




VALIDACIÓN E VERIFICACIÓN DO SOFTWARE
INFORME DE PRÁCTICAS

REPOSITORIO DO PROXECTO:

<https://github.com/adrilpz/vyv>

PARTICIPANTES NO PROXECTO:

Adrián López Martínez
Brais Rey Souto

	<p><u>VALIDACIÓN E VERIFICACIÓN DO SOFTWARE</u> <u>INFORME DE PRÁCTICAS</u></p>
---	---

1. DESCRIPCIÓN DO PROXECTO

Aplicación Java que permite xestionar unha páxina web de apostas deportivas. Se poden ver os eventos dispoñibles e non dispoñibles. Se poden facer apostas en aqueles que estean dispoñibles e comprobar o resultado de aqueles que non o estean. Ademais hai dous tipos de usuarios, un usuario normal e un usuario administrador, o cal pode crear y xestionar ditos eventos e os seus resultados

A parte do proxecto que imos testear correspóndese coa parte modelo da aplicación, encargada de aportar os servizos con toda a funcionalidade precisada na capa web.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

2. ESTADO ACTUAL

- Xestión de usuarios
 - Especificación: Inclúe funcións para rexistrar, loggear, buscar e actualizar a información de usuarios da aplicación.
 - Responsables de desenvolvemento: Adrián López Martínez e Brais Rey Souto.
 - Responsables do proceso de proba: Adrián López Martínez e Brais Rey Souto.
- Xestión de eventos e apostas
 - Especificación: Inclúe funcións para crear e buscar eventos, así como os seus tipos de aposta e as opcións de aposta correspondentes a cada un, ademais de poder marcar as opcións gañadoras dun tipo de aposta. Tamén hai funcións para apostar nunha opción de aposta concreta e poder ver as apostas feitas por un usuario.
 - Responsables de desenvolvemento: Adrián López Martínez e Brais Rey Souto.
 - Responsables do proceso de proba: Adrián López Martínez e Brais Rey Souto.

2.1. COMPOÑENTES AVALIADOS

- ApuestasService
 - Funcionalidades nas que participa: *Xestión de eventos e apostas*
 - N° de probas obxectivo: 41
 - N° de probas preparadas: 41
 - Porcentaxe executada: 100%
 - Porcentaxe superada: 100%
- UserService
 - Funcionalidades nas que participa: *Xestión de usuarios*
 - N° de probas obxectivo: 18
 - N° de probas preparadas: 18
 - Porcentaxe executada: 100%
 - Porcentaxe superada: 100%
- ApuestaDao
 - Funcionalidades nas que participa: *Xestión de eventos e apostas*
 - N° de probas obxectivo: 8
 - N° de probas preparadas: 8
 - Porcentaxe executada: 100%



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

- Porcentaxe superada: 100%
- CategoriaDao
 - Funcionalidades nas que participa: *Xestión de eventos e apostas*
 - N° de probas obxectivo: 8
 - N° de probas preparadas: 8
 - Porcentaxe executada: 100%
 - Porcentaxe superada: 100%
- EventoDao
 - Funcionalidades nas que participa: *Xestión de eventos e apostas*
 - N° de probas obxectivo: 8
 - N° de probas preparadas: 8
 - Porcentaxe executada: 100%
 - Porcentaxe superada: 100%
- OpcionApuestaDao
 - Funcionalidades nas que participa: *Xestión de eventos e apostas*
 - N° de probas obxectivo: 11
 - N° de probas preparadas: 11
 - Porcentaxe executada: 100%
 - Porcentaxe superada: 100%
- TipoApuestaDao
 - Funcionalidades nas que participa: *Xestión de eventos e apostas*
 - N° de probas obxectivo: 8
 - N° de probas preparadas: 8
 - Porcentaxe executada: 100%
 - Porcentaxe superada: 100%
- UserProfileDao
 - Funcionalidades nas que participa: *Xestión de usuarios*
 - N° de probas obxectivo: 9
 - N° de probas preparadas: 9
 - Porcentaxe executada: 100%
 - Porcentaxe superada: 100%



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

3. ESPECIFICACIÓN DE PROBAS

1.

- a) Código: "PR-UN-001"
- b) Unidade: ApuestasService
- c) Función/método: createEvento()
- d) Motivación: creación de un evento con valores correctos. Se consideran valores correctos: un valor no vacío en el campo nombre, una fecha posterior a la actual y un código de una categoría existente.
- e) Entrada(s): nombre(String), fecha(Calendar), categoria(Long) -> ("Barsa - Madrid", "24/12/2017", valor no libre)
- f) Saída(s): objeto de tipo Evento
- g) Inicialización: creación de objeto de tipo Categoria

2.

- a) Código: "PR-UN-002"
- b) Unidade: ApuestasService
- c) Función/método: createEventoWithEmptyName()
- d) Motivación: creación de un evento con el valor del campo nombre vacío (valor frontera) con el objetivo de que salte una excepción ya que no está permitido crear eventos sin nombre. Se consideran valores correctos: un valor no vacío en el campo nombre, una fecha posterior a la actual y un código de una categoría existente.
- e) Entrada(s): nombre(String), fecha(Calendar), categoria(Long) -> ("", "24/12/2017", valor no libre)
- f) Saída(s): excepción InputValidationException
- g) Inicialización: creación de objeto de tipo Categoria

3.

- a) Código: "PR-UN-003"
- b) Unidade: ApuestasService
- c) Función/método: createEventoWithNonExistentCategoria()
- d) Motivación: creación de un evento con un código de una categoría que no existe (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un valor no vacío o nulo en el campo nombre, una fecha posterior a la actual y un código de una categoría existente.
- e) Entrada(s): nombre(String), fecha(Calendar), categoria(Long) -> ("Gran premio China", "24/12/2017", -1);
- f) Saída(s): excepción InstanceNotFoundException
- g) Inicialización: dcreación de objeto de tipo Categoria

4.

- a) Código: "PR-UN-004"
- b) Unidade: ApuestasService
- c) Función/método: createEventoInDateBefore()
- d) Motivación: creación de un evento con una fecha anterior a la actual (valor frontera) con el objetivo de que salte una excepción ya que no está permitido crear eventos con una fecha anterior a la actual. Se consideran valores correctos: un valor no vacío en el campo nombre, una fecha posterior a la actual y un código de una categoría existente.
- e) Entrada(s): nombre(String), fecha(Calendar), categoria(Long) -> ("Sevilla-Betis", "24/11/2015", futbol.getCodCategoria());
- f) Saída(s): excepción InputValidationException
- g) Inicialización: creación de objeto de tipo Categoria.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

5.

- a) Código: "PR-UN-005"
- b) Unidad: ApuestasService
- c) Función/método: findEventosByCategory()
- d) Motivación: búsqueda de eventos por categoría de un usuario no administrador con valores correctos. Se consideran valores correctos: un nombre que debe ser nulo (puesto que solo se buscará por categoría), un código de categoría existente, el campo admin con valor false (que indica que el que realiza la búsqueda no es un usuario administrador), el valor del campo startIndex 0 o positivo y el valor del campo count positivo.
- e) Entrada(s): nombre(String), codCategoría(Long), admin(boolean), startIndex(int), count(int) -> (null, valor no libre, false, 0, 2)
- f) Salida(s): lista con objetos de tipo Evento
- g) Inicialización: creación de objetos de tipo Categoría y Evento

6.

- a) Código: "PR-UN-006"
- b) Unidad: ApuestasService
- c) Función/método: findEventosByCategoryAndName()
- d) Motivación: búsqueda de eventos por categoría y nombre de un usuario no administrador con valores correctos. Se consideran valores correctos: un valor no vacío o no nulo en el campo nombre, un código de categoría existente, el campo admin con valor false (que indica que el que realiza la búsqueda no es un usuario administrador), el valor del campo startIndex 0 o positivo y el valor del campo count positivo.
- e) Entrada(s): nombre(String), codCategoría(Long), admin(boolean), startIndex(int), count(int) -> ("Depor celt", valor no libre, false, 0, 2)
- f) Salida(s): lista con objetos de tipo Evento
- g) Inicialización: creación de objeto de tipo Categoría y Evento

7.

- a) Código: "PR-UN-007"
- b) Unidad: ApuestasService
- c) Función/método: findEventosByName()
- d) Motivación: búsqueda de eventos por nombre de un usuario no administrador con valores correctos. Se consideran valores correctos: un valor no vacío o no nulo en el campo nombre, un código de categoría que debe ser nulo (puesto que solo se buscará por nombre), el campo admin con valor false (que indica que el que realiza la búsqueda no es un usuario administrador), el valor del campo startIndex 0 o positivo y el valor del campo count positivo.
- e) Entrada(s): nombre(String), codCategoría(Long), admin(boolean), startIndex(int), count(int) -> ("federer", null, false, 0, 2)
- f) Salida(s): lista con objetos de tipo Evento
- g) Inicialización: creación de objeto de tipo Evento

8.

- a) Código: "PR-UN-008"
- b) Unidad: ApuestasService
- c) Función/método: findEventosByNameBeingAdmin()
- d) Motivación: búsqueda de eventos por nombre de un usuario administrador con valores correctos. Se consideran valores correctos: un valor no vacío o no nulo en el campo nombre, un código de categoría que debe ser nulo (puesto que solo se buscará por nombre), el campo admin con valor true (que indica que el que realiza la búsqueda es un usuario administrador), el valor del campo startIndex 0 o positivo y el valor del campo count positivo.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

e) Entrada(s): nombre(String), codCategoria(Long), admin(boolean), startIndex(int), count(int) -> ("federer", null, true, 0, 2);

f) Salida(s): lista con objetos de tipo Evento

g) Inicialización: creación de objeto de tipo Evento

9.

a) Código: "PR-UN-009"

b) Unidad: ApuestasService

c) Función/método: findEventosPagination()

d) Motivación:

e) Entrada(s): nombre(String), codCategoria(Long), admin(boolean), startIndex(int), count(int) -> (null, null, true, 0, 2)

f) Salida(s): lista con objetos de tipo Evento

g) Inicialización: creación de objetos de tipo Evento

10.

a) Código: "PR-UN-010"

b) Unidad: ApuestasService

c) Función/método: findEventosWithNonExistentCategoria()

d) Motivación: búsqueda de eventos por categoría de un usuario no administrador con un código de categoría que no existe (valor frontera). Se consideran valores correctos: un nombre que puede ser nulo o vacío si se quiere buscar por categoría o no serlo si se quiere buscar por nombre, un código de categoría existente si se quiere buscar por categoría o nulo si se quiere buscar por nombre, el campo admin con valor false (que indica que el que realiza la búsqueda no es un usuario administrador) o true (que indica que el que realiza la búsqueda es un usuario administrador), el valor del campo startIndex 0 o positivo y el valor del campo count positivo.

e) Entrada(s): nombre(String), codCategoria(Long), admin(boolean), startIndex(int), count(int) -> ("Depor celt", -1, false, 0, 10)

f) Salida(s): excepción InstanceNotFoundException

g) Inicialización: -

11.

a) Código: "PR-UN-011"

b) Unidad: ApuestasService

c) Función/método: createApuestaWithNonExistentOpcion()

d) Motivación: creación de una apuesta con un código de una opción de apuesta que no existe (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de una opción de apuesta existente, un valor para el campo cantidad mayor que 0 y un código de un usuario existente.

e) Entrada(s): opcion(Long), cantidad(double), userId(Long) -> (-1, 10, valor no libre);

f) Salida(s): excepción InstanceNotFoundException

g) Inicialización: creación de objeto de tipo UserProfile

12.

a) Código: "PR-UN-012"

b) Unidad: ApuestasService

c) Función/método: createApuestaWithNonExistentUser()

d) Motivación: creación de una apuesta con un código de un usuario que no existe (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de una opción de apuesta existente, un valor para el campo cantidad mayor que 0 y un código de un usuario existente.

e) Entrada(s): opcion(Long), cantidad(double), userId(Long) -> (valor no libre, 10, -1)

f) Salida(s): excepción InstanceNotFoundException

g) Inicialización: creación de objeto de tipo OpcionApuesta

13.

a) Código: "PR-UN-013"

b) Unidad: ApuestasService

c) Función/método: createApuesta()

d) Motivación: creación de una apuesta con valores correctos. Se consideran valores correctos: un código de una opción de apuesta existente, un valor para el campo cantidad mayor que 0 y un código de un usuario existente.

e) Entrada(s): opcion(Long), cantidad(double), userId(Long) -> (valor no libre, 10, valor no libre)

f) Salida(s): objeto de tipo Apuesta

g) Inicialización: creación de objetos de tipo OpcionApuesta y UserProfile

14.

a) Código: "PR-UN-014"

b) Unidad: ApuestasService

c) Función/método: createApuestaWithNegativeQuantity()

d) Motivación: creación de una apuesta con una cantidad igual a 0 (valor frontera) con el objetivo de que salte una excepción ya que no está permitido crear apuestas con una cantidad que no sea mayor que 0. Se consideran valores correctos: un código de una opción de apuesta existente, un valor para el campo cantidad mayor que 0 y un código de un usuario existente.

e) Entrada(s): opcion(Long), cantidad(double), userId(Long) -> (valor no libre, 0, valor no libre)

f) Salida(s): excepción InputValidationException

g) Inicialización: creación de objetos de tipo OpcionApuesta y UserProfile

15.

a) Código: "PR-UN-015"

b) Unidad: ApuestasService

c) Función/método: getTiposApuesta()

d) Motivación: obtención de los tipos de apuesta que ofrece un evento con valores correctos. Se consideran valores correctos: un código de un evento existente.

e) Entrada(s): codEvento(Long) -> (valor no libre)

f) Salida(s): objeto de tipo TipoApuestaDto

g) Inicialización: creación de objetos de tipo Evento

16.

a) Código: "PR-UN-016"

b) Unidad: ApuestasService

c) Función/método: getTiposApuestaWithNonExistentEvento()

d) Motivación: obtención de los tipos de apuesta que ofrece un evento con un código de evento que no existe (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un evento existente.

e) Entrada(s): codEvento(Long) -> (-1)

f) Salida(s): excepción InstanceNotFoundException

g) Inicialización: -



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

17.

a) Código: "PR-UN-017"

b) Unidad: ApuestasService

c) Función/método: createTipoApuesta()

d) Motivación: creación de un tipo de apuesta con valores correctos. Se consideran valores correctos: un tipo de apuesta con el campo nombre y multiple no nulo, una lista de opciones de apuesta y un código de un evento existente y que además no haya empezado.

e) Entrada(s): tipoApuesta(TipoApuesta), opciones(List<OpcionApuesta>), eventId(Long) -> (new TipoApuesta("Resultado", false), opcionesNoPersistentes, valor no libre)

```
*opcionesNoPersistentes -> List<OpcionApuesta> opcionesNoPersistentes = new ArrayList<OpcionApuesta>();
    OpcionApuesta opcionNoPersistente = new OpcionApuesta("2", 1.40, true);
    OpcionApuesta opcionNoPersistente2 = new OpcionApuesta("3", 3.4, true);
    OpcionApuesta opcionNoPersistente3 = new OpcionApuesta("4", 4.4, true);
```

f) Salida(s): objeto de tipo TipoApuesta

g) Inicialización: creación de objeto de tipo Evento

18.

a) Código: "PR-UN-018"

b) Unidad: ApuestasService

c) Función/método: createTipoApuestaWithEmptyName()

d) Motivación: creación de un tipo de apuesta con el valor del campo nombre vacío (valor frontera) con el objetivo de que salte una excepción ya que no está permitido crear tipos de apuesta sin nombre. Se consideran valores correctos: un tipo de apuesta con el campo nombre y multiple no nulo, una lista de opciones de apuesta y un código de un evento existente y que además no haya empezado.

e) Entrada(s): tipoApuesta(TipoApuesta), opciones(List<OpcionApuesta>), eventId(Long) -> (new TipoApuesta("", false), opcionesNoPersistentes, valor no libre)

```
*opcionesNoPersistentes -> List<OpcionApuesta> opcionesNoPersistentes = new ArrayList<OpcionApuesta>();
    OpcionApuesta opcionNoPersistente = new OpcionApuesta("2", 1.40, true);
    OpcionApuesta opcionNoPersistente2 = new OpcionApuesta("3", 3.4, true);
    OpcionApuesta opcionNoPersistente3 = new OpcionApuesta("4", 4.4, true);
```

f) Salida(s): excepción InputValidationException

g) Inicialización: creación de objeto de tipo Evento

19.

a) Código: "PR-UN-019"

b) Unidad: ApuestasService

c) Función/método: createTipoApuestaWithoutEvent()

d) Motivación: creación de un tipo de apuesta con un código de evento que no existe (los códigos, que son auto-generados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un tipo de apuesta con el campo nombre y multiple no nulo, una lista de opciones de apuesta y un código de un evento existente y que además no haya empezado.

e) Entrada(s): tipoApuesta(TipoApuesta), opciones(List<OpcionApuesta>), eventId(Long) -> (new TipoApuesta("Resultado", false), opcionesNoPersistentes, -1)

```
*opcionesNoPersistentes -> List<OpcionApuesta> opcionesNoPersistentes = new ArrayList<OpcionApuesta>();
    OpcionApuesta opcionNoPersistente = new OpcionApuesta("2", 1.40, true);
    OpcionApuesta opcionNoPersistente2 = new OpcionApuesta("3", 3.4, true);
    OpcionApuesta opcionNoPersistente3 = new OpcionApuesta("4", 4.4, true);
```



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

f) Saída(s): excepción InstanceNotFoundException

g) Inicialización: -

20.

a) Código: "PR-UN-020"

b) Unidad: ApuestasService

c) Función/método: createTipoApuestaWithStartedEvent()

d) Motivación: creación de un tipo de apuesta con un código de un evento que ya ha comenzado con el objetivo de que salte una excepción. Se consideran valores correctos: un tipo de apuesta con el campo nombre y multiple no nulo, una lista de opciones de apuesta y un código de un evento existente y que además no haya empezado.

e) Entrada(s): tipoApuesta(TipoApuesta), opciones(List<OpcionApuesta>), eventId(Long) -> (new TipoApuesta("Resultado", false), opcionesNoPersistentes, valor no libre)

```
*opcionesNoPersistentes -> List<OpcionApuesta> opcionesNoPersistentes = new ArrayList<OpcionApuesta>();
    OpcionApuesta opcionNoPersistente = new OpcionApuesta("2", 1.40, true);
    OpcionApuesta opcionNoPersistente2 = new OpcionApuesta("3", 3.4, true);
    OpcionApuesta opcionNoPersistente3 = new OpcionApuesta("4", 4.4, true);
```

f) Saída(s): excepción StartedEventException

g) Inicialización: creación de objeto de tipo Evento

21.

a) Código: "PR-UN-021"

b) Unidad: ApuestasService

c) Función/método: findApuestasByUserId()

d) Motivación: búsqueda de apuestas por código de un usuario con valores correctos. Se consideran valores correctos: un código de usuario existente, el valor del campo startIndex 0 o positivo y el valor del campo count positivo.

e) Entrada(s): userId(Long), startIndex(int), count(int) -> (valor no valido, 0, 2);

f) Saída(s): objeto de tipo ApuestaBlock

g) Inicialización: creación de objeto de tipo UserProfile

22.

a) Código: "PR-UN-022"

b) Unidad: ApuestasService

c) Función/método: marcarOpcionesGanadoras()

d) Motivación: marcación de las opciones de apuesta ganadoras de un tipo de apuesta con valores correctos. Se consideran valores correctos: un código de un tipo de apuesta existente cuyo evento al que hace referencia ya haya empezado y una lista no vacía de opciones de apuesta que no hayan sido marcadas ni invalidadas.

e) Entrada(s): codTipoApuesta(Long), opcionesGanadoras(List<Long>) -> (valor no libre, opcionesGanadoras)

```
*opcionesGanadoras -> List<Long> opcionesGanadoras = new ArrayList<Long>();
    opcionesGanadoras.add(valor no libre);
    opcionesGanadoras.add(valor no libre);
```

f) Saída(s): lista con objetos de tipo OpcionApuesta

g) Inicialización: creación de objetos de tipo TipoApuesta

23.

a) Código: "PR-UN-023"

b) Unidad: ApuestasService

c) Función/método: marcarOpcionesGanadorasMarcadasAnteriormente()



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

d) Motivación: marcación de las opciones de apuesta ganadoras de un tipo de apuesta que ya han sido marcadas con el objetivo de que salte una excepción ya que no se pueden marcar las opciones ganadoras si ya han sido marcadas. Se consideran valores correctos: un código de un tipo de apuesta existente cuyo evento al que hace referencia ya haya empezado y una lista no vacía de opciones de apuesta que no hayan sido marcadas ni invalidadas.

e) Entrada(s): `codTipoApuesta(Long)`, `opcionesGanadoras(List<Long>)` -> (valor no valido, `opcionesGanadoras`)
*`opcionesGanadoras` -> `List<Long>` `opcionesGanadoras = new ArrayList<Long>();`

`opcionesGanadoras.add(valor no libre);`

`opcionesGanadoras.add(valor no libre);`

f) Salida(s): excepción `ValidateOptionsException`

g) Inicialización: creación de objetos de tipo `TipoApuesta`

24.

a) Código: "PR-UN-024"

b) Unidad: `ApuestasService`

c) Función/método: `marcarOpcionesGanadorasMarcadasConEventoNoEmpezado()`

d) Motivación: marcación de las opciones de apuesta ganadoras de un tipo de apuesta cuyo evento al que hace referencia ya ha empezado (valor frontera) con el objetivo de que salte una excepción ya que no se pueden marcar opciones ganadoras si el evento ya ha comenzado. Se consideran valores correctos: un código de un tipo de apuesta existente cuyo evento al que hace referencia ya haya empezado y una lista no vacía de opciones de apuesta que no hayan sido marcadas ni invalidadas.

e) Entrada(s): `codTipoApuesta(Long)`, `opcionesGanadoras(List<Long>)` -> (valor no libre, `opcionesGanadoras`)

*`opcionesGanadoras` -> `List<Long>` `opcionesGanadoras = new ArrayList<Long>();`

`opcionesGanadoras.add(valor no libre);`

`opcionesGanadoras.add(valor no libre);`

f) Salida(s): excepción `NotStartedEventException`

g) Inicialización: creación de objetos de tipo `TipoApuesta`

25.

a) Código: "PR-UN-025"

b) Unidad: `ApuestasService`

c) Función/método: `marcarOpcionesGanadorasMarcadasConTipoApuestaInexistente()`

d) Motivación: marcación de las opciones de apuesta ganadoras de un tipo de apuesta con un código de tipo de apuesta que no existe (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un tipo de apuesta existente cuyo evento al que hace referencia ya haya empezado y una lista no vacía de opciones de apuesta que no hayan sido marcadas ni invalidadas.

e) Entrada(s): `codTipoApuesta(Long)`, `opcionesGanadoras(List<Long>)` -> (-1, `opcionesGanadoras`)

*`opcionesGanadoras` -> `List<Long>` `opcionesGanadoras = new ArrayList<Long>();`

`opcionesGanadoras.add(valor no libre);`

`opcionesGanadoras.add(valor no libre);`

f) Salida(s): excepción `InstanceNotFoundException`

g) Inicialización: creación de objetos de tipo `TipoApuesta`

26.

a) Código: "PR-UN-026"

b) Unidad: `ApuestasService`

c) Función/método: `marcarOpcionesGanadorasMarcadasConListaVacía()`



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

d) Motivación: marcación de las opciones de apuesta ganadoras de un tipo de apuesta con una lista de opciones de apuesta vacía con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un tipo de apuesta existente cuyo evento al que hace referencia ya haya empezado y una lista no vacía de opciones de apuesta que no hayan sido marcadas ni invalidadas.

e) Entrada(s): `codTipoApuesta(Long)`, `opcionesGanadoras(List<Long>)` -> (valor no libre, `listaVacía`)

*`listaVacía` -> `List<Long>` `listaVacía = new ArrayList<Long>();`

f) Salida(s): excepción `ValidateOptionsException`

g) Inicialización: creación de objetos de tipo `TipoApuesta`

27.

a) Código: "PR-UN-027"

b) Unidad: `ApuestasService`

c) Función/método: `marcarOpcionesGanadorasMarcadasConOpcionesInvalidas()`

d) Motivación: marcación de las opciones de apuesta ganadoras de un tipo de apuesta con una lista de opciones que han sido invalidadas con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un tipo de apuesta existente cuyo evento al que hace referencia ya haya empezado y una lista no vacía de opciones de apuesta que no hayan sido marcadas ni invalidadas.

e) Entrada(s): `codTipoApuesta(Long)`, `opcionesGanadoras(List<Long>)` -> (valor no libre, `opcionesGanadoras`)

*`opcionesGanadoras` -> `List<Long>` `opcionesGanadoras = new ArrayList<Long>();`

`opcionesGanadoras.add(opcion1.getCodOpcionApuesta());`

`opcionesGanadoras.add(opcion4.getCodOpcionApuesta());`

`opcionesGanadoras.add(opcion.getCodOpcionApuesta());`

f) Salida(s): excepción `ValidateOptionsException`

g) Inicialización: creación de objetos de tipo `TipoApuesta`

28.

a) Código: "PR-UN-028"

b) Unidad: `ApuestasService`

c) Función/método: `marcarOpcionesGanadorasDeApuestaSimpleConMultiplesOpciones()`

d) Motivación:

e) Entrada(s): `codTipoApuesta(Long)`, `opcionesGanadoras(List<Long>)`

f) Salida(s): excepción `ValidateOptionsException`

g) Inicialización: creación de objetos de tipo `TipoApuesta`

29.

a) Código: "PR-UN-029"

b) Unidad: `ApuestasService`

c) Función/método: `isOpcionesApuestaValidated()`

d) Motivación: comprobación de si las opciones de apuesta de un tipo de apuesta han sido validadas utilizando valores correctos. Se consideran valores correctos: un código de tipo de apuesta existente con sus pertenecientes opciones de apuesta.

e) Entrada(s): `codTipoApuesta(Long)` -> (valor no libre)

f) Salida(s): boolean

g) Inicialización: creación de objeto de tipo `TipoApuesta`

30.

a) Código: "PR-UN-030"

b) Unidad: `ApuestasService`

c) Función/método: `isOpcionesApuestaValidatedWithNonExistentTipoApuesta()`



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

d) Motivación: comprobación de si las opciones de apuesta de un tipo de apuesta han sido validadas con un código de tipo de apuesta que no existe (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de tipo de apuesta existente con sus pertenecientes opciones de apuesta.

e) Entrada(s): `codTipoApuesta(Long)` -> (-1)

f) Salida(s): excepción `InstanceNotFoundException`

g) Inicialización: -

31.

a) Código: "PR-UN-031"

b) Unidad: `ApuestasService`

c) Función/método: `getCategorias()`

d) Motivación: obtención de las categorías.

e) Entrada(s): -

f) Salida(s): Lista de objetos de tipo `Categoria`

g) Inicialización: creación de objetos de tipo `Categoria`

32.

a) Código: "PR-UN-032"

b) Unidad: `ApuestasService`

c) Función/método: `findEvento()`

d) Motivación: búsqueda de un evento con valores correctos. Se consideran valores correctos: un código de evento existente.

e) Entrada(s): `codEvento(Long)` -> (valor no libre)

f) Salida(s): objeto de tipo `Evento`

g) Inicialización: creación de objeto de tipo `Evento`

33.

a) Código: "PR-UN-033"

b) Unidad: `ApuestasService`

c) Función/método: `findCategoria()`

d) Motivación: obtención de una categoría con valores correctos. Se consideran valores correctos: un código de categoría existente.

e) Entrada(s): `codCategoria(Long)` -> (valor no libre)

f) Salida(s): objeto de tipo `Categoria`

g) Inicialización: creación de objeto de tipo `Categoria`

34.

a) Código: "PR-UN-034"

b) Unidad: `ApuestasService`

c) Función/método: `findTipoApuesta()`

d) Motivación: obtención de un tipo de apuesta con valores correctos. Se consideran valores correctos: un código de tipo de apuesta existente.

e) Entrada(s): `codTipoApuesta(Long)` -> (valor no libre)

f) Salida(s): objeto de tipo `TipoApuesta`

g) Inicialización: creación de objeto de tipo `TipoApuesta`

35.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- a) Código: "PR-UN-035"
b) Unidade: ApuestasService
c) Función/método: findOpcionApuesta()
d) Motivación: obtención de una opción de apuesta con valores correctos. Se consideran valores correctos: un código de opción de apuesta existente.
e) Entrada(s): codOpcionApuesta(Long) -> (valor no libre)
f) Saída(s): objeto de tipo OpcionApuesta
g) Inicialización: creación de objeto de tipo OpcionApuesta

36.

- a) Código: "PR-UN-036"
b) Unidade: ApuestasService
c) Función/método: findApuesta()
d) Motivación: obtención de una apuesta con valores correctos. Se consideran valores correctos: un código de apuesta existente.
e) Entrada(s): codApuesta(Long) -> (valor no libre)
f) Saída(s): objeto de tipo Apuesta
g) Inicialización: creación de objeto de tipo Apuesta

37.

- a) Código: "PR-UN-037"
b) Unidade: UserService
c) Función/método: registerUser()
d) Motivación: se registra un nuevo usuario con valores correctos. Se consideran valores correctos un valor no nulo o no vacío en el campo loginName y clearPassword y un UserProfileDetails con valores no nulos.
e) Entrada(s): loginName(String), clearPassword(String), userProfileDetails(UserProfileDetails) -> ("prueba", "password", new UserProfileDetails("adrian", "lopez", "adri@udc.es"))
f) Saída(s): objeto de tipo UserProfile
g) Inicialización: creación de objeto de tipo UserProfile

38.

- a) Código: "PR-UN-038"
b) Unidade: UserService
c) Función/método: registerDuplicatedUser()
d) Motivación: se registra un usuario que ya ha sido registrado (valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos un valor no nulo o no vacío en el campo loginName y clearPassword y un UserProfileDetails con valores no nulos.
e) Entrada(s): loginName(String), clearPassword(String), userProfileDetails(UserProfileDetails) -> ("user", "userPassword", new UserProfileDetails("name", "lastName", "user@udc.es"))
f) Saída(s): excepción DuplicateInstanceException
g) Inicialización: creación de objeto de tipo userProfile

39.

- a) Código: "PR-UN-039"
b) Unidade: UserService
c) Función/método: findUserProfile()
d) Motivación: búsqueda de un usuario con valores correctos. Se consideran valores correctos: un código de usuario existente.
e) Entrada(s): userProfileId(Long) -> (valor no libre)



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- f) Saída(s): objeto de tipo UserProfile
- g) Inicialización: creación de objeto de tipo userProfile

40.

- a) Código: "PR-UN-040"
- b) Unidad: UserService
- c) Función/método: loginClearPassword()
- d) Motivación: logeo de un usuario con contraseña en claro. Se consideran valores correctos: un valor no nulo o no vacío en el campo nombre y password y un valor false en el campo passwordIsEncrypted indicando que la contraseña no está encriptada.
- e) Entrada(s): loginName(String), password(String), passwordIsEncrypted(boolean) -> ("usuario", "userPassword", false)
- f) Saída(s): objeto de tipo UserProfile
- g) Inicialización: creación de objeto de tipo UserProfile

41.

- a) Código: "PR-UN-041"
- b) Unidad: UserService
- c) Función/método: loginEncryptedPassword()
- d) Motivación: logeo de un usuario con contraseña encriptada. Se consideran valores correctos: un valor no nulo o no vacío en el campo nombre, un valor encriptado en el campo password y un valor true en el campo passwordIsEncrypted indicando que la contraseña está encriptada.
- e) Entrada(s): loginName(String), password(String), passwordIsEncrypted(boolean) -> ("user", valor no libre, true)
- f) Saída(s): objeto de tipo UserProfile
- g) Inicialización: creación de objeto de tipo UserProfile

42.

- a) Código: "PR-UN-042"
- b) Unidad: ApuestasService
- c) Función/método: loginIncorrectPasword()
- d) Motivación: logeo de un usuario con contraseña incorrecta (valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un valor no nulo o no vacío en el campo nombre, un valor encriptado en el campo password y un valor true en el campo passwordIsEncrypted indicando que la contraseña está encriptada.
- e) Entrada(s): loginName(String), password(String), passwordIsEncrypted(boolean) -> ("user", 'X' + "userPassword", false)
- f) Saída(s): excepción IncorrectPasswordException
- g) Inicialización: creación de objeto de tipo UserProfile

43.

- a) Código: "PR-UN-043"
- b) Unidad: UserService
- c) Función/método: loginWithNonExistentUser()
- d) Motivación: logeo de un usuario que no existe (valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un valor no nulo o no vacío en el campo nombre, un valor encriptado en el campo password y un valor true en el campo passwordIsEncrypted indicando que la contraseña está encriptada.
- e) Entrada(s): loginName(String), password(String), passwordIsEncrypted(boolean) -> ("userNonExistent", "userPassword", false)



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

f) Saída(s): excepción InstanceNotFoundException
g) Inicialización: -

44.

a) Código: "PR-UN-044"
b) Unidad: UserService
c) Función/método: findNonExistentUser()
d) Motivación: búsqueda de un usuario no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de usuario existente.
e) Entrada(s): userProfileId(Long) -> (-1)
f) Saída(s): excepción InstanceNotFoundException
g) Inicialización: -

45.

a) Código: "PR-UN-045"
b) Unidad: ApuestasService
c) Función/método: update()
d) Motivación: actualización de un usuario con valores correctos. Se consideran valores correctos un código de usuario existente y un UserProfileDetails con todos sus campos no nulos ni vacíos.
e) Entrada(s): userProfileId(Long), userProfileDetails(UserProfileDetails) -> (valor no libre, new UserProfileDetails('X' + "nombre", 'X' + "apellido", 'X' + "usuario@udc.es"))
f) Saída(s): -
g) Inicialización: creación de objeto de tipo UserProfile

46.

a) Código: "PR-UN-046"
b) Unidad: UserService
c) Función/método: updateWithNonExistentUser()
d) Motivación: actualización de un usuario no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un usuario existente y un UserProfileDetails con todos sus campos no nulos ni vacíos.
e) Entrada(s): userProfileId(Long), userProfileDetails(UserProfileDetails) -> (-1, new UserProfileDetails("name", "lastName", "user@udc.es"))
f) Saída(s): excepción InstanceNotFoundException
g) Inicialización: creación de objeto de tipo UserProfile

47.

a) Código: "PR-UN-047"
b) Unidad: UserService
c) Función/método: changePassword()
d) Motivación: cambio de contraseña de un usuario con valores correctos. Se consideran valores correctos: un código de un usuario existente, un valor en el campo oldClearPassword que coincida con la contraseña actual de usuario y un valor no nulo y no vacío en el campo newClearPassword.
e) Entrada(s): userProfileId(Long), oldClearPassword(String), newClearPassword(String) -> (valor no libre, "userPassword", 'X' + "userPassword")
f) Saída(s): -
g) Inicialización: creación de objeto de tipo UserProfile



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

48.

- a) Código: "PR-UN-048"
- b) Unidad: UserService
- c) Función/método: changePasswordWithIncorrectPassword()
- d) Motivación: cambio de contraseña de un usuario con una contraseña que no coincide con la actual (valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un usuario existente, un valor en el campo oldClearPassword que coincida con la contraseña actual de usuario y un valor no nulo y no vacío en el campo newClearPassword.
- e) Entrada(s): userProfileId(Long), oldClearPassword(String), newClearPassword(String) -> (valor no libre, 'X' + "userPassword", 'Y' + "userPassword")
- f) Salida(s): -
- g) Inicialización: creación de objeto de tipo UserProfile

49.

- a) Código: "PR-UN-049"
- b) Unidad: UserService
- c) Función/método: changePasswordWithNonExistentUser()
- d) Motivación: cambio de contraseña de un usuario no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un usuario existente, un valor en el campo oldClearPassword que coincida con la contraseña actual de usuario y un valor no nulo y no vacío en el campo newClearPassword.
- e) Entrada(s): userProfileId(Long), oldClearPassword(String), newClearPassword(String) -> (-1, "userPassword", "XuserPassword")
- f) Salida(s): -
- g) Inicialización: -

50.

- a) Código: "PR-UN-050"
- b) Unidad: Apuesta
- c) Función/método: save()
- d) Motivación: almacenar una apuesta en base de datos con valores correctos. Se consideran valores correctos: una apuesta válida.
- e) Entrada(s): entity(E) -> (new Apuesta(opcion, 10, userProfile, fecha)
- f) Salida(s): -
- g) Inicialización: creación de objeto de tipo Opcion Apuesta y UserProfile

51.

- a) Código: "PR-UN-051"
- b) Unidad: Apuesta
- c) Función/método: find()
- d) Motivación: búsqueda de una apuesta en base de datos con valores correctos. Se consideran valores correctos: un código de una apuesta existente.
- e) Entrada(s): id(PK) -> (valor no libre)
- f) Salida(s): objeto de tipo E
- g) Inicialización: creación de objeto de tipo Apuesta

52.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- a) Código: "PR-UN-052"
b) Unidade: Apuesta
c) Función/método: findWithNoExistentId()
d) Motivación: búsqueda en base de datos de una apuesta no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de una apuesta existente.
e) Entrada(s): id(PK) -> (-1)
f) Saída(s): objeto de tipo E
g) Inicialización: -

53.

- a) Código: "PR-UN-053"
b) Unidade: Apuesta
c) Función/método: remove()
d) Motivación: borrado en base de datos de una apuesta con valores correctos. Se consideran valores correctos: un código de una apuesta existente.
e) Entrada(s): id(PK) -> (valor no libre)
f) Saída(s): -
g) Inicialización: creación de objeto de tipo Apuesta

54.

- a) Código: "PR-UN-054"
b) Unidade: Apuesta
c) Función/método: removeWithNoExistentId()
d) Motivación: borrado en base de datos de una apuesta no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de una apuesta existente.
e) Entrada(s): id(PK) -> (valor no libre)
f) Saída(s): -
g) Inicialización: -

55.

- a) Código: "PR-UN-055"
b) Unidade: Apuesta
c) Función/método: findByUserId()
d) Motivación: búsqueda en base de datos de las apuestas de un usuario con valores correctos. Se consideran valores correctos: un código de un usuario existente, el valor del campo startIndex 0 o positivo y el valor del campo count positivo.
e) Entrada(s): userId(Long), startIndex(int), count(int) -> (valor no libre, 0, 5)
f) Saída(s): lista de apuestas
g) Inicialización: creación de objetos de tipo Apuesta

56.

- a) Código: "PR-UN-056"
b) Unidade: Categoria
c) Función/método: save()
d) Motivación: almacenar una categoria en base de datos con valores correctos. Se consideran valores correctos: una categoria válida.
e) Entrada(s): entity(E) -> new Categoria("categoria")



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- f) Saída(s): -
- g) Inicialización: -

57.

- a) Código: "PR-UN-057"
- b) Unidade: Categoría
- c) Función/método: find()
- d) Motivación: búsqueda de una categoría en base de datos con valores correctos. Se consideran valores correctos: un código de una categoría existente.
- e) Entrada(s): id(PK) -> (valor no libre)
- f) Saída(s): objeto de tipo E
- g) Inicialización: creación de objeto de tipo Categoría

58.

- a) Código: "PR-UN-058"
- b) Unidade: Categoría
- c) Función/método: findWithNoExistentId()
- d) Motivación: búsqueda en base de datos de una categoría no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de una categoría existente.
- e) Entrada(s): id(PK) -> (-1)
- f) Saída(s): objeto de tipo E
- g) Inicialización: -

59.

- a) Código: "PR-UN-059"
- b) Unidade: Categoría
- c) Función/método: remove()
- d) Motivación: borrado en base de datos de una categoría con valores correctos. Se consideran valores correctos: un código de una categoría existente.
- e) Entrada(s): id(PK) -> (valor no libre)
- f) Saída(s): -
- g) Inicialización: creación de objeto de tipo Categoría

60.

- a) Código: "PR-UN-060"
- b) Unidade: Categoría
- c) Función/método: removeWithNoExistentId()
- d) Motivación: borrado en base de datos de una categoría no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de una categoría existente.
- e) Entrada(s): id(PK) -> (valor no libre)
- f) Saída(s): -
- g) Inicialización: -

61.

- a) Código: "PR-UN-061"
- b) Unidade: Categoría
- c) Función/método: findCategorías()



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

d) Motivación: búsqueda de todas las categorías existentes en base de datos con valores correctos. No recibe parámetros.

e) Entrada(s): -

f) Salida(s): lista con todas las categorías existentes

g) Inicialización: creación de objeto de tipo Categoría

62.

a) Código: "PR-UN-062"

b) Unidad: Evento

c) Función/método: save()

d) Motivación: almacenar un evento en base de datos con valores correctos. Se consideran valores correctos: un evento válido.

e) Entrada(s): entity(E) -> (new Evento("Barsa- Atletico", "24/12/2017", categoria))

f) Salida(s): -

g) Inicialización: creación de objeto de tipo Categoría

63.

a) Código: "PR-UN-063"

b) Unidad: Evento

c) Función/método: find()

d) Motivación: búsqueda de un evento en base de datos con valores correctos. Se consideran valores correctos: un código de un evento existente.

e) Entrada(s): id(PK) -> (valor no libre)

f) Salida(s): objeto de tipo E

g) Inicialización: creación de objeto de tipo Evento

64.

a) Código: "PR-UN-064"

b) Unidad: Evento

c) Función/método: findWithNoExistentId()

d) Motivación: búsqueda en base de datos de un evento no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un evento existente.

e) Entrada(s): id(PK) -> (-1)

f) Salida(s): objeto de tipo E

g) Inicialización: -

65.

a) Código: "PR-UN-065"

b) Unidad: Evento

c) Función/método: remove()

d) Motivación: borrado en base de datos de un evento con valores correctos. Se consideran valores correctos: un código de un evento existente.

e) Entrada(s): id(PK) -> (valor no libre)

f) Salida(s): -

g) Inicialización: creación de objeto de tipo Evento

66.

a) Código: "PR-UN-066"



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

- b) Unidade: Evento
c) Función/método: `removeWithNoExistentId()`
d) Motivación: borrado en base de datos de un evento no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un evento existente.
e) Entrada(s): `id(PK)` -> (valor no libre)
f) Salida(s): -
g) Inicialización: -

67.

- a) Código: "PR-UN-067"
b) Unidade: Evento
c) Función/método: `findEventosAbiertos()`
d) Motivación: búsqueda en base de datos de los eventos que están abiertos con valores correctos. Se consideran valores correctos un valor no nulo y no vacío para el campo nombre, un código de una categoría existente, un valor boolean en el campo `admin` true o false indicando si el que realiza la búsqueda es un usuario administrador o no, el valor del campo `startIndex` 0 o positivo y el valor del campo `count` positivo.
e) Entrada(s): `nombre(String)`, `codCategoria(Long)`, `admin(boolean)`, `startIndex(int)`, `count(int)` -> ("barsa", valor no libre, false, 0, 5)
f) Salida(s): lista de eventos
g) Inicialización: creación de objeto de tipo `Categoria`

68.

- a) Código: "PR-UN-068"
b) Unidade: `OpcionApuesta`
c) Función/método: `save()`
d) Motivación: almacenar una opcion de apuesta en base de datos con valores correctos. Se consideran valores correctos: una opcion de apuesta válida.
e) Entrada(s): `entity(E)` -> (`new OpcionApuesta("X", 1.70, null, tipoApuesta)`)
f) Salida(s): -
g) Inicialización: creación de objeto de tipo `TipoApuesta`

69.

- a) Código: "PR-UN-069"
b) Unidade: `OpcionApuesta`
c) Función/método: `find()`
d) Motivación: búsqueda de un evento en base de datos con valores correctos. Se consideran valores correctos: un código de un evento existente.
e) Entrada(s): `id(PK)` -> (valor no libre)
f) Salida(s): objeto de tipo `E`
g) Inicialización: creación de objeto de tipo `OpcionApuesta`

70.

- a) Código: "PR-UN-070"
b) Unidade: `OpcionApuesta`
c) Función/método: `findWithNoExistentId()`
d) Motivación: búsqueda en base de datos de una opcion de apuesta no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor fron-



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

tera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de una opción de apuesta existente.

e) Entrada(s): id(PK) -> (-1)

f) Salida(s): objeto de tipo E

g) Inicialización: -

71.

a) Código: "PR-UN-071"

b) Unidad: OpcionApuesta

c) Función/método: remove()

d) Motivación: borrado en base de datos de una opción de apuesta con valores correctos. Se consideran valores correctos: un código de una opción de apuesta existente.

e) Entrada(s): id(PK) -> (valor no libre)

f) Salida(s): -

g) Inicialización: creación de objeto de tipo OpcionApuesta

72.

a) Código: "PR-UN-072"

b) Unidad: OpcionApuesta

c) Función/método: removeWithNoExistentId()

d) Motivación: borrado en base de datos de una opción de apuesta no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de una opción de apuesta existente.

e) Entrada(s): id(PK) -> (valor no libre)

f) Salida(s): -

g) Inicialización: -

73.

a) Código: "PR-UN-073"

b) Unidad: OpcionApuesta

c) Función/método: validarOpciones()

d) Motivación: validación de opciones ganadoras en base de datos con valores correctos. Se consideran valores correctos: una lista con códigos de opciones de apuesta existentes.

e) Entrada(s): (opciones(List<Long>)) -> (ganadoras)

*ganadoras -> List<Long> ganadoras=new ArrayList<Long>()
ganadoras.add(valor no libre)

f) Salida(s): -

g) Inicialización: creación de objetos de tipo OpcionApuesta

74.

a) Código: "PR-UN-074"

b) Unidad: OpcionApuesta

c) Función/método: invalidarOpciones()

d) Motivación:

e) Entrada(s): codTipoApuesta(Long) -> (valor no libre)

f) Salida(s): -

g) Inicialización: creación de objeto de tipo TipoApuesta



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

75.

- a) Código: "PR-UN-075"
- b) Unidad: OpcionApuesta
- c) Función/método: getOpcionesApuesta()
- d) Motivación: búsqueda de todas las opciones de apuesta de un tipo de apuesta existentes en base de datos con valores correctos. Se consideran valores correctos: un código de tipo de apuesta existente.
- e) Entrada(s): codTipoApuesta(Long) -> (valor no libre)
- f) Salida(s): lista de opciones de apuesta
- g) Inicialización: creación de objetos de tipo TipoApuesta

76.

- a) Código: "PR-UN-076"
- b) Unidad: OpcionApuesta
- c) Función/método: isOpcionesValidated()
- d) Motivación: comprobación en base de datos de si las opciones de apuesta de un tipo de apuesta ya han sido validadas. Se consideran valores correctos: un código de tipo de apuesta existente.
- e) Entrada(s): codTipoApuesta(Long) -> (valor no libre)
- f) Salida(s): boolean
- g) Inicialización: creación de objeto de tipo TipoApuesta

77.

- a) Código: "PR-UN-077"
- b) Unidad: TipoApuesta
- c) Función/método: save()
- d) Motivación: almacenar un tipo de apuesta en base de datos con valores correctos. Se consideran valores correctos: un tipo de apuesta válido.
- e) Entrada(s): entity(E) -> (new TipoApuesta("Goleador", false, evento))
- f) Salida(s): -
- g) Inicialización: -

78.

- a) Código: "PR-UN-078"
- b) Unidad: TipoApuesta
- c) Función/método: find()
- d) Motivación: búsqueda de un tipo de apuesta en base de datos con valores correctos. Se consideran valores correctos: un código de un tipo de apuesta existente.
- e) Entrada(s): id(PK) -> (valor no libre)
- f) Salida(s): objeto de tipo E
- g) Inicialización: creación de objeto de tipo TipoApuesta

79.

- a) Código: "PR-UN-079"
- b) Unidad: TipoApuesta
- c) Función/método: findWithNoExistentId()
- d) Motivación: búsqueda en base de datos de un tipo de apuesta no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un tipo de apuesta existente.
- e) Entrada(s): id(PK) -> (-1)
- f) Salida(s): objeto de tipo E



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

g) Inicialización: -

80.

a) Código: "PR-UN-080"

b) Unidade: TipoApuesta

c) Función/método: remove()

d) Motivación: borrado en base de datos de un tipo de apuesta con valores correctos. Se consideran valores correctos: un código de un tipo de apuesta existente.

e) Entrada(s): id(PK) -> (valor no libre)

f) Saída(s): -

g) Inicialización: creación de objeto de tipo TipoApuesta

81.

a) Código: "PR-UN-081"

b) Unidade: TipoApuesta

c) Función/método: removeWithNoExistentId()

d) Motivación: borrado en base de datos de un tipo de apuesta no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un tipo de apuesta existente.

e) Entrada(s): id(PK) -> (valor no libre)

f) Saída(s): -

g) Inicialización: -

82.

a) Código: "PR-UN-082"

b) Unidade: TipoApuesta

c) Función/método: findTipoApuesta()

d) Motivación: búsqueda de todos los tipos de apuesta de un evento existentes en base de datos con valores correctos. Se consideran valores correctos: un código de un evento existente.

e) Entrada(s): codEvento(Long) -> (valor no libre)

f) Saída(s): lista con tipos de apuesta

g) Inicialización: creación de objetos de tipo TipoApuesta y Evento.

83.

a) Código: "PR-UN-083"

b) Unidade: UserProfile

c) Función/método: save()

d) Motivación: almacenar un perfil de usuario en base de datos con valores correctos. Se consideran valores correctos: un perfil de usuario válido.

e) Entrada(s): entity(E) -> (new UserProfile("user3", "userPassword3", "name3", "lastName3", "user3@udc.es"))

f) Saída(s): -

g) Inicialización: -

84.

a) Código: "PR-UN-084"

b) Unidade: UserProfile

c) Función/método: find()

d) Motivación: búsqueda de un perfil de usuario en base de datos con valores correctos. Se consideran valores correctos: un código de un perfil de usuario existente.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- e) Entrada(s): id(PK) -> (valor no libre)
- f) Salida(s): objeto de tipo E
- g) Inicialización: creación de objeto de tipo UserProfile

85.

- a) Código: "PR-UN-085"
- b) Unidade: UserProfile
- c) Función/método: findWithNoExistentId()
- d) Motivación: búsqueda en base de datos de un perfil de usuario no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de un perfil de usuario existente.
- e) Entrada(s): id(PK) -> (-1)
- f) Salida(s): objeto de tipo E
- g) Inicialización: -

86.

- a) Código: "PR-UN-086"
- b) Unidade: UserProfile
- c) Función/método: remove()
- d) Motivación: borrado en base de datos de un perfil de usuario con valores correctos. Se consideran valores correctos: un código de un perfil de usuario existente.
- e) Entrada(s): id(PK) -> (valor no libre)
- f) Salida(s): -
- g) Inicialización: creación de objeto de tipo UserProfile

87.

- a) Código: "PR-UN-087"
- b) Unidade: UserProfile
- c) Función/método: removeWithNoExistentId()
- d) Motivación: borrado en base de datos de un perfil de usuario no existente (los códigos, que son autogenerados, comienzan en 1 y siguen de forma incremental por lo que el código -1 no es un valor correcto; valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un código de perfil de usuario existente.
- e) Entrada(s): id(PK) -> (valor no libre)
- f) Salida(s): -
- g) Inicialización: -

88.

- a) Código: "PR-UN-088"
- b) Unidade: UserProfile
- c) Función/método: findByUserLogin()
- d) Motivación: búsqueda en base de datos de un perfil de usuario por su login con valores correctos. Se consideran valores correctos: un valor no nulo, no vacío y existente en el campo loginName.
- e) Entrada(s): loginName(String) -> ("user")
- f) Salida(s): -
- g) Inicialización: creación de objeto de tipo UserProfile

89.

- a) Código: "PR-UN-089"



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- b) Unidad: UserProfile
c) Función/método: findByUserLoginWithNoExistentLogin()
d) Motivación: búsqueda en base de datos de un perfil de usuario por un login que no existe (valor frontera) con el objetivo de que salte una excepción. Se consideran valores correctos: un valor no nulo, no vacío y existente en el campo loginName.
e) Entrada(s): loginName(String) -> ("")
f) Salida(s): excepción InstanceNotFoundException
g) Inicialización: creación de objeto de tipo UserProfile

90.

- a) Código: "PR-IN-090"
b) Unidad: ApuestaService
c) Función/método: findEventosWithoutAdmin
d) Motivación: búsqueda de eventos de un usuario no administrador con valores correctos. Se consideran valores correctos: un valor no vacío o no nulo en el campo nombre, un código de categoría existente, el campo admin con valor false (que indica que el que realiza la búsqueda no es un usuario administrador), el valor del campo startIndex 0 o positivo y el valor del campo count positivo.
e) Entrada(s): nombre(String), codCategoría(Long), admin(boolean), startIndex(int), count(int) -> ("madrid", valor no libre, false, 0, 5);
f) Salida(s): objeto de tipo EventoBlock
g) Inicialización: objeto de tipo Categoría

91.

- a) Código: "PR-IN-091"
b) Unidad: ApuestaService
c) Función/método: findEventosWithAdmin()
d) Motivación: búsqueda de eventos de un usuario no administrador con valores correctos. Se consideran valores correctos: un valor no vacío o no nulo en el campo nombre, un código de categoría existente, el campo admin con valor true (que indica que el que realiza la búsqueda no es un usuario administrador), el valor del campo startIndex 0 o positivo y el valor del campo count positivo.
e) Entrada(s): nombre(String), codCategoría(Long), admin(boolean), startIndex(int), count(int) -> ("madrid", valor no libre, true, 0, 5);
f) Salida(s): objeto de tipo EventoBlock
g) Inicialización: objeto de tipo Categoría y Evento

92.

- a) Código: "PR-IN-092"
b) Unidad: ApuestaService
c) Función/método: bet()
d) Motivación: creación de una apuesta con valores correctos. Se consideran valores correctos: un código de opción de apuesta existente, una cantidad positiva y un código de usuario existente.
e) Entrada(s): opcion(Long), cantidad(double), userId(Long) -> (valor no libre, 2, valor no libre)
f) Salida(s): objeto de tipo Apuesta
g) Inicialización: objeto de tipo Categoría

93.

- a) Código: "PR-IN-093"
b) Unidad: ApuestaService
c) Función/método: selectWinnerBetOptions()



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- d) Motivación: marcado de las opciones ganadoras de una apuesta con valores correctos. Se consideran valores correctos: un código de tipo de apuesta existente y una lista de códigos de opciones de apuesta existentes.
- e) Entrada(s): codTipoApuesta(Long), opcionesGanadoras(List<Long>) -> (valor no libre, valor no libre)
- f) Salida(s): Lista de opciones de apuesta
- g) Inicialización: objeto de tipo Categoria

94.

- a) Código: "PR-IN-094"
- b) Unidad: ApuestaService
- c) Función/método: findUserBets()
- d) Motivación: búsqueda de las apuestas realizadas por un usuario concreto con valores correctos. Se consideran valores correctos: un código de usuario existente, el valor del campo startIndex 0 o positivo y el valor del campo count positivo.
- e) Entrada(s): userId(Long), startIndex(int), count(int) -> (valor no libre, 0, 5)
- f) Salida(s): objeto de tipo ApuestaBlock
- g) Inicialización: objeto de tipo Categoria

95.

- a) Código: "PR-UN-095"
- b) Unidad: Apuesta
- c) Función/método: generator()
- d) Motivación: generar una serie de apuestas con valores correctos. Se consideran valores correctos: una apuesta válida.
- e) Entrada(s): entity(E) -> (valores aleatorios)
- f) Salida(s): -
- g) Inicialización: -

96.

- a) Código: "PR-UN-096"
- b) Unidad: Apuesta
- c) Función/método: saveRandomBets()
- d) Motivación: almacenar una serie de apuestas en base de datos con valores correctos. Se consideran valores correctos: una apuesta válida.
- e) Entrada(s): entity(E) -> (valores aleatorios)
- f) Salida(s): -
- g) Inicialización: -

97.

- a) Código: "PR-UN-097"
- b) Unidad: Categoria
- c) Función/método: generator()
- d) Motivación: generar una serie de categorias con valores correctos. Se consideran valores correctos: una categoria válida.
- e) Entrada(s): entity(E) -> (valores aleatorios)
- f) Salida(s): -
- g) Inicialización: -

98.

- a) Código: "PR-UN-098"



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- b) Unidade: Categoría
c) Función/método: saveRandomCategories()
d) Motivación: almacenar una serie de categorías en base de datos con valores correctos. Se consideran valores correctos: una categoría válida.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -

99.

- a) Código: "PR-UN-099"
b) Unidade: Evento
c) Función/método: generator()
d) Motivación: generar una serie de eventos con valores correctos. Se consideran valores correctos: un evento válido.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -

100.

- a) Código: "PR-UN-0100"
b) Unidade: Evento
c) Función/método: saveRandomEvents()
d) Motivación: almacenar una serie de eventos en base de datos con valores correctos. Se consideran valores correctos: un evento válido.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -

101.

- a) Código: "PR-UN-0101"
b) Unidade: OpcionApuesta
c) Función/método: generator()
d) Motivación: generar una serie de opciones de apuesta con valores correctos. Se consideran valores correctos: una opción de apuesta válida.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -

102.

- a) Código: "PR-UN-0102"
b) Unidade: OpcionApuesta
c) Función/método: saveRandomBetOptions()
d) Motivación: almacenar una serie de opciones de apuesta en base de datos con valores correctos. Se consideran valores correctos: una opción de apuesta válida.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -

103.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

- a) Código: "PR-UN-0103"
b) Unidade: TipoApuesta
c) Función/método: generator()
d) Motivación: generar una serie de tipos de apuesta con valores correctos. Se consideran valores correctos: un tipo de apuesta válido.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -

104.

- a) Código: "PR-UN-0104"
b) Unidade: TipoApuesta
c) Función/método: saveRandomBetTypes()
d) Motivación: almacenar una serie de tipos de apuesta en base de datos con valores correctos. Se consideran valores correctos: un tipo de apuesta válido.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -

105.

- a) Código: "PR-UN-0105"
b) Unidade: UserProfile
c) Función/método: generator()
d) Motivación: generar una serie de perfiles de usuario con valores correctos. Se consideran valores correctos: un perfil de usuario válido.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -

106.

- a) Código: "PR-UN-0106"
b) Unidade: UserProfile
c) Función/método: saveRandomUserProfiles()
d) Motivación: almacenar una serie de perfiles de usuario en base de datos con valores correctos. Se consideran valores correctos: un perfil de usuario válido.
e) Entrada(s): entity(E) -> (valores aleatorios)
f) Saída(s): -
g) Inicialización: -



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

4. REXISTRO DE PROBAS

a. Primeira semana

Na primeira semana de testing fixemos probas unitarias da aplicación utilizando a ferramenta JUnit.

Para isto modificamos casos de proba xa feitos e engadimos novos casos de proba, individualizando a funcionalidade a probar en cada caso de proba e estruturando cada caso de proba en tres partes: inicialización, chamada á funcionalidade e aserción. Ademais tamén incluínos nun método anotado con `@Before` a inicialización de aqueles datos necesarios para a execución de varios casos de proba. Desta maneira, obtivemos un total de 89 casos de proba (Dende PR-UN-001 ata PR-UN-089).

b. Segunda semana

Na segunda semana de testing usamos ferramentas de análise estática de caixa branca, mutación e estrés.

Para as probas de caixa branca usamos a ferramenta FindBugs para detectar bugs no noso código. Así, atopamos varios bugs, os cales foron resolto posteriormente. Os bugs detectados debíanse a algunha das seguintes causas:

- Comparacións de obxectos feita por referencia. A solución foi facer a comparación usando o método `equals`.
- Uso de variables con valor potencialmente nulo en algunha función. A solución foi facer a comprobación de se se a variable ten valor nulo antes do seu uso na función.
- Uso do valor “12” no campo “month” de algúns `Calendar`. A solución foi cambialo polo valor “11”, xa que dito campo pode tomar os valores 0..11.

Ademais tamén solucionamos tódolos warnings no código Java referidos a imports ou variables que non se usaban.

Para as probas de mutación de código usamos a ferramenta PITest. Para executar dita ferramenta usamos o comando `“mvn org.pitest:pitest-maven:mutationCoverage”`, usando tódolos tipos de operadores de mutación. A execución dura varios minutos e finalmente xera un report na carpeta `“target/pit-reports”`.

Unha vez visto o report atopamos que a maioría dos mutantes morrían. Aínda así, estudamos aqueles que vivían e resolvemos algún deles. Os mutantes que sobrevivían o facían ante a posibilidade de devolver un valor nulo en dous tipos de métodos:

- En algunhas entidades estaba sobrescrito o método `toString()`. Sen embargo este método non o estabamos usando implícitamente no noso código, polo que decidimos prescindir del nas entidades `“Categoría”`, `“UserProfile”` e `“Apuesta”`, quedando resolto o problema.
- En algúns getters na entidade `“TipoApuesta”`, na excepción `“IncorrectPasswordException”` e no `Dto “TipoApuestaDto”`. Decidimos que non produce ningún problema que neste caso os mutantes vivan, xa que soamente significa que nalgún caso faise un set da propiedade correspondente e non se recupera posteriormente co `get`.

Desta maneira obtivemos un 93% de Line Coverage e un 98% de Mutation Coverage, quedando 3/144 mutantes vivos e repartidas da seguinte forma:



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
20	93% <div><div></div></div> 339/366	98% <div><div></div></div> 141/144

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
es.udc.pa.p007.apuestasapp.model.apuesta	2	72% <div><div></div></div> 21/29	100% <div><div></div></div> 6/6
es.udc.pa.p007.apuestasapp.model.apuestasservice	4	95% <div><div></div></div> 92/97	100% <div><div></div></div> 58/58
es.udc.pa.p007.apuestasapp.model.categoria	2	85% <div><div></div></div> 11/13	100% <div><div></div></div> 3/3
es.udc.pa.p007.apuestasapp.model.evento	2	88% <div><div></div></div> 43/49	100% <div><div></div></div> 29/29
es.udc.pa.p007.apuestasapp.model.opcionApuesta	2	100% <div><div></div></div> 55/55	100% <div><div></div></div> 15/15
es.udc.pa.p007.apuestasapp.model.tipoApuesta	3	95% <div><div></div></div> 39/41	78% <div><div></div></div> 7/9
es.udc.pa.p007.apuestasapp.model.userprofile	2	94% <div><div></div></div> 33/35	100% <div><div></div></div> 8/8
es.udc.pa.p007.apuestasapp.model.userservice	3	96% <div><div></div></div> 45/47	94% <div><div></div></div> 15/16

Report generated by [PIT 1.1.10](#)

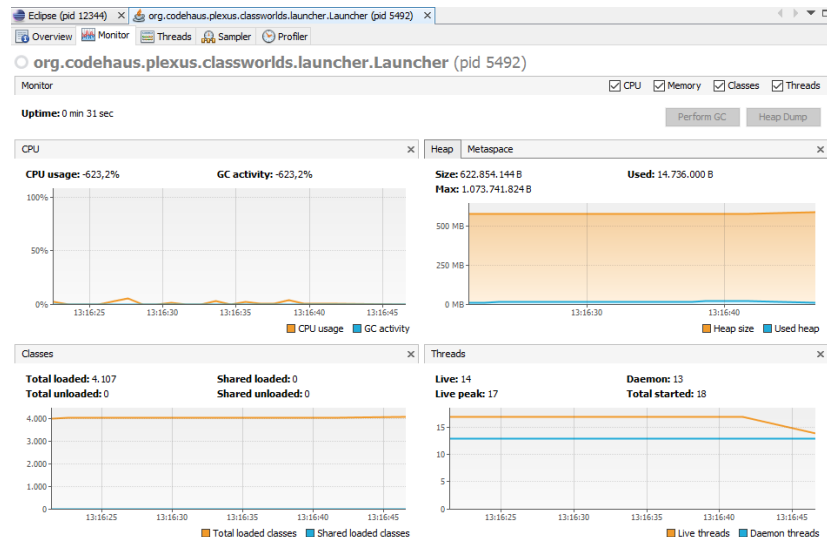
Sen embargo, esta gráfica ás veces cambia coa mesma execución, aparecendo máis ou menos mutantes, de maneira que non podemos controlalo. Así que tomamos unha das execucións para o anáise do uso da ferramenta.

Para as probas de estrés da aplicación usamos a ferramenta VisualVM . Con esta ferramenta fixemos un anáise do uso da CPU e do Heap coa execución dos tests implementados, é dicir, coa execución do comando “mvn test”. Sen embargo, como amosamos na seguinte gráfica, non se produce un gran incremento de uso coa execución dos tests, xa que unha única execución o computador pode realizala sen problema algún.

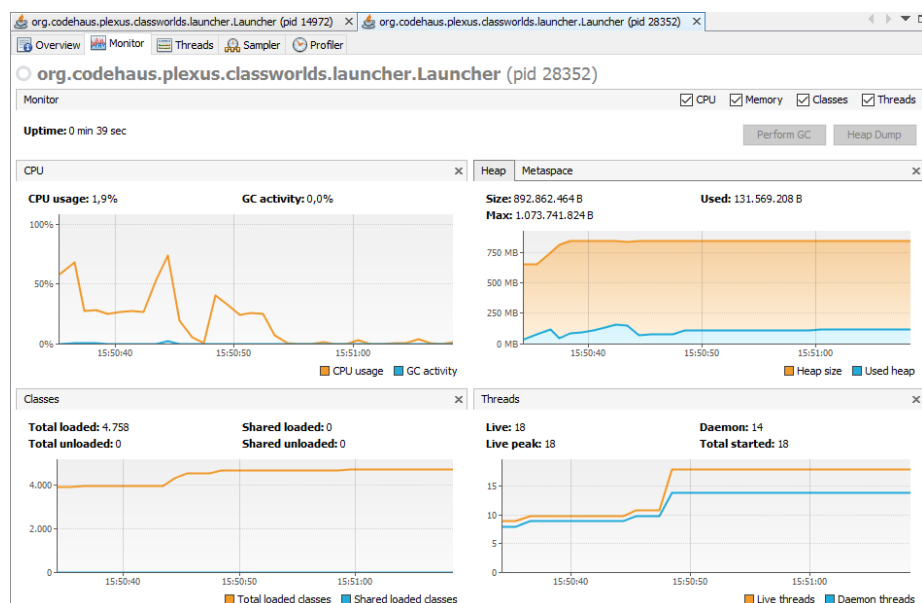


VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS



Ademais, dímonos conta que coa posterior incorporación da ferramenta Graphwalker, ó executar os tests co comando “`mvn graphwalker:generate-sources test`”, o uso da CPU e do Heap se incrementan bastante, sobre todo na primeira fase, é dicir na xeración de fontes para o GraphWalker. Desta maneira, o uso máximo da CPU elévase ao 74,4% e o uso da pila a 170.000.000 B. Podemos observar estes cambios na seguinte gráfica obtida:



Ademais, tamén intentamos realizar os tests de rendemento con varias execucións dos tests cun script do tipo: `for /l %x in (1, 1, 10) do mvn test`

Sen embargo, cada execución se facía nun proceso diferente e non se podía observar co VisualVM nunha mesma gráfica, polo que finalmente non analizamos os resultados das 10 execucións.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

Engadimos unhas poucas probas de integración, replicando aqueles comportamentos que máis se van repetir no uso da aplicación e chamando a distintas funcións dos servizos para replicar dito comportamento. Desta maneira, obtivemos un total de 5 casos de proba (Dende PR-IN-090 ata PR-IN-094).

c. Terceira semana

Na terceira semana de testing usamos ferramentas de selección de datos aleatoria, de probas baseadas en modelos e de cobertura.

Para as probas con selección de datos aleatoria usamos a ferramenta QuickCheck. Utilizamos esta ferramenta para comprobar a correcta creación de cada unha das entidades nos Daos da nosa aplicación, usando como parámetros para a súa construción datos xerados aleatoriamente. Para isto usamos para cada entidade unha clase encargada de xerar unha serie de obxectos totalmente aleatorios desa entidade; estas clases atópanse no mesmo paquete que o test de cada un dos Daos, e rematan co patrón *Generator. Unha vez implementados os xeneradores, engadimos dous novos tests por cada Dao, un deles para probar o correcto funcionamento do xenerador e outro para probar a correcta creación de datos da entidade en cuestión. Desta maneira, obtivemos un total de 12 casos de proba (Dende PR-UN-095 ata PR-UN-106).

Nesta ferramenta tivemos problemas co uso de xeradores e valores aleatorios únicos para o login dos usuarios, o que finalmente resolvemos creando o noso propio xerador.

Para as probas baseadas en modelos intentamos usar a ferramenta GraphWalker. Iniciamos a usar esta ferramenta, pero non puidemos realizar probas con ela, como se explica no apartado “Outros aspectos de interese”.

Esta ferramenta atrasounos bastante debido a non dar atopado o erro a partir do cal non funcionaba no noso proxecto.

Para as probas de cobertura usamos a ferramenta Jacoco. Desta maneira obtemos o seguinte gráfico de cobertura:



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

apuestas-app Project

apuestas-app Project

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
es.udc.pa.p007.apuestasapp.web.services	<div></div>	0%	<div></div>	0%	54	54	170	170	29	29	8	8
es.udc.pa.p007.apuestasapp.web.pages.apuestas	<div></div>	0%	<div></div>	0%	53	53	150	150	37	37	8	8
es.udc.pa.p007.apuestasapp.web.pages.search	<div></div>	0%	<div></div>	0%	78	78	148	148	54	54	5	5
es.udc.pa.p007.apuestasapp.web.pages.user	<div></div>	0%	<div></div>	0%	25	25	74	74	16	16	4	4
es.udc.pa.p007.apuestasapp.web.pages.eventos	<div></div>	0%	<div></div>	0%	20	20	62	62	13	13	3	3
es.udc.pa.p007.apuestasapp.web.util	<div></div>	0%	<div></div>	0%	28	28	54	54	27	27	3	3
es.udc.pa.p007.apuestasapp.model.userservice.util	<div></div>	99%	<div></div>	82%	7	30	9	146	3	16	0	2
es.udc.pa.p007.apuestasapp.web.components	<div></div>	0%	<div></div>	0%	7	7	10	10	4	4	1	1
es.udc.pa.p007.apuestasapp.web.pages.preferences	<div></div>	0%	<div></div>	n/a	4	4	6	6	4	4	1	1
es.udc.pa.p007.apuestasapp.model.apuestasservice	<div></div>	96%	<div></div>	88%	6	50	5	105	1	29	0	8
es.udc.pa.p007.apuestasapp.model.apuesta	<div></div>	77%	<div></div>	n/a	4	14	8	29	4	14	0	2
es.udc.pa.p007.apuestasapp.model.evento	<div></div>	94%	<div></div>	83%	9	30	6	49	3	12	0	2
es.udc.pa.p007.apuestasapp.web.pages	<div></div>	0%	<div></div>	n/a	4	4	4	4	4	4	4	4
es.udc.pa.p007.apuestasapp.model.userservice	<div></div>	95%	<div></div>	88%	2	16	2	47	1	12	0	3
es.udc.pa.p007.apuestasapp.model.tipoApuesta	<div></div>	95%	<div></div>	n/a	2	20	2	41	2	20	0	3
es.udc.pojo.modelutil.exceptions	<div></div>	86%	<div></div>	n/a	2	5	2	10	2	5	0	3
es.udc.pa.p007.apuestasapp.model.userprofile	<div></div>	95%	<div></div>	100%	1	17	2	35	1	16	0	2
es.udc.pa.p007.apuestasapp.model.categoria	<div></div>	88%	<div></div>	n/a	1	8	2	13	1	8	0	2
es.udc.pa.p007.apuestasapp.model.util	<div></div>	0%	<div></div>	n/a	1	1	1	1	1	1	1	1
es.udc.pa.p007.apuestasapp.model.opcionApuesta	<div></div>	100%	<div></div>	100%	0	21	0	55	0	18	0	2
es.udc.pojo.modelutil.dao	<div></div>	100%	<div></div>	100%	0	7	0	15	0	6	0	1
Total	2.781 of 10.087	72%	185 of 292	37%	308	492	717	1.224	207	345	38	68

Así, podemos obter un 72% de cobertura total e un 37% de rama. Aínda que o 72% non está de todo mal, estes resultados débense a que as probas implementadas só corresponden á capa modelo, e non á capa web. Desta maneira os paquetes da capa web teñen un 0% de cobertura, o que baixa as porcentaxes de cobertura totais do proxecto.

En definitiva, os valores de cobertura dos paquetes da capa modelo están en xeral bastante ben. En todo momento superiores ó 77%, excepto o paquete útil, do cal tampouco se fan probas.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE **INFORME DE PRÁCTICAS**

5. REXISTRO DE ERROS

No report xerado pola ferramenta Pitest atopamos algún mutantes que sobrevivían:

- No getter “getOpciones” da entidade “TipoApuesta”: Isto produce que un mutante sobreviva na entidade
- No getter da excepción “IncorrectPasswordException”.
- No getter “getCodEvento” do Dto “TipoApuestaDto”.

Atendimos estes erros e decidimos que como non produce ningún problema que neste caso os mutantes vivan, xa que soamente significa que nos tests nalgún caso faise un set da propiedade correspondente e non se recupera posteriormente co get.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE

INFORME DE PRÁCTICAS

6. ESTATÍSTICAS

- **Número de erros encontrados diariamente e semanalmente.** Na primeira semana atopamos multitude de erros coa ferramenta FindBugs, ó redor duns 20 erros, os cales foron resoltois completamente. Na segunda semana unicamente atopamos 6 erros co uso da ferramenta Pitest, 3 dos cales foron resoltois e os outros 3 están recollidos no rexistro de erros e quedaron rexistrados tamén como unha issue aberta no repositorio.
- **Nivel de progreso na execución das probas.** O 100% das probas deseñadas execútanse correctamente. As únicas probas que non fomos capaces de deseñar foron as probas basadas en modelos con GraphWalker, o cal se explica no último apartado do informe, e do cal tamén deixamos issues abertas no repositorio. A cobertura xeral do proxecto é dun 72%, o que está ben tendo en conta que ademais só probamos a capa modelo e esa cobertura inclúe tanto a parte modelo como a parte web.
- **Análise do perfil de detección de erros (lugares, compoñentes, tipoloxía):** Os bugs atopados coa ferramenta FindBugs correspondíanse a compoñentes variados e se concentraban aos 3 tipos de bugs mencionados na descrición do uso da ferramenta. Os bugs atopados con Pitest tampouco se concentraban en getters de entidades ou excepcións, polo que era o mesmo tipo de compoñente e tiñan a mesma tipoloxía, tamén explicada no apartado correspondente.
- **Informe de erros abertos e pechados por nivel de criticidade.** A issue que queda aberta de Pitest foi considerado de pouca gravidade debido a que consideramos que non afecta ó correcto funcionamento das probas e do proxecto en sí. As issues abertas de GraphWalker considerámoas de gravidade media, debido a que non contamos desta maneira de probas basadas en modelos para testear a nosa aplicación. A maior parte dos test realizados trátase de test de unidade, polo que a maior parte das issues correspóndense a este nivel de proba, aínda que taén fixemos algunhas probas de integración de funcións de servizos.
- **Avaliación global do estado de calidade e estabilidade actuais.** Realizamos un bo proceso de probas a nivel modelo da aplicación, o cal era o obxectivo inicial. En ningún momento intentou testearse a parte web por simplicidade. Foron utilizadas ferramentas de cada un dos apartados propostos, excepto dous: ferramentas de mocking por falta de aplicabilidade no noso proxecto, e ferramentas de probas basadas en modelos(GraphWalker) pola excesiva complexidade á hora de implementalo no noso proxecto. A medida que fomos creando issues, fomos resolvendo aquelas que consideramos mais importantes e cada vez con maior axilidade no manexo das issues, sobre todo aquelas relacionadas co uso de ferramentas, de resolución de bugs e de creación de tests necesarios para probar eficientemente a aplicación. En definitiva, o proxecto estaría listo para ser usado, xa que sobre el se realizou un correcto e exhaustivo proceso de proba, así como unha boa cobertura da funcionalidade do proxecto.



VALIDACIÓN E VERIFICACIÓN DO SOFTWARE INFORME DE PRÁCTICAS

7. OUTROS ASPECTOS DE INTERESE

Non usamos ferramentas de mocking para test, debido a que a maioría de casos de proba poden realizarse sobre unha funcionalidade independentemente das demais. Aqueles casos nos que isto non se permite, tampouco é adecuado o uso de mocks. Por exemplo, nalgúns métodos para crear datos, o dato creado non se devolve na función, polo que para facer a comprobación de se o dato se creou, é necesario chamar a outra función para consultar a súa existencia en base de datos. Neste caso, a solución sería cambiar a implementación da función que crea os datos dun void a unha función que devolvera o dato, pero habería que cambiar a implementación do método.

Como se mencionou no apartado de rexistro de probas, na terceira semana non puidemos utilizar a ferramenta GraphWalker. O que chegamos a facer foi crear un sencillo grafo que está subido ó repositorio na carpeta “es/udc/pa/p007/apuestasapp/model/vvs/testautomation”. A partir deste grafo xérase automaticamente unha interfaz para a realización do test a partir do grafo. Só puidemos executar o test usando prints en cada estado e en local. Tivemos o problema de que en Travis tampouco detectaba a interfaz xerada CreateAndSearchEvents, polo que deixamos comentada a clase CreateAndSearchEventsText. Isto si que funcionaba en local:

```
Results :

Tests run: 106, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 43.921 s
[INFO] Finished at: 2016-12-22T16:31:25+01:00
[INFO] Final Memory: 19M/795M
[INFO] -----
"cmd" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
C:\Users\ADRIAN\workspaceVv\apuestas-app>mvn graphwalker:generate-sources test
```

O problema atopouse á hora de intentar implementar este test, xa que para poder crear os datos en cada estado, precísase inicializar os daos e os servicios e por algunha razón, nesta clase non funciona a inxección de dependencias de Spring a partir da anotación @Autowired. Desta maneira, quedámonos neste punto debido a non poder continuar. Para poder continuar habería que reconfigurar o proxecto para poder inicializar os daos e servizos.