

Universidad ORT Uruguay

Facultad de Ingeniería

Obligatorio II

Ingeniería de Software 1

Entregado como requisito de la materia Ingeniería de
Software 1

Declaraciones de autoría

Nosotros, _____, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el Obligatorio I de Ingeniería de Software 1;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Resumen

Este documento especifica el desarrollo de prototipo de aplicación para la gestión de un calendario compartido de cuidado de mascotas caninas aplicando técnicas de ingeniería de software. Dicha aplicación se ocupa de la gestión de la información de las mascotas y los usuarios. Además de tener un calendario compartido donde aparecen las actividades que tengan atribuidas las distintas mascotas. Estas pueden ser paseos, alimentación o cualquier otro tipo de actividad, como puede ser agendar una hora con una veterinaria, o jugar. Por último, la aplicación se encarga de notificar al usuario cuando debe hacer una actividad a través de la aplicación y vía mail.

Índice general

1. Versionado	4
1.1. Repositorio utilizado	4
1.2. Criterios de versionado	4
1.3. Resumen del log de versiones	5
2. Codificación	7
2.1. Estándar de codificación	7
2.2. Pruebas unitarias	10
2.3. Análisis de código	11
3. Interfaz de usuario y usabilidad	13
3.1. Criterios de interfaz de usuario	13
3.2. Evaluación de usabilidad	14
3.2.1. Visibilidad del estado del sistema	14
3.2.2. Relación entre el sistema y el mundo real	14
3.2.3. Libertad y control por parte del usuario	14
3.2.4. Consistencia y estándares	14
3.2.5. Prevención de errores	15
3.2.6. Reconocer antes que recordar	15
3.2.7. Flexibilidad y eficiencia en el uso	15
3.2.8. Diseño estético y minimalista	15
3.2.9. Ayuda a los usuarios a reconocer, diagnosticar y recuperarse de los errores	15
3.2.10. Ayuda y documentación	16
4. Pruebas funcionales	17
4.1. Técnicas de prueba aplicadas	17
4.2. Casos de prueba	18
4.2.1. Caso de Uso 1	18
4.2.2. Caso de Uso 2	19
4.2.3. Caso de Uso 3	21
4.2.4. Caso de Uso 4	22
4.2.5. Caso de Uso 5	23
4.2.6. Caso de Uso 6	24
4.3. Sesiones de ejecución de pruebas	25
4.3.1. Sesión de prueba	25

4.3.2.	Sesión de prueba	25
4.3.3.	Sesión de prueba	26
4.3.4.	Sesión de prueba	26
4.3.5.	Sesión de prueba	26
4.3.6.	Sesión de prueba	27
4.3.7.	Sesión de prueba	27
4.3.8.	Sesión de prueba	27
4.3.9.	Sesión de prueba	28
4.3.10.	Sesión de prueba	28
4.3.11.	Sesión de prueba	28
4.3.12.	Sesión de prueba	29
4.3.13.	Sesión de prueba	29
4.3.14.	Sesión de prueba	29
4.3.15.	Sesión de prueba	30
4.3.16.	Sesión de prueba	30
4.3.17.	Sesión de prueba	30
4.3.18.	Sesión de Prueba Exploratoria	31
4.3.19.	Sesión de Prueba Exploratoria	31
4.3.20.	Sesión de Prueba Exploratoria	31
5.	Reporte de defectos	32
5.1.	Definición de categorías de defectos	32
5.2.	Defectos encontrados por iteración	32
5.2.1.	Reporte de defecto	32
5.2.2.	Reporte de defecto	33
5.2.3.	Reporte de defecto	33
5.2.4.	Reporte de defecto	33
5.2.5.	Reporte de defecto	34
5.2.6.	Reporte de defecto	34
5.2.7.	Reporte de defecto	34
5.2.8.	Reporte de defecto	35
5.2.9.	Reporte de defecto	35
5.2.10.	Reporte de defecto	35
5.2.11.	Reporte de defecto	36
5.2.12.	Reporte de defecto	36
5.2.13.	Reporte de defecto	36
5.2.14.	Reporte de defecto	37
5.2.15.	Reporte de defecto	37
5.2.16.	Reporte de defecto	37
5.2.17.	Reporte de defecto	38
5.2.18.	Reporte de defecto	38
5.2.19.	Reporte de defecto	38
5.2.20.	Reporte de defecto	39
5.2.21.	Reporte de defecto	39
5.2.22.	Reporte de defecto	39
5.2.23.	Reporte de defecto	40

5.2.24. Reporte de defecto	40
5.3. Estado de calidad global	41
6. Reflexión	42
6.1. Oportunidades de mejora	42
6.2. Conclusiones	42
7. Bibliografía	44

1. Versionado

1.1. Repositorio utilizado

El sistema de versionados fue realizado y administrado a través de la herramienta Git, utilizando un repositorio online en Bitbucket. El link al repositorio utilizado es

Los elementos de la configuración de software incluidos en el repositorio son el programa y su documentación. El programa tiene su código fuente con sus clases recursos y librerías.

1.2. Criterios de versionado

Para versionar los distintos elementos de la configuración de software utilizamos dos ramas. La rama Develop y la rama Master. La diferencia en estas ramas es el nivel de estabilidad del proyecto. En la rama Develop es donde se trabaja, se agregan nuevas features, se arreglan errores, se commitea seguido, el software sigue siendo inestable. Mientras que en la rama Master el software es estable, se commitea mucho menos seguido y es usado para hacer las pruebas funcionales. Cuando mergeamos a la rama Master y commiteabamos, el mensaje de los commit tiene la forma "Version 1.0", "Version 1.1". Para así indicar sobre qué versión del proyecto se hacen las pruebas funcionales. Posteriormente se empezó a utilizar los tags, para marcar con mayor facilidad las distintas versiones ya estables.

Para trabajar más ordenadamente nos dividimos el trabajo en diferentes partes del proyecto. Ambos integrantes trabajamos en todas las diferentes áreas del proyecto, pero no al mismo tiempo. Por ejemplo, mientras uno implementaba y diseñaba la interfaz, otro implementaba el dominio y las pruebas unitarias. Al trabajar con un repositorio distribuido, esto nos permitió tener menor cantidad de conflictos, y así trabajar más rápido, y no invertir tiempo en resolver estos conflictos.


En nuestro equipo commiteabamos a la rama Develop cada vez que implementábamos una feature o funcionalidad nueva, o un arreglo de algún

error. Siempre el software tiene que compilar. Al hacer los commits utilizamos mensajes que describan a grandes rasgos que fue lo que se implemento, se agrego o modifiko en este nuevo commit. Nuestros mensajes en los commit siguen una linea en común, se explica en tercera persona que cambios se realizaron, o clases se agregaron, de forma objetiva y concreta.

Sin embargo, con el proyecto ya casi terminado, encontramos en distintas fuentes, buenas practicas para los mensajes de los commits. Una de ellas es tener tres partes en el mensaje: Tipo , Cuerpo, Pie. En el tipo se titula que se hizo en este commit, por ejemplo bug fix. El cuerpo se especifica que se hizo y donde. Y en el pie se ponen referencias a distintas cosas del proyecto a las cuales están relacionadas el commit. Ver 6.2

1.3. Resumen del log de versiones

A continuación se adjuntará una imagen de los primeros commits subidos al repositorio. Se fueron realizando cambios y testeandolos en la rama develop, y luego al llegar a versiones funcionales se mergeó a la rama master.



ec701de	Se cambio el nombre del proyecto a Canu-Kugelmass	4 days ago
5be8556	Merge los cambios, y cambie las pruebas de PerroTest	2018-11-16
7e3a3e3 M	Merge branch 'develop' of https://bitbucket.org/MarcelCanu/canu-kugelmass into d...	2018-11-16
d75afbe	Se creo las clases JUnit ActividadCualquieraTest.java, PerroTest.java, PersonaTest.java.	2018-11-16
de89b27	Se actualizo la pestaña de perro	2018-11-16
5a3bac5	Se agrego la clase JUnit testPersona. No esta terminada. Se cambio distintos metodo...	2018-11-15
ce9b20f	Se creo el FechaTest, se incluyo el plugin JaCoCoverage. La clase fecha tiene 100% de...	2018-11-15
e8170e3	Se cambio la ventana de perros, ahora solo tiene un scroll para nombres y luego apa...	2018-11-14
50b27dc	Se agrego el jar Calendario, se puso el calendario en la pestaña calendario. Se hizo el ...	2018-11-07
4de7193	Se creó la ventana principal con sus pestañas calendario, perros, y personas, cada un...	2018-11-06
edfbca9	Se actualizo el nombre de los paquetes dominio e interfaz para que no comiencen co...	2018-11-06
e6ea3ff M	Agregue package Interfaz y clase VentanaPrincipal	2018-11-05
a7e7c18	Se agrego el paquete interfaz y la clase ventana principal	2018-11-05
1603053	Se hicieron pruebas sobre las personas, perros y actividades	2018-11-05
5881645	Se cambió el manejo de hora para que lo hagan las actividades	2018-11-05
409cc29	Se cambio el manejo de día hora año y horas, por una clase fecha, que manejará tod...	2018-11-05
30b4991	Se cambio la abstraccion, ahora esta la clase abstracta Actividad, con sus atributos, q...	2018-11-05
707c651	"Le puse atributos a la clase paseo y a la clase alimentacion"	2018-11-01
2c9dbf4	"Cree la clase paseo y la clase alimentacion"	2018-11-01
fc7208c	Persona y Mascota ya tienen atributos	2018-11-01
05ee3a5	Cambie la clase mascota	2018-11-01
3c186cd	Este es el pry	2018-11-01
bF3da00	Initial commit	2018-11-01

La mayoría de los commits al versionado del proyecto son hacia la rama develop, ya que la rama master se utilizó para hacer commits de versiones estables, para que se les haga testing.

El resto del log de versiones se encuentra en bitbucket.

2. Codificación

2.1. Estándar de codificación

El proyecto en su totalidad fue codificado bajo el estándar de codificación de java, usando tanto las buenas prácticas aprendidas en clase como las ya adquiridas en Programación I y II, y buscando en internet. Dicho estándar fue utilizado para poder crear software de calidad y facilitar tanto su mantenimiento como entendimiento tiempo después de haber sido creado. También es de gran utilidad al no haber un único codificador y así poder inferir como se puede llamar una variable o qué hace un método, sin haber sido el creador de los mismos. Estos estándares fueron discutidos entre los integrantes previo al comienzo de la codificación para ser consistentes a lo largo del proyecto. No obstante, algunos de ellos fueron elegidos durante la codificación por diferentes situaciones que fueron surgiendo. Los estándares que fueron elegidos son descriptos a continuación.

En cuanto a la nomenclatura, todos los paquetes fueron escritos en minúscula y sin caracteres especiales. Además de que divididas las clases en el paquete dominio y el paquete interfaz según correspondía. Por otro lado, todas las clases fueron nombradas en CamelCase, siendo la primer letra del nombre de cada clase puesta en mayúscula, y en caso del nombre estar compuesto por más de una palabra, cada primera letra de cada palabra interna también fue puesta en mayúscula, siendo el resto de la palabra escrito en minúscula, por ejemplo `ActividadCualquiera`. Las clases también fueron nombradas de manera simple y descriptiva, evitando el uso de acrónimos y abreviaturas.

Adicionalmente, los métodos fueron nombrados también de manera descriptiva, utilizando verbos que expliquen cual es su función y, a diferencia de las clases, se escriben en camelCase, lo cual quiere decir se escribe de la misma manera que las clases pero la primer letra del método es minúscula, como por ejemplo `inicializarPestanaUsuarios`. Sobre los atributos de las clases y los métodos de los objetos, son privados siempre que se pueda, evitando el hacerlos públicos de manera innecesaria. En lo que respecta a los métodos, es más importante la claridad de la descripción de lo que hace que su largo. Además los nombres fueron elegidos sin exponer detalles sobre su implementación si no que únicamente mencionan qué

hacen. Por su parte, las variables son escritas de manera similar que los métodos, con la diferencia que para ellas sí es importante que no sean excesivamente largas, siempre y cuando sean mnemotécnicas. Se evita el utilizar caracteres especiales tanto en las clases como en los métodos.

En cuanto al estilo de codificación, se eligió como indentación un tabulador. Además las llaves se utilizaron al estilo Kernighan y Ritchie/Kernel. Sobre los comentarios, se decidió usarlos únicamente cuando la complejidad lo amerite para documentar el código, pero también tratar de evitar dicha complejidad para no tener que hacer uso de ellos. En la versión final de la aplicación no fue necesario el uso de comentarios, sin embargo en pasos intermedios y métodos que no funcionaban de forma correcta sí fueron hechos. No se dejó comentado ningún método o variable que no fuera utilizado para un posible uso futuro, todos los comentarios que no son utilizados fueron borrados.

Por otra parte, en el manejo de las imágenes por ejemplo, en cuanto a sus direcciones fueron evitadas las direcciones absolutas, si no que fueron utilizadas direcciones relativas, para que sin importar donde se guarde la aplicación, sus imágenes podrán visualizarse. Las repeticiones fueron en su mayoría evitadas, encapsulando en métodos líneas de código que podrían ser repetidas, por ejemplo setear la lista de perros, que se hace cuando se agrega un perro, o cuando se ingresan los datos precargados. Esto, por otro lado, quita complejidad al código y evita el uso de comentarios. Además en todos los métodos fue utilizado un único punto de retorno y se trato de evitar la asignación de null, pero hubo casos que fue necesario hacerlo.

Sobre la calidad de código, una herramienta que es proporcionada tanto por el IDE Netbeans, el cual fue utilizado, como por otros son los hints, los cuales fueron tenidos en cuenta y decisiones sobre qué hacer fueron tomadas hasta que no quedasen advertencias. Ninguna advertencia fue ignorada para que de esta manera se pudiese prestar la atención necesaria a las advertencias de mayor importancia, ya que no todos son igual de relevantes.



Figura 2.1: El proyecto tiene un nombre identificador de los autores, todas las clases están dentro de paquetes, los paquetes comienzan con minúscula y las clases con mayúscula y CamelCase

```
public class Perro {  
  
    private String nombre;  
    private double altura;  
    private double peso;  
    private String comentarios;  
    private ImageIcon foto;  
}
```

Figura 2.2: Los atributos son privados y descriptivos de qué es lo que son

```

public ArrayList<Actividad> listaActividadesPorFecha(int dia, int mes, int ano) {
    ArrayList<Actividad> retLista = new ArrayList<>();
    if (dia >= 1 && dia <= 31 && mes >= 0 && mes <= 12 && ano >= 1) {
        for (int i = 0; i < listaActividades.size(); i++) {
            Actividad act = listaActividades.get(i);
            if (act.getFecha().getDia() == dia && act.getFecha().getMes() == mes && act.getFecha().getAño() == ano) {
                retLista.add(act);
            }
        }
    }
    return retLista;
}

```

Figura 2.3: Los parámetros que recibe son identificativos de lo que son, las variables definidas también, está bien indentado, el nombre escrito con camelCase y tiene un único punto de retorno.

```

CalBtnGroupRepetir : ButtonGroup
CalBtnGroupVeterinaria : ButtonGroup
CalBtnRealizadaNo : JRadioButton
CalBtnRealizadaSi : JRadioButton
CalBtnRuta : JButton
CalBtnVeterinariaNo : JRadioButton
CalBtnVeterinariaSi : JRadioButton
CalComboHora : JComboBox<String>
CalComboMotivo : JComboBox<String>
CalComboPerro : JComboBox<String>
CalComboResponsable : JComboBox<String>
CalComboTipo : JComboBox<String>

```

Figura 2.4: Los objetos de la ventana son identificativos de en que pestaña se encuentran, qué tipo de objeto son y qué hacen.

A continuación se mostrarán algunos ejemplos de uso de los estándares de codificación.

2.2. Pruebas unitarias

En un principio se empezó a implementar las primeras clases del dominio y luego hacer una clase JUnit para hacer la prueba unitaria y hacer correr pruebas sobre ella. Haciendo esto debíamos cambiar los métodos dentro de la clase probada para que cumpla con todas las pruebas hechas.

Luego de haber repetido este proceso se empezó a codificar las pruebas al mismo tiempo o hasta antes que la propia implementación que los métodos de clase. Así sabiendo que se espera del método se codificaba con menos errores, y menos corrección después. Así se logro seguir un Test-Driven Development, en el cual las pruebas sirven también para diseñar el código y no solamente probar.

Por cada método en la clase a probar, hay mas de una prueba en la clase JUnit. Estas pruebas se enfocan en probar casos normales, como casos limites. Se hicieron simples y que no dependieran de las otras pruebas.

JaCoCoverage analysis of project "ProyNetbeans" (powered by JaCoCo from EcEmma) > dominio

dominio







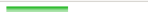















Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Main		0%		n/a	2	2	4	4	2	2	1	1
Sistema		100%		100%	0	61	0	96	0	34	0	1
Fecha		100%		100%	0	31	0	56	0	12	0	1
Veterinaria		100%		100%	0	26	0	51	0	13	0	1
VisitaVeterinaria		100%		100%	0	22	0	51	0	19	0	1
Paseo		100%		100%	0	21	0	49	0	19	0	1
Alimentacion		100%		100%	0	19	0	44	0	17	0	1
ActividadCualquiera		100%		100%	0	16	0	37	0	15	0	1
Perro		100%		100%	0	17	0	38	0	13	0	1
Usuario		100%		100%	0	11	0	24	0	9	0	1
Actividad		100%		n/a	0	1	0	1	0	1	0	1
Total	14 of 1.718	99%	0 of 146	100%	2	227	4	451	2	154	1	11

Figura 2.5: Análisis de cobertura con JaCoCovergae

Se presenta el análisis de cobertura de pruebas, utilizando la herramienta JaCoCoverage, el cual se llevo al 100 % en el dominio (Excluyendo a la clase main). Siempre sabiendo que llegar a un 100 % en las pruebas no significa que estas sean suficientes, nunca son.

2.3. Análisis de código

Para el análisis del código se utilizó la herramienta Java Hints. Esta herramienta ya se encuentra incorporada en Netbeans y permite visualizar si se respetan los estándares de codificación definidos en Oracle.

Estos son los estándares convencionales para la generación de código de calidad. Gracias a esta herramienta, encontramos algunas partes del código que se podrían mejorar, las cuales fueron cambiadas y serán descritas a continuación. Cabe destacar que no fueron muchos los hints recibidos por lo que se podría decir que fueron seguidos de manera relativamente certera los estándares de codificación.

Importaciones sin usar, por más de que evitamos usar importaciones del tipo, `import java.util.*`, lo cual importa toda la librería `java.util`, ocurrió que al comenzar haciendo un método de una forma y luego cambiarlo, algunos elementos que fueron importados, posteriormente no fueron usados. Esto significa que su importación no tiene sentido por lo que se eliminaron las importaciones sin usar.

Otra hint recibida, fue de una o dos variables que fueron definidas pero que no fueron modificadas, por lo que sería favorable hacerlas final. Esto fue arreglado porque genera un software de mayor calidad.

Además, en muchos métodos fue recibida la hint de que falta el javaDoc, lo cual no creemos necesario para este contexto de desarrollo de la aplicación por lo que ese tipo de hints fue descartado.

Debido a la constante modificación del código, al final de más de una línea, habían dos puntos y coma (;), genera una línea de código vacía e innecesaria, por lo que uno de ellos fue eliminado.

También, dentro de algunos métodos, cuando en un JTextField se hacía el método getText lo cual devuelve un String con el contenido de dicho campo, luego se hacía toString, lo cual es redundante debido a que lo que devuelve el método getText. El método toString fue removido de esa línea.

Por otro lado, en la generación de imágenes, la cual se rodea de try catch, la excepción que capturamos es Exception, lo cual no es específica, pero aún así se prefirió dejarlo ya que, de cambiarlo a las excepciones que se creen podrían ocurrir, en caso de ocurrir otra se caería el programa.

Para el método que genera una notificación después de cierto tiempo, se crea una clase interna anónima ActionListener, y surgió como hint convertirla a una expresión lambda. Luego de informarnos en la página <https://www.programcreek.com/2014/01/why-lambda-java-8/>, decidimos cambiarla a la expresión lambda correspondiente ya que por más de que previo a Java 8 esto no se podía hacer, trata mejor la funcionalidad de lo que se quería hacer en ese método.

En algunos métodos que sobrescribían otros, faltó agregarle Override antes de que comience el método. Por más de que en esta versión de Java se puede no hacer, se decidió ponerlo para que funcione en las próximas actualizaciones.

Consideramos que la calidad de código mejoró gracias al uso de la herramienta Java Hints.

3. Interfaz de usuario y usabilidad

3.1. Criterios de interfaz de usuario

Para la construcción de la interfaz de usuario varios criterios fueron utilizados basados en los datos en el curso.

En cuanto a la interacción del usuario con la aplicación, para que sea mas sencillo, decidimos utilizar únicamente tres pestañas: Usuario, Perro y Calendario. En estas se maneja toda la información de la aplicación y se pueden ver todos los datos. De esta manera, recordar dónde se encuentran las cosas y aprender a usar la aplicación es más sencillo y bastante intuitivo. Se puede acceder a la información requerida de manera rápida y sencilla.

Además, para ver y editar datos, tanto de los perros como de los usuarios, se hace de la misma manera que al agregarlos por primera vez. En el lugar donde se ingresó el nombre de la mascota para ingresarla al sistema, también es donde se ve el nombre del perro al seleccionarlo y donde se edita en caso de querer hacerlo.

Sobre la tolerancia de errores, se hizo que dentro de la aplicación el usuario pueda explorar de manera segura y las oportunidades de equivocarse y hacer mal las cosas son escasas. Por el fácil movimiento entre pestañas, es fácil darse cuenta si se esta en el lugar incorrecto. Además, en caso de ingresar datos incorrectos como una palabra en el lugar donde va la altura del perro, el sistema no lo permitirá. Por otro lado, en caso de que falte algun dato para hacer el ingreso, a través de mensajes en la pantalla se informará al usuario.

En caso de un botón no poder usarse, por ejemplo editar la ruta de una alimentación este botón no estará habilitado y el usuario no lo podrá ver para no seleccionarlo por error. Adicionalmente, si por ejemplo se quiere pedir una hora a una veterinara la cual no está disponible, también se avisa por pantalla.

Por último, en términos de consistencia, la terminología usada en la aplicación lo es, y de esta manera el usuario puede identificar correctamente

todos los elementos de ella. Existe un único lugar para mostrar sugerencias de la aplicación en todas las pestañas.

3.2. Evaluación de usabilidad

En esta sección evaluaremos el diseño de la interfaz de usuario siguiendo las 10 Reglas Heurísticas de Jakob Nielsen. (Ver Bibliografía - [6])

3.2.1. Visibilidad del estado del sistema

”La aplicación deberá siempre mantener informado al usuario de lo que esta ocurriendo y brindarle una respuesta en el menor tiempo posible.” (Ver Bibliografía - [7])

En el sistema no hay ninguna pantalla de carga, entonces en ningún momento al usuario se le informa que el sistema sigue en funcionamiento, por ejemplo al momento de iniciar la aplicación. Además ni cuando se agregan los datos precargados ni cuando se agrega un recordatorio, lo que enlentece un poco el sistema, se no se muestra ningún mensaje o símbolo de carga indicando que eso puede demorar. Por lo tanto no cumple con esta regla.

3.2.2. Relación entre el sistema y el mundo real

”La aplicación debe utilizar el lenguaje del usuario, con expresiones y palabras que le resulten familiares.” (Ver Bibliografía - [7])

La aplicación en todo momento usa lenguaje familiar para los usuarios como Usuario, Calendario, Paseo, Alimentación, Visita Médica, Perro. En ningún momento usa lenguaje técnico. Por lo tanto cumple con esta regla.

3.2.3. Libertad y control por parte del usuario

.”En caso de elegir alguna opción de la aplicación por error, el usuario debe disponer de una “salida de emergencia” para abandonar el estado no deseado en que se halla. Debe poder deshacer o repetir una acción realizada.” (Ver Bibliografía - [7])

La aplicación cumple con esta regla, debido a que la navegabilidad entre las diferentes partes del sistema es a través de pestañas, por lo que en caso de querer volver hacia la pestaña anterior, en todo momento tendrá opción.

3.2.4. Consistencia y estándares

”Los usuarios no tienen por qué saber que diferentes palabras, situaciones o acciones significan lo mismo. Es conveniente seguir convenciones.” (Ver Bibliografía - [7])

La aplicación cumple con esta regla. El uso de las palabras en la aplicación

es consistente. Por ejemplo, para referir a cualquier acción con respecto al perro, siempre se lo llama perro, y no perro, mascota, can. Lo mismo con el nombre de usuario.

3.2.5. Prevención de errores

Es importante ayudarle al usuario a que no caiga en un error.”(Ver Bibliografía - [7])

La aplicación previene algunos errores. Por ejemplo en caso de querer agregar un perro sin datos necesarios, no lo permite y pide los datos. Lo mismo sucede con los usuarios o actividades. Cuando se solicita la altura o peso del perro, se especifica que es en centímetros o kilogramos respectivamente. En cierta medida esta regla se cumple.

3.2.6. Reconocer antes que recordar

”Hacer visibles acciones y opciones para que el usuario no tenga que recordar información entre distintas secciones o partes de la aplicación.”(Ver Bibliografía - [7])

La aplicación cumple a medias con esta regla. Las pestañas superiores están siempre presentes y no es necesario recordar en que página esta el usuario actualmente, porque aparece resaltado.

3.2.7. Flexibilidad y eficiencia en el uso

”Los aceleradores o atajos pueden hacer más rápida la interacción para usuarios expertos, de tal forma que la aplicación sea útil tanto para usuarios básicos como avanzados.”(Ver Bibliografía - [7])

No se utilizan aceleradores, sin embargo para versiones posteriores estaría bueno implementarlos.

3.2.8. Diseño estético y minimalista

”Las páginas no deben contener información innecesaria. Cada información extra compite con la información relevante y disminuye su visibilidad.”(Ver Bibliografía - [7])

La aplicación no tiene ninguna pantalla cargada de información y es bastante minimalista. Por lo tanto cumple con esta regla.

3.2.9. Ayuda a los usuarios a reconocer, diagnosticar y recuperarse de los errores

”Los mensajes de error deben estar redactados con un lenguaje simple, deben ofrecerse alternativas para que el usuario pueda continuar realizando la tarea o recuperando lo último que hizo. ”(Ver Bibliografía - [7])

Los mensajes de error siempre dicen claramente cuál es el error y qué se debe hacer para solucionarlo.

3.2.10. Ayuda y documentación

"La ayuda debe ser fácil de localizar, especificar los pasos necesarios y no ser muy extensa." (Ver Bibliografía - [7])

La interfaz es bastante simple y fácil de aprender, no obstante no hay ningún botón de ayuda, por lo tanto no cumple con esta regla.

4. Pruebas funcionales

4.1. Técnicas de prueba aplicadas

La estrategia de pruebas realizadas es la siguiente. Primero se hace una planificación de las pruebas, se diseñan las pruebas, se ejecutan y luego se hace un análisis y reporte de defectos.

Se utilizan las diferentes técnicas de pruebas de caja negra:

- Escenarios de casos de uso
- Partición equivalente
- Análisis de valores límites
- Pruebas exploratorias

Los casos de prueba están asignados a los casos de uso de la letra del problema. Estos son:

1. Gestionar la información básica de la/las mascota/s de la familia con nombre, altura, peso, comentarios e imagen.
2. Calendario compartido de paseos. Se podrá registrar en un calendario quien de la familia es responsable del paseo de la/las mascota/s , en que horario.
3. Calendario de alimentación de la/las mascota/s (indicando cuando, quien y que alimento se le deberá proporcionar).
4. Registro de actividad realizada (paseo, baño, comida, etc) Posteriormente a la actividad de paseo se podrá registrar la distancia recorrida y ruta recorrida en un mapa.
5. Agendar servicio de cuidado en una veterinaria (corte de pelo, uñas, etc).
6. Recordatorios vía mail y notificaciones del sistema a los responsables de realizar las actividades agendadas previamente.

4.2. Casos de prueba

Listar los casos de prueba agrupados según la funcionalidad probada. Para cada caso de prueba, indicar entradas y salidas esperadas con datos concretos. Si varios casos comparten un mismo juego de datos de prueba, indicarlo como precondition.

En las próximas secciones se especificaran los diferentes casos de pruebas basados en escenarios y casos de uso del sistema. Dentro de ellos se utilizara la Partición Equivalente, donde se tomaran distintos valores para las pruebas, con ellos también analizaremos los limites.

4.2.1. Caso de Uso 1

En esta sección se especifican los casos de prueba asociados al caso de uso 1, "Gestionar la información básica de la/las mascota/s de la familia con nombre, altura, peso, comentarios e imagen."

Caso de prueba: Ingreso nombre perro

Clases Validas	Id	Clases No Validas	Id
Cualquier tipo de carácter	CV1	Entrada vacía	CNV1

Entrada	Salida Esperada	Id
"Rasta"	Se guarda el nombre	CV1
"a"	Se guarda el nombre	CV1
" "	Se pide el nombre nuevamente	CNV1

Caso de prueba: Ingreso altura perro

Clases Validas	Id	Clases No Validas	Id
Numero >0	CV1	Numero <= 0	CNV1
		No numero	CNV2
		No Entero	CNV3

Entrada	Salida Esperada	Id
3	Se guarde la altura	CV1
0	Se pida la altura nuevamente	CNV1
-1	Se pida la altura nuevamente	CNV1
"hola"	Se pida la altura nuevamente	CNV2
0.5	Se pida la altura nuevamente	CNV3

Caso de prueba: Ingreso peso perro

Clases Validas	Id	Clases No Validas	Id
Decimal >0	CV1	Numero <= 0	CNV1
		No numero	CNV2

Entrada	Salida Esperada	Id
3,2	Se guarde el peso	CV1
0	Se pida el peso nuevamente	CNV1
-1	Se pida el peso nuevamente	CNV1
"hola"	Se pida el peso nuevamente	CNV2
0.1	Se guarde el peso	CV1

Caso de prueba: Ingreso imagen perro

Clases Validas	Id	Clases No Validas	Id
Imagen	CV1	No imagen	CNV1

Entrada	Salida Esperada	Id
perro.png	Se guarde la imagen	CV1
archivo de texto	Se pida la imagen nuevamente	CNV1

Caso de prueba: Ingreso comentarios perro

Clases Validas	Id	Clases No Validas	Id
Cualquier carácter	CV1		
Vacía	CV2		

Entrada	Salida Esperada	Id
"Es rubio"	Se guarde el comentario	CV1
" "	No se agrega comentario	CV2

4.2.2. Caso de Uso 2

En esta sección se especifican los casos de prueba asociados al caso de uso 2, "Calendario compartido de paseos. Se podrá registrar en un calendario quien de la familia es responsable del paseo de la/las mascota/s , en que horario."

Precondición: Se asumen Usuario: Marcel, Perro: Rasta precargados.

Caso de prueba: Selecciono responsable del paseo

Clases Validas	Id	Clases No Validas	Id
Selecciona un responsable	CV1	No selecciona responsable	CNV1

Entrada	Salida Esperada	Id
Se selecciona Marcel	Se guarda el responsable	CV1
No se selecciona responsable	Se pide responsable nuevamente	CNV1

Caso de prueba: Selecciono perro del paseo

Clases Validas	Id	Clases No Validas	Id
Selecciona un perro	CV1	No selecciona perro	CNV1

Entrada	Salida Esperada	Id
Se selecciona Rasta	Se guarda el perro	CV1
No se selecciona perro	Se pide perro nuevamente	CNV1

Caso de prueba: Ingreso hora del paseo

Clases Validas	Id	Clases No Validas	Id
Hora <24	CV1	Hora >= 24	CNV1
Hora >= 0	CV2	Hora <0	CNV2
Minutos <60	CV3	Minutos >= 60	CNV3
Minutos >= 0	CV4	Minutos <0	CNV4
Enteros	CV5	Decimales	CNV5
		No numérico	CNV6

Entrada	Salida Esperada	Id
Ingreso hora 0	Se guarda hora	CV1 , CV2, CV5
Ingreso hora 23	Se guarda hora	CV1 , CV2, CV5
Ingreso minuto 0	Se guarda minuto	CV3, CV4, CV5
Ingreso hora 24	Se pide hora nuevamente	CNV1
Ingreso hora -1	Se pide hora nuevamente	CNV2
Ingreso minuto 60	Se pide minuto nuevamente	CNV3
Ingreso minuto -1	Se pide minuto nuevamente	CNV4
Ingreso hora 1,5	Se agrega hora 1	CNV5
Ingreso minuto 0,1	Se agrega minuto 0	CNV5
Ingreso hora "Hola"	Se pide hora nuevamente	CNV6
Ingreso minuto "Hola"	Se pide minuto nuevamente	CNV6

4.2.3. Caso de Uso 3

En esta sección se especifican los casos de prueba asociados al caso de uso 3, "Calendario de alimentación de la/las mascota/s (indicando cuando, quien y que alimento se le deberá proporcionar)."

Precondición: Se asumen Usuario: Marcel, Perro: Rasta precargados.

Caso de prueba: Selecciono responsable de la alimentación

Clases Validas	Id	Clases No Validas	Id
Selecciona un responsable	CV1	No selecciona responsable	CNV1

Entrada	Salida Esperada	Id
Se selecciona Marcel	Se guarda el responsable	CV1
No se selecciona responsable	Se pide responsable nuevamente	CNV1

Caso de prueba: Selecciono perro a alimentar

Clases Validas	Id	Clases No Validas	Id
Selecciona un perro	CV1	No selecciona perro	CNV1

Entrada	Salida Esperada	Id
Se selecciona Rasta	Se guarda el perro	CV1
No se selecciona perro	Se pide perro nuevamente	CNV1

Caso de prueba: Ingreso hora de la alimentación

Clases Validas	Id	Clases No Validas	Id
Hora <24	CV1	Hora >= 24	CNV1
Hora >= 0	CV2	Hora <0	CNV2
Minutos <60	CV3	Minutos >= 60	CNV3
Minutos >= 0	CV4	Minutos <0	CNV4
Enteros	CV5	Decimales	CNV5
		No numérico	CNV6

Entrada	Salida Esperada	Id
Ingreso hora 0	Se guarda hora	CV1 , CV2, CV5
Ingreso hora 23	Se guarda hora	CV1 , CV2, CV5
Ingreso minuto 0	Se guarda minuto	CV3, CV4, CV5
Ingreso hora 24	Se pide hora nuevamente	CNV1
Ingreso hora -1	Se pide hora nuevamente	CNV2
Ingreso minuto 60	Se pide minuto nuevamente	CNV3
Ingreso minuto -1	Se pide minuto nuevamente	CNV4
Ingreso hora 1,5	Se guarda hora 1	CNV5
Ingreso minuto 0,1	Se guarda minuto 0	CNV5
Ingreso hora "Hola"'	Se pide hora nuevamente	CNV6
Ingreso minuto "Hola"	Se pide minuto nuevamente	CNV6

4.2.4. Caso de Uso 4

En esta sección se especifican los casos de prueba asociados al caso de uso 4, "Registro de actividad realizada (paseo, baño, comida, etc) Posteriormente a la actividad de paseo se podrá registrar la distancia recorrida y ruta recorrida en un mapa."

Precondición: Se asume Actividad: BañarPerro ya precargada y registrada, ya realizada. Se asume Paseo precargado y registrado, no realizado

Caso de Prueba: Selecciono Actividad y marco como realizada

Clases Validas	Id	Clases No Validas	Id
Actividad seleccionada	CV1	Actividad no seleccionada	CNV1
Marcar como realizada	CV2		
actividad no realizada			
Marcar como realizada	CV3		
actividad ya realizada			

Entrada	Salida Esperada	Id
Selecciona BañarPerro	Sistema queda esperando que sea marcada como realizada	CV1
No selecciona actividad	Sistema no deja marcar como realizada	CNV1
Selecciona y marca Paseo	Paseo queda registrada como realizada	CV2
Selecciona y marca BañarPerro	BañarPerro queda registrada como no realizada	CV3

Caso de Prueba: Ingreso distancia en paseo

Clases Validas	Id	Clases No Validas	Id
Decimal >0	CV1	Numero <= 0	CNV1
		No numero	CNV2

Entrada	Salida Esperada	Id
3,2	Se guarda la distancia	CV1
0	Se pida la distancia nuevamente	CNV1
-1	Se pida la distancia nuevamente	CNV1
"hola"	Se pida la distancia nuevamente	CNV2
0.1	Se guarda la distancia	CV1

4.2.5. Caso de Uso 5

En esta sección se especifican los casos de prueba asociados al caso de uso 5, "Agendar servicio de cuidado en una veterinaria (corte de pelo, uñas, etc)."

Precondición: Se asume Veterinaria: VeterinariaPocitos ya precargada y registrada. Con horario disponible entre las 8hs - 16hs, además tiene una actividad ya agendada a las 10hs.

Caso de Prueba: Selección de Veterinaria

Clases Validas	Id	Clases No Validas	Id
Veterinaria seleccionada	CV1	Veterinaria no seleccionada	CNV1

Entrada	Salida Esperada	Id
Selecciona Veterinaria Pocitos	Sistema queda esperando que se ingresen los demás datos	CV1
No selecciona Veterinaria	Sistema pide que se seleccione una veterinaria	CNV1

Caso de Prueba: Ingreso de Motivo de Visita

Clases Validas	Id	Clases No Validas	Id
Cualquier carácter	CV1	Entrada vacía	CNV1

Entrada	Salida Esperada	Id
"Corte de Pelo"	Se guarda el motivo	CV1
" "	Se pide el motivo nuevamente	CNV1

Caso de Prueba: Ingresa hora para agendar con veterinaria

Clases Validas	Id	Clases No Validas	Id
Hora <17	CV1	Hora >= 17	CNV1
Hora >= 8	CV2	Hora <8	CNV2
Minutos <60	CV3	Minutos >= 60	CNV3
Minutos >= 0	CV4	Minutos <0	CNV4
Enteros	CV5	Decimales	CNV5
		No numérico	CNV6
		Hora = 10	CNV7

Entrada	Salida Esperada	Id
Ingreso hora 16	Se guarda hora	CV1 , CV2, CV5
Ingreso hora 8	Se guarda hora	CV1 , CV2, CV5
Ingreso minuto 0	Se guarda minuto	CV3, CV4, CV5
Ingreso hora 17	Se pide hora nuevamente	CNV1
Ingreso hora 7	Se pide hora nuevamente	CNV2
Ingreso minuto 60	Se pide minuto nuevamente	CNV3
Ingreso minuto -1	Se pide minuto nuevamente	CNV4
Ingreso hora 1,5	Se guarda hora 1	CNV5
Ingreso minuto 0,1	Se guarda minuto 0	CNV5
Ingreso hora "Hola"	Se pide hora nuevamente	CNV6
Ingreso minuto "Hola"	Se pide minuto nuevamente	CNV6
Ingreso hora 10	Se pide hora nuevamente	CNV7

4.2.6. Caso de Uso 6

En esta sección se especifican los casos de prueba asociados al caso de uso 5, "Recordatorios vía mail y notificaciones del sistema a los responsables de realizar las actividades agendadas previamente."

Caso de Prueba: Ingreso de nombre de usuario

Clases Validas	Id	Clases No Validas	Id
Cualquier tipo de carácter	CV1	Entrada vacía	CNV1

Entrada	Salida Esperada	Id
"Marcel"	Se guarda el nombre	CV1
"a"	Se guarda el nombre	CV1
" "	Se pide el nombre nuevamente	CNV1

Caso de Prueba: Ingreso de mail

Clases Validas	Id	Clases No Validas	Id
Cualquier string de la forma "algo@algo"	CV1	Entrada vacia	CNV1
		String no de la forma "algo@algo"	CNV2

Entrada	Salida Esperada	Id
	Se guarda el mail	CV1
	Se pide el mail nuevamente	CNV1
	Se pide el mail nuevamente	CNV2
	Se pide el mail nuevamente	CNV2

4.3. Sesiones de ejecución de pruebas

4.3.1. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP1
- Fecha: 21/11/2018
- Tiempo: 5 minutos
- Objetivo: Testear el caso de prueba 4.2.1 "Ingreso nombre perro" con los valores ya definidos

4.3.2. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP2
- Fecha: 21/11/2018
- Tiempo: 3 minutos
- Objetivo: Testear el caso de prueba 4.2.1 "Ingreso altura perro" con los valores ya definidos

4.3.3. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP3
- Fecha: 21/11/2018
- Tiempo: 3 minutos
- Objetivo: Testear el caso de prueba 4.2.1 "Ingreso peso perro" con los valores ya definidos

4.3.4. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP4
- Fecha: 21/11/2018
- Tiempo: 5 minutos
- Objetivo: Testear el caso de prueba 4.2.1 "Ingreso imagen perro" con los valores ya definidos

4.3.5. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP5
- Fecha: 21/11/2018
- Tiempo: 2 minutos
- Objetivo: Testear el caso de prueba 4.2.1 "Ingreso comentarios perro" con los valores ya definidos

4.3.6. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP6
- Fecha: 21/11/2018
- Tiempo: 5 minutos
- Objetivo: Testear el caso de prueba 4.2.2 "Selección de responsable de paseo" con los valores ya definidos

4.3.7. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP7
- Fecha: 21/11/2018
- Tiempo: 5 minutos
- Objetivo: Testear el caso de prueba 4.2.2 "Selección de perro de paseo" con los valores ya definidos

4.3.8. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP8
- Fecha: 21/11/2018
- Tiempo: 3 minutos
- Objetivo: Testear el caso de prueba 4.2.2 "Selección de hora de paseo" con los valores ya definidos

4.3.9. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP9
- Fecha: 21/11/2018
- Tiempo: 3 minutos
- Objetivo: Testear el caso de prueba 4.2.3 "Selección de responsable de alimentación" con los valores ya definidos

4.3.10. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP10
- Fecha: 21/11/2018
- Tiempo: 3 minutos
- Objetivo: Testear el caso de prueba 4.2.3 "Selección de perro a alimentar" con los valores ya definidos

4.3.11. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP11
- Fecha: 21/11/2018
- Tiempo: 7 minutos
- Objetivo: Testear el caso de prueba 4.2.3 "Selección de hora de alimentación" con los valores ya definidos

4.3.12. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP13
- Fecha: 21/11/2018
- Tiempo: 3 minutos
- Objetivo: Testear el caso de prueba 4.2.5 "Selección de veterinaria" con los valores ya definidos

4.3.13. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP14
- Fecha: 21/11/2018
- Tiempo: 3 minutos
- Objetivo: Testear el caso de prueba 4.2.5 "Selección de motivo de visita" con los valores ya definidos

4.3.14. Sesión de prueba

- Tester: l
- Versión: 1.0
- Id: SP15
- Fecha: 21/11/2018
- Tiempo: 7 minutos
- Objetivo: Testear el caso de prueba 4.2.5 "Ingresa hora para agendar con veterinaria" con los valores ya definidos

4.3.15. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP16
- Fecha: 21/11/2018
- Tiempo: 3 minutos
- Objetivo: Testear el caso de prueba 4.2.6 "Ingreso nombre de usuario" con los valores ya definidos

4.3.16. Sesión de prueba

- Tester:
- Versión: 1.0
- Id: SP17
- Fecha: 21/11/2018
- Tiempo: 5 minutos
- Objetivo: Testear el caso de prueba 4.2.6 "Ingreso de mail" con los valores ya definidos

4.3.17. Sesión de prueba

- Tester:
- Versión: 2.1
- Id: SP18
- Fecha: 21/11/2018
- Tiempo: 10 minutos
- Objetivo: Testear el caso de prueba 4.2.4 "Selecciono actividad y marco como realizada" con los valores ya definidos

4.3.18. Sesión de Prueba Exploratoria

- Tester:
- Versión: 2.1
- Id: SPE1
- Fecha: 21/11/2018
- Tiempo: 15 minutos
- Objetivo: Testear editar diferentes tipos de actividades en diferentes fechas. Probando casos limite. Intentar de editar una actividad mientras edito otra.

4.3.19. Sesión de Prueba Exploratoria

- Tester: l
- Versión: 2.2
- Id: SPE2
- Fecha: 21/11/2018
- Tiempo: 15 minutos
- Objetivo: Editar diferentes actividades, ver como afecta en las ventanas de usuario y perros.

4.3.20. Sesión de Prueba Exploratoria

- Tester:
- Versión: 2.4
- Id: SPE3
- Tiempo: 15 minutos
- Fecha: 22/11/2018
- Objetivo: Chequear como funciona el sistema de notificaciones.

5. Reporte de defectos

5.1. Definición de categorías de defectos

Definir las categorías de defectos que se van a usar en el reporte. Se debe definir en forma separada el tipo y la severidad del defecto.

A continuación se especificara los distintos tipos de defectos:

- Defecto de Chequeo: Este defecto ocurre cuando el sistema no chequea correctamente los datos ingresados, y los guarda. Ej: Nombre vació aceptado.
- Defecto de Sistema: Este defecto ocurre cuando el sistema deja de funcionar.
- Defecto de Función: Este defecto ocurre cuando el sistema no funciona como se esperaba. Ej: Marcar como realizada una actividad y que no se marque.

5.2. Defectos encontrados por iteración

Listar todos los defectos encontrados. La descripción del defecto debe ser reproducible. Los defectos deben estar categorizados.

5.2.1. Reporte de defecto

- Título: Ingreso nombre vació
- Sesión de prueba asociada: 4.3.1
- Descripción: Se ingreso nombre de perro vació y lo acepto
- Reproducible: Ingresar el nombre de perro con un espacio
- Categoría: Defecto de Chequeo
- Severidad: Media
- Estado: No corregido

5.2.2. Reporte de defecto

- Título: Ingreso peso negativo
- Sesión de prueba asociada: 4.3.3
- Descripción: Se ingreso peso negativo y lo acepto como peso 0
- Reproducible: Ingresar el peso del perro en -1
- Categoría: Defecto de Chequeo
- Severidad: Media
- Estado: Corregido parcialmente, en Versión 2.2, el usuario puede editar el peso del perro

5.2.3. Reporte de defecto

- Título: Ingreso imagen errónea
- Sesión de prueba asociada: 4.3.4
- Descripción: Se ingreso un archivo de texto y guardo un pero con imagen vacía, en vez de pedirla nuevamente
- Reproducible: Ingresar archivo que no sea imagen como imagen
- Categoría: Defecto de Chequeo
- Severidad: Baja
- Estado: No corregido

5.2.4. Reporte de defecto

- Título: No ingreso responsable
- Sesión de prueba asociada: 4.3.6
- Descripción: No avisa que ingrese un responsable cuando agrego un paseo sin responsable. Agrega el paseo pero no puedo ver la información.
- Reproducible: Ingresar paseo sin seleccionar ningún responsable
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido en Versión 2.5

5.2.5. Reporte de defecto

- Título: No ingreso perro
- Sesión de prueba asociada: 4.3.7
- Descripción: No avisa que ingrese un perro cuando agrego un paseo sin perro. Agrega el paseo pero no puedo ver la información.
- Reproducible: Ingresar paseo sin seleccionar ningún perro.
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido en Versión 2.5

5.2.6. Reporte de defecto

- Título: Ingreso hora errónea en paseo
- Sesión de prueba asociada: 4.3.8
- Descripción: Ingreso hora 24, el sistema no pide de nuevo la hora. Agrega con la ultima hora que se probó.
- Reproducible: Ingresar paseo con hora 24
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.7. Reporte de defecto

- Título: Ingreso hora errónea en paseo
- Sesión de prueba asociada: 4.3.8
- Descripción: Ingreso hora -1, el sistema no pide de nuevo la hora. Agrega con la ultima hora que se probó.
- Reproducible: Ingresar paseo con hora -1
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.8. Reporte de defecto

- Título: Ingreso minuto erróneo en paseo
- Sesión de prueba asociada: 4.3.8
- Descripción: Ingreso minuto 60, el sistema no pide de nuevo la hora. Agrega con el ultimo minuto que se probó.
- Reproducible: Ingresar paseo con minuto 60
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.9. Reporte de defecto

- Título: Ingreso minuto erróneo en paseo
- Sesión de prueba asociada: 4.3.8
- Descripción: Ingreso minuto -1, el sistema no pide de nuevo la hora. Agrega con el ultimo minuto que se probó.
- Reproducible: Ingresar paseo con minuto -1
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.10. Reporte de defecto

- Título: No ingreso responsable
- Sesión de prueba asociada: 4.3.9
- Descripción: No avisa que ingrese un responsable cuando agrego un visita a veterinaria sin responsable. Agrega el visita a veterinaria pero no puedo ver la información.
- Reproducible: Ingresar visita a veterinaria sin seleccionar ningún responsable
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido en Versión 2.5

5.2.11. Reporte de defecto

- Título: No ingreso perro
- Sesión de prueba asociada: 4.3.10
- Descripción: No avisa que ingrese un perro cuando agrego un alimentación sin perro. Agrega el alimentación pero no puedo ver la información.
- Reproducible: Ingresar alimentación sin seleccionar ningún perro.
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido en Versión 2.5

5.2.12. Reporte de defecto

- Título: Ingreso hora errónea en alimentación
- Sesión de prueba asociada: 4.3.11
- Descripción: Ingreso hora 24, el sistema no pide de nuevo la hora. Agrega con la ultima hora que se probó.
- Reproducible: Ingresar alimentación con hora 24
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.13. Reporte de defecto

- Título: Ingreso hora errónea en alimentación
- Sesión de prueba asociada: 4.3.11
- Descripción: Ingreso hora -1, el sistema no pide de nuevo la hora. Agrega con la ultima hora que se probó.
- Reproducible: Ingresar alimentación con hora -1
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.14. Reporte de defecto

- Título: Ingreso minuto erróneo en alimentación
- Sesión de prueba asociada: 4.3.11
- Descripción: Ingreso minuto 60, el sistema no pide de nuevo la hora. Agrega con el ultimo minuto que se probó.
- Reproducible: Ingresar alimentación con minuto 60
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.15. Reporte de defecto

- Título: Ingreso minuto erróneo en alimentación
- Sesión de prueba asociada: 4.3.11
- Descripción: Ingreso minuto -1, el sistema no pide de nuevo la hora. Agrega con el ultimo minuto que se probó.
- Reproducible: Ingresar alimentación con minuto -1
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.16. Reporte de defecto

- Título: Ingreso hora errónea en visita a veterinaria
- Sesión de prueba asociada: 4.3.12
- Descripción: Ingreso hora 24, el sistema no pide de nuevo la hora. Agrega con la ultima hora que se probó.
- Reproducible: Ingresar visita a veterinaria con hora 24
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.17. Reporte de defecto

- Título: Ingreso hora errónea en visita a veterinaria
- Sesión de prueba asociada: 4.3.12
- Descripción: Ingreso hora -1, el sistema no pide de nuevo la hora. Agrega con la ultima hora que se probó.
- Reproducible: Ingresar visita a veterinaria con hora -1
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.18. Reporte de defecto

- Título: Ingreso minuto erróneo en visita a veterinaria
- Sesión de prueba asociada: 4.3.12
- Descripción: Ingreso minuto 60, el sistema no pide de nuevo la hora. Agrega con el ultimo minuto que se probó.
- Reproducible: Ingresar visita a veterinaria con minuto 60
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.19. Reporte de defecto

- Título: Ingreso minuto erróneo en visita a veterinaria
- Sesión de prueba asociada: 4.3.12
- Descripción: Ingreso minuto -1, el sistema no pide de nuevo la hora. Agrega con el ultimo minuto que se probó.
- Reproducible: Ingresar visita a veterinaria con minuto -1
- Categoría: Defecto de Chequeo
- Severidad: Alta
- Estado: Corregido parcialmente, en Versión 2.1, el usuario puede editar la hora luego de agendar la actividad.

5.2.20. Reporte de defecto

- Título: Ingreso nombre de usuario vacío
- Sesión de prueba asociada: 4.3.15
- Descripción: Se ingreso nombre de usuario vacío y lo acepto
- Reproducible: Ingresar el nombre de usuario con un espacio
- Categoría: Defecto de Chequeo
- Severidad: Media
- Estado: No corregido

5.2.21. Reporte de defecto

- Título: Editar mientras edito
- Sesión de prueba asociada: 4.3.18
- Descripción: Se cayo el sistema cuando intente de editar una actividad mientras editaba otra. No deja seleccionar actividades.
- Reproducible: Editar una actividad. Antes de agregarla de nuevo apretar el botón editar de nuevo.
- Categoría: Defecto de Sistema
- Severidad: Alta
- Estado: Corregido en Versión 2.2

5.2.22. Reporte de defecto

- Título: Repite actividad en pestaña usuario
- Sesión de prueba asociada: 4.3.19
- Descripción: Cuando marque una actividad como realizada, apareció repetida en la lista de próximas actividades del usuario.
- Reproducible: Crear actividad no realizada. Marcar como realizada. Ir a la pestaña de usuarios. Seleccionar el usuario asignado a esa actividad.
- Categoría: Defecto de Función
- Severidad: Media
- Estado: No corregido

5.2.23. Reporte de defecto

- Título: No Actualización de Actividades
- Sesión de prueba asociada: 4.3.19
- Descripción: Cuando edito una actividad, no se actualiza automáticamente en la lista de actividades, recién cuando cambio de fecha y vuelvo a ella se actualiza.
- Reproducible: Editar actividad ya registrada. Cambiar de fecha y volver a ella
- Categoría: Defecto de Función
- Severidad: Baja
- Estado: No corregido

5.2.24. Reporte de defecto

- Título: Hora incorrecta en notificación
- Sesión de prueba asociada: 4.3.20
- Descripción: Cuando llega una notificación, si los minutos en la hora son menores que 10 aparecen sin el cero delante. Lo mismo sucede en la notificación por email. Ver Figura 5.1
- Reproducible: Crear actividad. Seleccionar hora con minutos menor que diez. Ej: 9:05. Esperar a que llegue al notificación.
- Categoría: Defecto de Función
- Severidad: Media
- Estado: Corregido en Versión 2.6

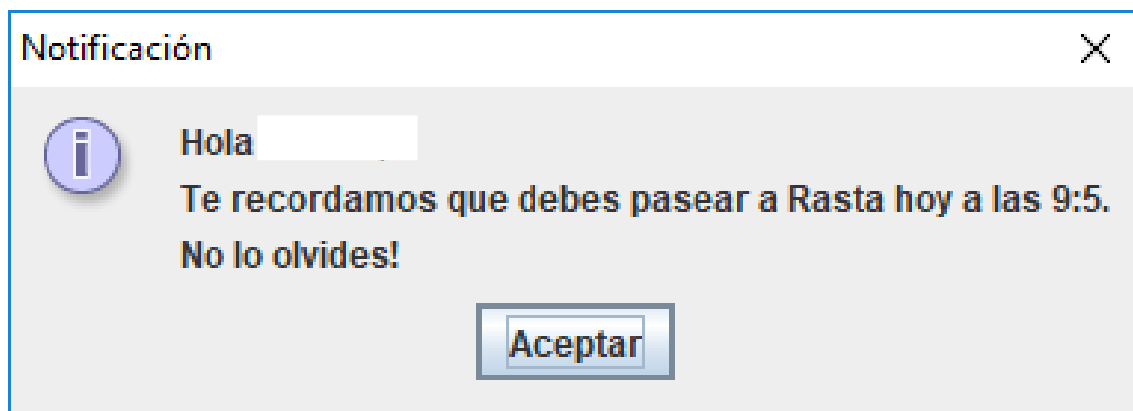


Figura 5.1: Error en la hora de Notificación

5.3. Estado de calidad global

El estado de calidad global de software es estable, no se encontraron mas defectos que hagan que se caiga el sistema. Todos los defectos categorizados con severidad alta fueron corregidos, o parcialmente corregidos.

Sin embargo, varios defectos quedaron sin corregir, por ejemplo el ingreso de imagen errónea y que no aparezca. No creemos que estos defectos no corregidos afecten a la funcionalidad total del sistema.

El total de los defectos encontrados según su severidad son los siguientes:

- Alta: 17
- Media: 5
- Baja: 2

El total de los defectos encontrados según su tipo son los siguientes:

- Defecto de Función: 3
- Defecto de Sistema: 1
- Defecto de Chequeo: 20

En total se encontraron 24 defectos y se corrigieron 19. La búsqueda de defectos nunca termina, y el sistema puede seguir teniendo defectos igualmente. Pero creemos que el sistema actual es significativamente estable.

6. Reflexión

6.1. Oportunidades de mejora

A pesar de haber terminado de implementar el software, siempre hay oportunidades de mejora, y campos en los cuales se podría haber desarrollado mas el proyecto. Por ejemplo, creemos que la interfaz se puede haber desarrollado mejor para ser mas atractiva para el ojo del usuario, se podría haber metido colores o iconos más lindos.

Otra oportunidad de mejora, es el poder corregir todos los defectos encontrados. Hubieron algunos que tuvimos que dejar sin corregir, por ejemplo el ingresar el nombre de usuario vació con un espacio, o que se tenga que cambiar de fecha para que se actualice la lista de actividades.

Por ultimo, el testing siempre es una oportunidad de mejora, porque siempre van a surgir nuevos defectos. Aunque con las pruebas que realizamos el sistema esta estable, no se puede predecir si con más pruebas siga siendo así.

6.2. Conclusiones

A lo largo del proyecto fuimos aplicando y aprendiendo las diferentes técnicas vistas en clase. Pudimos aplicar las pruebas unitarias, y como explicamos en la Sección 2.2, pudimos desarrollar esta técnica de pruebas automatizadas, y entendimos que es mejor hacer las pruebas unitarias mientras se implementan los métodos o hasta antes, así aplicar un Test-Driven Development, y poder desarrollar software más eficientemente y con menos errores.

A su vez, entendimos el funcionamiento de Git y nos sirvió en distintas situaciones. Pudimos trabajar eficientemente en simultaneo y diferenciar versiones según niveles de estabilidad gracias al branching.

Durante las sesiones de prueba y los reportes de defectos, pudimos comprender que con un diseño previo de las pruebas, y tomarse el tiempo para realizarlas se encuentran defectos en software donde no se creía que

había. Haciendo estas pruebas nos dimos cuenta que haciendo software pensamos que no tiene defectos, pero hay que tomarse tiempo para realizar el testing y mejorar la calidad del software.

Bibliografía

- [1] Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. [Online]. Available: <http://www.ort.xxedu.uy/fi/pdf/documento302facultaddeingenieria.pdf>
- [2] Fherz. Buenas Practicas de Commit en Git. [Online]. Available: <https://codigofacilito.com/articulos/buenas-practicas-en-commits-de-git>
- [3] I. de Software 1, “Practicas de codificación,” 2013, Universidad ORT Uruguay.
- [4] I. de Software 1, “Prueba de software,” 2013, Universidad ORT Uruguay.
- [5] Ingeniería de Software 1, “Pruebas unitarias,” 2013, Universidad ORT Uruguay.
- [6] Mauricio Hernandez. 10 HEURISTICAS O PRINCIPIOS BASICOS DE USABILIDAD. [Online]. Available: <http://www.uxabilidad.com/usabilidad/10-heuristicas-o-principios-basicos-de-usablidad.html>