INFORME DE PRÁCTICAS

Repositorio de proxecto: https://github.com/carlasalgado/VVS
Participantes no proxecto: Carla Salgado López : Pedro Fernández Luces

Validación e Verificación de Software

1. Descrición do proxecto

El sitio Web permite visualizar en cada momento el conjunto de eventos deportivos sobre los que se pueden realizar consultas y comentarios.

De cada evento deportivo interesa almacenar su nombre (eg. Deportivo – Real Madrid), la fecha en la que tendrá lugar (día, mes, año, hora y minuto), la categoría a la que pertenece (eg. Fútbol, Baloncesto, Motor, etc.) y una pequeña reseña del mismo. En aras de la simplificación se asumirá un sistema de categorías plano, es decir, sin subcategorías.

La búsqueda de eventos, y la consulta de información asociada a los mismos, estarán accesibles para todos los usuarios sin necesidad de registro previo. Si un usuario desea realizar comentarios acerca de algún evento deportivo, deberá registrarse en el sitio Web especificando cierta información de registro.

La aplicación Web que se propone desarrollar consiste en un sitio que permite a múltiples usuarios intercambiar información acerca de eventos deportivos, que podría ser relevante de cara a las posibles apuestas sobre dichos eventos. Este sitio Web permitirá a los usuarios comentar aquellos eventos deportivos que todavía no hayan comenzado. Sólo los usuarios autenticados podrán añadir comentarios a los eventos deportivos.

De cara a facilitar el intercambio de información de interés, la aplicación permitirá la creación de grupos de usuarios con intereses comunes. Cualquier usuario autenticado podrá crear un nuevo grupo. Un usuario podrá formar parte de ninguno, uno o varios grupos y podrá recomendar eventos a cualquiera de los grupos a los que pertenece.

Un usuario se registra en la aplicación de comentarios sobre eventos deportivos especificando cierta información de registro, a saber, su seudónimo (login), contraseña, nombre, apellidos y dirección de correo electrónico. Opcionalmente podrá requerirle información relativa a su idioma y país para gestionar la internacionalización en base a las preferencias de usuario y no en base a la configuración del navegador.

2. Estado actual

Todos los casos de uso de la aplicación fueron desarrollados empleando la técnica de pair programming y probados por Carla Salgado López y Pedro Fernández Luces.

Las funcionalidades implementadas son las siguientes:

- **1. Registro de usuarios.** Debe permitir registrar nuevos usuarios, así como permitir actualizar la información de registro de los usuarios ya registrados.
- **2. Autenticación y salida.** Un usuario se autenticará indicando su seudónimo y contraseña, con la posibilidad de recordar la contraseña para no tener que introducirla la siguiente vez. El usuario podrá salir explícitamente de la aplicación, lo que provoca que ya no se recuerde su contraseña, en caso de haber seleccionado dicha opción con anterioridad.
- **3. Gestión de grupos de usuarios.** Un usuario autenticado podrá crear nuevos grupos. Un grupo consta de un nombre y una descripción. La aplicación dispondrá de

un enlace “Ver grupos” que mostrará todos los grupos existentes en la aplicación, proporcionando para cada uno de ellos el número de miembros del grupo, el número de recomendaciones dirigidas al grupo y un enlace para que el usuario pueda darse de alta en el grupo. Si el usuario ya pertenece al grupo, no se mostrará el enlace “Darse de alta”. Cualquier usuario podrá ver los grupos existentes, pero únicamente los usuarios autenticados podrán darse de alta en un grupo. La aplicación dispondrá también de un enlace “Mis grupos” que mostrará los grupos a los que pertenece un usuario, proporcionando un enlace “Darse de baja” por cada grupo, que permitirá eliminar la suscripción del usuario a ese grupo.

- **4. Búsqueda de eventos.** El usuario podrá buscar eventos deportivos por palabras clave del nombre del evento, especificando opcionalmente la categoría mediante un desplegable. Las palabras clave especificadas en la búsqueda tienen que estar todas contenidas en el nombre del evento como palabras o parte de palabras, sin distinguir entre mayúsculas y minúsculas, y en cualquier orden. Opcionalmente, el usuario podrá especificar una categoría, en cuyo caso, la búsqueda se restringirá a los eventos de dicha categoría. El resultado de la búsqueda mostrará para cada evento: nombre, categoría, fecha, un enlace para añadir un nuevo comentario y un enlace para ver los comentarios existentes. Por último se incluirá un enlace para recomendar el evento a un grupo.
- **5. Añadir comentario.** Un usuario puede añadir comentarios relativos a un evento. Si el usuario no estaba autenticado, cuando selecciona el enlace para añadir un comentario, se le redirige al formulario de autenticación, y tras autenticarse correctamente, se le muestra el formulario para añadir comentario. Un usuario también podrá modificar o eliminar los comentarios realizados por él mismo.
- **6. Ver comentarios de un evento.** A través del enlace para ver los comentarios de un evento se mostrarán, en una nueva página, los comentarios ordenados por la fecha de realización, apareciendo el más reciente primero (orden cronológico descendente). Por cada comentario se mostrará el seudónimo del usuario que lo realizó, la fecha en la que se insertó y el texto del comentario. Esta opción aparecerá disponible únicamente cuando un evento tenga comentarios asociados.
- **7. Recomendar un evento.** En la línea de las aplicaciones de carácter social un usuario podrá recomendar eventos a los miembros de un grupo. Para ello, se mostrarán los grupos a los que pertenece el usuario y se le permitirá seleccionar aquellos a los que desea recomendar el evento. Cada recomendación puede llevar un texto asociado, justificando la recomendación.
- **8. Mostrar recomendaciones.**La aplicación dispondrá de un enlace “Mostrar recomendaciones” que permitirá a cada usuario consultar las recomendaciones de eventos recibidas, en función de los grupos a los que pertenezca el usuario. El nombre del evento será un enlace para visualizar los comentarios asociados al mismo. Las recomendaciones se mostrarán ordenadas, de más reciente a más antigua.

2.1. Compoñentes avaliados

2.1.1. Comentario Dao

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**10

- **Número de pruebas preparadas:** 10
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.2. Etiqueta Dao

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**2
- **Número de pruebas preparadas:**2
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.3. Evento Dao

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**5
- **Número de pruebas preparadas:**5
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.4. Grupo Dao

- **Funcionalidades en las que participa:** Servicio eventos y servicio grupos.
- **Número de pruebas objetivo:**4
- **Número de pruebas preparadas:**4
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.5. Recomendacion Dao

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**2
- **Número de pruebas preparadas:**2
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.6. User Profile Dao

- **Funcionalidades en las que participa:** Servicio usuarios y servicio eventos.
- **Número de pruebas objetivo:**2
- **Número de pruebas preparadas:**2
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.7. Bloque Grupos

- **Funcionalidades en las que participa:** Servicio grupos.
- **Número de pruebas objetivo:**1
- **Número de pruebas preparadas:**1
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.8. Bloque Comentarios

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**3
- **Número de pruebas preparadas:**3
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.9. Bloque Eventos

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**2
- **Número de pruebas preparadas:**2
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.10. UserProfileDetails

- **Funcionalidades en las que participa:** Servicio usuarios.
- **Número de pruebas objetivo:**5
- **Número de pruebas preparadas:**5
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.11. Grupo DTO

- **Funcionalidades en las que participa:** Bloque grupos, servicio grupos
- **Número de pruebas objetivo:**2
- **Número de pruebas preparadas:**2
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.12. Comentario DTO

- **Funcionalidades en las que participa:** Bloque comentarios, servicio eventos.
- **Número de pruebas objetivo:**6
- **Número de pruebas preparadas:**6
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.13. Recomendacion DTO

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**2
- **Número de pruebas preparadas:**2
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.14. Evento DTO

- **Funcionalidades en las que participa:** Bloque eventos y servicio eventos.
- **Número de pruebas objetivo:**4
- **Número de pruebas preparadas:**4
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.15. Grupo Service

- **Funcionalidades en las que participa:** Ninguna funcionalidad.
- **Número de pruebas objetivo:**18
- **Número de pruebas preparadas:**18
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.16. User Service

- **Funcionalidades en las que participa:** Ninguna funcionalidad.
- **Número de pruebas objetivo:**11
- **Número de pruebas preparadas:**11
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.17. Evento Service

- **Funcionalidades en las que participa:** Ninguna funcionalidad.
- **Número de pruebas objetivo:** 51
- **Número de pruebas preparadas:** 51
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.18. SinGruposException

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**1
- **Número de pruebas preparadas:**1
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

2.1.19. UsuarioNoPropietarioException

- **Funcionalidades en las que participa:** Servicio eventos.
- **Número de pruebas objetivo:**2
- **Número de pruebas preparadas:**2
- **Porcentaje ejecutadas:** 100 %
- **Porcentaje superadas:** 100 %

Para este análisis, no hemos tenido en cuenta las pruebas de caja blanca y las pruebas de mutación, ya que han sido generadas automáticamente por las herramientas de Visual Studio, y no han sido diseñadas por nosotros.

3. Especificación de probas

3.1. Pruebas Unitarias

PR-UN-01

- Unidad: ComentarioDAO
- Función/método: verComentarios
- Motivación: comprobación de que devuelve los comentarios adecuados mediante particiones equivalentes y valores frontera para el número de resultados devueltos [<0, = 0, >0]
- Entradas:
  - identificador del evento del que se quiere ver los comentarios
  - 0 (índice del primer resultado de la búsqueda)
  - 10 (número máximo de resultados devueltos)
- Salida esperada:
  - lista de comentarios iguales a los asociados al evento
  - el mismo número de comentarios que los asociados al evento (10 como máximo)
- Inicialización del contexto: El evento tiene que existir en la base de datos con comentarios

PR-UN-02

- Unidad: ComentarioDAO
- Función/método: verComentarios
- Motivación: comprobación de que devuelve los comentarios adecuados mediante particiones equivalentes y valores frontera para el número de resultados devueltos [<0, = 0, >0]
- Entradas:
  - identificador del evento del que se quiere ver los comentarios
  - 0 (índice del primer resultado de la búsqueda)
  - 0 (número máximo de resultados devueltos)
- Salida esperada: se devuelven todos los comentarios del evento
- Inicialización del contexto: El evento tiene que existir en la base de datos con comentarios

PR-UN-03

- Unidad: ComentarioDAO
- Función/método: verComentarios
- Motivación: comprobación de que devuelve los comentarios adecuados mediante particiones equivalentes y valores frontera para el número de resultados devueltos [<0, = 0, >0]

- Entradas:
  - identificador del evento del que se quiere ver los comentarios
  - 0 (índice del primer resultado de la búsqueda)
  - -1 (número máximo de resultados devueltos)
- Salida esperada: se devuelven todos los comentarios del evento
- Inicialización del contexto: El evento tiene que existir en la base de datos con comentarios

PR-UN-04

- Unidad: ComentarioDAO
- Función/método: verComentarios
- Motivación: comprobación de que no devuelve los comentarios cuando el evento no tiene mediante particiones equivalentes
- Entradas:
  - identificador del evento del que se quiere ver los comentarios
  - 0 (índice del primer resultado de la búsqueda)
  - 0 (número máximo de resultados devueltos)
- Salida esperada: lista de comentarios vacía
- Inicialización del contexto: El evento tiene que existir en la base de datos con comentarios

PR-UN-05

- Unidad: ComentarioDAO
- Función/método: verComentarios
- Motivación: comprobación de que no devuelve los comentarios cuando el evento no existe mediante particiones equivalentes
- Entradas:
  - identificador del evento no existente
  - 0 (índice del primer resultado de la búsqueda)
  - 0 (número máximo de resultados devueltos)
- Salida esperada: lista de comentarios vacía
- Inicialización del contexto: El evento no existe en la base de datos

PR-UN-06

- Unidad: ComentarioDAO
- Función/método: verComentarios

- Motivación: comprobación de que devuelve el número de comentarios indicado mediante particiones equivalentes y valores frontera para el número de resultados devueltos[<0, = 0, >0]
- Entradas:
  - identificador del evento existente
  - 1 (índice del primer resultado de la búsqueda)
  - 1 (número máximo de resultados devueltos)
- Salida esperada: lista con 1 comentario habiendo más comentarios asociados al evento
- Inicialización del contexto: El evento tiene que existir en la base de datos con dos o más comentarios

PR-UN-07

- Unidad: ComentarioDAO
- Función/método: buscarPorUsuario
- Motivación: comprobación de que devuelve los comentarios adecuados mediante particiones equivalentes
- Entradas:
  - identificador del usuario del que se quiere ver los comentarios
  - identificador del evento del que se quiere ver los comentarios
- Salida esperada:
  - lista de comentarios iguales a los asociados al evento y escritos por el usuario indicado
  - el mismo número de comentarios que los asociados al evento y escritos por el usuario indicado
- Inicialización del contexto: un evento con un comentario asociado a un usuario en base de datos

PR-UN-08

- Unidad: ComentarioDAO
- Función/método: buscarPorUsuario
- Motivación: comprobación de que no devuelve comentarios cuando el evento no tiene comentarios de ese usuario mediante particiones equivalentes
- Entradas:
  - identificador del usuario del que se quiere ver los comentarios
  - identificador del evento del que se quiere ver los comentarios
- Salida esperada: lista de comentarios vacia.
- Inicialización del contexto: un evento sin comentarios asociados en base de datos y un usuario en base de datos

PR-UN-09

- Unidad: ComentarioDAO
- Función/método: buscarPorUsuario
- Motivación: comprobación de que no devuelve comentarios cuando el usuario no existe mediante particiones equivalentes
- Entradas:
  - identificador del usuario inexistente
  - identificador del evento del que se quiere ver los comentarios
- Salida esperada: lista de comentarios vacia.
- Inicialización del contexto: el usuario no existe en base de datos y el evento tiene comentarios

PR-UN-10

- Unidad: ComentarioDAO
- Función/método: buscarPorUsuario
- Motivación: comprobación de que no devuelve comentarios cuando el evento no existe mediante particiones equivalentes
- Entradas:
  - identificador del usuario del que se quiere ver los comentarios
  - identificador del evento inexistente
- Salida esperada: : lista de comentarios vacia.
- Inicialización del contexto: el evento no existe y el usuario tiene comentarios asociados

PR-UN-11

- Unidad: EtiquetaDAO
- Función/método: buscarEtiquetaPorNombre
- Motivación: comprobación de que devuelve la etiqueta adecuada mediante porciones equivalentes
- Entradas: nombre de la etiqueta
- Salida esperada: : la etiqueta buscada
- Inicialización del contexto: la etiqueta existe en base de datos

PR-UN-12

- Unidad: EtiquetaDAO
- Función/método: nubeEtiquetas

- Motivación: comprobación de que devuelve la lista de etiquetas ordenada mediante porciones equivalentes
- Entradas: -
- Salida esperada:
  - lista de etiquetas ordenadas
  - aparecen todas las etiquetas
- Inicialización del contexto: 4 con nombres que coinciden parcialmente con las palabras buscadas y 1 no

PR-UN-13

- Unidad: EventoDAO
- Función/método: buscarEventos
- Motivación: comprobación de que devuelve los eventos que coinciden y la paginación, mediante porciones equivalentes
- Entradas:
  - palabras clave mediante las cuales queremos buscar los eventos
  - 1 (índice del segundo resultado de la búsqueda, obviamos el primero)
  - 2 (número máximo de resultados devueltos)
- Salida esperada: lista con 2 eventos
- Inicialización del contexto: 4 eventos con nombres que coinciden parcialmente con las palabras buscadas y 1 no.

PR-UN-14

- Unidad: EventoDAO
- Función/método: buscarEventosSinPalabrasClave
- Motivación: comprobación de que devuelve todos los eventos que coinciden y la paginación mediante porciones equivalentes
- Entradas:
  - 0 (índice del primer resultado de la búsqueda)
  - 0 (número máximo de resultados devueltos)
- Salida esperada: todos los eventos que coinciden
- Inicialización del contexto: 4 con nombres que coinciden parcialmente con las palabras buscadas y 1 no

PR-UN-15

- Unidad: EventoDAO
- Función/método: buscarEventosSinPalabrasClave

- Motivación: comprobación de que devuelve todos los eventos mediante porciones equivalentes
- Entradas:
  - 0 (índice del primer resultado de la búsqueda)
  - 10 (número máximo de resultados devueltos)
- Salida esperada:
  - lista de eventos
  - el mismo número de eventos que los creados (10 como máximo)
- Inicialización del contexto: 4 con nombres que coinciden parcialmente con las palabras buscadas y 1 no

PR-UN-16

- Unidad: EventoDAO
- Función/método: buscarEventosSinPalabrasClave
- Motivación: comprobación de que devuelve todos los eventos mediante porciones equivalentes
- Entradas:
  - 0 (índice del primer resultado de la búsqueda)
  - 0 (número máximo de resultados devueltos)
- Salida esperada: todos los eventos
- Inicialización del contexto: 4 con nombres que coinciden parcialmente con las palabras buscadas y 1 no

PR-UN-17

- Unidad: EventoDAO
- Función/método: buscarEventosSinPalabrasClave
- Motivación: comprobación de que devuelve todos los eventos mediante porciones equivalentes
- Entradas:
  - 0 (índice del primer resultado de la búsqueda)
  - -1 (número máximo de resultados devueltos)
- Salida esperada: todos los eventos
- Inicialización del contexto: 4 con nombres que coinciden parcialmente con las palabras buscadas y 1 no

PR-UN-18

- Unidad: GrupoDAO

- Función/método: mostrarGrupos
- Motivación: comprobación de que devuelve todos los grupos mediante porciones equivalentes
- Entradas:
  - 0 (índice del primer resultado de la búsqueda)
  - 10 (número máximo de resultados devueltos)
- Salida esperada:
  - lista de grupos
  - el mismo número de grupos que los creados (10 como máximo)
- Inicialización del contexto: 2 grupos en base de datos

PR-UN-19

- Unidad: GrupoDAO
- Función/método: buscarPorNombre
- Motivación: comprobación de que devuelve el grupo que coincide por nombre mediante porciones equivalentes
- Entradas: nombre del grupo
- Salida esperada: el grupo cuyo nombre coincide con el de ebúsqueda
- Inicialización del contexto: grupo buscado existente en base de datos

PR-UN-20

- Unidad: GrupoDAO
- Función/método: buscarPorNombre
- Motivación: comprobación de que si un grupo no existente se lanza una excepción mediante porciones equivalentes
- Entradas: grupo no existente
- Salida esperada: Error: InstanceNotFoundException
- Inicialización del contexto: grupo buscado no existente en base de datos

PR-UN-21

- Unidad: GrupoDAO
- Función/método: buscarPorUsuario
- Motivación: comprobación de que devuelve todos los grupos de un usuario mediante porciones equivalentes
- Entradas: identificador del usuario
- Salida esperada: lista de todos los grupos pertenecientes al usuario

- Inicialización del contexto: usuario asociado a un grupo en base de datos

PR-UN-22

- Unidad: RecomendacionDAO
- Función/método: buscarRecomendacion
- Motivación: comprobación de que devuelve cierto cuando se efectuó una recomendación de un evento a un grupo mediante porciones equivalentes
- Entradas:
  - identificador del grupo
  - identificador del evento
- Salida esperada: cierto que se recomendó al grupo el evento
- Inicialización del contexto: una recomendación del evento indicado al grupo indicado, en base de datos

PR-UN-23

- Unidad: RecomendacionDAO
- Función/método: buscarRecomendacion
- Motivación: comprobación de que devuelve falso cuando no se efectuó una recomendación de un evento a un grupo mediante porciones equivalentes
- Entradas:
  - identificador del grupo
  - identificador del evento
- Salida esperada: falso que se recomendó al grupo el evento
- Inicialización del contexto: el grupo indicado tiene recomendaciones al evento indicado en base de datos

PR-UN-24

- Unidad: UserProfileDAO
- Función/método: findByLoginName
- Motivación: comprobación de que dado un login de usuario se obtiene el perfil del usuario asociado mediante porciones equivalentes
- Entradas: login de usuario registrado
- Salida esperada: perfil de usuario correspondiente
- Inicialización del contexto: el login indicado existe en base de datos

PR-UN-25

- Unidad: UserProfileDAO

- Función/método: findByLoginName
- Motivación: comprobación de que dado un login no existente se lanza una excepción mediante porciones equivalentes
- Entradas: login de usuario no registrado
- Salida esperada: Error: InstanceNotFoundException
- Inicialización del contexto: el login indicado no existe en base de datos

Esta solución web emplea un contenedor unity para resolver dependencias entre clases, esto provoca una complejidad elevada a la hora de aplicar Mocks para realizar las pruebas unitarias. Por ese motivo, no se han creado pruebas unitarias de los servicios.

3.2. Pruebas de Integración

PR-IN-01

- Unidad: EventoService
- Función/método: añadirComentario
- Motivación: comprobación de que se añaden correctamente comentarios a eventos mediante porciones equivalentes
- Entradas:
  - identificador de usuario que realiza el comentario
  - identificador de evento sobre el que se realiza el comentario
  - comentario
- Salida esperada:
  - el comentario coincide con el texto escrito
  - el usuario indicado coincide con el que indica el comentario
  - el evento indicado coincide con el que indica el comentario
- Inicialización del contexto: usuario y eventos indicados existentes en base de datos

PR-IN-02

- Unidad: EventoService
- Función/método: añadirComentario
- Motivación: comprobación de que se lanza una excepción cuando el usuario no existe mediante porciones equivalentes
- Entradas:
  - identificador de usuario no existente
  - identificador de evento
  - comentario
- Salida esperada:

    - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado no existente y evento indicado existente en base de datos

PR-IN-03

- Unidad: EventoService
- Función/método: añadirComentario
- Motivación: comprobación de que se lanza una excepción cuando el evento no existe mediante porciones equivalentes
- Entradas:
    - identificador de usuario
    - identificador de evento no existente
    - comentario
- Salida esperada:
    - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado existente y evento no existente en base de datos

PR-IN-04

- Unidad: EventoService
- Función/método: modificarComentario
- Motivación: comprobación de que se modifica correctamente un comentario mediante porciones equivalentes
- Entradas:
    - identificador de comentario existente
    - identificador de usuario que realizó el comentario
    - comentario modificado
- Salida esperada:
    - el comentario contenga el nuevo texto
    - el usuario que escribió el comentario siga siendo el mismo
    - el comentario sigue perteneciendo al evento al que fue asociado
- Inicialización del contexto: comentario indicado existente y asociado al usuario indicado y existente en base de datos

PR-IN-05

- Unidad: EventoService
- Función/método: modificarComentario

- Motivación: comprobación de que se lanza un error cuando el usuario no existe mediante porciones equivalentes
- Entradas:
  - identificador de comentario existente
  - identificador de usuario no existente
  - comentario modificado
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: comentario indicado existente y usuario indicado inexistente en base de datos

PR-IN-06

- Unidad: EventoService
- Función/método: modificarComentario
- Motivación: comprobación de que se lanza correctamente un error cuando el comentario no existe mediante porciones equivalentes
- Entradas:
  - identificador de comentario inexistente
  - identificador de usuario que realizó el comentario
  - comentario modificado
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: comentario indicado inexistente y usuario indicado existente en base de datos

PR-IN-07

- Unidad: EventoService
- Función/método: modificarComentario
- Motivación: comprobación de que lanza un error cuando el usuario que modifica el comentario no es el autor del mismo mediante porciones equivalentes
- Entradas:
  - identificador de comentario existente
  - identificador de usuario existente pero que no es el autor del comentario
  - comentario modificado
- Salida esperada:
  - Error: UsuarioNoPropietarioException

Inicialización del contexto: comentario indicado existente pero no asociado al usuario indicado y existente en base de datos

PR-IN-08

- Unidad: EventoService
- Función/método: eliminarComentario
- Motivación: comprobación de que se elimina correctamente un comentario mediante porciones equivalentes
- Entradas:
  - identificador de comentario existente
  - identificador de usuario que realizó el comentario
- Salida esperada:
  - Al buscar el comentario: Error: InstanceNotFoundException
- Inicialización del contexto: comentario indicado existente y asociado al usuario indicado y existente en base de datos

PR-IN-09

- Unidad: EventoService
- Función/método: eliminarComentario
- Motivación: comprobación de que se lanza un error cuando el usuario no es el que creó el comentario mediante porciones equivalentes
- Entradas:
  - identificador de comentario existente
  - identificador de usuario que existe pero no realizó el comentario
- Salida esperada:
  - Error: UsuarioNoPropietarioException
- Inicialización del contexto: comentario indicado existente pero no asociado al usuario indicado y existente en base de datos

PR-IN-10

- Unidad: EventoService
- Función/método: eliminarComentario
- Motivación: comprobación de que se lanza un error cuando el usuario no existe mediante porciones equivalentes
- Entradas:
  - identificador de comentario existente
  - identificador de usuario inexistente

- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: comentario indicado existente y usuario indicado inexistente en base de datos

PR-IN-11

- Unidad: EventoService
- Función/método: eliminarComentario
- Motivación: comprobación de que se lanza un error cuando el comentario no existe mediante porciones equivalentes
- Entradas:
  - identificador de comentario inexistente
  - identificador de usuario que realizó el comentario
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: comentario indicado inexistente y usuario indicado existente en base de datos

PR-IN-12

- Unidad: EventoService
- Función/método: verComentarios
- Motivación: comprobación de que se muestran correctamente los comentarios de un evento mediante porciones equivalentes
- Entradas:
  - identificador de evento existente
  - 0 (índice del primer elemento a devolver)
  - 0 (número de elementos devueltos)
- Salida esperada: todos los comentarios del evento y no existen más comentarios
- Inicialización del contexto: evento indicado existente con comentarios asociados

PR-IN-13

- Unidad: EventoService
- Función/método: verComentarios
- Motivación: comprobación de que se muestran correctamente los comentarios de un evento mediante porciones equivalentes
- Entradas:
  - identificador de evento existente

    - 0 (índice del primer elemento a devolver)
    - 1 (número de elementos devueltos)
- Salida esperada: 1 comentario del evento y existen más eventos
- Inicialización del contexto: evento indicado existente con más de un comentarios asociados

PR-IN-14

- Unidad: EventoService
- Función/método: verComentarios
- Motivación: comprobación de que no se obtienen comentarios de un evento en el que no se han realizado comentarios mediante porciones equivalentes
- Entradas:
    - identificador de evento existente
    - 0 (índice del primer elemento a devolver)
    - 0 (número de elementos devueltos)
- Salida esperada: ningún comentario
- Inicialización del contexto: eventoi indicado existente sin comentarios asociados

PR-IN-15

- Unidad: EventoService
- Función/método: verComentario
- Motivación: comprobación de que se lanza un error cuando el evento no existe mediante porciones equivalentes
- Entradas: identificador de evento inexistente
- Salida esperada:
    - Error: InstanceNotFoundException
- Inicialización del contexto: evento indicado inexistente en base de datos

PR-IN-16

- Unidad: EventoService
- Función/método: buscarComentario
- Motivación: comprobación de que se obtiene el comentario buscado mediante porciones equivalentes
- Entradas: identificador de comentario existente
- Salida esperada: DTO del comentario
- Inicialización del contexto: comentario indicado existente en base de datos

PR-IN-17

- Unidad: EventoService
- Función/método: buscarComentario
- Motivación: comprobación de que se lanza un error cuando el comentariono existe mediante porciones equivalentes
- Entradas: identificador de comentario inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: comentario indicado inexistente en base de datos

PR-IN-18

- Unidad: EventoService
- Función/método: buscarComentarioPorUsuario
- Motivación: comprobación de que se obtienen los comentarios asociados a un usuario a un evento dado mediante porciones equivalentes
- Entradas:
  - identificador de evento existente
  - identificador de usuario existente
- Salida esperada: lista de comentarios
- Inicialización del contexto: evento indicado existente y asociado mediante uno o más comentarios al usuario indicado y existente en base de datos

PR-IN-19

- Unidad: EventoService
- Función/método: buscarComentariosPorUsuario
- Motivación: comprobación de que se lanza un error cuando el evento no existe mediante porciones equivalentes
- Entradas:
  - identificador de evento inexistente
  - identificador del usuario existente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: evento indicado inexistente y usuario indicado existente en base de datos

PR-IN-20

- Unidad: EventoService

- Función/método: buscarComentariosPorUsuario
- Motivación: comprobación de que se lanza un error cuando el usuario no existe mediante porciones equivalentes
- Entradas:
  - identificador de evento existente
  - identificador del usuario inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: evento indicado existente y usuario indicado inexistente en base de datos

PR-IN-21

- Unidad: EventoService
- Función/método: buscarComentarioPorUsuario
- Motivación: comprobación de que no se obtienen comentarios cuando un usuario no realizó ninguno en el evento dado mediante porciones equivalentes
- Entradas:
  - identificador de evento existente
  - identificador de usuario existente
- Salida esperada: lista de comentarios vacía
- Inicialización del contexto: evento indicado existente pero no asociado mediante comentarios al usuario indicado y existente en base de datos

PR-IN-22

- Unidad: EventoService
- Función/método: crearEtiqueta y buscarEtiquetaPorId
- Motivación: comprobación de que se crea y se encuentra correctamente una etiqueta mediante porciones equivalentes
- Entradas: nombre de la etiqueta inexistente
- Salida esperada: la nueva etiqueta
- Inicialización del contexto: ninguna etiqueta en base de datos

PR-IN-23

- Unidad: EventoService
- Función/método: crearEtiqueta
- Motivación: comprobación de que se lanza un error cuando la etiqueta creada ya existía previamente mediante porciones equivalentes

- Entradas: nombre de la etiqueta existente
- Salida esperada:
  - DuplicateInstanceException
- Inicialización del contexto: etiqueta que se quiere crear, ya existente en base de datos

PR-IN-24

- Unidad: EventoService
- Función/método: buscarEtiquetaPorId
- Motivación: comprobación de que se lanza un error cuando la etiqueta buscada no existe mediante porciones equivalentes
- Entradas: identificador de la etiqueta inexistente
- Salida esperada: InstanceNotFoundException
- Inicialización del contexto: ninguna etiqueta en base de datos

PR-IN-25

- Unidad: EventoService
- Función/método: anadirEtiquetas
- Motivación: comprobación de que se le añade una lista de etiquetas a un comentario mediante porciones equivalentes
- Entradas:
  - identificador de comentario existente
  - lista de identificadores de etiquetas
- Salida esperada: el comentario contenga todas las etiquetas asociadas
- Inicialización del contexto: comentario indicado existente y etiquetas existentes en base de datos

PR-IN-26

- Unidad: EventoService
- Función/método: anadirEtiquetas
- Motivación: comprobación de que lanza un error cuando el comentario no existe mediante porciones equivalentes
- Entradas:
  - identificador de comentario inexistente
  - lista de identificadores de etiquetas
- Salida esperada:
  - Error: InstanceNotFoundException

- Inicialización del contexto: comentario indicado inexistente y etiquetas existentes en base de datos

PR-IN-27

- Unidad: EventoService
- Función/método: mostrarComentariosEtiqueta
- Motivación: comprobación de que se muestran los comentarios asociados con una misma etiqueta mediante porciones equivalentes
- Entradas: nombre de etiqueta existente
- Salida esperada: aparecen todos los comentarios asociados
- Inicialización del contexto: nombre de la etiqueta existente en base de datos con comentarios asociados

PR-IN-28

- Unidad: EventoService
- Función/método: mostrarComentariosEtiqueta
- Motivación: comprobación de que se lanza un error cuando la etiqueta no existe mediante porciones equivalentes
- Entradas: nombre de etiqueta inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: la etiqueta indicada no existe en base de datos

PR-IN-29

- Unidad: EventoService
- Función/método: nubeEtiqueta
- Motivación: comprobación de que se muestran en orden descendente las etiquetas por número de apariciones en los comentarios mediante porciones equivalentes
- Entradas: -
- Salida esperada: todas las etiquetas ordenadas descendentemente
- Inicialización del contexto: etiquetas existentes y asociadas a comentarios

PR-IN-30

- Unidad: EventoService
- Función/método: buscarEventos
- Motivación: comprobación de que se muestran los eventos buscados por palabras clave mediante porciones equivalentes

- Entradas: palabras clave
- Salida esperada: todas los eventos cuyo nombre contenga las palabras clave
- Inicialización del contexto: eventos en base de datos cuyos nombres coincidan parcialmente con las palabras clave y eventos que no.

PR-IN-31

- Unidad: EventoService
- Función/método: buscarEventos
- Motivación: comprobación de que no se muestran los eventos cuando el nombre no coincide con las palabras clave mediante porciones equivalentes
- Entradas: palabras clave con las que no coincide ningún evento
- Salida esperada: ningún evento
- Inicialización del contexto: eventos en base de datos cuyos nombres no coincidan total o parcialmente con las palabras clave.

PR-IN-32

- Unidad: EventoService
- Función/método: buscarEvento
- Motivación: comprobación de que se devuelve el evento adecuado mediante porciones equivalentes
- Entradas: identificador del evento existente
- Salida esperada: el evento buscado
- Inicialización del contexto: evento indicado existente en base de datos

PR-IN-33

- Unidad: EventoService
- Función/método: buscarEvento
- Motivación: comprobación de que lanza una excepción cuando no existe el evento mediante porciones equivalentes
- Entradas: identificador del evento inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: evento indicado inexistente en base de datos

PR-IN-34

- Unidad: EventoService
- Función/método: recomendarEvento

- Motivación: comprobación de que los grupos indicados recibieron una recomendación para el evento indicado mediante porciones equivalentes
- Entradas:
  - identificador del evento existente
  - lista de grupos
  - texto de comentario
- Salida esperada: todos los grupos de la lista tienen una recomendación para ese evento
- Inicialización del contexto: evento indicado existente y grupos existentes en base de datos

PR-IN-35

- Unidad: EventoService
- Función/método: recomendarEvento
- Motivación: comprobación de que lanza una excepción cuando no existe el evento mediante porciones equivalentes
- Entradas:
  - identificador del evento inexistente
  - lista de gruos
  - texto de comentario
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: evento indicado inexistente y grupos existentes en base de datos

PR-IN-36

- Unidad: EventoService
- Función/método: recomendarEvento
- Motivación: comprobación de que lanza una excepción cuando la lista de grupos no contiene ningún grupo mediante porciones equivalentes
- Entradas:
  - identificador del evento existente
  - ningún grupo
  - texto de comentario
- Salida esperada:
  - Error: SinGruposException
- Inicialización del contexto: evento indicado existente

PR-IN-37

- Unidad: EventoService
- Función/método: mostrarRecomendaciones
- Motivación: comprobación de que muestra todas las recomendaciones de un usuario mediante porciones equivalentes
- Entradas: identificador del usuario existente
- Salida esperada: todas las recomendaciones realizadas por un usuario
- Inicialización del contexto: usuario indicado existente con recomendaciones asociadas

PR-IN-38

- Unidad: EventoService
- Función/método: mostrarRecomendaciones
- Motivación: comprobación de que lanza error cuando el usuario no existe mediante porciones equivalentes
- Entradas: identificador de usuario inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado inexistente

PR-IN-39

- Unidad: EventoService
- Función/método: grupoRecomendado
- Motivación: comprobación de que un grupo ha recibido una recomendación para un determinado evento mediante porciones equivalentes
- Entradas:
  - identificador de grupo existente
  - identificador de evento recomendado para el grupo
- Salida esperada: true para el evento recomendado y false para el evento no recomendado
- Inicialización del contexto: un grupo indicado existente y asociado mediante una recomendación al evento indicado y existente en base de datos; y otro grupo indicado existente pero sin estar asociado al evento mediante ninguna recomendación.

PR-IN-40

- Unidad: EventoService
- Función/método: grupoRecomendado
- Motivación: comprobación de que lanza error cuando el evento no existe mediante porciones equivalentes

- Entradas:
  - identificador de grupo existente
  - identificador de evento inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: grupo indicado existente y evento indicado inexistente en base de datos.

PR-IN-41

- Unidad: EventoService
- Función/método: grupoRecomendado
- Motivación: comprobación de que lanza error cuando el grupo no existe mediante porciones equivalentes
- Entradas:
  - identificador de grupo inexistente
  - identificador de evento existente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: grupo indicado inexistente y evento indicado existente en base de datos.

PR-IN-42

- Unidad: GrupoService
- Función/método: crearGrupo
- Motivación: comprobación de que un grupo se crea correctamente mediante porciones equivalentes
- Entradas:
  - grupo a crear
  - identificador de usuario existente
- Salida esperada: nuevo grupo creado y asociado con el usuario
- Inicialización del contexto: usuario indicado existente y grupo indicado inexistente en base de datos

PR-IN-43

- Unidad: GrupoService
- Función/método: crearGrupo
- Motivación: comprobación de que se lanza un error cuando el grupo está duplicado mediante porciones equivalentes

- Entradas:
  - grupo a crear duplicado
  - identificador de usuario existente
- Salida esperada:
  - Error: DuplicateInstanceException
- Inicialización del contexto: usuario indicado existente y grupo indicado existente en base de datos

PR-IN-44

- Unidad: GrupoService
- Función/método: verGrupos
- Motivación: comprobación de que se obtienen los grupos en un rango determinado mediante porciones equivalentes
- Entradas:
  - 0 (índice del primer resultado de la búsqueda)
  - 20 (número máximo de resultados devueltos)
- Salida esperada: los 20 primeros grupos como máximo, si los hay
- Inicialización del contexto: grupos existentes en base de datos

PR-IN-45

- Unidad: GrupoService
- Función/método: altaGrupo
- Motivación: comprobación de que un usuario se da de alta en un grupo correctamente mediante porciones equivalentes
- Entradas:
  - identificador del usuario existente
  - identificador del grupo existente
- Salida esperada: el usuario contiene al grupo y el grupo al usuario
- Inicialización del contexto: usuario indicado existente y grupo indicado existente en base de datos

PR-IN-46

- Unidad: GrupoService
- Función/método: altaGrupo
- Motivación: comprobación de que se lanza un error cuando el usuario no existe
-

- Entradas:
  - identificador del usuario inexistente
  - identificador del grupo existente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado inexistente y grupo indicado existente en base de datos

PR-IN-47

- Unidad: GrupoService
- Función/método: altaGrupo
- Motivación: comprobación de que se lanza un error cuando el grupo no existe mediante porciones equivalentes
- Entradas:
  - identificador del usuario existente
  - identificador del grupo inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado existente y grupo indicado inexistente en base de datos

PR-IN-48

- Unidad: GrupoService
- Función/método: bajaGrupo
- Motivación: comprobación de que se da un usuario de baja de un grupo correctamente mediante porciones equivalentes
- Entradas:
  - identificador del usuario existente
  - identificador del grupo existente
- Salida esperada: El grupo no contiene al usuario y el usuario no contiene al grupo
- Inicialización del contexto: usuario indicado existente y asociado al grupo indicado existente en base de datos

PR-IN-49

- Unidad: GrupoService
- Función/método: bajaGrupo
- Motivación: comprobación de que se lanza un error cuando el grupo no existe mediante porciones equivalentes

- Entradas:
  - identificador del usuario existente
  - identificador del grupo inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado existente y grupo indicado inexistente en base de datos

PR-IN-50

- Unidad: GrupoService
- Función/método: bajaGrupo
- Motivación: comprobación de que se lanza un error cuando el usuario no existe mediante porciones equivalentes
- Entradas:
  - identificador del usuario inexistente
  - identificador del grupo existente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado inexistente y grupo indicado existente en base de datos

PR-IN-51

- Unidad: GrupoService
- Función/método: bajaGrupo
- Motivación: comprobación de que se elimina un grupo cuando se va el ultimo usuario del mismo mediante porciones equivalentes
- Entradas:
  - identificador del usuario existente
  - identificador del grupo existente
- Salida esperada: el grupo no existe
- Inicialización del contexto: usuario indicado existente y asociado al grupo indicado existente en base de datos

PR-IN-52

- Unidad: GrupoService
- Función/método: buscarPorUsuario
- Motivación: comprobación de que se obtienen los grupos de un usuario mediante porciones equivalentes

- Entradas: identificador del usuario existente
- Salida esperada: todos los grupos del usuario
- Inicialización del contexto: usuario indicado existente asociado a uno o más grupos.

PR-IN-53

- Unidad: GrupoService
- Función/método: buscarPorUsuario
- Motivación: comprobación de que se lanza un error cuando el usuario no existe mediante porciones equivalentes
- Entradas: identificador del usuario inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado inexistente en base de datos

PR-IN-54

- Unidad: GrupoService
- Función/método: buscarGrupo
- Motivación: comprobación de que se obtiene el grupo correctamente mediante porciones equivalentes
- Entradas: identificador del grupo existente
- Salida esperada: el grupo buscado
- Inicialización del contexto: grupo indicado existente en base de datos

PR-IN-55

- Unidad: GrupoService
- Función/método: buscarGrupo
- Motivación: comprobación de que se lanza un error cuando el grupo no existe mediante porciones equivalentes
- Entradas: identificador del grupo inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: grupo indicado inexistente en base de datos

PR-IN-56

- Unidad: GrupoService
- Función/método: buscarGrupoPorId

- Motivación: comprobación de que se obtiene el grupo correctamente mediante porciones equivalentes
- Entradas: identificador del grupo existente
- Salida esperada: el grupo buscado
- Inicialización del contexto: grupo indicado existente en base de datos

PR-IN-57

- Unidad: GrupoService
- Función/método: buscarGrupoPorId
- Motivación: comprobación de que se lanza un error cuando el grupo no existe mediante porciones equivalentes
- Entradas: identificador del grupo inexistente
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: grupo indicado inexistente en base de datos

PR-IN-58

- Unidad: UserProfileService
- Función/método: registerUser
- Motivación: comprobación de que se registra correctamente un usuario mediante porciones equivalentes
- Entradas:
  - login inexistente
  - contraseña
  - detalles del perfil
- Salida esperada: Se encuentra el usuario registrado
- Inicialización del contexto: usuario indicado inexistente en base de datos

PR-IN-59

- Unidad: UserProfileService
- Función/método: registerUser
- Motivación: comprobación de que se lanza una excepción cuando el login ya existe mediante porciones equivalentes
- Entradas:
  - login existente
  - contraseña

    - detalles del perfil
- Salida esperada:
    - Error: DuplicateInstanceException
- Inicialización del contexto: usuario indicado existente en base de datos

PR-IN-60

- Unidad: UserProfileService
- Función/método: login
- Motivación: comprobación de que se autentica correctamente un usuario mediante porciones equivalentes
- Entradas:
    - login existente
    - contraseña
    - false (indica que la contraseña no está encriptada)
- Salida esperada: Se encuentra el usuario autenticado
- Inicialización del contexto: usuario indicado existente en base de datos

PR-IN-61

- Unidad: UserProfileService
- Función/método: login
- Motivación: comprobación de que se autentica correctamente un usuario mediante porciones equivalentes
- Entradas:
    - login existente
    - contraseña correcta
    - true (indica que la contraseña está encriptada)
- Salida esperada: Se encuentra el usuario autenticado
- Inicialización del contexto: usuario indicado existente en base de datos

PR-IN-62

- Unidad: UserProfileService
- Función/método: login
- Motivación: comprobación de que se lanza un error cuando la contraseña no coincide mediante porciones equivalentes
- Entradas:
    - login existente

    - contraseña incorrecta
    - false (indica que la contraseña no está encriptada)
- Salida esperada:
    - Error: IncorrectPasswordException
- Inicialización del contexto: usuario indicado existente en base de datos

PR-IN-63

- Unidad: UserProfileService
- Función/método: login
- Motivación: comprobación de que no se autentica correctamente un usuario que no está registrado mediante porciones equivalentes
- Entradas:
    - login inexistente
    - contraseña
    - false (indica que la contraseña no está encriptada)
- Salida esperada:
    - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado inexistente en base de datos

PR-IN-64

- Unidad: UserProfileService
- Función/método: findUserProfileDetails
- Motivación: comprobación de que se devuelven los datos del perfil del usuario correctamente mediante porciones equivalentes
- Entradas: identificador de usuario existente
- Salida esperada: datos del perfil del usuario
- Inicialización del contexto: usuario indicado existente en base de datos

PR-IN-66

- Unidad: UserProfileService
- Función/método: updateUserProfileDetails
- Motivación: comprobación de que se actualizan los datos del perfil del usuario mediante porciones equivalentes
- Entradas:
    - identificador de usuario existente
    - nuevos datos del perfil del usuario

- Salida esperada: Se obtiene el perfil del usuario con los nuevos datos
- Inicialización del contexto: usuario indicado existente en base de datos

PR-IN-67

- Unidad: UserProfileService
- Función/método: updateUserProfileDetails
- Motivación: comprobación de que se lanza una excepción cuando el usuario no existe mediante porciones equivalentes
- Entradas:
  - identificador de usuario inexistente
  - nuevos datos del perfil del usuario
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado inexistente en base de datos

PR-IN-68

- Unidad: UserProfileService
- Función/método: changePassword
- Motivación: comprobación de que cambia satisfactoriamente la contraseña mediante porciones equivalentes
- Entradas:
  - identificador de usuario existente
  - contraseña en claro antigua
  - contraseña en claro nueva
- Salida esperada: El usuario es autenticado satisfactoriamente con la nueva contraseña
- Inicialización del contexto: usuario indicado existente en base de datos

PR-IN-69

- Unidad: UserProfileService
- Función/método: changePassword
- Motivación: comprobación de que no se cambia la contraseña cuando la antigua contraseña no es la correcta mediante porciones equivalentes
- Entradas:
  - identificador de usuario existente
  - contraseña en claro antigua incorrecta
  - contraseña en claro nueva

- Salida esperada:
  - Error: IncorrectPasswordException
- Inicialización del contexto: usuario indicado inexistente en base de datos

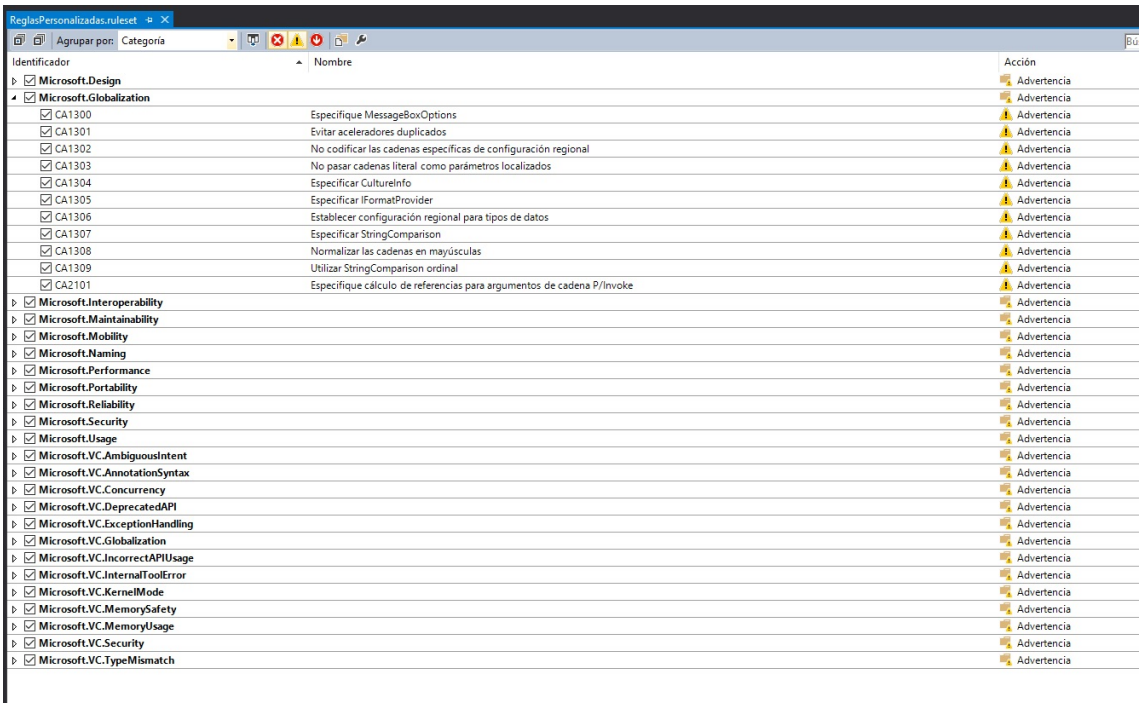
PR-IN-70

- Unidad: UserProfileService
- Función/método: changePassword
- Motivación: comprobación de que no se cambia la contraseña cuando el usuario no existe mediante porciones equivalentes
- Entradas:
  - identificador de usuario inexistente
  - contraseña en claro antigua
  - contraseña en claro nueva
- Salida esperada:
  - Error: InstanceNotFoundException
- Inicialización del contexto: usuario indicado inexistente en base de datos

3.3. Pruebas de Caja Blanca

3.3.1. Descripción

Se han comprobado numerosas reglas de programación sobre nuestro código, como podemos ver en la siguiente imagen.



Debido a las numerosas correcciones resultantes, hemos expuesto en este documento un ejemplo de cada tipo.

3.3.2. Marcar los tipos ISerializable con SerializableAttribute

Agregue [Serializable] a 'SinGruposException', ya que este tipo implementa Iserializable. Las clases de excepción deben ser serializables para que funcionen correctamente en los dominios de aplicación.

3.3.3. Las propiedades de la colección deben ser de solo lectura

Cambie 'Comentario.Etiqueta' para que sea de solo lectura quitando el establecedor de propiedades.
No hemos corregido esto porque perderíamos la navegabilidad que nos proporciona el edmx.

3.3.4. Las operaciones no deben desbordarse

Corrija el desbordamiento potencial en la operación 'count+1' en 'EventoService.VerComentarios(long, int, int)'.
Sustituímos la operación count + 1 por count ++

3.3.5. Desechar (Dispose) objetos antes de perder el ámbito

En el método 'PasswordEncrypter.Crypt(string)', llame a System.IDisposable.Dispose en el objeto 'hashAlg' antes de que todas las referencias a él estén fuera de ámbito.
Hemos hecho el Dispose del objeto hashAlg en el finally de un try/catch para que pase lo que pase, siempre se libere ese objeto. Esto reducirá el uso de memoria de la aplicación.

3.3.6. No llamar a métodos reemplazables en constructores

'Comentario.Comentario()' contiene una cadena de llamada que da como resultado un llamada a un método virtual definido por la clase. Revise la siguiente pila de llamadas para comprobar las consecuencias no intencionadas: Comentario.ctor() $Comentario.set\_Etiqueta(ICollection < Etiqueta >) : Void$

Hemos eliminado el riesgo de que se llame a una colección que no estéa inicializada, se ha solucionado quitando la posibilidad de sobreescribir la colección en una posible herencia (Quitando la etiqueta "virtual")

3.3.7. Comprobar si las cadenas están vacías mediante la longitud de cadena

Reemplace la llamada a 'string.Equals(string)' en 'EventoService.BusquedaEventos(string, int, int)' con una llamada a 'String.IsNullOrEmpty'.

Hemos encontrado un error en el código gracias a esto, porque no se había controlado el caso de que el string fuese nulo, en tal caso la aplicación se paraba inesperadamente.

3.3.8. Los identificadores deberían tener la ortografía correcta

Considere proporcionar un nombre más significativo que el nombre del miembro 'IeventoService.A()'.
Hemos mejorado la legibilidad del código gracias a esta regla ya que algunas variables, tenian nombres que no eran suficientemente descriptivos.

3.3.9. No debería utilizar parámetros predeterminados

Reemplace el método 'EventoDaoEntityFramework.BuscarEventos(List¡string¿, int, int)' por una sobrecarga que proporcione todos los argumentos predeterminados.

Con esto se mejora la legibilidad del código, ya que los valores por defecto no eran realmente necesarios.

3.3.10. Los nombres de tipo no deberían coincidir con los espacios de nombres

El nombre de tipo 'UserService' está en conflicto total o parcialmente con el nombre del espacio de nombres 'Es.Udc.DotNet.PracticaMaD.Model.UserService'. Cambie el nombre para eliminar el conflicto.

Se ha mejorado la legibilidad del código porque se han sustituído nombres de clases que coincidían con nombres del espacio de nombres.

3.3.11. Los identificadores deberían utilizar las mayúsculas y minúsculas correctamente

Corrija el uso de mayúsculas y minúsculas en 'fecha' en el nombre del miembro 'Recomendacion.fecha' cambiándolo a 'Fecha'.

Hemos seguido el sistema de nombrado pascal case por lo que hemos ignorado esta regla.

3.3.12. Utilizar términos preferidos

Reemplace el término 'login' en el nombre del miembro 'UserProfile.loginName' por la alternativa preferida 'LogOn'.

Hemos ignorado este aviso porque consideramos que debemos usar los nombres que hemos asignado nosotros.

3.3.13. Implementar constructores de excepción estándar

Agregue el siguiente constructor a 'SinGruposException': public SinGruposException(String, Exception).

Se han implementado los constructores de excepción estándar que faltaban, esto mejora la flexibilidad de la aplicación.

3.3.14. Validar argumentos de métodos públicos

En el método 'ComentarioDTO.ComentarioDTO(ComentarioDTO)' visible externamente, valide el parámetro 'dto' antes de usarlo.
Se ha comprobado que los parámetros tipos no primitivos introducidos en las funciones no fuesen nulos, ya que podían dar fallos no controlados cuando se usaban métodos de estos parámetros. Por ejemplo en el caso expuesto en este apartado, si intentamos acceder a un atributo del dto introducido en la función, al ser nulo, lanzaba una excepción no controlada.

3.3.15. Marque los ensamblados con CLSCompliantAttribute

Marque 'PracticaMaD.Model.dll' con CLSCompliant(true) porque expone tipos visibles externamente.
$[assembly : CLSCompliant(true)]$ Con esto indicamos que nuestros ensamblados están sujetos a las reestricciones de nomenclatura del Common Language Specification (CLS).

3.3.16. Los ensamblados deben tener nombres seguros válidos

Firme 'PracticaMaD.Model.dll' con una clave de nombre seguro.

No hemos podido firmar PracticaMaD.Model.dll porque depende de ModelUtil que es una librería en la que confiamos pero sobre la que no tenemos control.

3.3.17. Quitar variables locales no utilizadas

'EventoService.RecomendarEvento(long, List¡Grupo¿, string)' declara una variable, 'grupo', de tipo 'Grupo', que no se usa nunca o sólo se asigna. Use esta variable o quítela.
Hemos detectado que había variables que no estabamos usando, pero necesitabamos. Se han corregido gracias a esta regla.

3.3.18. Especificar CultureInfo

Puesto que el comportamiento de 'string.ToLower()' podría variar en función de la configuración regional del usuario actual, reemplace esta llamada de 'EventoService.CrearEtiqueta(string)' por una llamada a 'string.ToLower(CultureInfo)'. Si el resultado de 'string.ToLower(CultureInfo)' se mostrará al usuario, especifique 'CultureInfo.CurrentCulture' como parámetro 'CultureInfo'. De lo contrario, si el resultado se va a almacenar y software obtendrá acceso a él, como cuando se guarda en un disco o una base de datos, especifique 'CultureInfo.InvariantCulture'.
Al especificar el CultureInfo, tenemos en cuenta a la hora de dar formato las convenciones del país/región que se especifique, lo que favorece la experiencia del usuario.

3.3.19. Las propiedades no deben ser de solo escritura

Puesto que el captador de propiedad de 'EventoService.comentarioDao' es menos visible que su establecedor, aumente la accesibilidad de su captador o reduzca la de su establecedor.

3.3.20. No debería utilizar parámetros predeterminados

Reemplace el método 'EventoService.BusquedaEventos(string, int, int)' por una sobrecarga que proporcione todos los argumentos predeterminados.

Se han cambiado métodos con parámetros de entrada por defecto, por un método sin parámetros por defecto y un método sobrecargado sin los parámetros por defecto, que llaman al primer método añadiendo los valores por defecto.

3.3.21. No exponer listas genéricas

Cambie "List¡Comentario¿" en 'ComentarioDaoEntityFramework.VerComentarios(long, int, int)' para que use Collection¡T¿, ReadOnlyCollection¡T¿o KeyedCollection¡K,V¿

En general, se pueden usar de manera externa listas sólo si mejoran de forma notable el rendimiento. Como no es así, las cambiamos por colecciones genéricas, ya que están diseñadas para la herencia y pudiera la aplicación ser ampliada en el futuro y/o reutilizarse.

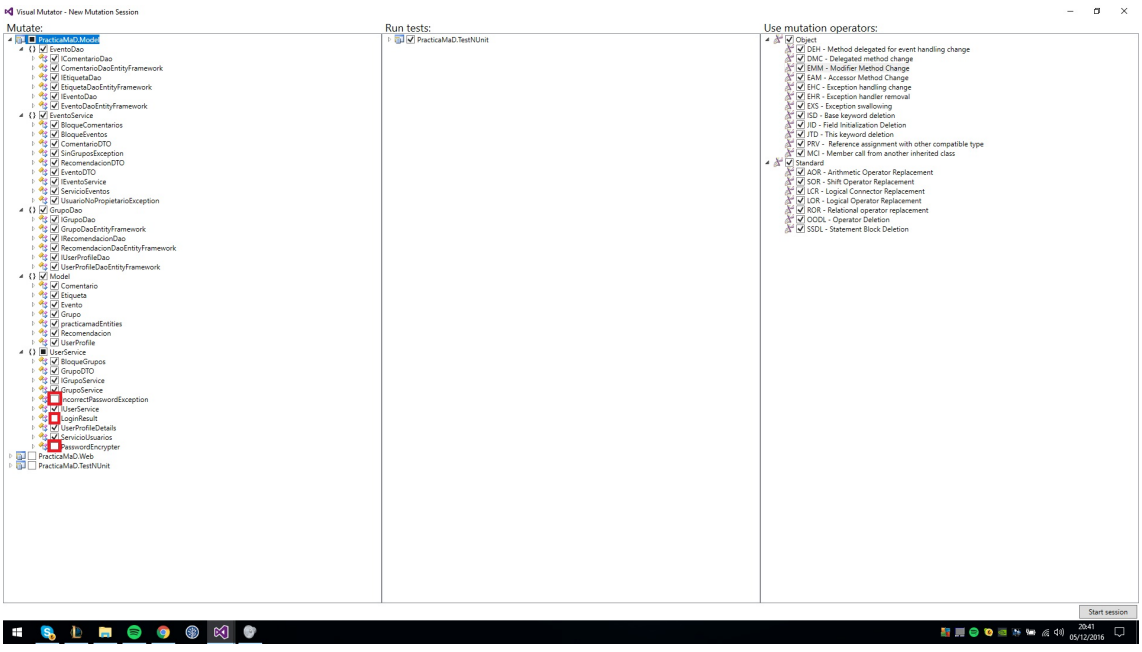
3.3.22. Evitar espacios de nombres con pocos tipos

Considere combinar los tipos definidos en 'Es.Udc.DotNet.PracticaMaD.Model.ComentarioDao' con otro espacio de nombres.
Se han agrupado las clases en función de la relación que tienen con otras clases. De esta forma, se mejora la visibilidad y el acceso a los métodos de dichas clases.

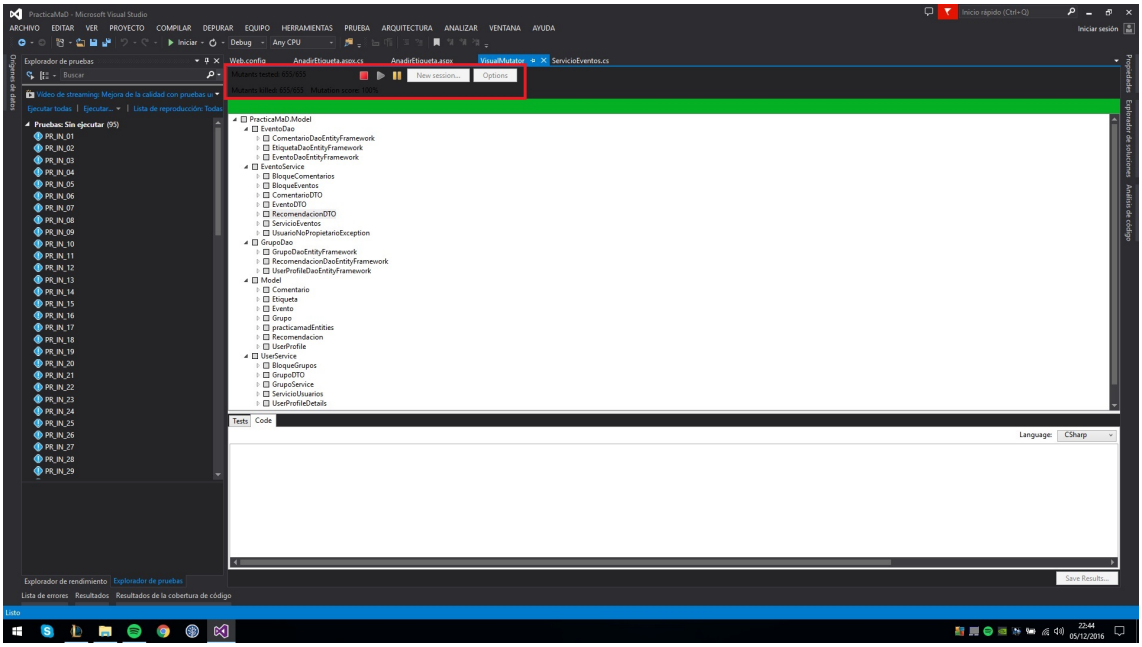
3.4. Tests Mutantes

Los tests mutantes se utilizan para comprobar la calidad de las pruebas existentes. Estos hacen pequeñas modificaciones en el código, estas modificaciones se llaman mutantes, y luego vuelven a pasar las pruebas diseñadas. Si la prueba pasa, hay un caso en el que el código está haciendo mal su función, pero las pruebas no han detectado ese error, se dice que el mutante ha sobrevivido. Si la prueba falla, el mutante es asesinado.

Para realizar las mutaciones hemos utilizado VISUAL MUTATOR, hemos ignorado IncorrectPasswordException, LoginResult y PasswordEncrypter, ya que es un código ageno a nosotros y del que confiamos ya que se ha utilizado en muchos proyectos y está muy probado.



Se han creado 655 mutantes y se han asesinado a todos, por lo que podemos decir que la calidad de las pruebas diseñadas es buena.



Como se han creado 655 mutantes, tan solo vamos a exponer un ejemplo de cada tipo de mutante creado. No se han generado todos los tipos de mutantes que ofrece la herramienta, ya que no se da el caso en nuestra aplicación.

Objeto

3.4.1. EMM - Modifier method change

```
(-) this.login = dto.login;
(+) this.texto = dto.login;
```

3.4.2. EAM - Accessor method change

```
(-) this.texto = dto.texto;
(+) this.texto = dto.idComentario
```

3.4.3. MCI - Member call from another inherited class

```
(-) (!(this.idComentario != p.idComentario) && this.login.Equals(p.login))
(+) (!(this.idComentario != p.idComentario) && this.<login>k__BackingFieldEquals(p.login)
```

Standard

3.4.4. AOR - Arithmetic operator replacement

Multiplication

```
(-) eventos = this.eventoDao.BuscarEventos(startIndex, count + 1);
(+) eventos = this.eventoDao.BuscarEventos(startIndex, count * 1);
```

Division

```
(-) eventos = this.eventoDao.BuscarEventos(startIndex, count + 1);
(+) eventos = this.eventoDao.BuscarEventos(startIndex, count / 1);
```

Subtraction

```
(-) eventos = this.eventoDao.BuscarEventos(startIndex, count + 1);
(+) eventos = this.eventoDao.BuscarEventos(startIndex, count - 1);
```

Right param

```
(-) eventos = this.eventoDao.BuscarEventos(startIndex, count + 1);
(+) eventos = this.eventoDao.BuscarEventos(startIndex, count);
```

Right param

```
(-) eventos = this.eventoDao.BuscarEventos(startIndex, count + 1);
(+) eventos = this.eventoDao.BuscarEventos(startIndex, 1);
```

Modulus

```
(-) eventos = this.eventoDao.BuscarEventos(startIndex, count + 1);
(+) eventos = this.eventoDao.BuscarEventos(startIndex, count % 1);
```

3.4.5. ROR - Relational operator replacement

Equality

```
(-) if (count > 0)
(+) if (count == 0)
```

False

```
(-) if (obj == null)
(+) if (false)
```

True

```
(-) if (count > 0)
(+) if (true)
```

3.4.6. OODL - Operator deletion

Right Operand removed

```
(-) if (count > 0)
(+) if (0)
```

Left Operand removed

```
(-) if (count > 0)
(+) if (count)
```

3.5. Tests Aleatorios

PR-AL-01

- Unidad: EventoService
- Función/método: buscarEventos
- Motivación: validar que la función funcione con keywords aleatorias
- Entradas: palabras clave aleatorias
- Salida esperada: todas los eventos cuyo nombre contenga las palabras clave
- Inicialización del contexto: no necesita inicialización

PR-AL-02

- Unidad: EventoService
- Función/método: buscarEventos
- Motivación: validar que la función funcione con keywords aleatorias
- Entradas: palabras clave aleatorias
- Salida esperada: todas los eventos cuyo nombre contenga las palabras clave
- Inicialización del contexto: no necesita inicialización

PR-AL-03

- Unidad: EventoService
- Función/método: buscarEventos
- Motivación: validar que la función funcione con start index aleatorio
- Entradas: start index aleatorio
- Salida esperada: todas los eventos cuyo nombre contenga las palabras clave
- Inicialización del contexto: no necesita inicialización

PR-AL-04

- Unidad: EventoService
- Función/método: buscarEventos
- Motivación: validar que la función funcione con count aleatorio
- Entradas: count aleatorio
- Salida esperada: todas los eventos cuyo nombre contenga las palabras clave
- Inicialización del contexto: no necesita inicialización

PR-AL-05

- Unidad: EventoService

- Función/método: añadirComentario
- Motivación: validar que la función funcione con un texto aleatorio.
- Entradas:
  - identificador de usuario
  - identificador de evento
  - comentario generado aleatoriamente.
- Salida esperada: idComentario
- Inicialización del contexto: usuario indicado existente y evento existe en base de datos

PR-IN-06

- Unidad: EventoService
- Función/método: modificarComentario
- Motivación: validar que la función funcione con un texto aleatorio.
- Entradas:
  - identificador de comentario existente
  - identificador de usuario que realizó el comentario
  - comentario aleatorio
- Salida esperada:
  - el comentario contenga el nuevo texto
  - el usuario que escribió el comentario siga siendo el mismo
  - el comentario sigue perteneciendo al evento al que fue asociado
- Inicialización del contexto: comentario indicado existente y asociado al usuario indicado y existente en base de datos

PR-AL-07

- Unidad: EventoService
- Función/método: VerComentarios
- Motivación: validar que la función funcione con start index aleatorio
- Entradas:
  - start index aleatorio
  - count válido
- Salida esperada: lista de comentarios
- Inicialización del contexto:no necesita inicialización

PR-AL-08

- Unidad: EventoService

- Función/método: VerComentarios
- Motivación: validar que la función funcione con count aleatorio
- Entradas:
  - start index válido
  - count aleatorio
- Salida esperada: lista de comentarios
- Inicialización del contexto:no necesita inicialización

PR-AL-09

- Unidad: EventoService
- Función/método: recomendarEvento
- Motivación: validar que la función funcione con un texto de comentario aleatorio
- Entradas:
  - identificador del evento existente
  - lista de grupos
  - texto de comentario aleatorio
- Salida esperada: todos los grupos de la lista tienen una recomendación para ese evento
- Inicialización del contexto: evento indicado existente y grupos existentes en base de datos

PR-AL-10

- Unidad: EventoService
- Función/método: crearEtiqueta
- Motivación: comprobación de que se crea una etiqueta con un nombre aleatorio
- Entradas: nombre de la etiqueta aleatoria inexistente
- Salida esperada: la nueva etiqueta
- Inicialización del contexto: ninguna etiqueta en base de datos

PR-AL-11

- Unidad: GrupoService
- Función/método: VerGrupos
- Motivación: validar que la función es correcta cuando se introduce un start index aleatorio
- Entradas:
  - start index aleatorio
  - count válido

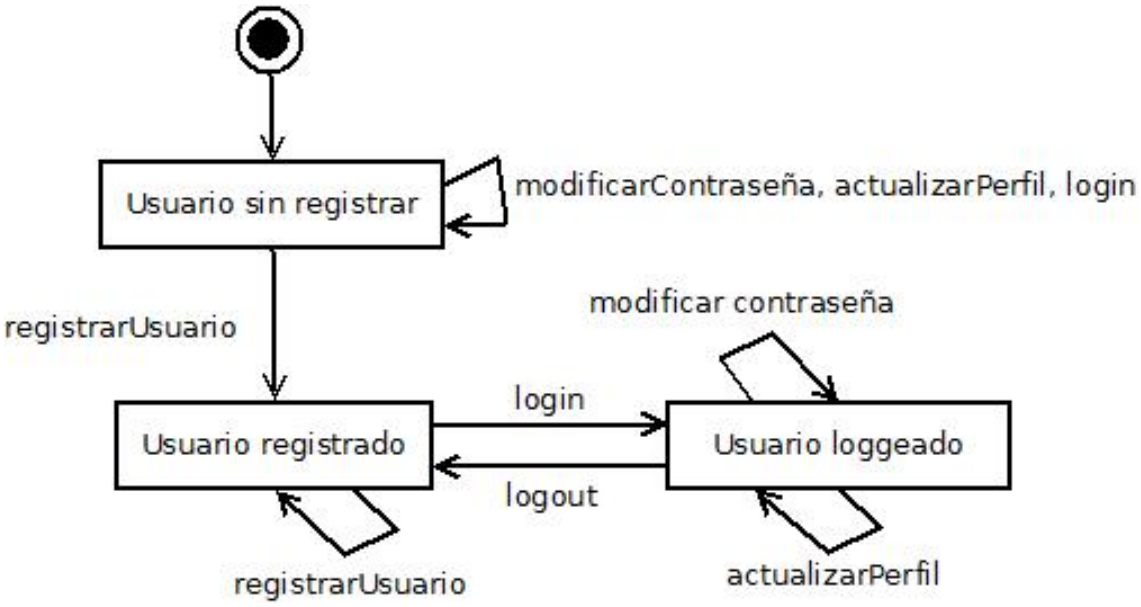
- Salida esperada: lista de grupos
- Inicialización del contexto: no necesita inicialización

PR-AL-12

- Unidad: GrupoService
- Función/método: VerGrupos
- Motivación: validar que la función es correcta cuando se introduce un count aleatorio
- Entradas:
  - start index válido
  - count aleatorio
- Salida esperada: lista de grupos
- Inicialización del contexto: no necesita inicialización

3.6. Tests basados en modelos

Las pruebas basadas en modelos las hemos realizado sobre la entidad usuario, que se encuentra en diferentes estados en función de los métodos que se llevan a cabo. El paso de un estado a otro lo hemos reflejado en el siguiente diagrama:



Debido a que para reflejar el flujo entre un usuario registrado registrado y loggeado (así como las operaciones que se pueden realizar cuando el usuario está loggeado) se necesita una sesión y no es posible emularlo de forma sencilla, hemos decidido hacer tests en los que se refleje el correcto paso del usuario por los diferentes estados mediante tests.

1. Si el usuario no está registrado y realiza algunas de las operaciones de modificar contraseña, actualizar perfil o loggearse: esto se ve reflejado en los tests PR_IN_70, PR_IN_67 y PR_IN_63, respectivamente.

2. Si el usuario no está registrado y realiza la operación de registrarse: esto se ve reflejado en el test PR_IN_58

3. Si el usuario está registrado y realiza la operación de registrarse: esto se ve reflejado en el test PR_IN_59

4. Si el usuario está registrado y realiza la operación de logearse: esto se ve reflejado en el test PR_IN_60

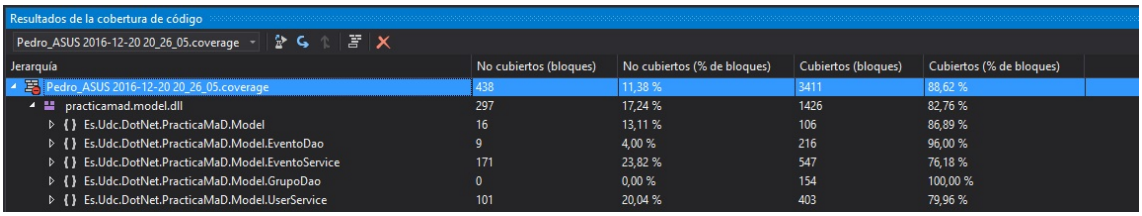
5. Si el usuario está loggeado y modifica la contraseña: esto se controla a nivel de sesión lo que no es objeto de prueba. Que el usuario tiene que estar al menos registrado se ve reflejado en el test PR_IN_68.

6. Si el usuario está loggeado y actualiza el perfil: esto se controla a nivel de sesión lo que no es objeto de prueba. Que el usuario tiene que estar al menos registrado se ve reflejado en el test PR_IN_66.

7. Si el usuario está loggeado puede hacer logout: esto se controla a nivel de sesión lo que no es objeto de prueba.

3.7. Cobertura

Debido a la aleatoriedad de algunas funcionalidades, y a funcionalidades ajenas a nosotros de la que confiamos su procedencia, el resultado de las pruebas no es el 100 % pero podemos decir que el código está probado en profundidad, un 88 % del total del código.


Resultados de la cobertura de código

Pedro_ASUS 2016-12-20 20_26_05.coverage

Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% de bloques)
Pedro_ASUS 2016-12-20 20_26_05.coverage	438	11,38 %	3411	88,62 %
practicamad.model.dll	297	17,24 %	1426	82,76 %
{ } Es.Udc.DotNet.PracticaMaD.Model	16	13,11 %	106	86,89 %
{ } Es.Udc.DotNet.PracticaMaD.Model.EventoDao	9	4,00 %	216	96,00 %
{ } Es.Udc.DotNet.PracticaMaD.Model.EventoService	171	23,82 %	547	76,18 %
{ } Es.Udc.DotNet.PracticaMaD.Model.GrupoDao	0	0,00 %	154	100,00 %
{ } Es.Udc.DotNet.PracticaMaD.Model.UserService	101	20,04 %	403	79,96 %

3.8. Estrés

Las pruebas de carga mediante Visual Studio no son posibles en todas las versiones. En el Visual Studio Community (que es el que tenemos) no hemos encontrado por ningún lado la forma de generar tests de carga. Hemos probado a descargarnos la plantilla de la prueba pero tampoco aparece, además de buscar herramientas alternativas con idénticos resultados. Las licencias que nos proporciona la FIC sólo nos permite descargarnos un Visual Studio, por lo que no podemos hacernos con dos versiones diferentes del mismo.

4. Rexisto de probas

La realización de las pruebas, si no bien han sido diarias por la dedicación parcial de horas a la práctica, al menos si ha sido constante semanalmente, a excepción de dos semanas que, como comentaremos más adelante, no hemos podido dedicarle tiempo a la asignatura debido a otras entregas.

5. Rexistro de erros

Al ser una práctica de otra materia que incluían pruebas, y además, ya había sido presentada y defendida en dos iteraciones, el código no mostraba un alto número de errores, siendo los más comunes, errores de estilo y estándares propios del lenguaje que no se han tenido en cuenta en dicha asignatura.

Los errores nombrados anteriormente se han detectado usando la herramienta de análisis de código integrada en el Visual Studio que realiza pruebas de caja blanca sobre la solución. Estos errores y sus correcciones se han descrito en el apartado 3.3

A mayores de esto, se ha detectado un error en un índice en grupoService por el cuál, con un count = 1, el programa fallaba. Este error se ha detectado gracias a las pruebas aleatorias ya que count = 1 pertenece a un número dentro de un conjunto válido y no se había probado ese valor en concreto.

6. Estatísticas

SEMANA 1

En esta semana, se ha configurado el proyecto, y la integración con TravisCI.

Se ha realizado un 0 % de las pruebas realizadas.

SEMANA 2

Se han traducido las pruebas unitarias MSUnit a NUnit para que fuesen compatibles con TravisCI, y se han completado algunos casos que no se habían contemplado.

En esta semana no se ha encontrado ningún error.

Se ha realizado 20 % de las pruebas realizadas.

SEMANA 3

Se han implementado las pruebas de integración de la aplicación. No se han encontrado errores.

SEMANA 4

Se han acabado las pruebas de integración y ejecutado las pruebas de caja blanca abriendo las issues correspondientes.

- Espacio de nombres #5
- Los ensamblados deben tener nombres seguros válidos #6
- Listas genéricas #7
- Marque los ensamblados con CLSCompilantAttribute #8
- Parámetros predeterminados #9

- Validar argumentos de métodos públicos #10
- Propiedades de solo escritura #11
- Implementar constructores de excepción estándar #12
- Especificar cultureinfo #13
- Utilizar términos preferidos #14
- Los identificadores deberían utilizar las mayúsculas y minúsculas correctamente #15
- Los nombres de tipo no deberían coincidir con los espacios de nombres #16
- Parámetros #17
- Ortografía identificadores #18
- Comprobar si las cadenas están vacías mediante la longitud de cadena #19
- Variables no utilizadas #20
- No llamar a métodos reemplazables en constructores #21
- Desechar objetos antes de perder el ámbito #22
- Las operaciones no deben desbordarse #23
- Propiedades de solo lectura #24
- Marcar los tipos ISerializable con SerializableAtribute #25

Se ha realizado un 65 % de las pruebas realizadas.

Los errores localizados esta semana han sido errores de estilo y estándares. Se han detectado repartidos por todo el programa, y eran de de criticidad baja.

SEMANA 5 y 6

Debido a prácticas de otras materias y de esta misma también, no hemos podido dedicarle tiempo a esta práctica.

Se ha realizado un 0 % de las pruebas realizadas.

SEMANA 7

Se han corregido algunas issues abiertas en la semana 4.
En esta semana se han realizado además las pruebas aleatorias y las pruebas de mutación de código.
En esta semana se han ejecutado el 15 % de las pruebas.
Se ha detectado un error en GrupoService de criticidad elevada, ya que producía un fallo que interrumpía la ejecución de la aplicación.

SEMANA 8

En esta semana se ha realizado la documentación, cobertura de los tests y las pruebas de estrés.

Estado Actual

El sistema se encuentra en un punto estable, ya que no se han detectado nuevos errores en las últimas pruebas ejecutadas.

7. Outros aspectos de interese

No se han hecho pruebas de unidad de los servicios dada la complejidad de dependencias que han empleado en la práctica, puesto que realizar tests empleando moqs presenta una gran dificultad.

Herramienta de integración continua TRAVIS CI: https://travis-ci.org/carlasalgado/VVS