solo atributo	excepción que controle la casuística	excepción	

listaTramo - HorarioServicio - Juan

En este método pasamos una variable dia de tipo integer y una lista de tipo Horario el cual posee una variable tramo de tipo integer, una asignatura y una variable esAlta booleana, el método se encargara de cargar una lista con los horarios extraídos dependiendo del dia que especifiquemos.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
El método debe devolver una lista dependiendo del dia	listaHorarioconAlta	un array de tipo Horario, una variable de tipo integer	un array de tipo Horario	un array	
Le pasaremos una lista vacía y devolverá otra vacía	listaVacia	un array vacío de tipo Horario, una variable de tipo integer	un array vacío de tipo Horario	un array vacío	
Le pasaremos solo horarios de baja	listaHorariosdeBaja	un array de tipo Horario, una variable de tipo integer	un array de tipo Horario	un array	
Le pasaremos como dia un número negativo	listaHorarios	un array de tipo Horario, una variable de tipo integer	un array de tipo Horario	un array	

solapaHora - HorarioServicio - Juan

En este método pasaremos un objeto de tipo Horario que posee un tramo de tipo integer, una asignatura y una variable booleana, devolverá un true si es que el curso y título que contienen existen en la base de datos, caso contrario devolverá false.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
el método recibe un horario y esta debe comprobar si se encuentra en la base de datos	horarionoExistentedeAlta	un objeto de tipo Horario	False	False	
el método recibe un horario y esta	horarionoExistentedeBaja	un objeto de tipo Horario	False	False	

debe comprobar si se encuentra en la base de datos					
recibira un objeto horario de alta y los demás objetos que contenga de baja	horarionoExistentedeBajayAlta	un objeto de tipo Horario	False	False	

horariosPorAlumno - HorarioServicio - Adrián

En este método pasaremos un objeto de tipo Alumno que contiene un Curso con una Lista de asignaturas que componen las horas específicas del Curso. Prolongadamente en este método encontrará una asignatura distinta a la del Curso encontrada por ampliación de notas que añadirá esta asignatura al horario del alumno (El método devuelve un List<Horario>) que conforma el horario que tendrá durante una semana. También puede desaparecer esta asignatura si esa asignatura existe en una lista de Excepción ya sea porque se encuentre convalidada.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
Vamos a comprobar cual es el horario de un alumno por defecto sin ningún tipo de dato elaborado y con un curso que tenga una lista de asignatura vacía, además de una lista de ampliaciones de asignaturas que se encuentra vacía	horarioCursoAlumnoSinAsignaturasListaAmpliacionesVaciaListaVacia()	Un alumno con la lista de asignaturas del curso vacía y una List<Ampliacion> vacía	List<Ampliacion> vacía	List<Ampliacion> vacía	
Vamos a comprobar cual es el horario de un	horarioCursoAlumnoAsignaturasListaAmpliacionesVacia	Un alumno con la lista de asignaturas del curso llena	List<Ampliacion> con la 2 asignaturas, la del curso y	List<Ampliacion> con la 2 asignaturas, la del curso y	

alumno con un curso que contenga la misma asignatura que contenga una ampliación de otra asignatura y en estado aceptado que se espera que se añada su horario a una lista junto al horario de la asignatura que contiene el curso del alumno.	ListaLLena	y una List<Ampliacion> llena de una Ampliacion en estado “Aceptado”	la de la ampliación	la de la ampliación	
Vamos a comprobar cual es el horario de un alumno con un curso que contenga la misma asignatura y que no contenga ninguna ampliación de asignatura. Se espera que se añada su horario a una lista	horarioCursoAlumnoAsignaturasListaAmpliacionesVacia	Un alumno con la lista de asignaturas del curso llena y una List<Ampliacion> vacía”	List<Ampliacion> con la asignatura del curso	List<Ampliacion> con la asignatura del curso	
Vamos a comprobar cual es el horario de un alumno con un curso que contenga la misma asignatura que contenga una ampliación de	horarioCursoAlumnoAsignaturasListaAmpliacionesLLenaNoAprobado	Un alumno con la lista de asignaturas del curso llena y una List<Ampliacion> llena de una Ampliacion en estado “Diferente de aceptado”	List<Ampliacion> con la asignatura del curso	List<Ampliacion> con la asignatura del curso	

otra asignatura y en estado diferente a aceptado. Se espera que se añada a la lista solo el horario de la asignatura del curso.					
Vamos a comprobar cual es el horario de un alumno con un curso que contenga la misma asignatura que contenga una ampliación de otra asignatura y en estado aceptado. Se espera que se añada a la lista solo el horario de la ampliacion del curso.	horarioCursoAlumnoAsignaturasListaAmpliacionesLLenaExcepcionesPrevistas	Un alumno con la lista de asignaturas del curso llena y una List<Ampliacion> llena de una Ampliacion en estado “Aceptado”	List<Ampliacion> con la asignatura de la ampliación (D ebido a que se Mockea el metodo buscarExistenciaTerminadaExcepcion() de ExcepcionServicio para que devuelva la asignatura del Curso)	List<Ampliacion> con la asignatura de la ampliación	
Vamos a comprobar cual es el horario de un alumno con un curso que contenga la misma asignatura que no contenga ninguna ampliación. Se espera que no se añada nada.	horarioCursoAlumnoAsignaturasListaAmpliacionesVaciaExcepcionesPrevistas	Un alumno con la lista de asignaturas del curso llena y una List<Ampliacion> vacia	List<Ampliacion> vacía (Debido a que se Mockea el metodo buscarExistenciaTerminadaExcepcion() de ExcepcionServicio para que devuelva la asignatura del Curso)	List<Ampliacion> vacía (Debido a que se Mockea el metodo buscarExistenciaTerminadaExcepcion() de ExcepcionServicio para que devuelva la asignatura del Curso)	

buscarExistenciaTerminadaExcepcionExc - ExcepcionRepository - Adrián

La consulta consiste en buscar en todas las excepciones guardadas en el repositorio correspondiente las que pertenezcan al alumno y asignatura indicados y que el estado de la misma sea aceptado y que el tipo sea Convalidación.

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
Comprobar una lista vacía	comprobarVacio	Un alumno y asignatura	null	null	No se va a guardar ninguna excepción
Que haya una situación excepcional de tipo “Convalidación”, aceptada, para el alumno y asignatura buscados	comprobarIndividual	Un alumno y asignatura	La excepción a buscar	La excepción a buscar	
Que haya varias y una sea situación excepcional de tipo “Convalidación”y aceptada para el alumno y asignatura buscados	comprobarIndividualMult	Un alumno y asignatura	La excepción a buscar	La excepción a buscar	
Que haya varias y una sea situación excepcional de tipo “Convalidación”y aceptada para el alumno, pero no para la asignatura buscada	comprobarNoValidAsign	Un alumno y asignatura	null	null	
Que haya varias y una sea situación excepcional aceptada para el alumno y asignatura	comprobarNoValidTipo	Un alumno y asignatura	null	null	

buscados pero el tipo es convalidación					
Que haya varias y una sea situación excepcional para el alumno y asignatura buscados pero el tipo está pendiente	comprobarVariasNoAcep	Un alumno y asignatura	null	null	
Que haya varias y más de séa situación excepcional de tipo “Convalidación”y aceptada para el alumno y asignatura buscados	comprobarValidVar	Un alumno y varias asignatura	Las varias excepciones	Las varias excepciones	

UsuarioRepository - Clase entera con sus métodos básicos de un JpaRepository - Adrián

Casuística	Nombre del test	Valores de entrada	Valor esperado de salida	Valor final de salida	Comentarios
Probamos a guardar un alumno y comprobamos que si lo devuelve es porque lo ha guardado con éxito	probarGuardaAlumno()	Un Usuario que guarda UsuarioRepository	El mismo Usuario	El mismo Usuario	
Probamos a eliminar un alumno y comprobamos buscandolo en la lista porque lo ha borrado con	eliminarAlumno()	Un método de UsuarioRepository que busca por email	Optional.empty()	Optional.empty()	

éxito					
Probamos a encontrar todos los alumnos	encontrarTodosLosAlumnos()	Una List<Usuario> resultante de usuarioRepository.findAll()	List<Usuario> con los usuarios guardados	List<Usuario> con los usuarios guardados	
Probamos a encontrar un alumno por email	encontrarAlumnoPorEmail()	Un método de UsuarioRepository que busca por email	El usuario que contiene ese email	El usuario que contiene ese email	
Probamos a encontrar el alumno por email (estando el repositorio vacio)	encontrarAlumnoListaVaciaPorEmail()	Un método de UsuarioRepository que busca por email	Optional.empty()	Optional.empty()	
Probamos a encontrar el alumno de un email que es un string vacio	encontrarAlumnoPorEmailVacio()	Un método de UsuarioRepository que busca por email (Con una cadena vacía)	Optional.empty()	Optional.empty()	
Probamos a encontrar el alumno por su id	encontrarAlumnoPorId()	Un método de UsuarioRepository que busca por id (Y cogiendo el id del usuario como referencia)	Optional.of(Usuario)	Optional.of(Usuario)	
Probamos a eliminar todos de la lista	eliminarTodos()	Una List<Usuario> resultante de usuarioRepository.findAll()	List<Usuario> vacía	List<Usuario> vacía	