INFORME DE PRÁCTICAS

Repositorio de proxecto: https://github.com/heishiro-slash/VVS-Social
Participantes no proxecto: Uxía Ponte Villaverde, José Miguel del Río

Validación e Verificación de Software

1. Descrición del proyecto

Una pequeña red social con interfaz gráfica escrita en Java. Permite añadir amigos, publicar en el muro, ver las últimas actualizaciones y hacer un broadcast. Para la creación del proyecto se utilizo Netbeans junto con la librería de Swing.

2. Estado actual

Las funcionades llevadas a cabo en este proyecto fueron:

- Usuarios. Cualquier persona podrá registrarse como usuario. Para ello debe proporcionar sus datos reales y los datos que conformarán su perl, así como una fotografía para este. También se permitirá darse de alta usando un certicado digital (o el DNIe), de esa forma se podrá asegurar la identidad de la cuenta y esta aparecerá con un icono de cuenta vericada. Después de suministrar la información personal el usuario puede elegir como quiere que se presente su perl de un conjunto de plantillas seleccionadas (para resaltar mas algunos aspectos sobre otros: fotografías, información personal, etc.). El perl también incluirá la posibilidad de escribir una frase de estado y cambiarla o borrarla en cualquier momento.

- Amigos. Los usuarios podrán hacer solicitudes de amistad a otros usuarios que, por ejemplo, se encuentren a través de un buscador. La solicitudes de amistad se comunicarán, además, vía correo electrónico al interesado. Una vez hecha la solicitud, se deberá esperar la respuesta del destinatario. En caso de ser armativa, los dos usuarios quedan registrados como amigos. Para decidir si hacernos amigos o no podemos consultar el perl del solicitante, aunque solo podremos ver su nombre, fotografía de perl y lista de amigos (esta última podrá presentarse en forma de grafo para comprobar si es amigo de nuestros amigos o que grado de separación tenemos con esta persona). Una vez conrmada la amistad ya podremos ver el resto de datos del perl, excepto aquellos que se marquen explícitamente como privados.

- Muro. Cada usuario poseerá un muro. El muro es una zona donde se van registrando las acciones que realiza un usuario. Algunos ejemplos de estas acciones son:
  - Cambios en la frase de estado del usuario.
  - Cambios en los datos o en la foto de perl
  - Hacer nuevos amigos
  - Hacer comentarios (ver más adelante)
  - También es posible escribir una entrada en el muro propio (ver más adelante).

- Entradas. Cualquier usuario podrá crear una entrada en su muro en la que podrá subir un texto, un enlace a una web, una fotografía, un lugar o un vídeo. Dependiendo del tipo de contenido la entrada se mostrará de una forma u otra (por ejemplo, de una web se mostrará una entradilla y una fotografía). En el futuro se incluirán nuevos tipos de entradas. Cada entrada indicará si es pública, si es visible para los amigos, para un círculo determinado o para una o varias personas en concreto. Cada entrada podrá clasicarse añadiéndole una categoría. Existirán categorías estándar (que serán gestionadas por el administrador de la red) pero el usuario podrá añadir nuevas categorías si así lo considera necesario.

- Comentarios. Cualquier acción llevada a cabo que se registre en el muro puede ser objetivo de comentarios por parte de otros usuarios (que puedan ver la publicación). No se podrán anidar comentarios para mantener el interfaz sencillo pero sí se podrá responder a un usuario concreto introduciendo su nombre al principio (o entre el texto) de dicho comentario.
- Página de inicio. Cada usuario dispondrá de una página de inicio en la que están reejadas todas la acciones relacionadas con sus amigos. De esta forma se está informado de lo que hacen sin necesidad de visitar el muro de cada uno de ellos. Será una página de scroll innito, a medida que bajemos por ella se irán cargando más resultados. La información de la página de inicio podrá ser ltrada por distintos criterios (ver siguiente apartado).
- Filtros. Los ltros nos permitirán seleccionar qué información queremos que se muestre en la página de inicio. Podrán ser ltros de personas, de grupos, o de categorías. Podremos indicar si queremos que se muestren o que se oculten las entradas que provengan de personas, grupos o categorías concretas.

Personas encargadas del desarrollo

- José Miguel del Río
- Victor Blanco
- Faustino Castro
- Esteban Abanqueiro

Personas encargadas de las pruebas

- Uxía Ponte Villaverde
- José Miguel del Río

2.1. Compoñentes avaliados

Se realizarán las pruebas en los componestes GestorMuro y GestorUsuario.

- Gestor Muro.
  - Funcionalidades en las que participa:
    - Muro
    - Cambios del perfil
    - Comentarios
    - Entradas
  - Número pruebas objetivo: 8
  - Número de pruebas preparadas: 8
  - Porcentaje ejecutada: 100 %
  - Porcentaje superada: 100 %

- Gestor Usuario.
  - Funcionalidades en las que participa:
    - Usuarios
    - Amigos
    - Solicitudes
  - Número pruebas objetivo: 25
  - Número de pruebas preparadas: 25
  - Porcentaje ejecutada: 100 %
  - Porcentaje superada: 100 %

3. Especificación de probas

- PR-UN-001
  - Descripción escenarios: Método GetMuro en GestorMuro
  - Entradas utilizadas: Usuario no inicializado
  - Salidas deseadas: Excepción controlada
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-002
  - Descripción escenarios: Método PublicarEntrada en GestorMuro
  - Entradas utilizadas: String Vacío
  - Salidas deseadas: Excepción controlada
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-003
  - Descripción escenarios: Método PublicarEntrada (5Args) en GestorMuro
  - Entradas utilizadas: Strings Vacíos (Probar 1 a 1)
  - Salidas deseadas: Excepción controlada
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-004
  - Descripción escenarios: Método GetUsuarios en GestorUsuarios
  - Entradas utilizadas:
  - Salidas deseadas: Lista usuarios
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-005

    - Descripción escenarios: Método GetUsuarios en GestorUsuarios
    - Entradas utilizadas:
    - Salidas deseadas: Lista vacía
    - Herramienta: JUnit
    - Criterio éxito/suspensión/abandono:
- PR-UN-006
    - Descripción escenarios: Método AltaUsuario en GestorUsuarios
    - Entradas utilizadas: Strings Vacíos (Probar 1 a 1)
    - Salidas deseadas: Excepción controlada
    - Herramienta: JUnit
    - Criterio éxito/suspensión/abandono:
- PR-UN-007
    - Descripción escenarios: Método GetUsuario en GestorUsuarios
    - Entradas utilizadas: String Vacío
    - Salidas deseadas: Excepción controlada
    - Herramienta: JUnit
    - Criterio éxito/suspensión/abandono:
- PR-UN-008
    - Descripción escenarios: Método ExisteUsuario en GestorUsuarios
    - Entradas utilizadas: Nombre de usuario existente
    - Salidas deseadas: True
    - Herramienta: JUnit
    - Criterio éxito/suspensión/abandono:
- PR-UN-009
    - Descripción escenarios: Método ExisteUsuario en GestorUsuarios
    - Entradas utilizadas: Nombre de usuario no existente
    - Salidas deseadas: False
    - Herramienta: JUnit
    - Criterio éxito/suspensión/abandono:
- PR-UN-010
    - Descripción escenarios: Método BuscarUsuario en GestorUsuarios
    - Entradas utilizadas: Strings Vacíos (Probar 1 a 1)
    - Salidas deseadas: Excepción controlada
    - Herramienta: JUnit
    - Criterio éxito/suspensión/abandono:
- PR-UN-011

- Descripción escenarios: Método BuscarUsuario en GestorUsuarios
- Entradas utilizadas: Nombre de usuario existente
- Salidas deseadas: Lista usuarios
- Herramienta: JUnit
- Criterio éxito/suspensión/abandono:

- PR-UN-012
  - Descripción escenarios: Método BuscarUsuario en GestorUsuarios
  - Entradas utilizadas:Nombre de usuario no existente
  - Salidas deseadas: Lista vacía
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-013
  - Descripción escenarios: Método AceptarSolicitud en GestorUsuarios
  - Entradas utilizadas: Solicitud no inicializada
  - Salidas deseadas: Excepción controlada
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-014
  - Descripción escenarios: Método RechazarSolicitud en GestorUsuarios
  - Entradas utilizadas: Solicitud no inicializada
  - Salidas deseadas: Excepción controlada
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-015
  - Descripción escenarios: Método VerAmigos en GestorUsuarios
  - Entradas utilizadas: Usuario no inicializado
  - Salidas deseadas: Excepción controlada
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-016
  - Descripción escenarios: Método VerAmigos en GestorUsuario
  - Entradas utilizadas: Usuario existente
  - Salidas deseadas: Lista amigos
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-017

- Descripción escenarios: Método VerAmigos en GestorUsuario
- Entradas utilizadas: Usuario "Eduardo"
- Salidas deseadas: Lista vacía
- Herramienta: JUnit
- Criterio éxito/suspensión/abandono:

- PR-UN-018
  - Descripción escenarios: Método AñadirAmigos en GestorUsuario
  - Entradas utilizadas: Usuario no inicializado
  - Salidas deseadas: Excepción controlada
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- PR-UN-019
  - Descripción escenarios: Método AñadirAmigos en GestorUsuario
  - Entradas utilizadas: String vacío
  - Salidas deseadas: Excepción controlada
  - Herramienta: JUnit
  - Criterio éxito/suspensión/abandono:
- Probas de análisis de caja blanca
  - Descripción escenarios: Se realizan en todo los componentes del software.
  - Entradas utilizadas:
  - Salidas deseadas:
  - Herramienta: FindBugs
  - Criterio éxito/suspensión/abandono:

4. Rexisto de probas

A continuación detallaremos las herramientas utilizadas para el proceso de pruebas especificando los problemas encontrados y su solución.

4.1. FINDBUGS

Se utilizo está herramienta para la realización de pruebas de caja blanca. Debido a que nuestro programa está desarrollado en netbeans en un principio tuvimos problemas para integrar la herramienta con el programa. Finalmente se encontró un plugin de Findbugs para el IDE netbeans, que fue el utilizado.

A partir del uso de esta herramienta se encontraron en el código 42 bugs, se resolvieron un total de 3. Debido a que otras pruebas nos parecían más relevantes y que los bugs encontrados con esta herramienta no son muy perjudiciales decidimos centrarnos en otros y el resto de bugs serán corregidos en un futuro.

4.2. VISUALVM

Se utilizó herramienta de VisualVM integrada en netbeans (La herramienta cuenta con plugins para los principales IDEs).

Los pasos seguidos para la realización de esta prueba fueron la de configurar una sesión de pruebas, seleccionando telemetría, y dejando el programa ejecutado por más de 1 hora.

A su vez también se probó a ejecutar el programa y testear todos los métodos múltiples veces hasta para comprobar la correcta gestión del Heap.

Ambas situaciones terminaro de manera favorable, sin dar signos de alguna fuga de memoria clara.

4.3. JCHECK

Para los test de entradas aleatorias se hizo uso de JCHECK. Para poder usarlo, después de añadir el jar a la carpeta lib, hubo que añadirlo manualmente a la librería de tests del proyecto de netbeans.

Despues de eso hubo que añadir '@RunWith(org.jcheck.runners.JCheckRunner.class)' antes de la definición de la clase. Para controlar el tamaño de los aleatorios también se incluyó '@Configuration(tests = 10, size = 10)'.

Su uso en los test fue la de proveer de valores aleatorios a los test que necesitasen alguna entrada de texto o numérica.

4.4. GRAPHWALKER

Debido a la fuerte dependencia del GraphWalker con Maven y a que nuestro proyecto estaba realizado con ANT, para poder aprovecharnos del uso de esta herramienta tuvimos que ejecutar el cliente de forma independiente al IDE. Con él generamos un modelo del prueba con el cual debería hacerse, de manera manual, un test en el netbeans.

Debido a la falta de tiempo, este test no fue realizado.

4.5. JACOCO

Para la cobertura del sistema se recurrió al JACOCO, el cual dispone de un plugin para netbeans (TikiOne JACOCO), gracias al cual su uso es muy sencillo.

Este programa permite 2 tipos de cobertura, cobertura durante los test y cobertura durante la ejecución.

Una vez terminado genera un html el cual te permite ver función por función qué porcentaje de línea y de rama tienes cubierto, así como los detalles de la ejecución.

5. Rexistro de erros

Durante la realización de las pruebas de unidad se encontraron varios bugs que están especificados en las issues del repositorio Github con el nombre de la prueba en la que se encontraron.

Además, durante la ejecución del JCheck se llegó a obtener un desbordamiento de HEAP, causado seguramente por la ejecución sin control de los test con esta herramienta. Se solucionó limitando el número de llamadas a la misma, así como el tamaño del generador.

En los test de Cobertura se descubrió que existía una baja cobertura del sistema en los test, tanto en la parte de la interfaz, así como en la parte de gestión. Se abrieron los respectivos issues en Github y se les dió solución. Para la parte de la interfaz se recurrió a

la ejecución del programa con la herramienta corriendo hasta que se obtuvo una cobertura superior al 80 %, pero al ser una ejecución en vivo, este tipo de test no sería considerado válido en muchas situaciones, ya que dificilmente sería repetible.

6. Estatísticas

- Número de errores encontrados: 40 Errores
  - Diariamente: 2 errores diarios.
  - Semanalmente: 13 errores semanales.
- Nivel de progreso en la de las pruebas: Actualmente se han solventado 28 de los 40 erroes.
- Perfil de los errores encontrados: La mayor parte de los errores encontrados en el programa fueron detectados por el findbugs, y errores de funciones nunca usadas, o de errores de estilo.
- Errores cerrado: Debido a que sólo se encontró 1 error crítico (Falta de implementación de una función), este fue solucionado y cerrado rápidamente, para así poder centrarnos en los errores más pequeños del código.
- Evaluación global: Viendo que en número de errores encontrado fue bastante alto, y que los más críticos ya han sido solucionados, dejando principalmente errores de estilo, considero que la calidad del código inicial era baja y esta es actualmente media.

7. Outros aspectos de interese

7.1. Reorganización de las carpetas del proyecto

Para que el proyecto tuviera una estructura estándar se reorganizaron las carpetas, eliminando aquellas que genera netbeans de forma automática y reubicando el build.xml en la carpeta root.

7.2. Inclusión de soporte para travis

Para este apartado, tras solicitar ayuda incluyendo el tag correspondiente en el issue de Github, conseguimos solucionar el problema.

7.3. Ejecución de la aplicación

Para ejecutar la aplicación hay que crear un nuevo proyecto desde origenes existentes, seleccionar el proyecto reorganizado y correrlo.

7.4. Inclusion del librerías externas

Para la correcta ejecución de los test, antes de ejecutarlos hay que hacer click en la carpeta de librerías de test e incluir las librerías que están incluidas en la carpeta lib.