

Obligatorio II

Ingeniería de Software en la Práctica

Universidad ORT Uruguay – Facultad de Ingeniería



Matías Gutierrez

(200244)



Mario Souto

(168772)



Bruno Vezoli

(201150)

CONTENTS

1.	Introducción	4
1.1	Objetivo del documento	4
1.2	Descripción del problema	4
1.3	User Stories	4
2.	Análisis de Riesgos	6
2.1	Usuarios finales	6
2.2	Uso esperado	7
2.3	Mal uso predecible	7
2.4	Análisis de riesgos	7
2.5	Contingencia	7
3.	Riesgos concretados	8
4.	Planning / Resultados	8
4.1	Releases	8
4.2	Release 1	9
4.2.1	Avances Back End	9
4.2.2	Avances Front End	9
4.2.3	Conclusiones de la Retro	10
4.3	Release 2	10
4.3.1	Avances Back End	10
4.3.2	Avances Front End	11
4.3.3	Conclusiones de la Retro	11
4.4	Release 3	12
4.4.1	Avances Back End	12
4.4.2	Avances Front End	12
4.5	Conclusiones del proceso	13
5.	Bocetos de interfaz de usuario	14
5.1	Login - Register	14
5.2	Home – Home [Admin]	15
5.3	Home [Crew] – Home with Menu	15
5.4	Add Damage – Review Damage [Admin]	16
5.5	Accept Damage [Admin] – Assign Damage [Crew]	16
5.6	Repair Damage [Crew] – Damages List	17

6.	Screenshots reales de la aplicación	18
6.1	Login – Register	18
6.2	Home	18
6.3	Review Damage	19
6.4	My Reports.....	20
6.5	Repair Damage	20
7.	Arquitectura	21
7.1	Arquitectura de la API	22
7.1.1	Introducción.....	22
7.1.2	Descripción de alto nivel	22
7.1.3	Modelo de tablas diseñado	24
7.1.4	Modelo de tablas actual	27
7.1.5	Diagrama de deploy.....	27
7.2	Arquitectura de la aplicación Android	28
7.2.1	Diseño inicial.....	28
7.2.2	Implementación.....	29
7.3	Implementación y principales decisiones de diseño	29
7.3.1	Autenticación y autorización	29
7.3.2	Alta de usuarios	32
7.3.3	Gestión de reportes de daños	34
7.3.4	Otros.....	35
8.	Gestión de la configuración	35
8.1	Control de Código	35
6.2	Versionado	36
8.2	Artefactos	36
8.3	Reporte de defectos.....	37
9.	Aseguramiento de calidad.....	37
9.1	Introducción.....	37
9.2	Plan de métricas.....	37
9.2.1	Introducción.....	37
9.2.2	Planificación	38
9.2.3	Definición.....	38
9.3	Plan de aseguramiento de la calidad	39

9.4	Análisis de usabilidad	39
9.4.1	Introduccion.....	39
9.4.2	Resultado esperado	40
9.5	Análisis de calidad de código.....	40
9.5.1	Guías de codificación	40
9.5.2	Calidad de las pruebas unitarias	40
9.6	Calidad del producto	41
9.6.1	Revisiones de código.....	41
9.6.2	Convenciones y uso de estándares.....	41
9.6.3	Código limpio.....	41
9.6.4	Usabilidad.....	42
9.7	Resultado de calidad	42
9.7.1	Cobertura del código	42
9.7.2	Convenciones y uso de estándares.....	43
9.7.3	Conclusion	44
9.8	Calidad del proceso	44
9.8.1	User stories en formato BDD	44
9.8.2	Estimación con planning poker	44
9.8.3	Trackeo de horas.....	44
9.8.4	Flujo de trabajo en Trello	45
9.8.5	Gestión de la configuración.....	47
9.8.6	Organización de releases	49
9.8.7	Retrospectivas	49
9.9	Definición de un modelo	49
9.10	Análisis de usabilidad	49
9.10.1	Heurísticas de Nielsen.....	49
10.	Referencias.....	51

1. INTRODUCCIÓN

1.1 OBJETIVO DEL DOCUMENTO

En el documento presentaremos un contraste de los resultados esperados, definidos en la fase de planificación, y de los resultados obtenidos durante el proceso de desarrollo.

1.2 DESCRIPCIÓN DEL PROBLEMA

El cliente desea hacer una aplicación colaborativa en Android en la que usuarios pueden registrarse y subir reportes de daños en distintas ubicaciones (que incluyen contenido multimedia), luego esos reportes son validados por un administrador y funcionarios se asignan esos reportes para repararlos.

1.3 USER STORIES

A continuación, pueden ver las user stories y su valor dado como resultado del planning poker que realizamos entre los integrantes del equipo.

1. Como Usuario quiero registrarme en la aplicación con un nombre de usuario y contraseña para poder luego loguearme en la aplicación. (3 puntos)
2. Como Usuario quiero loguearme en la aplicación con un nombre de usuario y contraseña para poder acceder a la aplicación. (3 puntos)
3. Como Usuario quiero registrar reportes de daños en una ubicación para poder contribuir a mi comunidad. (3 puntos)
4. Como Usuario quiero recibir notificaciones cuando un daño reportado por mí es reparado para estar al tanto de mis contribuciones. (5 puntos)
5. Como Usuario quiero registrar un reporte de daño con una imagen para mostrar el daño a los encargados de repararlo. (3 puntos)
6. Como Usuario quiero registrar un reporte de daño con un audio para expresar a la comunidad mis sentimientos sobre el daño. (3 puntos)
7. Como Usuario quiero registrar un reporte de daño con un video para mostrar a la comunidad las implicancias que el daño conlleva. (5 puntos)
8. Como Usuario quiero ver en un mapa en tiempo real los reportes de daño cercanos a mi ubicación para tener conocimiento de lo que sucede en mi comunidad. (5 puntos)
9. Como Usuario quiero ver los reportes de daños que he reportado para tener conocimiento del estado de los mismos. (2 puntos)
10. Como Usuario quiero acceder al detalle (imagen/audio/video) de los reportes de daños registrados por otros usuarios para estar al tanto de la gravedad de los mismos. (3 puntos)

11. Como Usuario quiero acceder a las imágenes subidas por los funcionarios para expresar mis sentimientos sobre los arreglos. (2 puntos)
12. Como Usuario quiero compartir por Whatsapp un reporte de daño para que mis contactos estén al tanto de los daños de la comunidad. (3 puntos)
13. Como Administrador quiero asignar el rol de ADMINISTRADOR a otro Usuario, para poder recibir ayuda en el manejo de los reportes de daño. (3 puntos)
14. Como Administrador quiero aceptar, y marcar como ACEPTADO, un reporte de daño para verificar los datos ingresados por el usuario. (2 puntos)
15. Como Administrador quiero ignorar un reporte de daño para filtrar los reportes de daños que no son inválidos. (2 puntos)
16. Como Administrador quiero dar de alta un Funcionario con un nombre de usuario para poder atender los problemas reportados. (2 puntos)
17. Como Administrador quiero dar de baja un Funcionario para eliminar funcionarios que ya no son necesarios. (2 puntos)
18. Como Administrador quiero bloquear un Usuario para poder eliminar del sistema usuarios que suben reportes de daños falsos. (2 puntos)
19. Como Administrador quiero asignar una prioridad ALTA, MEDIA o BAJA en un reporte de daño para que sean atendidos primero los daños con mayor prioridad. (3 puntos)
20. Como Administrador quiero visualizar los reportes reparados por cada funcionario para estar al tanto del rendimiento de los mismos. (3 puntos)
21. Como Funcionario quiero marcar un reporte de daño como REPARADO para informar a los usuarios que el problema fue solucionado. (2 puntos)
22. Como Funcionario quiero marcar un reporte de daño como EN REPARACION para informar a los usuarios que el problema está siendo solucionado. (2 puntos)
23. Como Funcionario quiero ver en un mapa en tiempo real los reportes de daño sin asignar cercanos a mi ubicación para poder reparar los que se encuentren más cercanos. (5 puntos)
24. Como Funcionario quiero asignarme un reporte de daño para evitar que otros funcionarios se dirijan al mismo. (2 puntos)
25. Como Funcionario quiero filtrar por prioridad los reportes de daño para poder atender primero aquellos daños más graves. (3 puntos)
26. Como Funcionario quiero recibir notificaciones cuando se acepta un reporte de daño para estar al tanto de reportes de daños sin asignar. (5 puntos)

27. Como Funcionario quiero subir una imagen cuando el daño fue reparado para que los usuarios puedan ver el antes y después del arreglo. (2 puntos)

28. Como Funcionario quiero visualizar los reportes de daño que he reparado para estar al tanto de mi rendimiento.
(2 puntos)

2. ANÁLISIS DE RIESGOS

2.1 USUARIOS FINALES

La aplicación posee 3 grupos de usuarios finales los que se mapearán con roles dentro del sistema con sus respectivas operaciones permitidas en la aplicación. Estos son (descendentemente ordenados de acuerdo con su nivel de acceso):

1. Administrador
2. Funcionario
3. Usuario

Vale aclarar que no todas las acciones pueden ser realizadas por el administrador. En particular utilizamos la siguiente asignación de permisos:

Administrador (Admin) puede:

- Crear usuarios
- Reportar daños
- Consultar daños
- Modificar daños
- Bloquear usuarios
- Eliminar daños (esto sería como no aceptarlos)

Funcionario (Crew) puede:

- Modificar daños
- Consultar daños
- Reparar daños
- Reportar daños

Usuario (User) puede:

- Reportar daños
- Consultar daños

A su vez cada uno de los roles tiene diferentes condiciones que se deben cumplir de parte de un daño para ser consultado por alguno de los roles.

- Un usuario Administrador puede consultar cualquier daño.

- Un usuario Funcionario puede consultar solamente daños aceptados por un administrador previamente, daños ya reparados, daños que él/ella está reparando o reportó.
- Un usuario con rol Usuario puede solamente consultar daños reparados, aceptados o daños que haya reportado él/ella.

2.2 USO ESPERADO

- Se espera que se utilice en la versión 5.1.1 de Android en un smartphone con 1.2 GHz de velocidad de CPU Quad-Core con 1Gb de RAM y 8GB de almacenamiento.
- Se espera que el smartphone tenga una unidad de almacenamiento estable.
- La API recibirá un número de peticiones que pueda manejar.
- La aplicación estará disponible en Google Play.
- No se usará la misma cuenta simultáneamente en múltiples dispositivos.
- Se espera que los usuarios sean el rol que más abunde en el sistema, seguido por funcionarios y administradores.

2.3 MAL USO PREDECIBLE

- Subida de información errónea acerca de daños inexistentes o bromas de mal gusto.
- Subida de daños duplicados.
- Instalación en un smartphone que no cumple los requerimientos.
- Intento de utilización de la misma cuenta en múltiples dispositivos.

2.4 ANÁLISIS DE RIESGOS

Nº	Riesgo	Efecto	Probabilidad
1	Adopción de una tecnología nueva en el equipo (Android)	Baja velocidad de desarrollo del front-end	Media
2	Adopción de un lenguaje de programación nuevo (Kotlin)	Baja velocidad de desarrollo del front-end	Media
3	Solapado de tareas que baje productividad	Baja velocidad de desarrollo del front-end	Media
4	No llegar a la entrega final	Perder la materia / muchos puntos	Baja
5	Uno de los integrantes abandone el proyecto por voluntad	Scope demasiado largo para dos personas	Baja
6	Uno de los integrantes abandone el proyecto por fuerza mayor	Scope demasiado largo para dos personas	Baja
7	Que se dificulte las tareas por pérdida de materiales	Baja velocidad de desarrollo global	Baja

2.5 CONTINGENCIA

Riesgo Número	Medida para mitigarlo
1 y 2	Contacto de alguien con conocimiento para evacuar dudas.

	Plan de ejercicios de capacitación
3	Reuniones cada 3 días para mejorar comunicación. Crear subareas independientes en cada HU para minimizar solapamiento.
4	División de tareas en releases razonables Mantener controles de avance a nuestros pares
5	Apoyarnos mutuamente y colaborar a la buena convivencia
6	Hacer la mayor cantidad de código al principio y así disminuir el código que tendrían que hacer dos integrantes solos
7	Mantener el código sincronizado con el repositorio remoto para minimizar la cantidad que se podría perder del mismo

3. RIESGOS CONCRETADOS

1, 2 y 3 fueron los riesgos que mas percibimos a lo largo del proceso. La utilización de una tecnología con la que no estábamos familiarizados enlenteció el proceso bastante, además de que la programación de funcionalidades similares en paralelo generó conflictos que no siempre fueron rápidos de resolver.

Gracias a la planificación y previsión de riesgos pudimos sortearlos de una manera relativamente aceptable.

4. PLANNING / RESULTADOS

4.1 RELEASES

Decidimos utilizar Kanban como metodología de trabajo y de dividir el proceso de desarrollo (que tiene de fecha límite el 20 de junio) en 3 releases bien marcados. Se puede apreciar gráficamente el tiempo que dedicaremos a cada uno en la siguiente figura:

APRIL 2018						
SUN	MON	TUE	WED	THU	FRI	SAT
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY 2018						
SUN	MON	TUE	WED	THU	FRI	SAT
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

JUNE 2018						
SUN	MON	TUE	WED	THU	FRI	SAT
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

- **Punto Azul:** Inicio del desarrollo
- **Punto Verde:** Primer release y comienzo de desarrollo para el segundo
- **Punto Violeta:** Segundo release y comienzo de desarrollo para el tercero
- **Punto Rojo:** Release final (entrega)
- **R blanca:** Retrospectiva del release.

También dividimos las historias en cada uno de los releases para tener bien definidas las funcionalidades que cada uno va a tener. Previo a esta división, hubo un pienso con respecto a la realidad de los procesos de desarrollo en nuestra experiencia. Sabemos que tenemos 82 puntos de historias en total y que en general la mayoría del trabajo se termina realizando al final (por condición humana). Como consecuencia decidimos dividir los puntos de la siguiente manera:

Release 1 (21 puntos): Stories 1, 2, 3, 5, 8, 9 y 16. Teniendo como objetivo del release un MVP en el que se pueda gestionar los daños por parte de los usuarios.

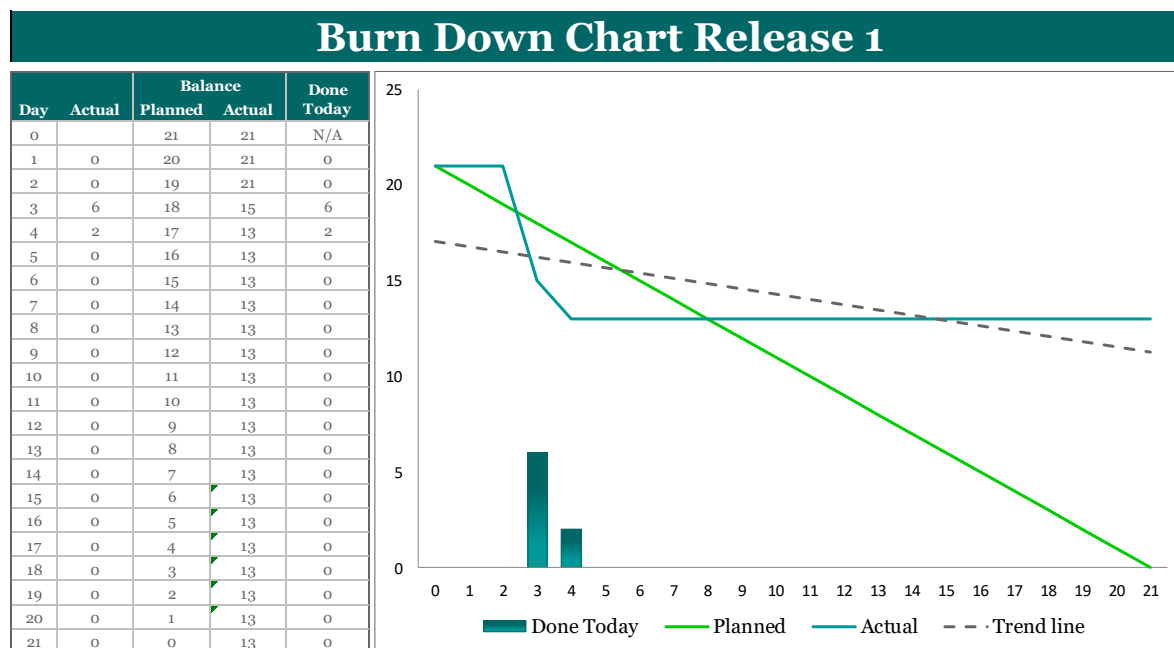
Release 2 (30 puntos): Stories 4, 6, 7, 10, 14, 18, 19, 20, 21, 22. Manejo de notificaciones y multimedia, además de reparaciones por funcionario y funciones de administrador.

Release 3 (31 puntos): Stories 11, 12, 13, 15, 17, 23, 24, 25, 26, 27, 28. El resto de las funcionalidades.

4.2 RELEASE 1

4.2.1 AVANCES BACK END

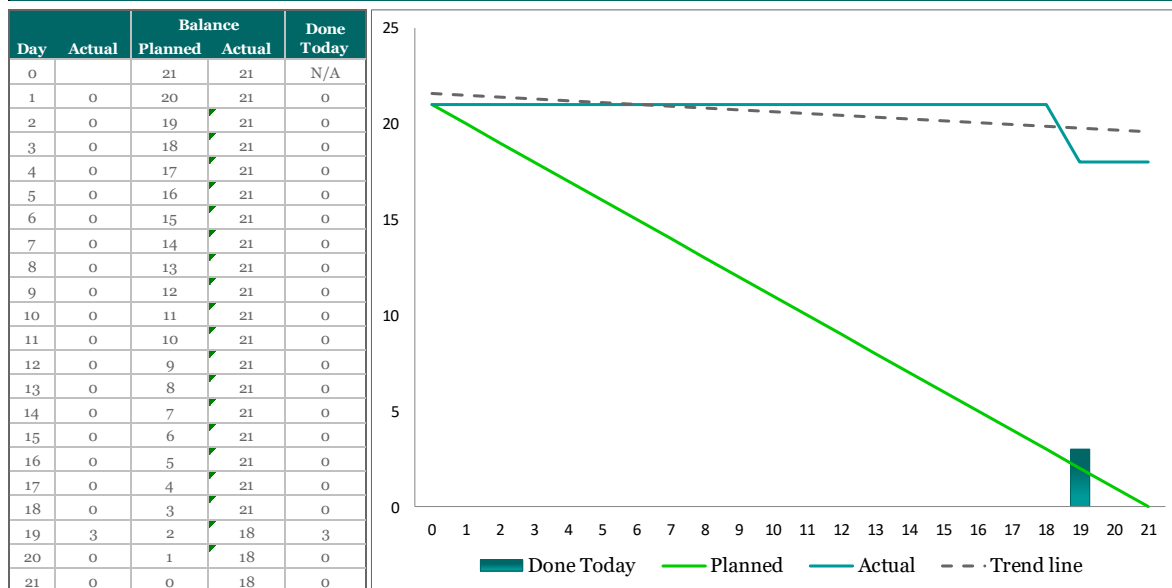
Las features realizadas en el primer release fueron la 1, 2 y 16 por un total de 8 puntos.



4.2.2 AVANCES FRONT END

Se realizó la feature 2.

Burn Down Chart Release 1



4.2.3 CONCLUSIONES DE LA RETRO

4.2.3.1 ¿QUÉ SALIÓ BIEN?

Se logró realizar el diseño inicial de la arquitectura que será reutilizable el resto de la aplicación, a eso se debe la baja velocidad.

4.2.3.2 ¿QUÉ SE PUEDE MEJORAR?

El compromiso y la velocidad de desarrollo.

4.2.3.3 ¿A QUÉ NOS COMPROMETEMOS PARA LA PRÓXIMA ITERACIÓN?

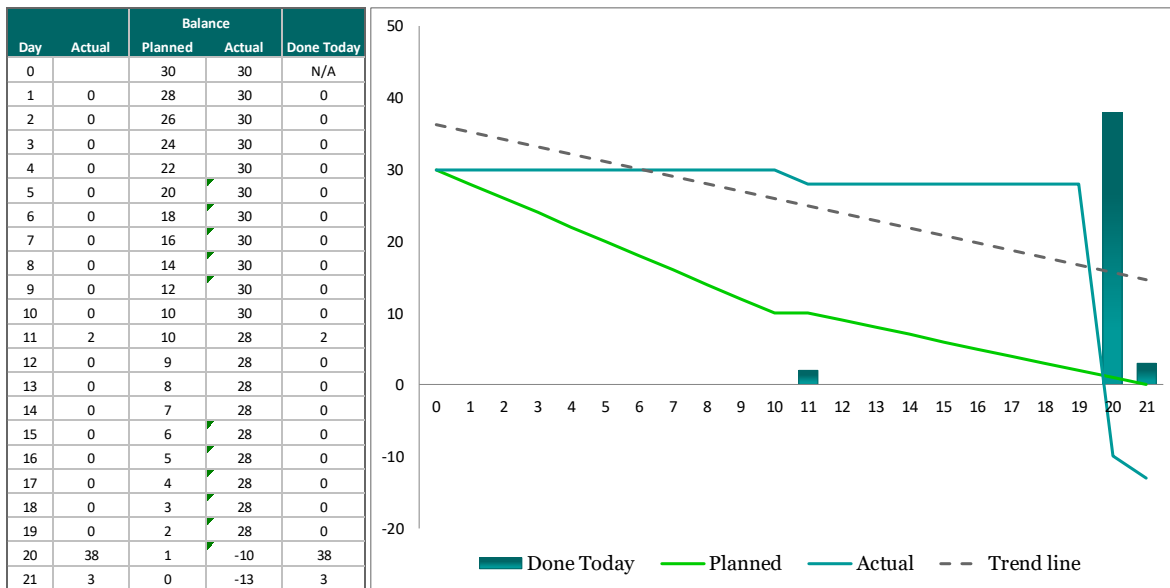
Aumentar la cantidad de features logradas intentando acercarnos a la meta.

4.3 RELEASE 2

4.3.1 AVANCES BACK END

Las features realizadas en el segundo release fueron la 3, 5, 6, 7, 8, 9, 10, 13, 14, 18, 19, 20, 21 y 22 por un total de 43 puntos.

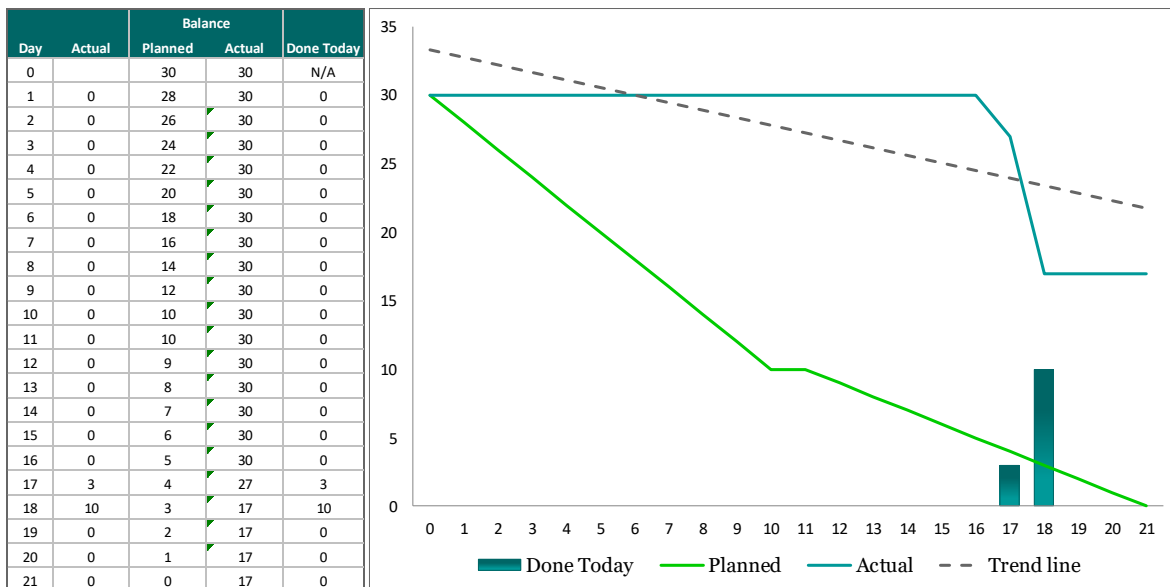
Burn Down Chart Release 2



4.3.2 AVANCES FRONT END

Se hicieron las features 1, 8 y la 23

Burn Down Chart Release 2



4.3.3 CONCLUSIONES DE LA RETRO

4.3.3.1 ¿QUÉ SALIÓ BIEN?

Se logró superar la cantidad de features esperadas para el sprint en el back end.

4.3.3.2 ¿QUÉ SE PUEDE MEJORAR?

Podríamos distribuir el trabajo en el tiempo en lugar de hacer tanta cosa junta y mejorar la productividad del front end.

4.3.3.3 ¿A QUÉ NOS COMPROMETEMOS PARA LA PRÓXIMA ITERACIÓN?

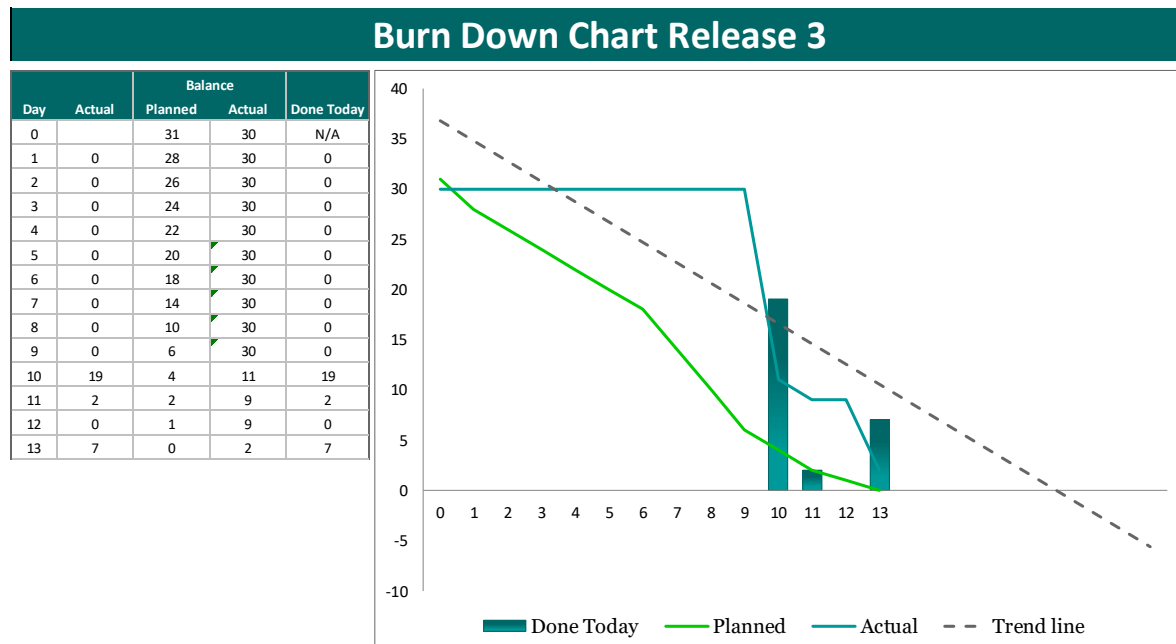
A hacer lo posible por terminar las features restantes.

4.4 RELEASE 3

4.4.1 AVANCES BACK END

Se realizaron las historias 4, 11, 15, 17, 23, 24, 25, 26, 27 y 28 por un total de 28 puntos.

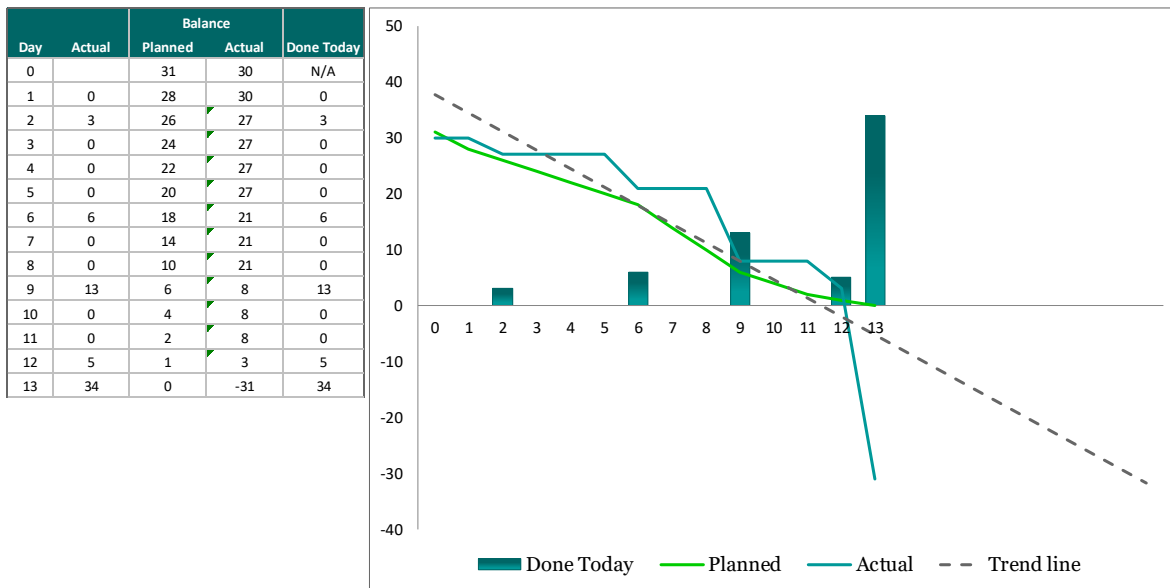
La única feature que no se hizo fue la 12.



4.4.2 AVANCES FRONT END

Se hicieron las features 4, 5, 6, 7, 10, 14, 24, 3, 9, 13, 15, 16, 17, 18, 19, 20, 21, 22, 25, 26, 28-

Burn Down Chart Release 3



4.5 CONCLUSIONES DEL PROCESO

Se pudo haber sido más prolijo con los tiempos, pero se logró un producto que creemos aceptable basados en las premisas iniciales y en la lista de user stories.

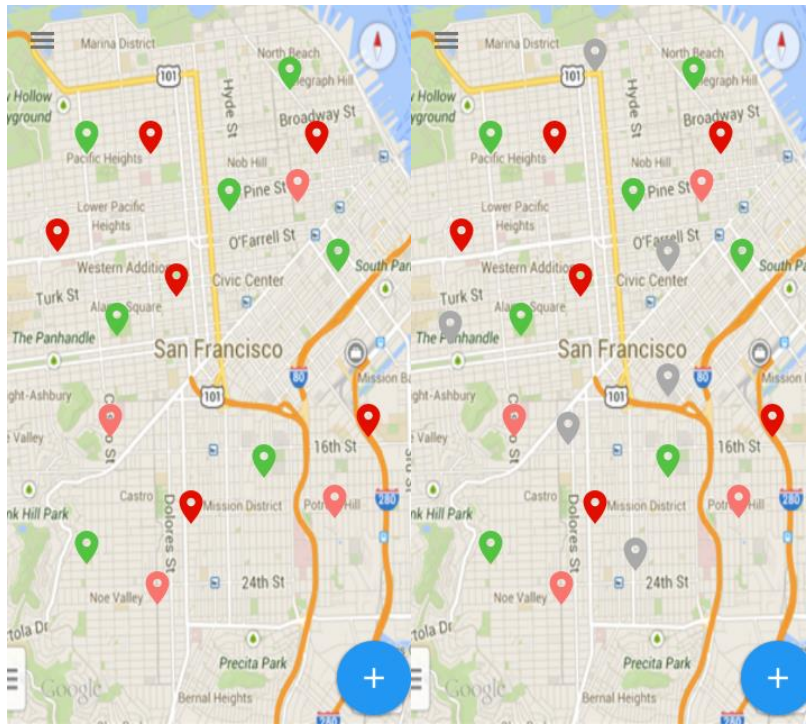
Las únicas stories que no pudimos hacer fueron la 11, la 12 y la 27. En el caso de la 12 se decidió que no aportaba valor con respecto al beneficio que brindaba por el hecho de necesitar una pagina web para mostrar el link del reporte de daños.

5. BOCETOS DE INTERFAZ DE USUARIO

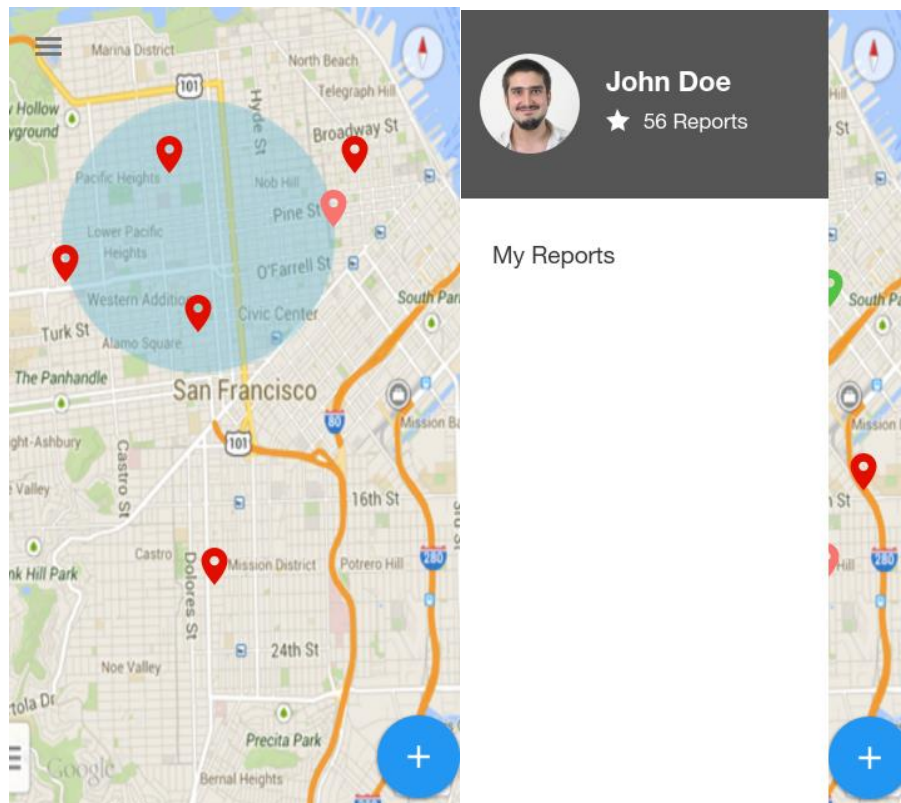
5.1 LOGIN - REGISTER

TeLoArreglo		TeLoArreglo	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="password"/>	<input type="password"/>	<input type="password"/>	<input type="password"/>
<input type="button" value="Log In"/>	<input type="button" value="Register"/>	<input type="button" value="Register"/>	<input type="button" value="Register"/>

5.2 HOME – HOME [ADMIN]



5.3 HOME [CREW] – HOME WITH MENU



5.4 ADD DAMAGE – REVIEW DAMAGE [ADMIN]


Add City Damage

Description

Donec rhoncus vitae eros mattis vulputate. Interdum et malesuada fames ac ante ipsum primis in faucibus. Quisque in nibh aliquet, scelerisque arcu ut, mattis eros. Etiam lacinia quis erat nec ornare. In ac tincidunt quam. Suspendisse fermentum quam viverra feugiat varius. Suspendisse vulputate enim sed neque imperdiet, tempor elementum erat volutpat. Donec in diam nulla. Curabitur dui ipsum, imperdiet eget dui in, dignissim molestie odio. Nunc blandit pretium fermentum.

Submit

Review City Damage




Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce convallis pellentesque metus id lacinia. Nunc dapibus pulvinar auctor. Duis nec sem at orci commodo viverra id in ipsum.

Accept

Reject

5.5 ACCEPT DAMAGE [ADMIN] – ASSIGN DAMAGE [CREW]

Review City Damage



Priority

Low


Medium

High

Accept

Reject

2854 Cartwright Curve




Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce convallis pellentesque metus id lacinia. Nunc dapibus pulvinar auctor. Duis nec sem at orci commodo viverra id in ipsum.

Assign To Me

5.6 REPAIR DAMAGE [CREW] – DAMAGES LIST

2854 Cartwright Curve


My Damages




In repair

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce convallis pellentesque metus id lacinia. Nunc dapibus pulvinar auctor. Duis nec sem at orci commodo viverra id in ipsum.

Repair





763 Lowe Skyway

80041 Stark Crescent

13914 Murray Court

515 Kariane Villages

5821 Jerde Inlet

11358 Ruth Glens

9947 Remington Inlet

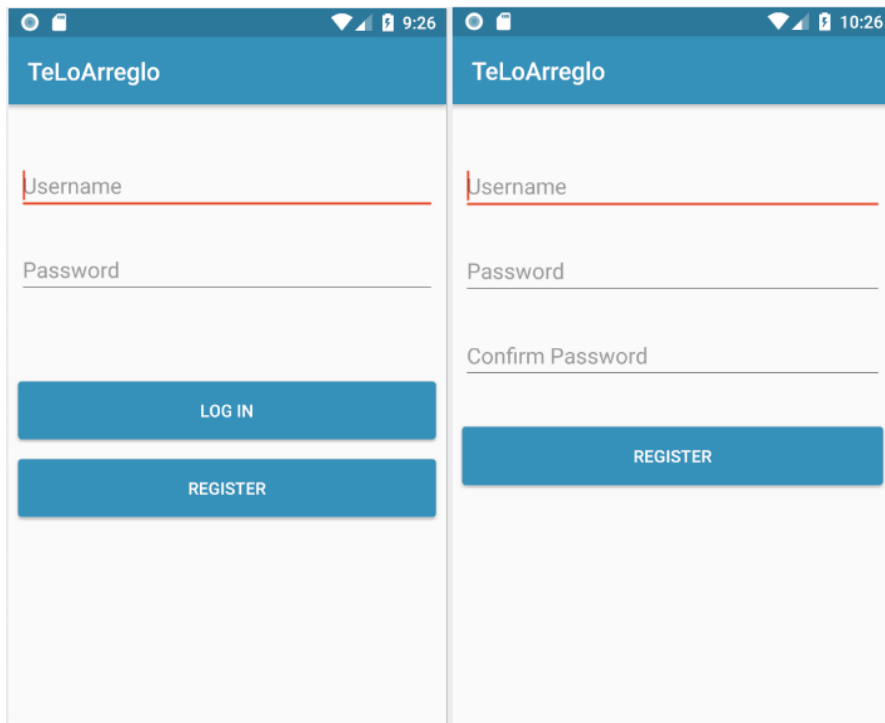
83110 Tess Haven

2414 Anastasia Inlet

2854 Cartwright Curve

6. SCREENSHOTS REALES DE LA APLICACIÓN

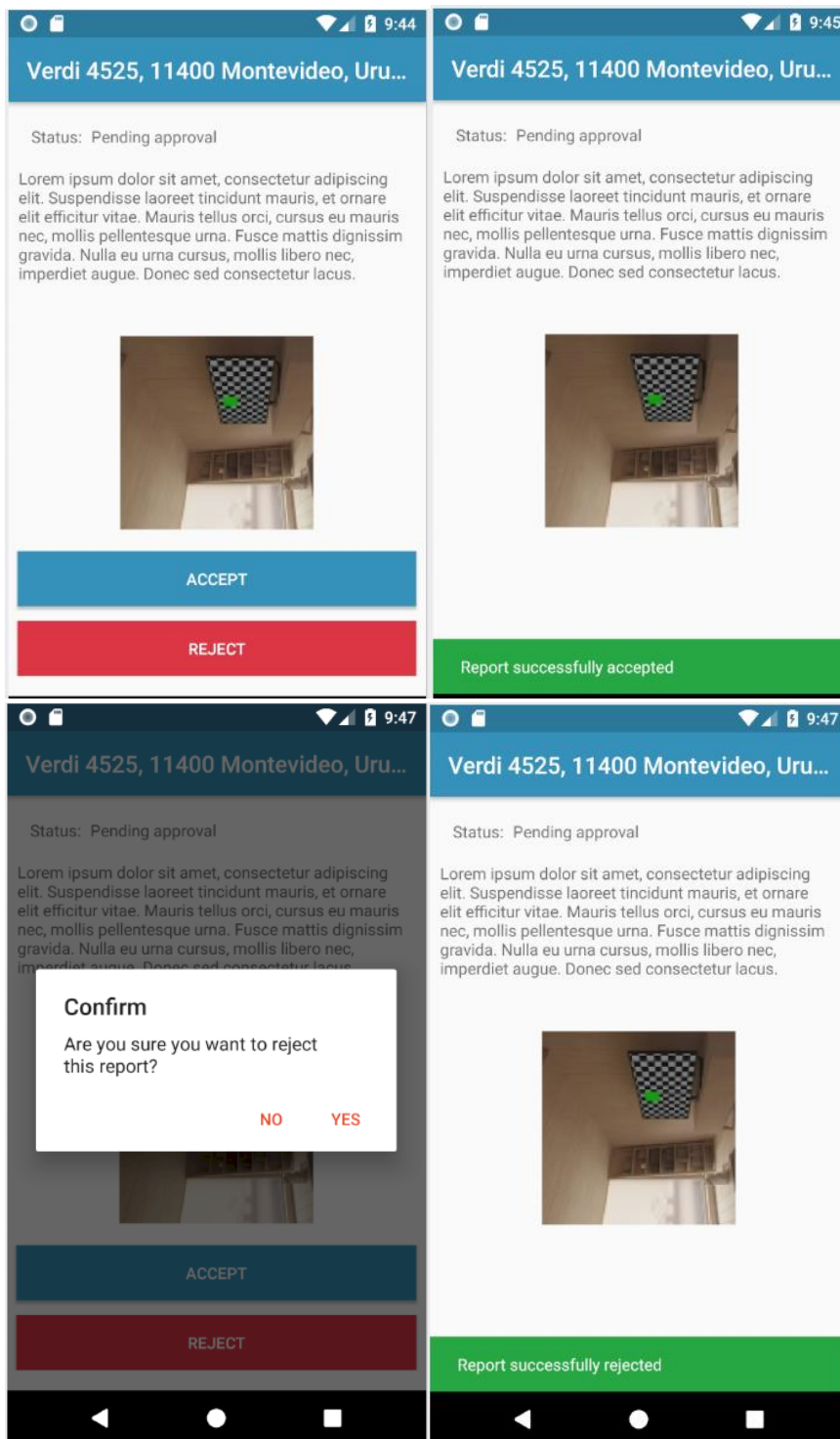
6.1 LOGIN – REGISTER

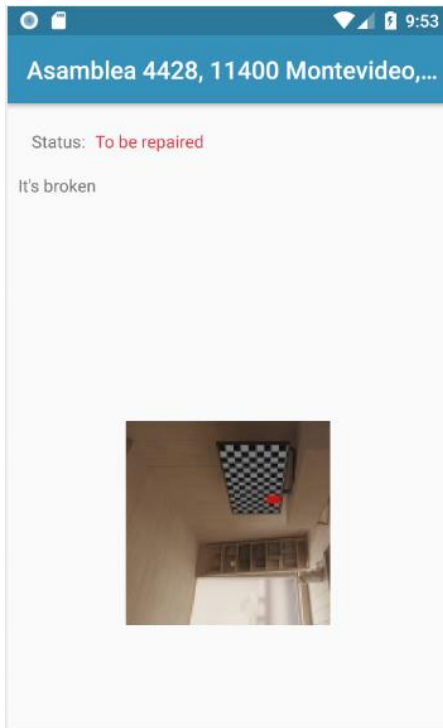


6.2 HOME

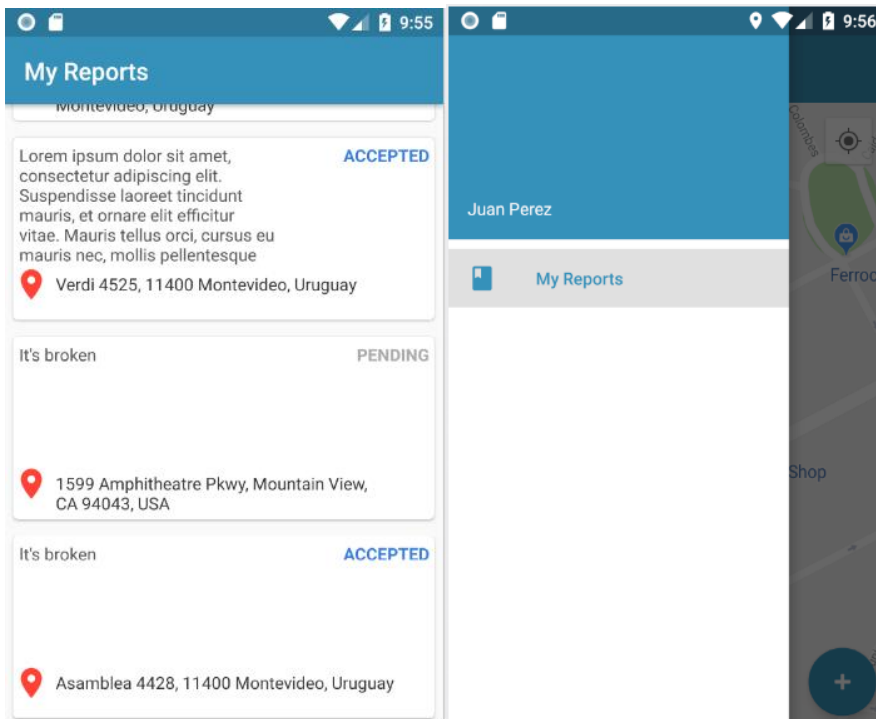


6.3 REVIEW DAMAGE

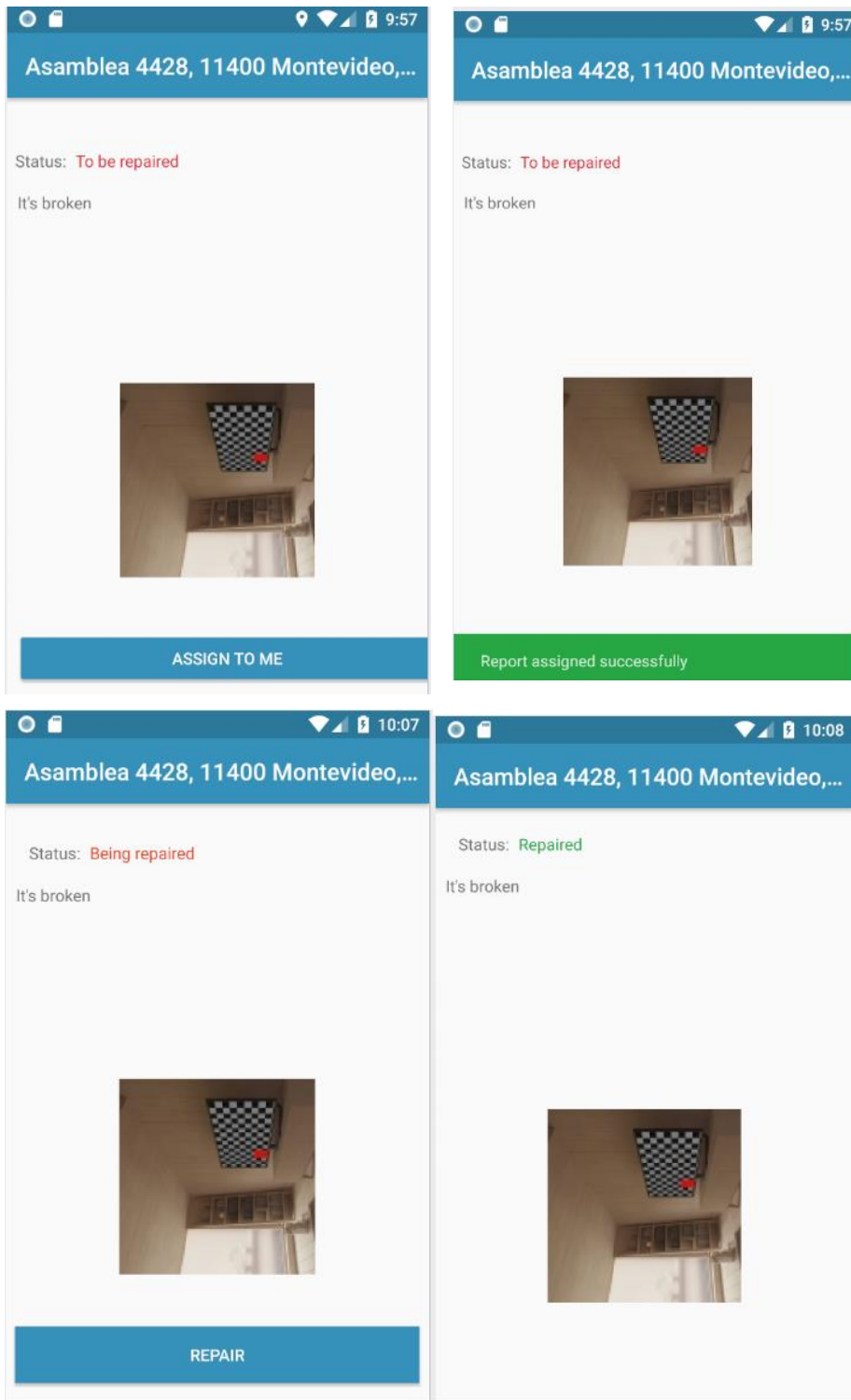




6.4 MY REPORTS



6.5 REPAIR DAMAGE



7. ARQUITECTURA

7.1 ARQUITECTURA DE LA API

7.1.1 INTRODUCCIÓN.

Como la letra indica implementaremos el back end con una API en .NET.

Utilizaremos para la misma una layered architecture que se apoya principalmente en los principios SOLID y en los patrones GRASP.

7.1.2 DESCRIPCIÓN DE ALTO NIVEL

7.1.2.1 IDEA ORIGINAL

Utilizaremos la división de responsabilidades a nivel macro que se muestra en la figura 1.

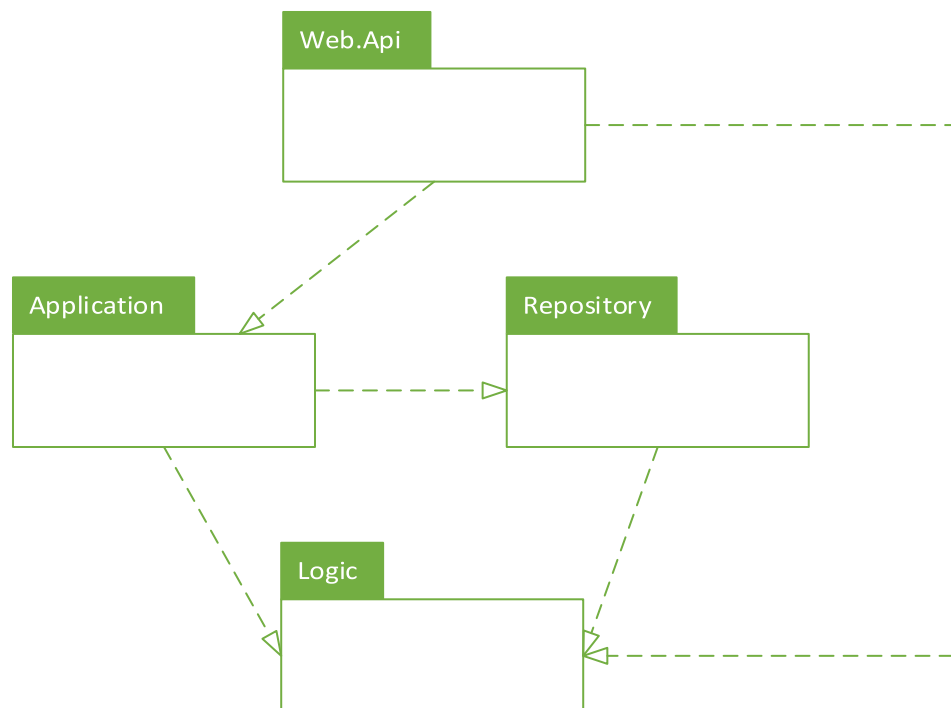


Figura 1 - Diagrama de Paquetes Macro 1

En ella podemos apreciar que se respeta DIP (Dependency Inversion Principle) ya que el paquete que contiene la lógica de negocio no dependerá de otros paquetes más cercanos a la tecnología específica (bajo nivel – alto nivel). Esto nos permitiría, por ejemplo, el día de mañana cambiar la tecnología de la conexión a la base de datos, sin afectar otros componentes de la solución.

También vale aclarar que utilizaremos [ASP.NET Boilerplate](#) (previamente consensuado con el docente) el cual es un paquete nuGet que nos permitirá implementar de manera relativamente simple la inyección de dependencias por medio de reflection.

Lo vemos como una ventaja ya que a pesar de que las dependencias siguen siendo como se muestra en la figura 1, los servicios como por ejemplo repositorios de bases de datos, application services y domain services serán inyectados por medio de los constructores, utilizando interfaces. De esta manera el acoplamiento entre capas (que no son la de lógica) se dará por medio de interfaces y no de clases concretas lo que disminuye el acoplamiento.

Por otro lado, creemos que otro aspecto importante a explicar de nuestra arquitectura es el por qué de la capa de aplicación y su dependencia con el repositorio, en lugar de utilizar una arquitectura en la que la API le delega peticiones a la capa de lógica y ésta al repositorio. En nuestro caso creemos que es bueno tener esta capa intermedia que serviría como orquestador de los diferentes elementos que no necesariamente son lógica de negocio, dejando la misma más limpia de operaciones que no son su responsabilidad.

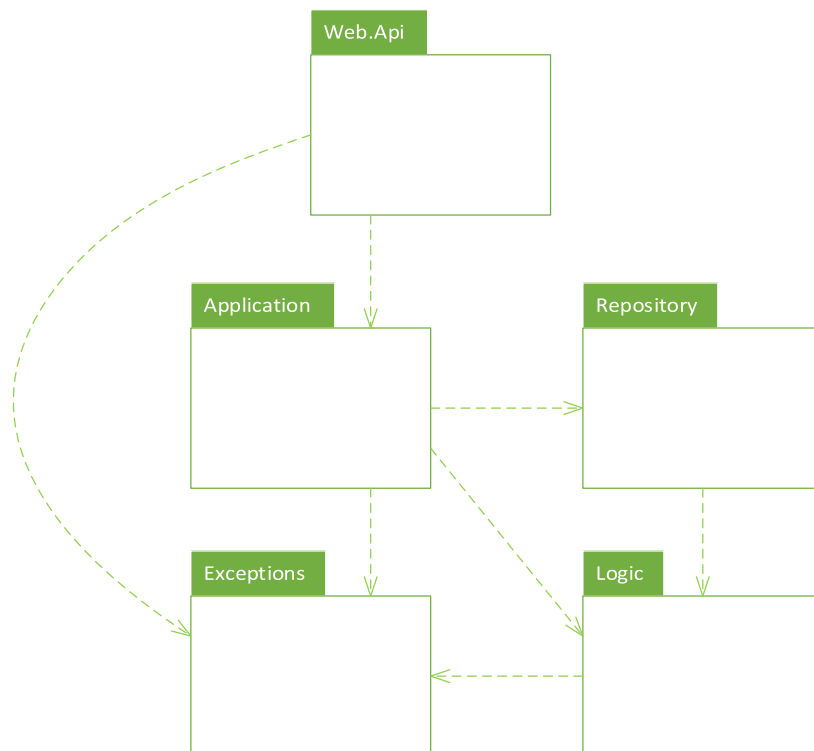
Esa decisión fue inspirada por el siguiente post:

<http://enterprisecraftsmanship.com/2016/09/08/domain-services-vs-application-services/>

Uno puede compartirla o no, pero creemos que los argumentos son fuertes y nos van a ayudar a mantener paquetes altamente cohesivos.

7.1.2.2 IMPLEMENTACIÓN

El diagrama final quedó definido de la siguiente manera:



Estas son las decisiones de diseño que nos llevaron a realizar las modificaciones que se ven entre el diagrama planificado y el final:

- Se creó un paquete de excepciones porque sabemos que va a ser un paquete estable y muchos van a depender de él. Además, la lógica no tenía porque tener la responsabilidad de conocer todas las excepciones, sino las que se lanzan desde ella.
- No fue necesario conocer el paquete de lógica desde la Web.Api ya que solo conoce los dtos que están en la capa Application.

7.1.2.3 WEB API

- Es el punto de entrada de la aplicación.
- Routea las peticiones entrantes a su correspondiente destino y devuelve respuestas dadas por las capas inferiores.
- Las entradas y salidas de información se harán utilizando JSON (mapea JSON a DTOS y viceversa).
- Realiza los correspondientes parseos de headers para manejar la autenticación.
- Es simple, delegando la mayoría del trabajo a la capa de aplicación.

Entrando más en el área de los endpoints en particular, nos basaremos en el libro "Web API Design Crafting Interfaces that Developers Love" para el diseño de los endpoints por recurso y sus correspondientes mensajes http esperados.

7.1.2.4 APPLICATION

- Maneja el mapeo de DTOS a elementos del modelo (capa de lógica).
- Maneja que los usuarios tengan permiso de realizar las operaciones ayudada por las capas de lógica y repositorios.
- Orquesta el manejo de transacciones.
- Orquesta el flujo de trabajo, pero delega a la capa de dominio cada paso del mismo.

7.1.2.5 LOGIC

- Contiene la lógica de negocio de la aplicación.
- Contiene los modelos.
- Mantiene la integridad de datos.

7.1.2.6 REPOSITORY

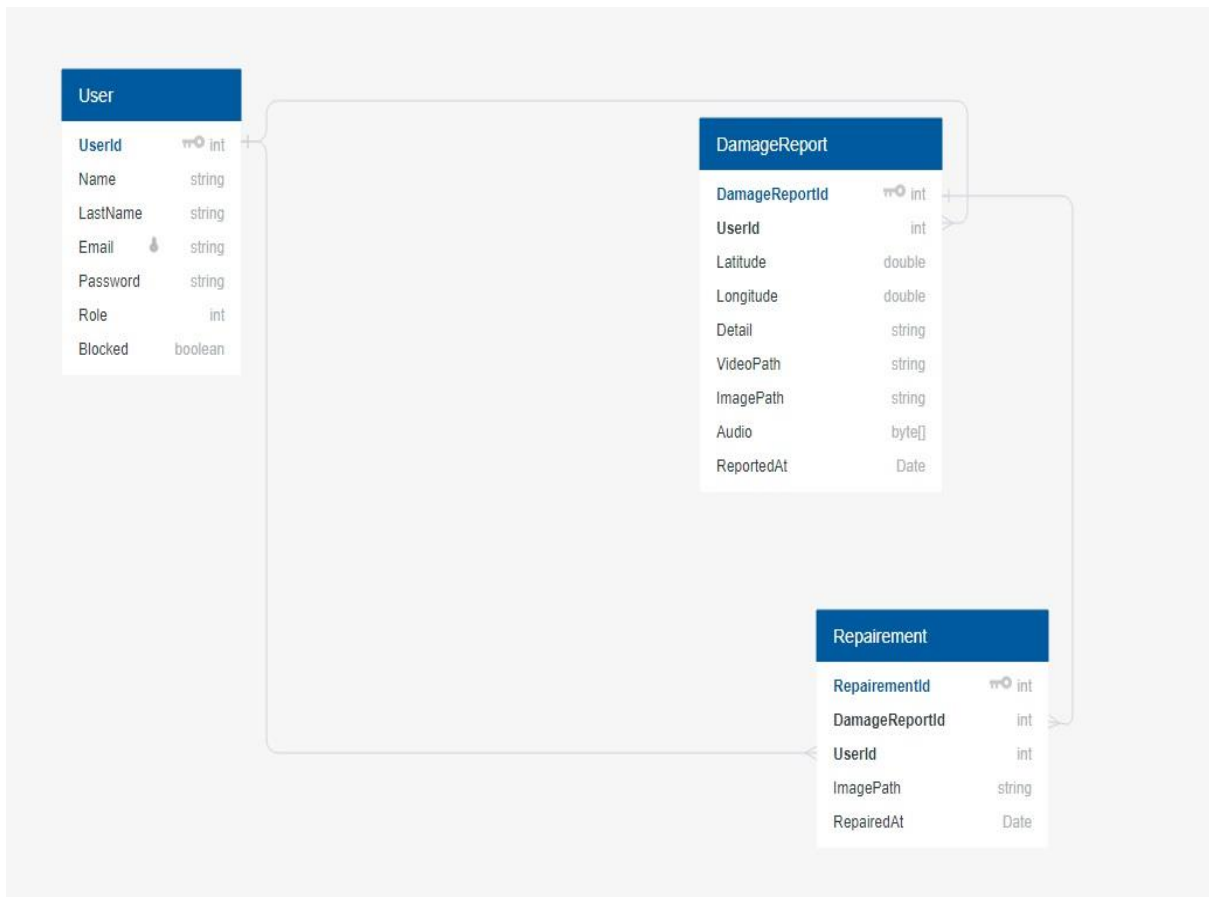
- Provee una interfaz programática para el acceso a datos persistentes en una base de datos.
- Maneja migraciones.

7.1.2.7 EXCEPTIONS

- Contiene todas las excepciones, tanto de lógica como de aplicación.

7.1.3 MODELO DE TABLAS DISEÑADO

A continuación, se presenta el modelo de tablas de la API REST y se describen las principales decisiones efectuadas



User	
UserId	int
Name	string
LastName	string
Email	string
Password	string
Role	int
Blocked	boolean

Se tendrá una tabla USER, donde el correo electrónico será único y en donde cada usuario tendrá un rol definido (Usuario común, Administrador ó Funcionario) representado por un valor entero.

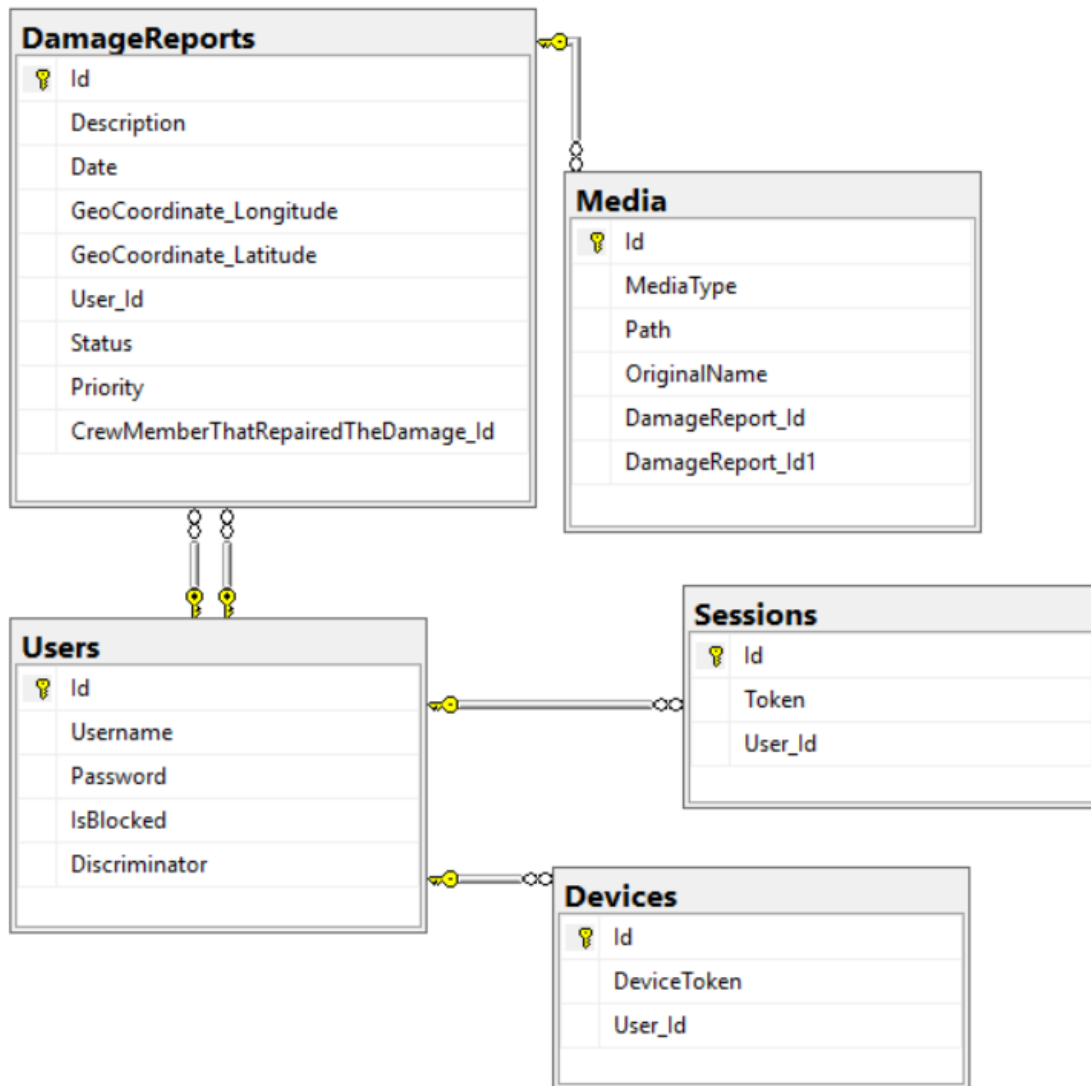
DamageReport	
DamageReportId	int
UserId	int
Latitude	double
Longitude	double
Detail	string
VideoPath	string
ImagePath	string
Audio	byte[]
ReportedAt	Date

Se tendrá una tabla DAMAGE REPORT, que contiene los reportes de daños subidos por un usuario registrado, por lo que la tabla tiene clave foránea USERID a la table USER. El reporte de daño almacena su latitud y longitud para representarse en la aplicación mobile.

Repairement	
RepairementId	int
DamageReportId	int
UserId	int
ImagePath	string
RepairedAt	Date

Finalmente la tabla REPAIRMENT hace referencia a los arreglos de los reporte de daño. Por lo que tienen una clave foránea DAMAGEREPORTID a la tabla DAMAGEREPORT y una clave foránea USERID a la tabla USER ya que el arreglo es realizado por un Funcionario, que es un rol de usuario.

7.1.4 MODELO DE TABLAS ACTUAL

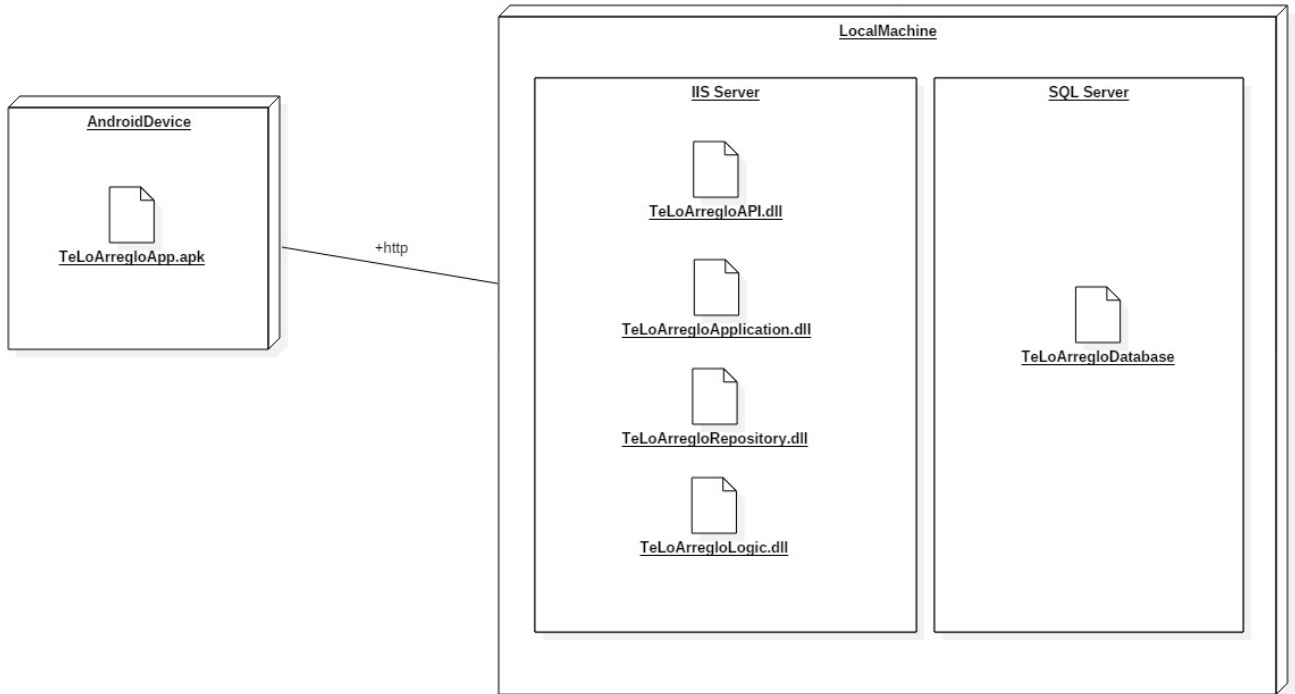


Preguntas que se pueden generar al ver el diagrama:

- ¿Por qué los archivos multimedia tienen dos DamageReportId?

Esto se dio porque en realidad los DamageReports tienen dos listas, una de los recursos multimedia que el usuario sube cuando reporta el daño, y otra de los recursos que suben los funcionarios cuando lo reparan.

7.1.5 DIAGRAMA DE DEPLOY



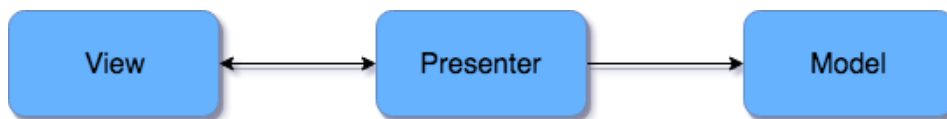
Se deployará localmente la API REST, en el IIS local, desde Visual Studio. Allí se ejecutarán los 4 artefactos necesarios, según la justificación brindada en el diseño de alto nivel.

La API REST se conecta a una base de datos SQL Server deployada en el servidor local de Microsoft SQL Server de la misma máquina local.

Esta API interactúa, vía http, con una aplicación mobile corriendo en un dispositivo Android, representada por el artefacto .apk que se ve en el diagrama.

7.2 ARQUITECTURA DE LA APLICACIÓN ANDROID

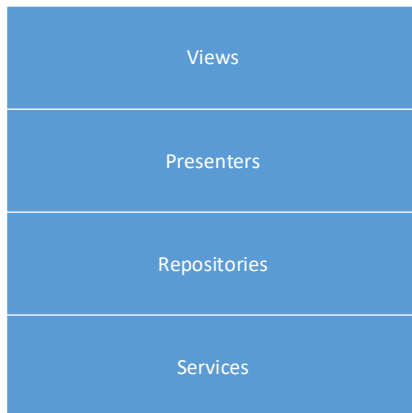
7.2.1 DISEÑO INICIAL



Para el desarrollo de la aplicación Android utilizaremos la arquitectura MVP, Model View Presenter, donde

- **Model** es la interfaz responsable del manejo de los datos, el acceso a bases de datos, caching e interacción con APIS.
- **View**, es la encargada de mostrar la información de una forma definida por el presenter. Puede ser implementado por activities o fragments.
- **Presenter**, es la capa intermedia entre el modelo y la vista. La lógica de la vista reside en el presenter que es el encargado de interactuar con el modelo y actualizar la vista.

7.2.2 IMPLEMENTACIÓN



La aplicación en Android puede ser simplificada en las 4 capas que se ven en la figura (las cuales son una simplificación de la arquitectura Model-View-Presenter).

Las responsabilidades son las siguientes:

Services: Son los encargados de realizar las llamadas HTTP a la api y parsear las respuestas a modelos que entiende la aplicación.

Repositories: Abstraen la capa de servicios, siguiendo el repository pattern.

Presenters: Funcionan como una capa intermedia entre las views y los repositories, orquestando el comportamiento de la view según las interacciones del usuario.

Views: Muestran información de manera amigable y reciben interacción del usuario.

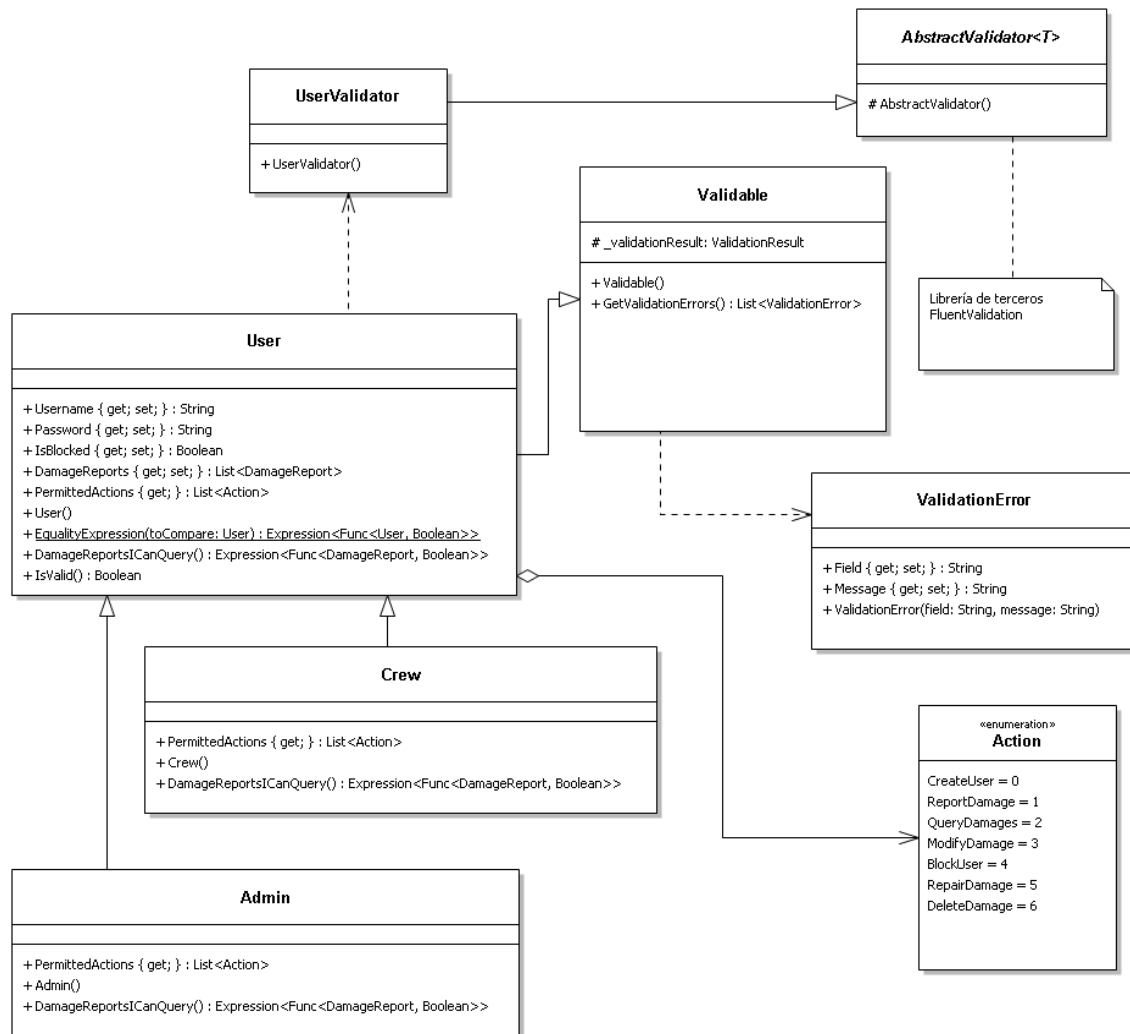
7.3 IMPLEMENTACIÓN Y PRINCIPALES DECISIONES DE DISEÑO

Vale aclarar que omitimos varios nombres de métodos que nos parecieron intrascendentes a la explicación general de la arquitectura de los componentes. Para ver una implementación completa recomendamos ir al código. El propósito de esta sección es brindar una explicación general y justificar las decisiones que tomamos en ciertas secciones cruciales del software.

7.3.1 AUTENTICACIÓN Y AUTORIZACIÓN

7.3.1.1 BACK END

Particularmente la sección de usuarios fue implementada de la siguiente manera:



Utilizamos la librería FluentValidation porque permite implementar de manera simple y escalable validaciones “clásicas” para las diferentes clases de nuestro dominio. En este caso se muestra la utilización para los usuarios, pero en realidad diseñamos la arquitectura para que sea reutilizable para otras clases y así tomar provecho del código que es claramente necesario en otras secciones de la solución.

Este caso funciona básicamente creando una clase UserValidator, en cuyo constructor declara las reglas de validación que se impondrán.

Ej.:

```

public class UserValidator : AbstractValidator<User>
{
    public UserValidator()
    {
        RuleFor(user => user.Username)
        .NotEmpty()
        .WithMessage("Username cannot be empty");
    }
}

```

```

    }
}

```

Esta clase es utilizada por la clase User en su método IsValid(), dejando el resultado de esta validación en la variable _validationResult (de tipo FluentValidation.ValidationResult), heredada de la clase Validable.

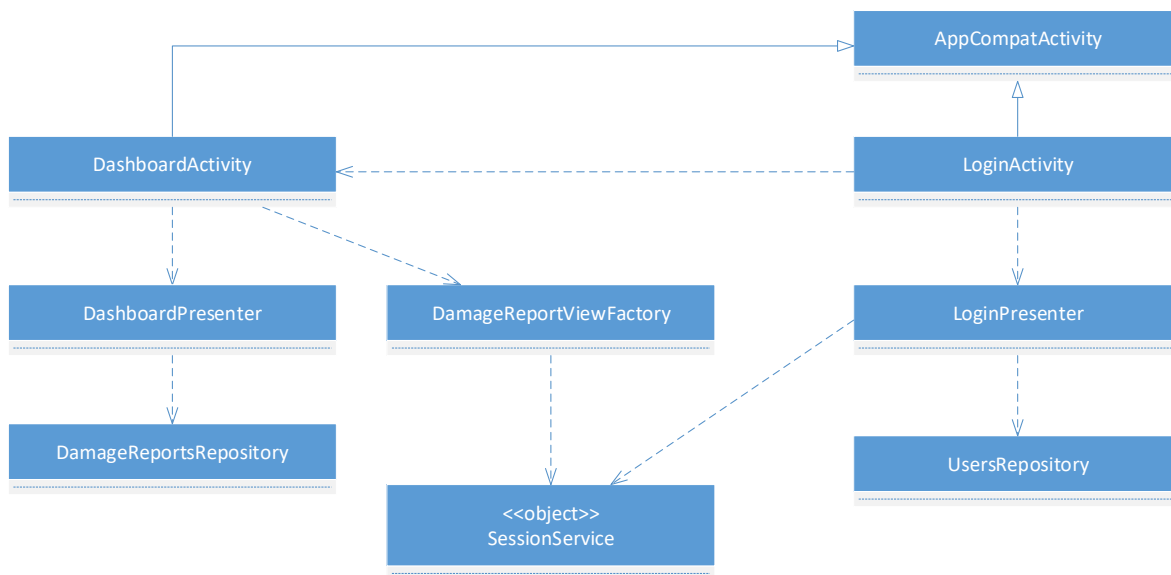
Nos es útil ya que en caso de que un usuario NO sea válido, se puede acceder a un mapeo de los errores a nuestra clase de dominio ValidationError, que se utilizan para enviarle a los clientes de la API cuáles fueron los campos en los que fallaron validaciones todos al mismo tiempo y que ellos programáticamente puedan mapearlos con los campos del formulario que muestran en la interfaz.

Con respecto a la autenticación, fue implementada como lo diseñamos al principio, utilizando x-authentication. Enviando usuario y contraseña se recibe un token que luego se envía en el header en el resto de las requests y la api chequea que tenga permiso de realizar esas acciones.

Otro punto interesante a aclarar es que cuando un usuario se loguea, el cliente de la api recibe una lista de todas las acciones que ese usuario puede realizar, para facilitar el correcto control de elementos a visualizar en la interfaz de usuario. Estas acciones se pueden ver en el diagrama de clases presentado.

7.3.1.2 FRONT END

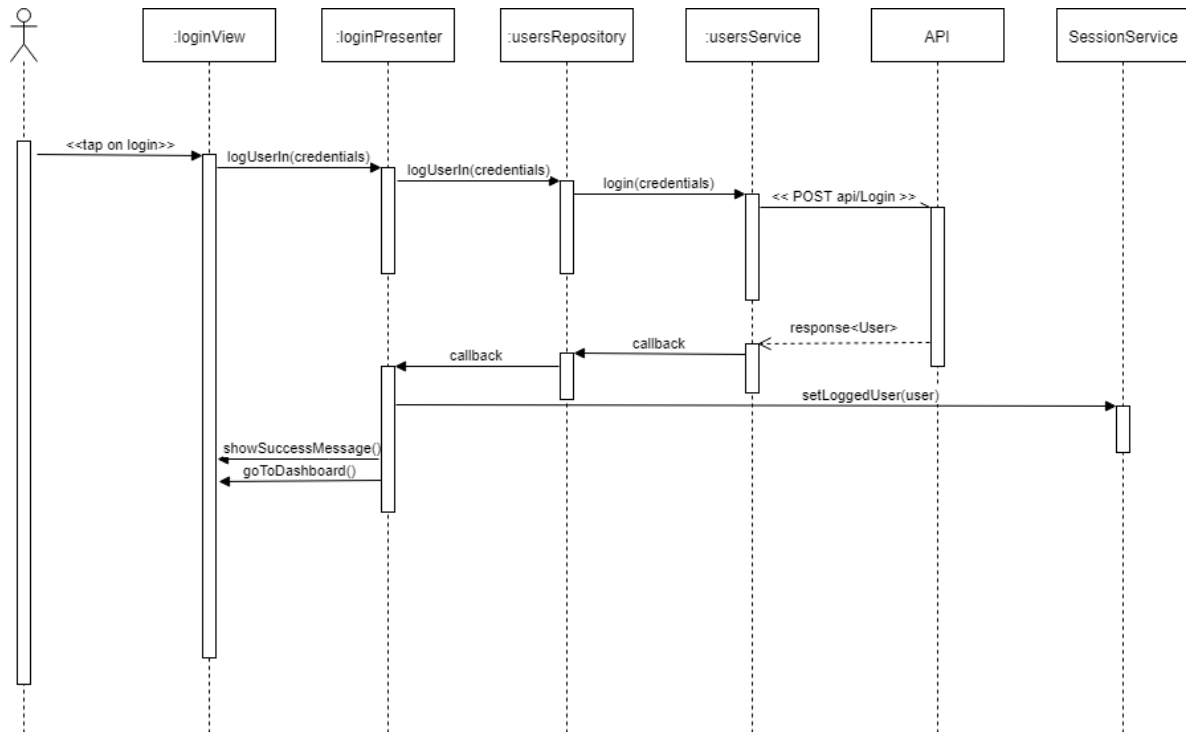
El siguiente es un ejemplo de un flujo normal en la aplicación:



Cuando se inicializa la aplicación se muestra el LoginActivity, orquestrado por el LoginPresenter, que luego de realizado el login exitoso, carga los datos del usuario en el SessionService.

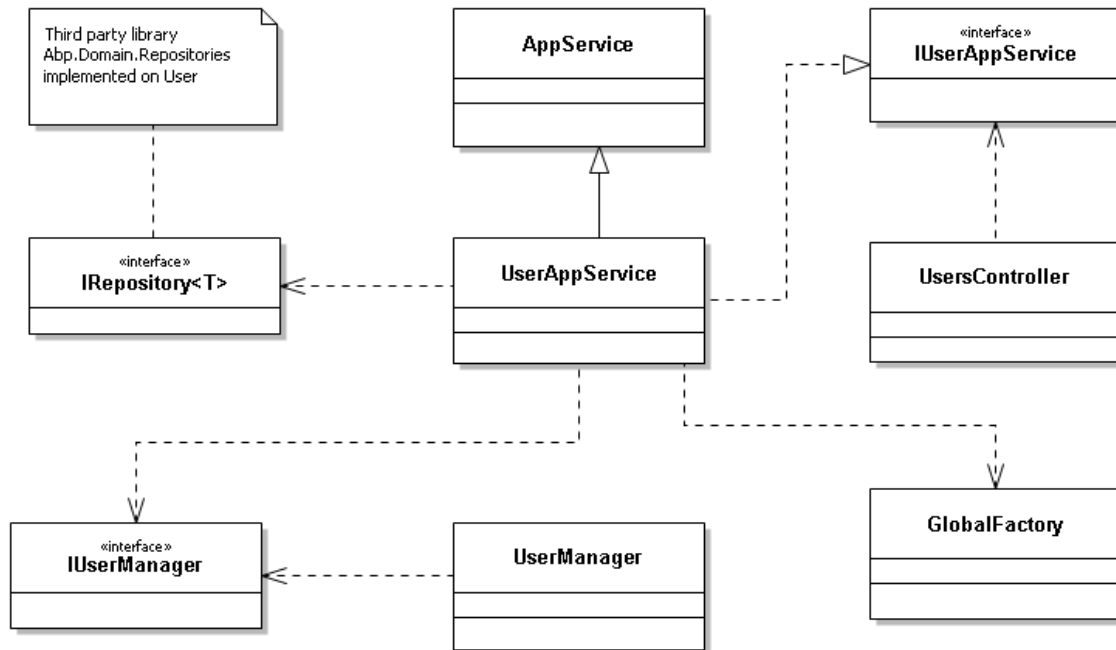
Luego, se realiza una request a la api para popular la vista de damage reports y se crea una clase DamageReportViewFactory, que cuando el usuario hace click en un daño, ésta utiliza el SessionService para definir según el tipo de usuario y el estado de damage report, que activity debe ser renderizada para mostrar la vista correcta.

Adjuntamos un diagrama de interacción para demostrar un proceso que demuestra el funcionamiento estándar de todo el Front End:



7.3.2 ALTA DE USUARIOS

7.3.2.1 BACK END



Básicamente la implementación es una clásica utilización del framework boilerplate para creación de usuarios. UsersController recibe por inyección de dependencias el UserAppService, al cual le envía los pedidos de creación de usuarios que luego gestiona con la ayuda del repositorio de usuarios y el UserManager (capa de lógica). Un punto a destacar es que, dependiendo del rol recibido en la request, se levanta por reflection (utilizando la Global Factory) una instancia de la clase User (que puede ser Admin, Crew o User) y se guarda. De esta manera evitamos RTTI y se asigna dinámicamente el rol.

El método que realiza esta instanciación es el siguiente:

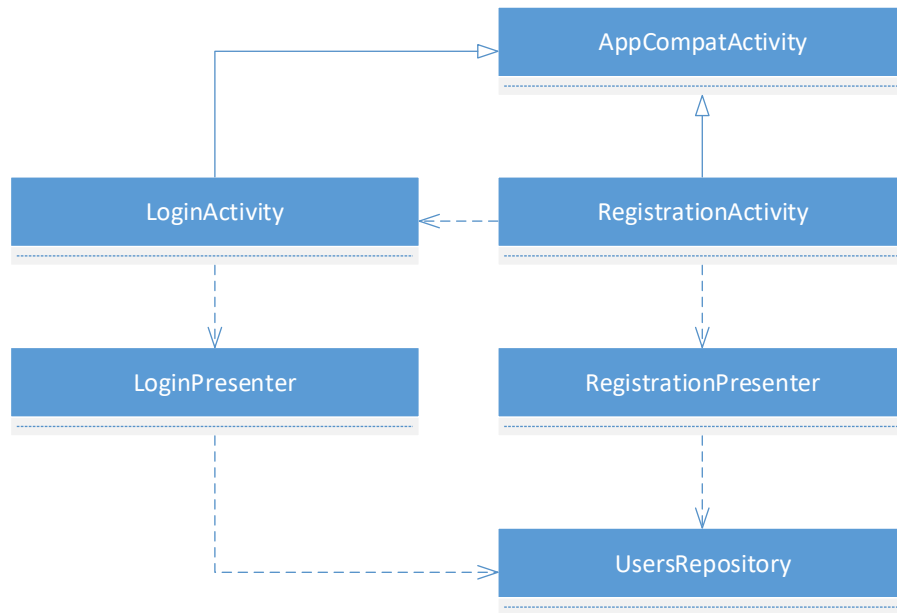
```

private static object CreateInstance(Assembly assembly, string className)
{
    var type = assembly.GetType().First(t => t.Name == className);

    return Activator.CreateInstance(type);
}

```

7.3.2.2 FRONT END

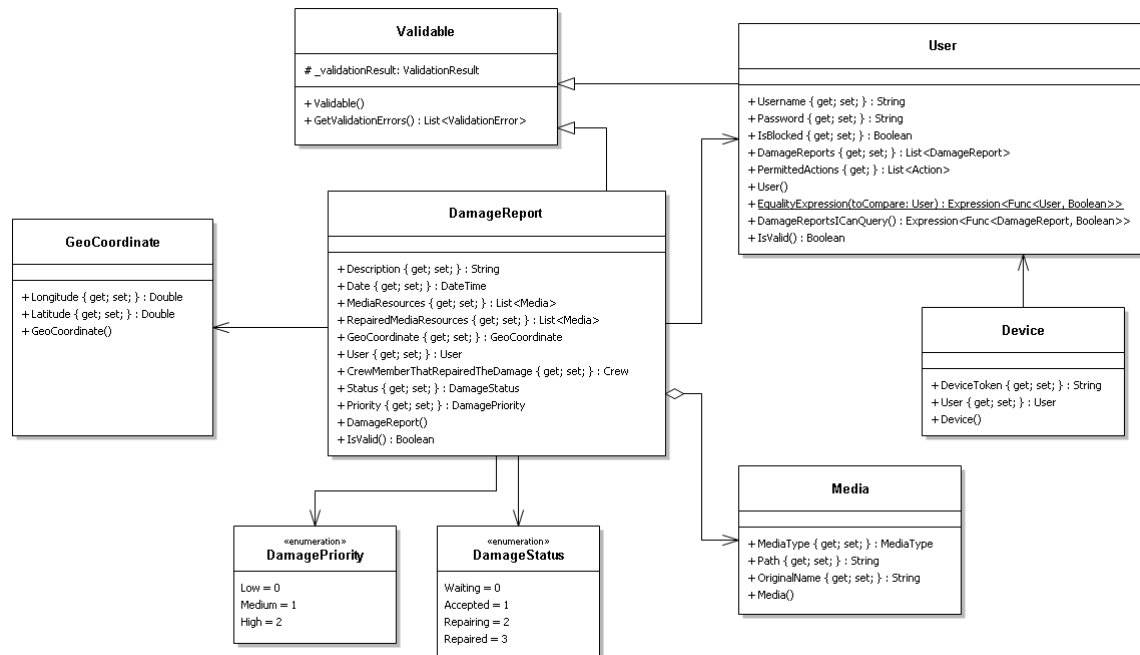


Luego de realizada el registro, siguiendo la arquitectura descrita previamente, se envía a la pantalla de login.

7.3.3 GESTIÓN DE REPORTES DE DAÑOS

7.3.3.1 BACK END

En esta sección es pertinente hablar sobre todo el diseño de las relaciones de las entidades del dominio ya que todas rodean a esta funcionalidad que es la principal y el objetivo de la aplicación.



El diagrama es explicativo, creemos que lo que vale aclarar es que los devices son los dispositivos Android conectados, de los cuales necesitamos el DeviceToken para poder enviarles push notifications (usando Firebase).

El registro en particular y las modificaciones siguen el mismo modelo arquitectónico que el resto de las features, respetando las capas y las responsabilidades de cada capa.

7.3.4 OTROS

7.3.4.1 MANEJO GLOBAL DE EXCEPCIONES EN API

Registramos en la api una clase que se llama `ExceptionHandlerAttribute` que recibe todas las excepciones que se arrojan en los controllers o en capas inferiores y las traduce a respuestas HTTP. Esto nos permite prescindir de try-catch repetitivos en todos los métodos de los controllers, facilitando la lectura del código.

7.3.4.2 MANEJO DE MULTIMEDIA EN API

8. GESTIÓN DE LA CONFIGURACIÓN

8.1 CONTROL DE CÓDIGO

Para el control del versionado se usará git como herramienta con repositorios remotos en Github. Se tendrán dos repositorios diferentes, uno para la aplicación de Android y otro para el back end del cual esta consumirá. Esto permite tener un control muy granular de que cambios se han hecho en el código y saber en qué estado se encuentra cada uno mirando los commits que componen el log dentro del repositorio.

Asimismo, se usarán tags a la hora de construir release para marcar las diferentes versiones estables de la aplicación en caso de que sea necesario volver a una de estas por cualquier error.

Se ha invitado al docente a nuestros dos repositorios:

<https://github.com/mariosouto/tla-api/>

<https://github.com/mariosouto/tla-android/>

6.2 VERSIONADO

El versionado se hará de acuerdo con el estándar establecido en <https://semver.org/>. Por lo que esperamos marcar para cada release una versión menor y en caso de necesitar hacer hotfixes un patch. De esta forma se puede tener un control del orden de los releases así como de cuál es la compatibilidad entre un release y el siguiente. Asimismo, que sea una notación estandarizada provee una manera sencilla de que una persona externa pueda entender el sistema con una curva de aprendizaje relativamente baja.

8.2 ARTEFACTOS

Los artefactos serán contruidos con el proceso por defecto de la IDE de desarrollo, Android Studio, y serán almacenados de forma local ya que no entra en el alcance de este proyecto poder descargar un ejecutable directamente desde una tienda de aplicaciones.

El manejo de ambientes dentro de la aplicación móvil no está determinado debido a la carencia de conocimiento sobre el método de construcción y por lo tanto será determinado cuando el momento sea más oportuno.

8.3 REPORTE DE DEFECTOS

El reporte de defectos se hará en la herramienta Trello, en el tablero Kanban usado por el equipo. Estos defectos se diferenciarán con un tag con el fin de no ser confundidos con las funcionalidades a implementar. A su vez, estos serán arreglados en el tiempo sobrante dentro de las iteraciones luego de haber hecho el release correspondiente.

Estos defectos a su vez serán estimados de la misma manera que se hace con las funcionalidades a modo de tener una manera de elegir aquellos que pueden ser atacados con respecto al tiempo del que se disponga. También serán priorizados de acuerdo con su gravedad en “Baja”, “Media” y “Alta” para poder así distinguir aquellos que deben ser solucionados antes

9. ASEGURAMIENTO DE CALIDAD

9.1 INTRODUCCIÓN

Cada acción que realizaremos requiere esfuerzo humano (horas). Como el tiempo disponible para invertir en la realización del proyecto es finito, debemos utilizar un conjunto de métricas que nos permitan visualizar el estado actual en el que se encuentra el software en cada etapa del proceso de desarrollo para saber si estaría avanzando de la manera que nos gustaría a lo largo del mismo. Con esto en mente, podremos enfrentar mejor la situación y los escenarios que nos encontraremos.

9.2 PLAN DE MÉTRICAS

9.2.1 INTRODUCCIÓN

Utilizaremos un Enfoque GQM (Goal - Question - Metric) el cual es un paradigma para desarrollar y mantener un programa de métricas que ayudan a:

- Alinear las métricas con los negocios de la organización y las metas técnicas.
- Mejorar el proceso del software
- Gerenciar el riesgo
- Mejorar la calidad del producto (QIP)

Proporciona una manera útil para definir mediciones tanto del proceso como de los resultados de un proyecto. Considera que un programa de medición puede ser más satisfactorio si es diseñado teniendo en mente las metas (objetivo perseguido). Las preguntas ayudarán a medir si se está alcanzando en forma exitosa la meta definida por esta razón se considerarán preguntas potencialmente medibles.

Puede incluso ser utilizada por los miembros individuales de un equipo de proyecto para enfocar su trabajo y para determinar su progreso hacia la realización de sus metas específicas

Se divide en 3 partes:

1. Fijarse un objetivo de la medición (comprender, controlar, predecir o mejorar)

2. Hacerse preguntas para alcanzar los objetivos
3. Establecer las métricas contesten las preguntas

9.2.2 PLANIFICACIÓN

La Planificación GQM supervisa la implementación de GQM dentro del contexto del proyecto. Para esto identificamos dos áreas de enfoque para nuestras métricas:

Del producto o proceso:

- Mantener una interfaz limpia siguiendo criterios conocidos de usabilidad.

Basada en los objetivos de negocio (costos, tiempo, riesgos, calidad):

- Mantener una calidad de código alta.

9.2.3 DEFINICIÓN

Calidad de la GUI	
Analizar	La interfaz
Con el propósito de	Mejorar la usabilidad
Con respecto a	Las Heurísticas de Nielsen
Desde el punto de vista de	Los usuarios finales de la aplicación
En el contexto de	En la aplicación de reportes de daño, en particular su ejecución
Preguntas	
Pregunta 1	¿El software cumple con los requisitos de usabilidad?
Métricas	
Pregunta 1	Heurísticas de Nielsen

Calidad de código	
Analizar	La calidad del código
Con el propósito de	Poder mejorarla para aumentar la velocidad y facilidad de desarrollo de funcionalidades nuevas y mantenimiento de las existentes.
Con respecto a	Los estándares y métricas mundiales de calidad de código.
Desde el punto de vista de	Los desarrolladores
En el contexto de	La realización de cambios en el software en cuestión.
Preguntas	
Pregunta 1	¿Qué tanto cumple los estándares el código?
Pregunta 2	¿Las pruebas unitarias son correctas?
Métricas	
Pregunta 1	Estándar de .NET y estándar de codificación de Kotlin
Pregunta 2	Cobertura de pruebas con Visual Studio

9.3 PLAN DE ASEGURAMIENTO DE LA CALIDAD

Durante el proceso se realizará un chequeo de las métricas para corroborar que se esté cumpliendo con los objetivos de calidad propuestos. Estos chequeos se realizarán antes de hacer un release con herramientas y procesos definidos en las secciones que siguen a continuación.

9.4 ANÁLISIS DE USABILIDAD

9.4.1 INTRODUCCION

Estaremos utilizando las heurísticas de Nielsen para analizar la usabilidad del software a mantener. Para poder ver la evolución de estas se harán cuestionarios conteniendo todas las heurísticas.

Estos deberán ser completados por todos los integrantes del equipo asignando un valor del 1 al 5 a cada heurística indicando el grado en el que se cumple. Asimismo, se deberá incluir una breve descripción de las razones por las cuales se decidió asignar el mismo.

Para obtener un puntaje total se hará un promedio por métrica del puntaje de todos los estudiantes.

Nos basamos en un artículo publicado por el mismo Jakob Nielsen en el que menciona que es más performante la realización de una evaluación heurística personal para luego compararla con la de otros especialistas para mejorar los resultados, que hacer una conjunta con todos los integrantes del equipo.

9.4.2 RESULTADO ESPERADO

Se espera que la mayoría de las heurísticas se mantengan con un puntaje mayor a 3 durante todo el proceso, pudiendo haber algunas que decaigan por poca aplicabilidad a la plataforma o al proyecto.

9.5 ANÁLISIS DE CALIDAD DE CÓDIGO

9.5.1 GUIAS DE CODIFICACION

La calidad de código se medirá con respecto a las infracciones hechas a la guía de estilos correspondiente al lenguaje escogido.

Para Kotlin se usará la herramienta Spellchecker integrado con Android Studio y siguiendo la guía de estilos establecida.

Para el back-end se realizará un proceso parecido usando Re-Sharper como herramienta de análisis estático de código también configurado con la guía de estilos seleccionada previamente.

9.5.1.1 RESULTADO ESPERADO

Debido a que las IDEs de desarrollo usadas proveen una ayuda en tiempo de desarrollo sobre las infracciones que se están realizando, las mismas pueden ser corregidas de manera eficaz durante la codificación. Por lo tanto, se espera que el número de infracciones sea bastante bajo a la hora de realizar las mediciones. En concreto, la suma de infracciones para cada proyecto no debería superar las 50.

9.5.2 CALIDAD DE LAS PRUEBAS UNITARIAS

Se realizarán pruebas unitarias en el backend y solo en el backend debido a la complejidad que las pruebas tienen cuando se introduce el elemento de la interfaz de usuario. Para medir las mismas se usará la métrica de cobertura de código. La misma será tomada usando la funcionalidad de Visual Studio para proveer estos valores. Se tomará como representativo de la calidad el paquete de lógica de negocio ya que es donde se encuentra la parte más importante de la aplicación y la que está sometida a más cambios.

9.5.2.1 RESULTADO ESPERADO

Se espera que la cobertura de código esté por encima del 75% en todas las releases para el paquete de lógica de negocio.

9.6 CALIDAD DEL PRODUCTO

9.6.1 REVISIONES DE CÓDIGO

La revisión de código es una práctica que permite la detección temprana de defectos en el código y que mejora la legibilidad del mismo ya que obliga a que una persona, aparte del autor, sea capaz de comprenderlo.

Se trabajará con Pull Requests en GitHub de forma de que cada funcionalidad a implementar sea revisada y aprobada por al menos una persona. De ser necesarios cambios, los mismos se especificarán en la Pull Request para que sean corregidos.

9.6.2 CONVENCIONES Y USO DE ESTÁNDARES

Para el mantenimiento de la consistencia del código se seguirán las siguientes guías y buenas prácticas de fuentes actualmente reconocidas en la industria.

Guía de Kotlin oficial de Android

<https://android.github.io/kotlin-guides/style.html>

Guía de C# de la herramienta ReSharper de JetBrains

La misma es una herramienta automática que realiza análisis de código, recomendando posibles mejoras y asegurándose que el código fuente siga la guía de estilo de C#.

Guía de Best Practices de Thoughtbot

<https://github.com/thoughtbot/guides/tree/master/best-practices>

Guía de REST Apis de Api Gee

<https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf>

Así mismo se establece entre los integrantes que el código base tanto de la API Rest como de la aplicación mobile esté enteramente escrito en inglés.

9.6.3 CÓDIGO LIMPIO

Las revisiones de código se realizarán tomando en cuenta los principios vistos en el libro Clean Code de Robert Martin. Se hará especial énfasis en:

- Nombres de variables expresivos.
- Funciones que tengan solamente una razón por la que cambiar.
- Clases que solamente tengan una razón por la que cambiar.
- No más de 3 parámetros en funciones.
- Funciones cortas.
- Nombres de funciones expresivos.
- No utilizar comentarios. Escribir el código lo suficientemente claro para no tener que comentarlo.

9.6.4 USABILIDAD

Dada la importancia de la experiencia de usuario en una aplicación mobile se realizan bocetos de interfaz de usuario siguiendo las heurísticas de Nielsen y las prácticas establecidas en la guía Material Design de Google (<https://material.io/guidelines>)

9.7 RESULTADO DE CALIDAD

9.7.1 COBERTURA DEL CÓDIGO

El porcentaje de cobertura de las pruebas en la capa de aplicación es de 78%. Fue la capa que elegimos para testear porque utilizando Xunit junto con el diseño arquitectónico que nos brinda la utilización de entity framework, testear la capa de aplicación (orquestrador de lógica y repositorios) nos permite tener la visión mas acertada de que la funcionalidad cumple con lo esperado.

Podemos manipular directamente el contexto mockeado que está utilizando y analizar su estado una vez finalizada la prueba.

Vale aclarar que cumple los objetivos planteados en la etapa de análisis, y no es mayor en este caso ya que no se han realizado pruebas automáticas para la subida de imágenes ya que está muy acoplado al file system el código.

TeLoArreglo.Application	78%	90/403
TeLoArreglo.Application.Media	0%	21/21
TeLoArreglo.Application.Dtos.Device	63%	3/8
TeLoArreglo.Application.Dtos.DamageReport	67%	14/42
TeLoArreglo.Application.Users	78%	17/77
TeLoArreglo.Application.DamageReports	82%	29/157
TeLoArreglo.Application.Dtos.User	83%	4/24
TeLoArreglo.Application	86%	1/7
TeLoArreglo.Application.Devices	95%	1/21
TeLoArreglo.Application.Dtos	100%	0/46

Ejemplo de prueba para que se entienda de lo que hablamos:

```
[Fact]
public void UserAppService_Login_Successful()
{
    User user = UserFactory.NewUser();

    UsingDbContext(context => context.Users.Add(user));

    string token = _userAppService.Login(
        new UserLoginDto
        {
            Username = user.Username,
            Password = user.Password
        }).Token;

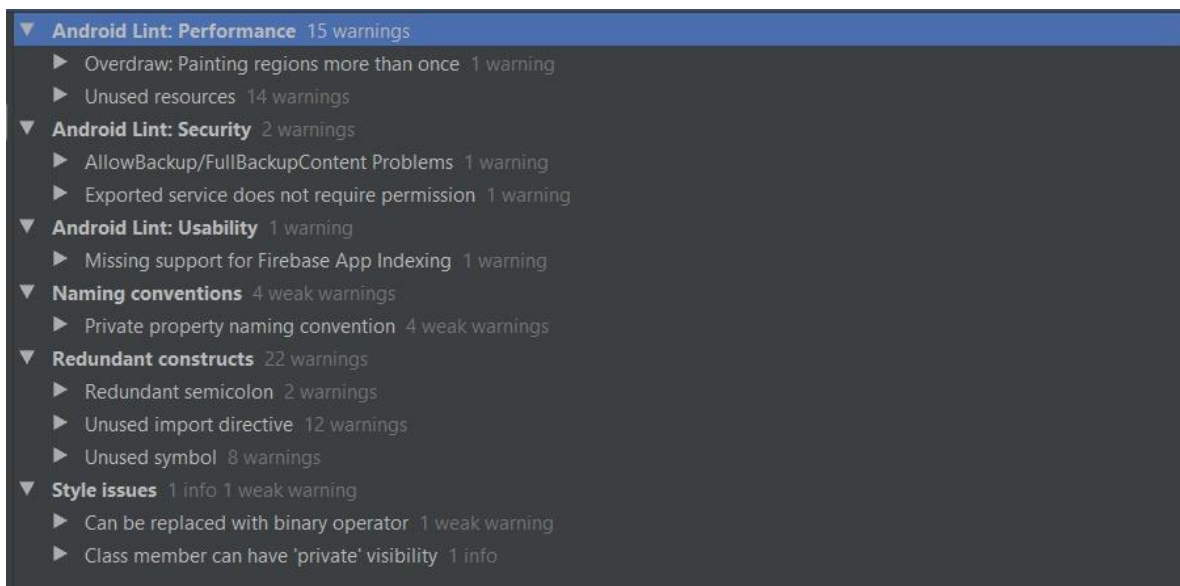
    Assert.False(token.IsNullOrEmpty());

    UsingDbContext(context =>
    {
        Assert.Equal(1, context.Sessions.Count());
    });
}
```

9.7.2 CONVENCIONES Y USO DE ESTÁNDARES

Lista de errores en las convenciones de código:

- 45 violaciones obtenidas (detectadas por la IDE).



9.7.3 CONCLUSION

El enfoque GQM no nos fue efectivo porque los tests y los análisis de estilos fueron hechos en las etapas finales del proceso de desarrollo por descuidos del equipo. A pesar de esto podemos estar conformes con los resultados porque cumplen los parámetros establecidos en la planificación.

9.8 CALIDAD DEL PROCESO

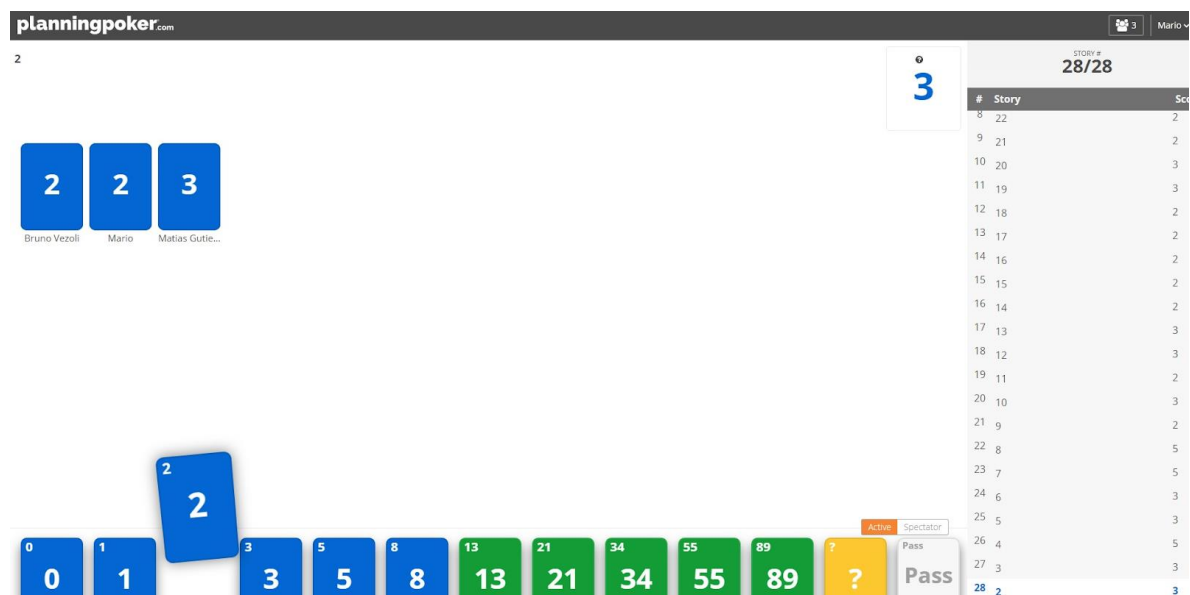
9.8.1 USER STORIES EN FORMATO BDD

Se redactó las historias de usuario en el formato recomendado por BDD. Para cada historia de Usuario se identifica un rol, un evento y el beneficio de la historia de usuario. El uso de este estándar facilitó la comprensión de las historias para todos los miembros del equipo y permite mantener las historias enfocadas, de forma de evitar escribir historias extremadamente complejas. Así mismo, este formato facilita la planificación de los releases, ya que permite visualizar fácilmente la separación conceptual de cada user story y planificar los releases de forma que la aplicación tenga un sentido como producto.

9.8.2 ESTIMACIÓN CON PLANNING POKER

Se estimó las historias de usuario utilizando Planning Poker entre los integrantes del equipo, de forma de obtener un estimativo para cada historia basado en el promedio del puntaje dado por cada integrante.

Se siguió la escala basada en Fibonacci (1, 2, 3, 5, 8, etc) donde el valor es la complejidad de la historia de usuario.



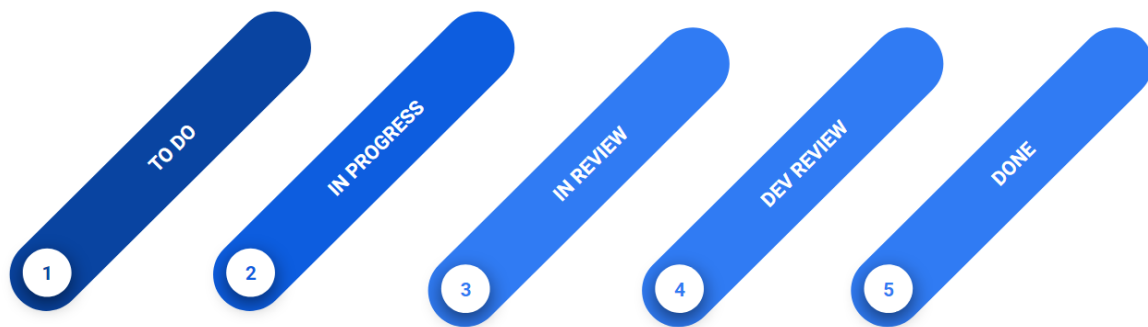
9.8.3 TRACKEO DE HORAS

Utilizaremos la aplicación Toggl para el trackeo de horas cada vez que se realice una actividad relacionada con el proyecto. De esta forma se podrán detectar riesgos de burnout de un integrante del equipo, que es uno de los riesgos detectados en el informe de gestión de riesgos. Se puede visualizar si hay un miembro del equipo que esté dedicando mayor cantidad de horas y así redistribuir mejor el trabajo.

El trackeo de horas también nos ayuda a tomar una dimensión real en tiempo de la complejidad de las historias de usuario.

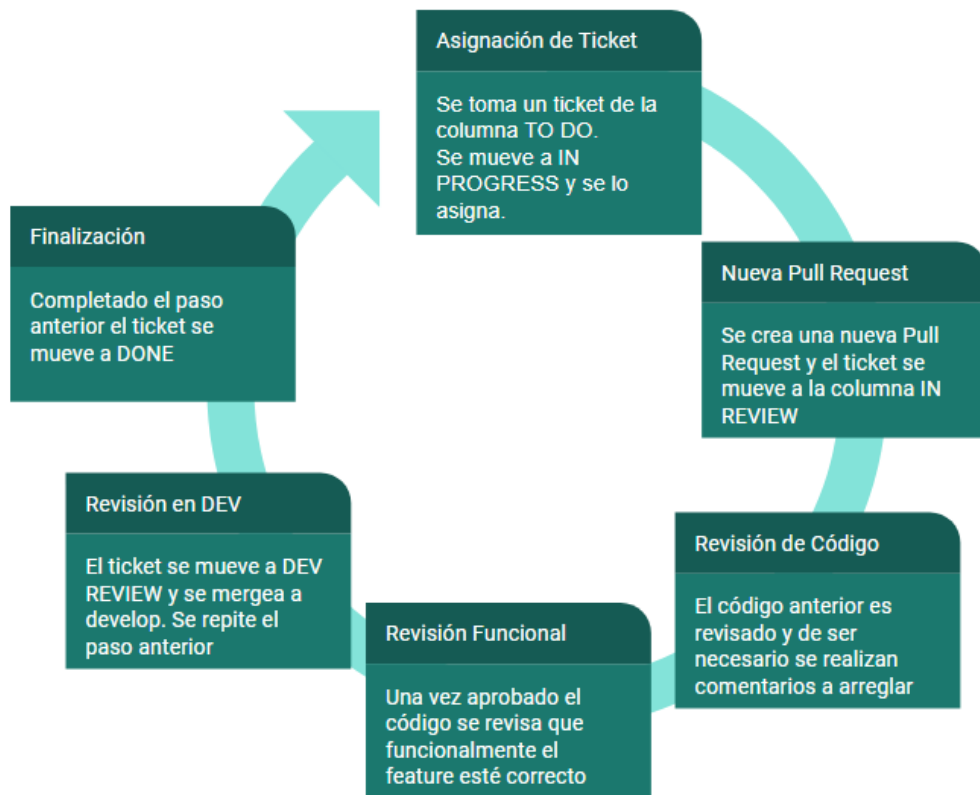
Como vimos que no éramos un equipo prolijo con el seguimiento de las horas decidimos abandonar la idea de Toggl y enfocarnos a la correctitud del burnout chart, que a nuestro entender refleja el avance con la suficiente precisión como para sacar las conclusiones que necesitamos.

9.8.4 FLUJO DE TRABAJO EN TRELLO



Se definen en Trello 5 columnas para la utilización de Bugs, Hotfixes y User Stories

- **TO DO**, significa que el ticket no está asignado y está listo para ser desarrollado.
- **IN PROGRESS**, significa que el ticket está asignado y siendo desarrollado por otro integrante.
- **IN REVIEW**, el ticket está siendo evaluado, mediante revisión de código o pruebas de QA.
- **DEV REVIEW**, el ticket se encuentra mergeado en la branch **develop** y será testeado con la demás funcionalidad.
- **DONE**, el ticket fue finalizado correctamente.

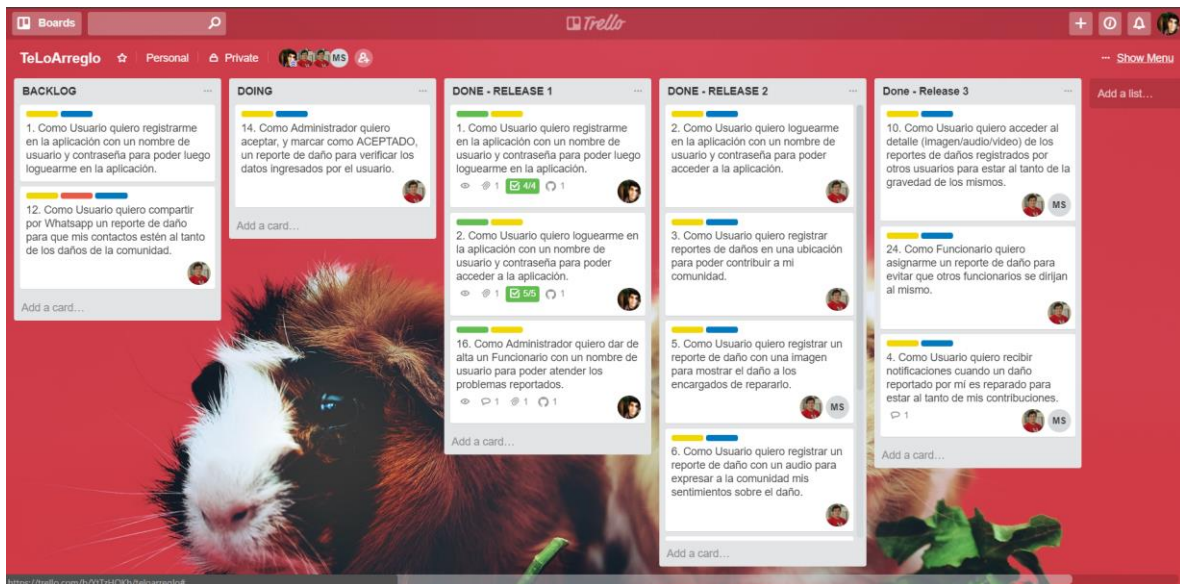


A continuación se agrega el flujo de trabajo desde que se asigna un ticket hasta que el mismo es finalizado y mergeado a la branch **develop**.

Luego con la funcionalidad realizada se hace un release a **master** tal como lo indica el flujo de GitFlow.

De encontrarse defectos (bugs) se crean los tickets correspondientes.

Captura del Trello en funcionamiento:



9.8.5 GESTIÓN DE LA CONFIGURACIÓN

Uso de Git y Gitflow, y branches protegidas y reviews requeridos y etc

Se utilizará Git bajo las prácticas recomendadas por GitFlow

<https://datasift.github.io/gitflow/IntroducingGitFlow.html>

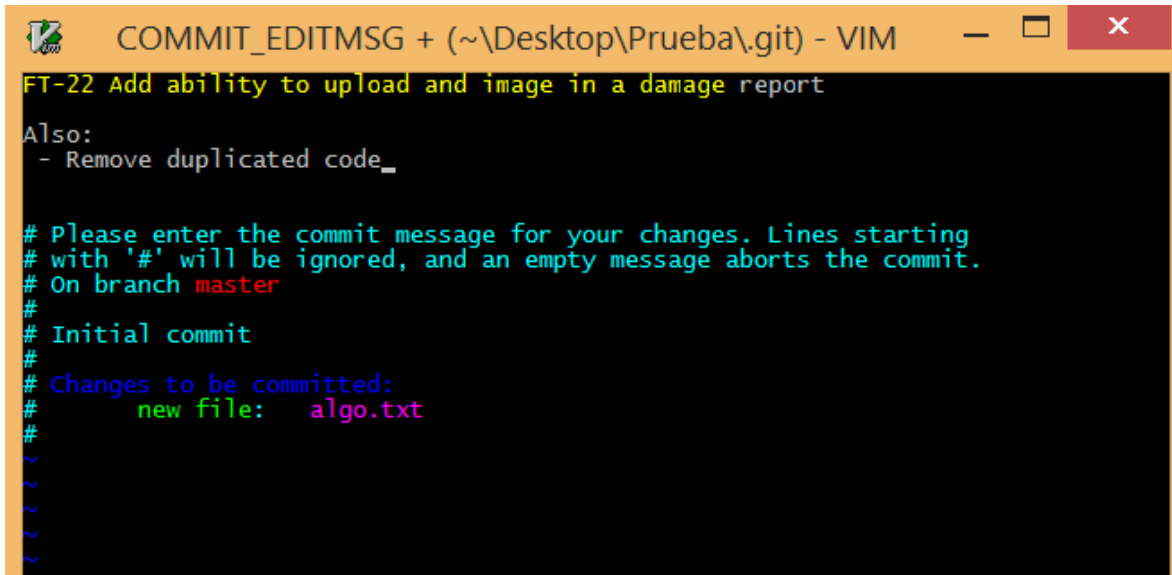
Así mismo se utilizará GitHub con las branches **master** y **develop** protegidas de forma de que no esté permitido el push directo a estas branches y sea necesario crear una Pull Request para cada funcionalidad.

Se definió un estándar propio para los mensajes de Commit.

[Tipo de Ticket]-[Número]

Resumen del commit

Detalles adicionales del commit



```

COMMIT_EDITMSG + (~\Desktop\Prueba\.git) - VIM
FT-22 Add ability to upload and image in a damage report
Also:
- Remove duplicated code_

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   algo.txt
#
~
~
~
~

```

Donde distinguimos tres tipos de ticket

- Features, que van con la sigla FT
- Bugs, que van con la sigla BUG
- Hotfixes, que van con la sigla HT

Y el correspondiente número de ticket especificado en Trello.

Luego se indica el resumen con lo más importante del cambio y, si es necesario, detalles explicando los restantes cambios.

Ejemplo de nuestra aplicación de GitFlow:



9.8.6 ORGANIZACIÓN DE RELEASES

Los releases se organizan de forma de que cada release contenga un conjunto de historias de usuario que le genere un valor real a la aplicación. Es decir, que la funcionalidad entregada tenga sentido como producto (Minimum Viable Product) y que la aplicación se pueda utilizar desde el primer release.

9.8.7 RETROSPECTIVAS

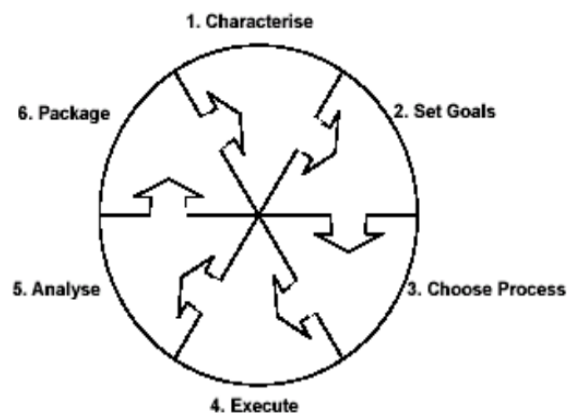
Se realizarán retrospectivas para inspeccionar el proceso de desarrollo y detectar posibles mejoras y/o correcciones que permitan una mayor eficiencia en el trabajo de equipo. Cada Retrospectiva se realizará al final de cada release, excepto en el release final.

9.9 DEFINICIÓN DE UN MODELO

Hemos utilizado el paradigma de mejora de la calidad (QIP)

QIP es una aproximación a la calidad, que enfatiza la mejora continua por medio del aprendizaje de la experiencia dentro de los proyectos y de la organización (Basili 1994).

Consiste en un ciclo de seis pasos, que está basado en el Plan-Do-Check-Act.



9.10 ANÁLISIS DE USABILIDAD

9.10.1 HEURÍSTICAS DE NIELSEN

9.10.1.1 VISIBILIDAD DEL ESTADO DEL SISTEMA

Se usarán alertas en caso de errores para hacer saber al usuario de cualquier cambio en el estado del sistema, ya sea por causas externas o por acciones realizadas por el mismo. De esta forma se puede tener una respuesta instantánea hacia el usuario al encontrar un problema.

9.10.1.2 UTILIZAR EL LENGUAJE DE LOS USUARIOS

La interfaz se basa mucho en iconos conocidos para los usuarios, por ejemplo, el icono de cámara para agregar una foto, lo cual resulta muy natural para un usuario acostumbrado a usar dispositivos móviles.

Se usan frases explicativas en el título de las pantallas para explicar el contexto en que se encuentra al usuario

9.10.1.3 CONTROL Y LIBERTAD PARA EL USUARIO

Los dispositivos Android proveen un botón (físico o virtual en algunos casos) con la opción de volver hacia atrás, este botón permitiría deshacer cualquier opción que el usuario haya realizado por equivocación para que pueda volver al estado anterior

9.10.1.4 CONSISTENCIA Y ESTÁNDARES

Se usan convenciones seguidas a través de otras aplicaciones móviles, como el signo de más para agregar ítems en el mapa, para que el usuario continúe en el mismo entorno al usar la aplicación y no tenga que familiarizarse desde cero con la misma

9.10.1.5 PREVENCIÓN DE ERRORES

Se manejan las acciones que puede realizar el usuario con sus permisos desde el principio para así evitar que tenga funcionalidades en las que se le tenga que mostrar un error de no autorizado o algo similar. Un ejemplo de esto es la pantalla de inicio donde se muestran solo aquellos reportes de daños con los que el usuario puede interactuar.

9.10.1.6 MINIMIZAR LA CARGA DE LA MEMORIA DEL USUARIO

Como se puede ver en las pantallas, se sigue el mismo patrón que otras aplicaciones usan por lo tanto no se sobrecarga al usuario con lo que tiene que aprender. Asimismo, se proveen etiquetas en todos los campos a rellenar para que el usuario sepa que información es requerida en el mismo.

9.10.1.7 FLEXIBILIDAD Y EFICIENCIA DE USO

No se provee personalización de acciones frecuentes, pero se puede ver que cualquier acción que se quiera realizar no está a más de tres pasos, por lo tanto, no debería ser necesario disminuir todavía más el acceso a las mismas.

9.10.1.8 DIÁLOGOS ESTÉTICOS Y DISEÑO MINIMALISTA

El diseño es muy minimalista, se muestra solo la mínima información necesaria para lo que el usuario desea hacer en el contexto en el que se encuentra.

9.10.1.9 AYUDAR A LOS USUARIOS A RECONOCER, DIAGNOSTICAR Y RECUPERARSE DE LOS ERRORES

Los mensajes de error incluirán una descripción detallada de que fue lo que salió mal, con esto el usuario debería ser capaz de corregirlos y volver a intentar. Un ejemplo son los formularios, donde ante un campo erróneo se mostrará el nombre de este campo y por qué se lo considera de esta forma

9.10.1.10 AYUDA Y DOCUMENTACIÓN

No se cumple esta heurística ya que no se considera parte del alcance de esta entrega.

10. REFERENCIAS

- Sommerville, Ian. (2013). Software Engineering, (9th ed.)
- Pressman, Roger S. (2009). Software Engineering: A Practitioner's Approach, (7th ed.)
- Article "How to Conduct a Heuristic Evaluation" by JAKOB NIELSEN on January 1, 1995
- <https://medium.com/@cervonefrancesco/model-view-presenter-android-guidelines-94970b430ddf>