Trabajo de Fin de Grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas

Universidad de Barcelona

APPQUEOLOGY 2.0

Fernando Boullosa García

Director: Oriol Pujol

Resumen:

La aplicación Android AppQueology 2.0 es el resultado de quince semanas de trabajo enmarcadas dentro de un Proyecto de Fin de Grado de Ingeniería Informática de la Universidad de Barcelona.

Realizada en colaboración con investigadores de la Facultad de Historia de la UB, es una aplicación diseñada por y para arqueólogos que pretende ayudar en la gestión y documentación de los diferentes procesos que conforman una excavación arqueológica.

La elaboración de este proyecto ha puesto de relieve la falta de software destinado a estos menesteres y la necesidad de acercar las diferentes posibilidades que ofrecen los dispositivos móviles a esta disciplina.

Palabras clave:

Excavación, Arqueología, Android, GPS, fotos, unidades estratigráficas, relaciones, estratos, negativas, estructuras, grafo, diario excavación, Ingeniería Informática, Historia, Universidad de Barcelona, cronoestratigraf, …

Abstract:

The Android application AppQueology 2.0 is the result of fifteen weeks of work framed within a project by end of Degree in Computer Engineering from the University of Barcelona.

The application has been developed in collaboration with researchers from the Faculty of History at UB, this is an application designed by archaeologists for themselves which aims to help in the management and documentation of the different processes than make up an archaeological dig.

The development of this project has highlighted about the lack of software aimed at these purposes and the need to bring the different possibilities that smartphones offer to this discipline.

Keywords:

Excavation, Archeology, Android, GPS, potos, stratigraphic units, relations, strata, negative, structures, graph, excavation diary, Computer Engineering, History, University of Barcelona, cronoestratigraf, …

Estructura de la Memoria:

En este apartado se especificará, como su título indica, la estructura que plantea este breve trabajo o memoria.

El objetivo de dicho punto es acercar al lector las diferentes partes que componen este proyecto, así como ayudarle en la comprensión de este.

Se espera que sea lo suficientemente clarificador y que permita una lectura comprensible, rápida y amena del trabajo que nos ocupa.

Además de los puntos iniciales de esta memoria que tratan de acercar al lector el tema a tratar, se ha decido crear los puntos de Análisis, Diseño, Implementación y Pruebas. Ya que estos siguen más o menos el ciclo de vida de una aplicación y se entendía que era necesario tratarlos como puntos individualizados dentro de este breve proyecto.

Por último se han desarrollado dos puntos que tratarán aspectos relacionados con la posible ampliación de esta aplicación en un momento futuro y siguiendo en colaboración con la Facultad de Historia de la Universidad de Barcelona y las conclusiones obtenidas tras la realización de este Proyecto de Fin de Grado.

Comentar que en el Índice de contenidos cada apartado ha sido enlazado con su sección correspondiente para que con un simple clic se dirija al lector al punto donde se explique ese aspecto.

Del mismo modo, se ha realizado con algunos conceptos a lo largo de la memoria con la intención de facilitar su comprensión y su lectura.

ÍNDICE DE CONTENIDOS

1. Introducción

Este apartado hará un breve resumen de que es AppQueology 2.0, comentará cuales han sido las motivaciones que me han llevado a desarrollarla, explicará que se pretende realizando este trabajo y que tempos se han sido necesarios para su realización, así como enumerará las razones de dichos objetivos.

Con él se pretende poner en contexto al lector y situarlo dentro del tema que será tratado en esta breve memoria.

1.1 Descripción

AppQuelology 2.0 es una aplicación Android pensada para facilitar la gestión de una excavación arqueológica.

Entre otras cosas, ofrece:

- La posibilidad de crear Unidades Estratigráficas con todos los parámetros necesarios para su posterior comprensión y análisis.
- Geo localizarlas y fotgrafiarlas.
- Relacionarlas entre sí dando lugar a la elaboración de la secuencia estratigráfica.
- Controlar automáticamente los posibles errores que el arqueólogo pueda ocasionar durante la elaboración de dicha secuencia.
- Diferentes tipos de representación de la secuencia estratigráfica. Por ejemplo, mediante un grafo dirigido, una tabla que asocia las Unidades Estratigráficas a partir de su contemporaneidad y no por su relaciones físicas, etc.
- Aspectos relacionados con el análisis de dicha secuencia, como puede ser su estudio a partir de fases cronológicas.
- Y mucho más…

En un momento en el que el desarrollo de aplicaciones Android ha experimentado un alto crecimiento, AppQueology 2.0 no aparece como una aplicación más, ya que ha contado para su desarrollo con la inestimable colaboración de investigadores de la Facultad de Historia de la Universidad de Barcelona. Gracias a ellos ha sido posible acotar y definir cuáles deberían ser los requisitos que AppQueology 2.0 tendría que cumplir para que realmente fuera una aplicación útil y funcional que verdaderamente agilice y facilite el trabajo de los arqueólogos durante el proceso de estudio de una excavación arqueológica. Por otro lado, y gracias a su valiosa colaboración, también ha sido posible probar la aplicación con casos reales y constatar, de este modo, su funcionalidad y utilidad.

Por todo esto, AppQueology 2.0 nace como una aplicación diseñada por arqueólogos y para arqueólogos. Que no pretende simplemente mostrar datos históricos como las muchas ya existentes en el mercado Android, sino que quiere dar un paso más y acercar los recursos computacionales y de movilidad que ofrecen los dispositivos móviles, a las necesidades que los arqueólogos se encuentran en su día a día. Así como renovar los anclados hábitos metodológicos existentes en esta disciplina, acercándola un poco más a un tiempo presente que sigue estudiando el pasado con herramientas realmente contemporáneas.

No hay que olvidar que AppQueology 2.0 se enmarca dentro de un Trabajo de Fin de Grado y más concretamente dentro de un Trabajo de Fin de Grado de la Facultad de Ingeniería Informática de la Universidad de Barcelona, por lo que a lo ya dicho anteriormente hay que sumar la desinteresada colaboración de los investigadores de dicha Facultad para que esta se adapte a las últimas tendencias existentes en el mercado.

Todo lo mencionado anteriormente ha hecho que AppQueology 2.0, siendo una aplicación novedosa que trata de solventar un problema real, no solo vaya destinada al mundo profesional, sino que aprovechando que ha sido creada desde el mundo Universitario también repercuta en él y sea para él. Por lo que ha sido pensada para que también pueda ser utilizada por los estudiantes de Historia y Arqueología que quieran acercarse un poco más a este mundo y comprender un poco mejor como las nuevas tecnologías pueden adaptarse a este campo. Así como facilitarle la comprensión y elaboración de sus trabajos y prácticas.

1.2 Motivación

En primer lugar, mencionar que este Trabajo de Fin de Grado me ha permitido conjugar dos temas que me han hecho ser lo que soy.

Para comprender esto mejor, creo que es necesario retroceder un poco en el tiempo y explicar que desde siempre me ha gustado la Historia y las historias. Cuando era niño, he tenido la suerte de tener unos tíos jóvenes que nos llevaban de excursión a todos los primos. En las noches de ríos de agua caliente y de tienda de campaña, mi tío, profesor de latín, nos contaba mitos e historias de época clásica. Esto, probablemente, despertó en mí una inquietud por estos temas que hizo que finalmente me decantara por estudiar Historia en Santiago de Compostela. Una vez terminada la carrera, decidí venirme a vivir a Barcelona para buscar trabajo y estudiar un Título Propio en Arqueología que ofrecía la Facultad de Historia de la UB. Una vez finalizados mis estudios me dediqué ya de forma profesional a la arqueología durante 7 años.

La informática y la tecnología era algo que siempre me llamara la atención, pero para un chico nacido en un pueblo de Galicia, en los ochenta, no era habitual tener, en su contexto cercano, a alguien que le despertara la inquietud por la informática. Realmente, es algo que tardó en llegar. El haberme dedicado durante unos años al dibujo de campo en arqueología, y el pasarme horas delante de un muro dibujando piedra por piedra, hizo que me preguntara si dicha tarea no podría automatizarse de algún modo a partir de fotos y la superposición de estas. Durante esa época, tuve un compañero de trabajo que tenía las mismas inquietudes que yo y empezamos a hacer nuestros primeros pinitos en estos temas. La inquietud por la informática aumentaba durante esos años y de repente el 2009 llegó y la arqueología se desmoronó al desatarse la crisis en el sector de la construcción. Tras un año pasando de empresa a empresa, con unas condiciones de trabajo más que precarias y tras alguna crisis de confianza en la objetividad Histórica y en el trabajo que estaba realizando, decidí empezar a estudiar informática, ya que la crisis parecía que había llegado para quedarse.

El poder hacer este proyecto me permite rescatar unas pasiones ya casi olvidadas después de estos 3 años y medio. Me permite creer en un sueño que he tenido desde hace tiempo, poder crear software que llevará el estudio del pasado a un momento más contemporáneo. La Historia y la Arqueología en España son unas disciplinas que casi no han bebido de la revolución tecnológica del 2000. Que no han sabido adaptarse por el momento a esta nueva tendencia. Que han permanecido ancladas en metodologías de principios del siglo pasado. Y creo que es el momento de que esto cambie.

Por otro lado, la idea de que fuera un proyecto basado en Android me gustaba por el simple hecho de trabajar con una plataforma abierta. Además era un campo que no había explorado mucho. Un campo en expansión que tenía ganas de conocer más. Había entrado en contacto con él al cursar la asignatura de Proyecto Integrado de Software durante el Grado, pero era una asignatura en grupo y yo me había dedicado a adaptar las vistas y aspectos no muy relacionados con el motor del juego que hicimos. Esta también es una razón por lo que no he priorizado estos aspectos en este proyecto, ya que realmente tenía ganas de dedicarme a otros menesteres más relacionados con el funcionamiento en sí de la aplicación.

También me parecía interesante el hecho de que el trabajo se realizara en colaboración con la Universidad de Historia, ya que esto me permitiría ensayar la relación con los clientes. Además recientemente he podido aprender métodos para la planificación y desarrollo del software, como Scrum o Kanban y creía que este proyecto era una excelente oportunidad para intentar ponerlos en práctica.

1.3 Planificación inicial y real

Comentar en cuanto a la planificación dos aspectos fundamentales:

- Tiempos y plazos de entrega.
- Como organizar el desarrollo del proyecto.

En cuanto a los tiempos y plazos decir que la idea inicial era seguir el consejo de planificación que se nos había dado en la reunión orientativa del TFG:

- 4 Semanas para documentarse y aprender.
- 7 Semanas para desarrollar el proyecto.
- 4 Semanas para la escritura de la memoria.

Pero finalmente no ha sido posible y los tempos han tenido que ir adaptándose a medida que se iba desarrollando el proyecto. Esto fue debido fundamentalmente a que la fase de obtención de los requisitos ha sido un poco más costosa, temporalmente hablando, que la pensada inicialmente. Por ejemplo, ha sido necesario dejar la fase de testeo de la aplicación para las cuatro semanas que estaban destinadas para únicamente la escritura de la memoria.

Digamos que la división final ha quedado del siguiente modo:

- 2 Semanas dedicadas a entender cómo sería el proyecto que nos ocupa.
- 2 Semanas dedicadas a dedicarme a documentarme y a aprender.
- 8 Semanas dedicadas al desarrollo del proyecto.
- 3 Semanas dedicadas a la elaboración de la memoria.

En cuanto a cómo organizar y desarrollar el proyecto, tal y como se ha dejado ya intuir anteriormente, comentar que se ha intentado seguir un modelo incremental, que se centrase inicialmente en las tareas con mayor valor de negocio pero intercalándolas, a su vez, con algunas de menor valor, más ligeras, pero también con menor coste temporal, con la intención de buscar un desarrollo lo más fluido y ágil posible.

A partir de la combinación de estas tareas mencionadas anteriormente se han ido creando diferentes versiones beta que han sido entregadas para obtener con la mayor rapidez posible un feedback del cliente y de este modo adaptar “en la mayor medida posible” la aplicación a los requisitos dados. En este punto me gustaría destacar como Scrum, recogiendo ideas del Postmodernismo, acepta que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada, y como, por tanto, adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido.

Se ha optado por crear código de forma rápida siendo conscientes que tras el feedback mencionado habría que re-factorizarlo. Pero entendiendo también que con esto se obtendría un código lo más elaborado posible ya que necesariamente tendría que ser re-implementado en sucesivas ocasiones.

Se han mantenido reuniones semanales con los clientes (investigadores de la Facultad de Historia) y con el tutor, que han permitido recoger los requisitos que debería cumplir la aplicación.

Simulando el Kanban se ha intentado visualizar el flujo de trabajo para intentar analizar cómo avanza este y detectar posibles errores organizativos.

1.4 Objetivo principal

El objetivo fundamental para la realización de este trabajo, es familiarizarme con la plataforma Android y adquirir más conocimientos sobre este campo.

De este modo, el objetivo principal de este proyecto es desarrollar una aplicación Android, con una interfaz intuitiva, que permita gestionar un yacimiento arqueológico. Con la que se puedan realizar los principales pasos en la documentación de una excavación arqueológica y que, a su vez, permita ordenar la información de tal manera que facilite el análisis posterior de esta.

Así como incluso llegar a mejora la documentación de los restos encontrados, ya que hasta el momento, por mi experiencia vivida, hay muy pocos arqueólogos que utilicen, por ejemplo, un GPS para localizar una foto que hagan sobre una Unidad Estratigráfica (UE). Cosa que es muy sencillo con un móvil.

1.5 Objetivos secundarios

Gracias a que el trabajo es en colaboración con la Facultad de Historia de la Universidad de Barcelona, aprovechar, como se había comentado, para familiarizarme con la obtención de los requisitos de una aplicación.

Así como, mediante esta fase de elaboración de la aplicación, acostumbrarme a relacionarme con el cliente, aprender a negociar, a saber marcarme los tiempos y hasta donde puedo llegar a comprometerme. Aprender a saber decir “no”.

Por otro lado, iniciarme en la organización de un proyecto de estas características, ya que no he tenido que hacer ninguno similar durante los años en los que he cursado el grado.

A su vez, ya que durante el desarrollo del proyecto se ha puesto en práctica un sistema similar al kanban, experimentar con este tipo de métodos organizativos. Al igual que ocurre con la variante aplicada de Scrum.

También intentar, por medio de una aplicación sencilla que realmente sea útil para los arqueólogos y que cubra las necesidades de estos, acercar las posibilidades que ofrece la tecnología y más concretamente la de los smartphones, a una disciplina que no ha aprovechado mucho la expansión y desarrollo de esta. Cualquiera que haya vivido el día a día de una excavación arqueológica sabe que la lluvia y la tierra no son muy amigos del papel, soporte principal de la documentación creada por un arqueólogo hasta el momento. Desarrollar esta aplicación permitirá facilitar este proceso documental. Además de ofrecer nuevos recursos.

Además, este proyecto me brinda la posibilidad de nuevamente estar en contacto con la arqueología y refrescar conceptos cada vez ya más olvidados.

Por otro lado, es una oportunidad para ver si es viable crear una aplicación que verse sobre estos temas, que sea vendible. Saber si realmente hay un nicho de usuarios lo suficientemente grande como para poder desarrollar una aplicación de este tipo y que sea rentable. Ya que siendo una aplicación para móvil y que estas no suelen alcanzar un precio alto en el mercado para su descarga y uso, y debido a que, como comentaba, el número de personas interesadas en una aplicación de arqueología es reducido, se me plantea el dilema de si podré dedicarme a ello.

Por último, consolidar los conocimientos adquiridos durante la carrera.

1.6 Razones

Como había mencionado anteriormente, solo hay que dar una vuelta por Barcelona, subirse en el metro, o dar un paseo por sus calles, para darse cuenta del creciente uso de los dispositivos móviles.

Un ejemplo de este incremento puede apreciarse en la siguiente tabla (figura 1) extraída del Instituto Nacional de Estadística:

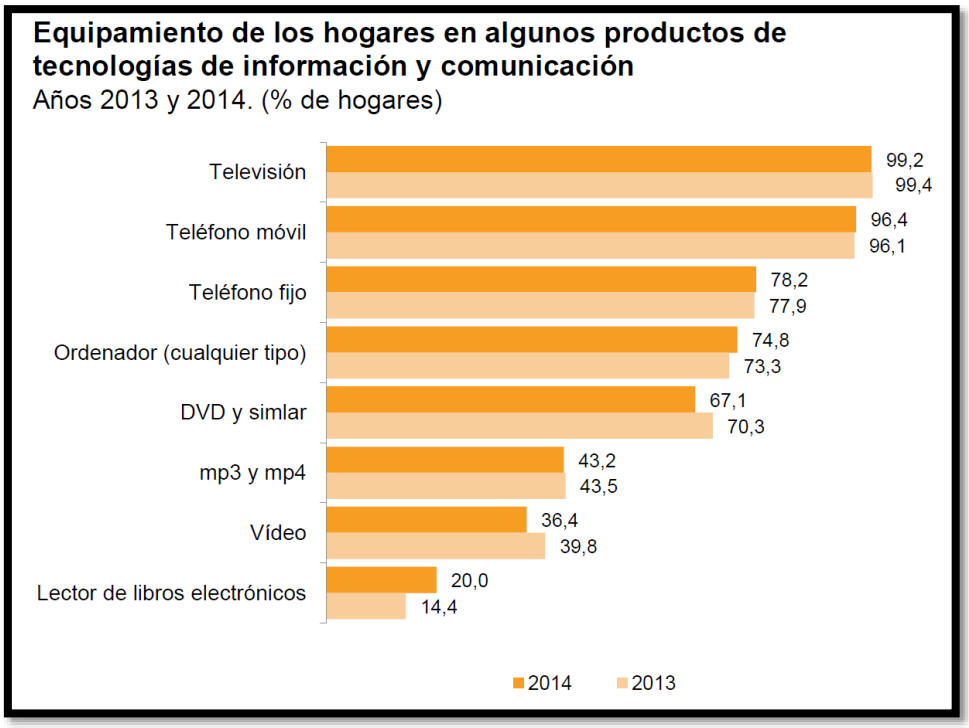

Ilustración 1 - Datos estraídos del Instituto Nacional de Estadística

Se constata, para empezar, que se produce un crecimiento del 2013 al 2014 en el uso de la telefonía móvil. Que el teléfono móvil ha superado con creces ya al teléfono fijo en los hogares españoles. Que el porcentaje de hogares que dispone de teléfonos móviles, llega casi a igualar ya, al de los que tienen televisores.

Por primera vez, según el INE, el móvil se ha convertido en el principal tipo de conexión a Internet. En 2014, un 67,2% de los hogares accede a la Red a través de su Smartphone, superando al ADSL (66,2%) y la red de cable o fibra óptica (20,9%), según la Encuesta sobre Equipamiento y Uso de Tecnologías de Información y Comunicación en los Hogares, del instituto mencionado. De estos porcentajes, que no son excluyentes, se entiende que el móvil es el dispositivo que más utilizan los hogares españoles para conectarse a internet.[1]

Además, según el mismo estudio, el móvil también es el primer dispositivo que usan los españoles para conectarse a la Red. El 81,7% de los hogares lo usa, frente al 72,2% que lo hace a través de su portátil o la tableta y el 53,5% con el ordenador de sobremesa.

Este uso creciente de teléfonos móviles, y más concretamente de los Smartphone, produjo también un fuerte crecimiento en la producción de aplicaciones Android.

Según el informe "Radiografía del mercado móvil en España," que ha sido presentado en el Mobile World Centre de Barcelona en el 2014 por IAB Spain, se ha producido un

[1] [INE14] Encuesta sobre Equipamiento y Uso de Tecnologias de Información y Comunicación en los Hogares.

espectacular incremento del uso de aplicaciones. Este estudio afirma que en el 2012, con una penetración del smarphone en un 59%, el 41% de los usuarios se conectaba a Internet a través de aplicaciones. El año 2013, con una penetración en el mercado del 80%, esa cifra alcanzaba ya el 71% de los usuarios.[2]

Esto puede ser debido, como se muestra en el informe, a la tendencia consumista por parte de los clientes de este tipo de productos.

Uso de aplicaciones:

Ilustración 2 - Muestra extraída del informe realizado por IAB SPain, 2014.

Como se puede apreciar, en la figura 2, el 71% de los usuarios propietarios de smartphones utilizan aplicaciones.

Si a esto le sumamos que, según el informe elaborado por el IDC en el 2014, android superó desde noviembre del 2013 la barrera del 80% en cuaota de mercado mundial de smarphones. Lo que supone que 4 de cada 5 teléfonos vendidos montan el sistema operativo de Google.

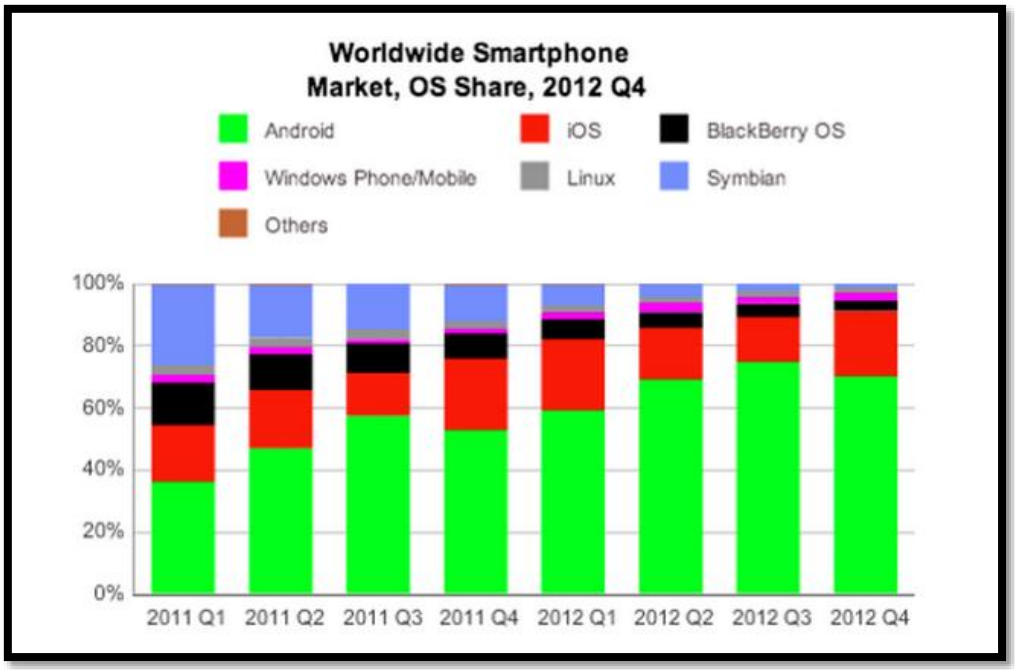

Ilustración 3 - Muestra sacada del informe del IDC publicado en el 2013 sobre el mercado de los Smartphones.

[2] [IAB13] – InteractiveAdvetising Bureau.

Por otro lado, todo esto genera un contexto propicio para que exista una fuerte demanda, dentro del mundo laboral, de desarrolladores de aplicaciones Android. Con un simple vistazo a cualquiera de los más populares portales de ofertas de trabajo se encuentran numerosas oportunidades relacionadas con este campo.

2. Conceptos teóricos acerca de la Arqueología

En este apartado nos encontraremos, como su título indica, con aspectos relacionados con cuestiones, un tanto teóricas, del campo de la Arqueología. Y que permitirán al lector familiarizarse con los diferentes conceptos que serán utilizados durante la redacción de la memoria.

De este modo, se le permitirá al lector una mejor comprensión del trabajo que nos ocupa y, se entiende, una asimilación más ágil y amena, sin la necesidad de ir buscando los diferentes conceptos que irán apareciendo y con los que probablemente no se encontrará familiarizado.

La Arqueología es una disciplina que estudia los cambios físicos que se producen desde las sociedades antiguas hasta las actuales, a través de restos materiales distribuidos en el espacio y conservados a través del tiempo.

En Arqueología se denomina excavación, al proceso de análisis de las estratigrafías naturales y antrópicas que se sedimentan en un determinado lugar. El proceso de excavación consiste en remover los depósitos en el orden inverso a como se han ido formando.

Por tanto, los objetivos de una excavación son[3]:

- Las actividades humanas en un periodo determinado del pasado.
- Los cambios experimentados por esas actividades de una época a otra.

Como se trata de una actividad destructiva (cada vez que se excava se destruye la posición original de los depósitos) es preciso documentar y registrar con toda atención los distintos elementos que componen la estratificación de un yacimiento.

2.1 Las leyes estratigráficas

Hasta hace unos años en la documentación arqueológica se tomaban en consideración solamente los estratos, construcciones y otros elementos dotados de materialidad. A partir de la contribución de Edward Harris se ha introducido la categoría de Unidad Estratigráfica para definir tanto las acciones estratigráficas que comportan una aportación de materia (Unidades Estratigráficas positivas), como la aportación de la misma (Unidades Estratigráficas negativas).[4]

La estratigrafía arqueológica se basa en los principios de la estratigrafía geológica. Esta contempla unas leyes o principios que determinaron y determinan de qué forma se sucede la superposición de estratos geológicos:

- Principio de la superposición de estratos: los niveles superiores serán más recientes que los inferiores.
- Ley de la horizontalidad original: los estratos se forman originalmente de forma horizontal.
- Ley de la continuidad original: cada depósito es originalmente un conjunto informe sin aristas.
- Principio de sucesión faunística: evolución regular de las especies gracias a la selección natural.

En Arqueología con anterioridad a Harris, se habían apuntado dos principios:

- Si el estrato A cubre al estrato B, es que B se depositó antes.
- Cada nivel o estrato data de un tiempo posterior al de la manufactura del objeto más reciente que en él se halle.

Harris enunciará entonces cuatro leyes básicas para la estratigrafía arqueológica, las tres primeras adaptadas de la Geología y la cuarta propiamente arqueológica (muy necesaria ya que el principio de superposición no es suficiente a la hora de establecer la secuencia estratigráfica):[5]

[3] [CP98] – Capítulo: ¿Dónde? Prospección y Excavación de Yacimientos y Estructuras.
[4] [MV04] – Apartado: Las leyes de la estratificación arqueológica.
[5] [R99] – Apartado: Los principios estratigráficos.

- La “ley” de superposición: si los estratos e interfaces se encuentran en el estado original de deposición, las unidades superiores son más recientes y las inferiores más antiguas. Como vemos este principio no tiene en cuenta el contenido artefactual de los estratos. Se trata, en definitiva, de la constatación del orden de deposición de dos estratos cualesquiera, la confirmación de las relaciones físicas entre depósitos superpuestos. Su importancia radica en que define las relaciones interfaciales entre elementos y depósitos de un yacimiento.
- La “ley” de horizontalidad original: al igual que la anterior, se refiere únicamente a los estratos y su forma de deposición. Los estratos en su estado original, tienden a depositarse formando planos horizontales; aquéllos con superficies inclinadas yacerán así debido a la forma de la cuenca de deposición preexistente. Deben tenerse en cuenta todas aquellas variaciones producidas por la acción antrópica.
- La “ley” de continuidad original: todo estrato o interface (se refiere a estratos horizontales) está delimitado originalmente por una cuenca de deposición o bien su grosor irá disminuyendo en los extremos hasta acabar en una cuña. Si cualquier extremo presenta una cara vertical, debe buscarse la causa que provocó este corte en la unidad estratigráfica. Este principio es la base sobre la que se construyen las correlaciones estratigráficas entre partes separadas de depósitos o interfaces originales. Según Harris tiene una gran utilidad a la hora de analizar elementos interfaciales como las fosas o, en “arqueología vertical”, para el estudio de los muros.
- La “ley” de sucesión original: nos dice que un estrato o interface ocupa su lugar exacto en la secuencia estratigráfica entre la más antigua de las unidades que la cubren (la más baja) y la más reciente de las unidades a las que cubre (la más alta), manteniendo contacto físico con ambas y siendo redundante cualquier otra relación de superposición.

Como se ha comentado a mediados de los años setenta Harris propuso representar mediante un diagrama arborescente la secuencia estratigráfica de un yacimiento arqueológico. El Matrix Harris es un instrumento de representación y análisis muy útil para contextos pluriestratificados.

Parte del principio, como se ha dejado intuir, de transformar las relaciones físicas existentes entre las unidades estratigráficas (cubre, corta, rellena, se adosa) en un esquema de relaciones temporales (anterior, posterior o contemporáneo) de manera que todas las unidades estratigráficas puedan ser reunidas en un mismo diagrama.

2.1.1 Relaciones básicas entre Unidades Estratigráficas

A continuación se mostrarán cuatro casos que expondrán los tipos básicos de relaciones estratigráficas.

En primer lugar, se empezará mostrando en la figura 4, una representación gráfica de los 4 tipos de relaciones estratigráficas que serán presentadas.

Y posteriormente se hará una descripción de que tipos de relaciones son y cómo son consideradas estas.

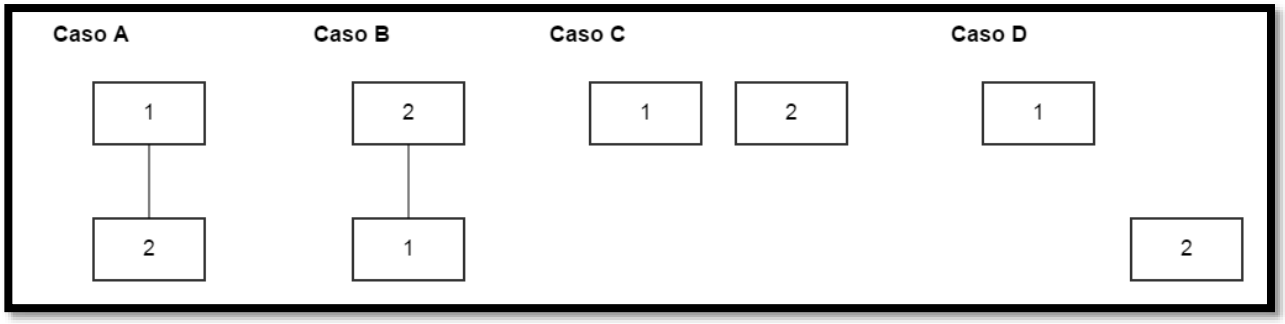

Ilustración 4 - Representación gráfica de los diferentes tipos de relaciones.

En la figura 4 pueden apreciarse como se han presentado cuatro casos diferentes (Caso A, Caso B, Caso C y Caso D) que serán descritos a continuación.

- En el caso A: 1 es posterior a 2.
- En el caso B: 2 es posterior a 1.
- En el caso C: 1 y 2 no tienen relación y son contemporáneas.
- En el caso D: 1 y 2 no tiene relación y no son contemporáneas.

2.1.2 Propiedades de las relaciones estratigráficas

- Irreflexivas: 1 no puede ser más moderna que ella misma.
- Antisimétricas: Si 1 es más moderna que 2, es imposible que 2 sea más moderna que 1.
- Transitivas: Si 1 es más moderno que 2 y 2 es más moderno que 3, entonces 1 es más modenro que 3.
- Acíclicas: Si q es más moderno que 2 y 2 es más moderno que 3, entonces 3 no puede ser más moderno que 1.

2.1.3 Sucesión estratigráfica

Las relaciones de orden que tiene una Ue con otras UEs pueden ser redundantes o inmediatas.

Las relaciones inmediatas son aquellas que son más antiguas que todas las UEs con las que tiene una relación de anterioridad o más moderna que todas las UEs con las que tiene una relación de posterioridad (están por debajo o son más antiguas).

2.1.4 Secuencia estratigráfica

Lista ordenada de relaciones binarias resultante de eliminar de la lista de adyacencia aquellos pares que son redundantes, que a pesar de ser útiles, válidas e imprescindibles, no se utilizan para dibujar la secuencia estratigráfica.

A continuación será presentado en las figuras 5a, 5b y 6 un ejemplo que pretende detectar cuáles y cómo se eliminan las relaciones redundantes.

Caso de ejemplo:

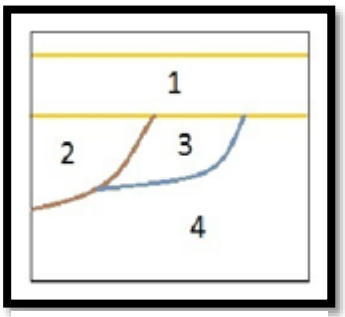

Ilustración 5a - Ejemplo secuencia estratigráficaa

Relaciones estratigráficas

(1,2)

(1,3)

(1,4)

(2,3)

(2,4)

(3,4)

Estratigrafía:

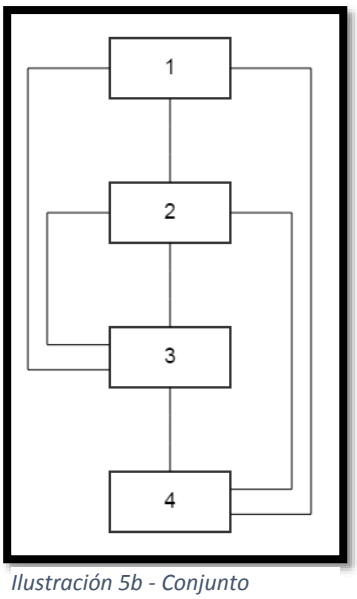

Ilustración 5b - Conjunto ordenado de todas las relaciones estratigráficas

Secuencia estratigráficas:

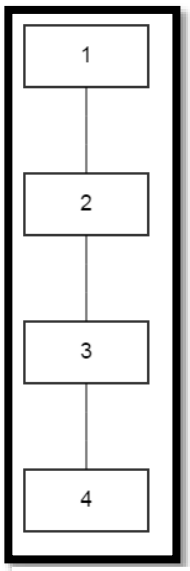

Ilustración 6 - Sin relaciones redundantes.

Como se puede apreciar en las figuras 5a, 5b y 6, las relaciones (1,3), (1,4) y (2,4) son relaciones redundantes y por tanto no serán guardadas en la nueva lista de secuencia estratigráfica y, por tanto, tampoco serán dibujadas. De esta manera se obtiene una secuencia de parejas ordenadas inmediatas.

2.2 Documentación arqueológica

Como se ha dicho, todo esto se realiza para documentar la secuencia de un yacimiento arqueológico. Pero para documentar una excavación, no solo basta con esto, sino que es necesario documentar todos los elementos que compone el yacimiento para poder obtener datos que ayuden a reconstruir la historia en ese mismo lugar en determinado momento.

Es muy importante recoger la mayor cantidad de datos posibles de cada elemento; por esta razón es necesario documentar fotográficamente todo lo que aparece: muros, fosas, los restos cerámicos, fauna…

Dentro este campo no ha de olvidarse el dibujo de campo y las planimetrías. Con planimetrías me refiero a la representación gráfica a escala sobre un plano de todo aquello que es objeto de estudio. Así se dibujarán cada una de las estructuras constructivas aparecidas en la excavación (muros, cortes, superficies, etc.), estableciendo el criterio de qué y en qué momento se dibuja el arqueólogo responsable de la intervención arqueológica.

Todas las estructuras dibujadas deben llevar cotas y su correspondiente número de UE. Todos los planos deben incluir indicación del norte, escala, leyenda de tramas, cotas de estructuras, números de UE representadas, nombre del dibujante y fecha de realización del dibujo. La planimetría se presentará debidamente georreferenciada.

3. Tecnología empleada

En este apartado se describirán las diferentes tecnologías empleadas para el desarrollo de este proyecto.

Así como describir el IDE seleccionado para la implementación de la aplicación y el lenguaje elegido y el por qué.

Con este punto se pretende familiarizar al lector con los temas que serán tratados a lo largo de la memoria y que hacen referencia a aspectos más técnicos desde el punto de vista de la programación en Android.

3.1 Android

En este apartado se tratará este sistema operativo inicialmente pensado para teléfonos móviles.

3.1.1 Un poco de Historia

Android es el sistema operativo más utilizado en telefonía móvil a día de hoy, pero lo cierto es que no siempre fue así. La compañía lanzó su primera versión de Android en 2003, y desde entonces han estado luchando por conseguir el mayor número de usuarios posibles.

A continuación se hará un breve repaso de la historia de Android destacando sus momentos más significativos:

- Google adquiere Android Inc. en el año 2005.
- En el año 2007 se crea el consorcio Handset Alliance con el objetivo de desarrollar estándares abiertos para móviles. Formado por Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel y otros. Sus miembros se comprometen a publicar una parte importante de su propiedad intelectual como código abierto bajo licencia Apache v2.0.
- En noviembre del 2007 se lanza una primera versión del Android SDK.
- En el 2008 aparece el primer móvil con Android (T-Mobile G1) y se abre el Android Market.
- Durante el 2010 Android se consolida como uno de los sistemas operativos para móviles más utilizados.
- En el 2011 se lanzan las versiones 2.0, 3.1 y 3.2 específicas para tabletas y la 4.0 tanto para móviles como para tabletas.
- En 2012 Google cambia su estrategia en su tienda de descargas online reemplazando Android Market por Google Play Store.

3.1.2 Características Android

Como se ha mencionado anteriormente, existen muchas plataformas para móviles (IPhone, Symbian, Windows Phone, BlackBerry, Palm, Javav Mobile Edition, Linux Mobile, etc); sin embargo Android presenta una serie de características que lo hacen diferente:

- Es una plataforma realmente de desarrollo libre, basada en Linux y de código abierto.
- Es adaptable a cualquier hardware ya que no ha sido diseñado para su uso exclusivo en teléfonos y tabletas. Hoy en día hay relojes, cámaras, electrodomésticos y gran variedad de sistemas empotrados que se basan en este sistema operativo.
- Portabilidad asegurada ya que las aplicaciones finales son desarrolladas en Java, lo que asegura que podrán ser ejecutadas en cualquier tipo de CPU. Esto se consigue gracias al concepto de máquina virtual.
- Arquitectura basada en componentes inspirados en Internet.
- Filosofía de dispositivo siempre conectado a Internet.
- Gran cantidad de servicios incorporados, por ejemplo localización basada tanto en GPS como en redes, bases de datos con SQL, etc.

- Aceptable nivel de seguridad. Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación.
- Cada vez más optimizado para abaja potencia y poca memoria. Android utiliza la Máquina Virtual Dalvik, se trata de una implementación de Google de la máquina virtual de Java optimizada para dispositivos móviles.
- Alta calidad de gráficos y sonido. Gráficos vectoriales suavizados, animaciones inspiradas en Flash, gráficos en tres dimensiones basados en OpenGL. Incorpora codecs estándar más comunes de audio y video.

3.1.3 Arquitectura Android

El siguiente gráfico muestra la arquitectura de Android. Como se puede ver está formado por cuatro capas.

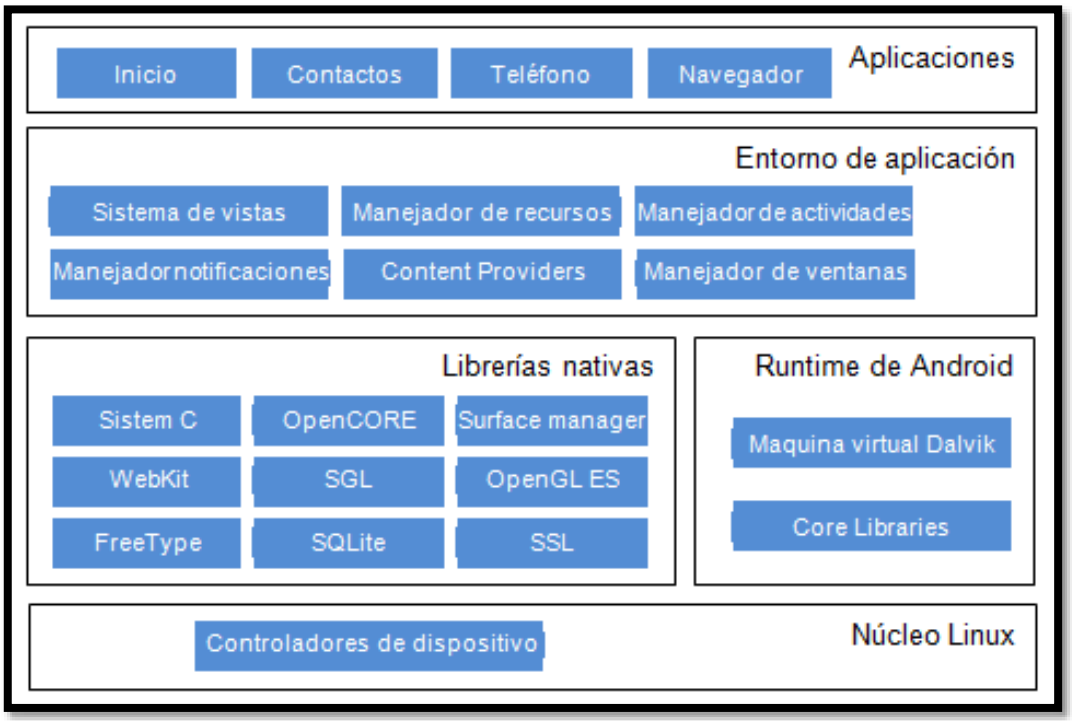

Ilustración 7 - Arquitectura de Android

Como se puede apreciar en la figura 7, Android es una plataforma para dispositivos móviles que contiene una pila de software donde se incluye un sistema operativo, middleware y aplicaciones básicas para el usuario.

En las siguientes líneas se dará una visión global por capas de cuál es la arquitectura empleada en Android. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores.

3.1.3.1 El núcleo Linux

Está formado por el sistema operativo Linux versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de drivers para dispositivos.

Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del hardware.

Algunas de las características de la mencionada máquina virtual es que facilita la optimización de recursos ya que ejecuta ficheros .dex, que es un formato optimizado para ahorrar memoria.

3.1.3.2 Runtime de Android

Está basado en el concepto de máquina virtual utilizado en Java. Dado las limitaciones de los dispositivos donde ha de correr Android (poca memoria y procesador limitado) no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual Dalvik, que responde mejor a estas limitaciones.

3.1.3.3 Librerías nativas

Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Algunas de estas librerías son:

- System C library: una derivación de la librería BSD de C estándar adaptada para dispositivos embebidos basados en Linux.
- Media Framework: librería basada en PacketVideo's OpenCORE; soporta codecs de reproducción y grabación de multitud de formatos de audio y vídeo e imágenes MPEG4, H.264, MP·, AAC, AMR, JPG y PNG.
- Surface Manager: maneja el acceso al subsistema de representación gráfica 2D y 3D.
- WebKit: soporta un moderno navegador web utilizado en el navegador Android y en la vista webview. Se trata de la misma librería que utiliza Google Chrome y Safari de Apple.
- SGL: motor de gráficos 2D.
- Librerías 3D: implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- FreeType: fuentes en bitmap y renderizado vectorial.
- SQLite: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- SSL: proporciona servicios de encriptación Secure Socket Layer.

3.1.3.4 Entorno de aplicación

Proporciona una plataforma de desarrollo libre de aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, etc.).

Esta capa ha sido diseñada para simplificar la reutilización de componentes.

Los servicios más importantes que incluye son:

- Views: extenso conjunto de vistas, (parte visual de los componentes).
- Resource Manager: proporciona acceso a recursos que no son en código.
- Activity Manager: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.

- Notificación Manager: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- Content Providers: mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

3.1.3.5 Aplicaciones

Este nivel está formado por el conjunto de aplicaciones instaladas en una máquina Android. Todas las aplicaciones han de correr en la máquina virtual Dalvik para garantizar la seguridad del sistema.

Normalmente las aplicaciones Android están escritas en Java. Para desarrollar aplicaciones en Java podemos utilizar el Android SDK. Existe otra opción consistente en desarrollar las aplicaciones utilizando C/C++. Para esta opción podemos utilizar el Android NDK (Native Devlopment kit).

3.1.4 Las versiones de Android y niveles de API

Antes de empezar un proyecto en Android hay que elegir la versión del sistema para la que deseamos realizar la aplicación. Es muy importante observar que hay clases y métodos que están disponibles a partir de una versión, si las vamos a usar hemos de conocer la versión mínima necesaria.

Cuando se ha lanzado una nueva plataforma siempre ha sido compatible con las versiones anteriores. Es decir, solo se añaden nuevas funcionalidades y en caso de modificar alguna funcionalidad no se elimina, se etiquetan como obsoletas pero se pueden continuar utilizando.

3.1.4.1 Distribución en el mercado de las APIs Android

A continuación, en la figura 8, se representará gráficamente el uso de cada API Android.

Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%

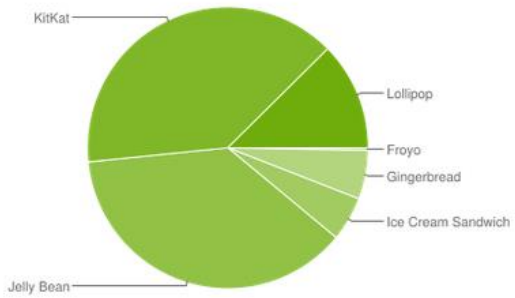

Ilustración 8 - Estadística sobre cada versión de Android - Mayo 2015

Como se puede apreciar, en la figura 8, el mercado se lo reparte Jelly Bean y KitKat. En un principio la aplicación que nos ocupa, AppQueology 2.0, se empezó a desarrollar con la API 17 para que diera soporte al mayor número de usuarios posible. Ya que la intención es que sea lo más usable posible. Pero en el momento de introducir aspectos relacionados con la vista, con la intención de darle un aspecto moderno a la aplicación, que era uno de los requisitos no funcionales que se habían obtenido durante la fase correspondiente, ha sido necesario pasar la aplicación a la API 5.0 perdiendo cuota de mercado. Aunque como se puede apreciar en la gráfica y a pesar de los problemas que presenta esta nueva versión del sistema operativo, ha sufrido un rápido crecimiento e instauración en los diferentes dispositivos móviles desde su reciente lanzamiento.

3.2 Bases de datos

Las bases de datos son una herramienta de gran potencia en la creación de aplicaciones informáticas. Hasta hace muy poco resultaba muy costoso y complejo incorporar bases de datos a nuestras aplicaciones. No obstante, como se ha mencionado, Android incorpora la librería SQLite que nos permite agilizar esta tarea.

SQLite es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesita servidor, precisa poca configuración, ser transacional y por supuesto, ser de código libre.

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas una completa API para llevar a cabo de manera sencilla todas las tareas necesarias.

En Android, la forma típica para crear, actualizar, y conectar con una base de datos SQLite será a través de una clase auxiliar llamada SQLiteOpenHelper, o para ser más exactos, de una clase propia que derive de ella y que debemos personalizar con el código necesario para crear nuestra base de datos y para actualizar su estructura respectivamente.

3.3 Componentes de una Aplicación

Existen una serie de elementos clave que resultan imprescindibles para desarrollar aplicaciones Android. En este apartado se desarrollarán una breve descripción de algunos de los más importantes.

3.3.1 Vista (View)

Las vistas son los elementos que componen la interfaz de usuario de una aplicación. Son por ejemplo, un botón, una entrada de texto,… Todas las vistas van a ser objetos descendientes de la clase View, y por tanto, pueden ser definidos utilizando código Java. Sin embargo, lo habitual va a ser definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero. Esta forma de trabajar es muy similar a la definición de una página web utilizando código HTML.[6]

[6] [J13] – Capítulo: Visión general y entorno de desarrollo.

3.3.2 Layout

Un Layout es un conjunto de vistasa grupadas de una determinada forma. Vamos a disponer de diferentes tipos de Layouts para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los Layouts también son objetos descendientes de la clase View. Igual que las vistas los Layouts pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML.[7]

3.3.3 Activity

Una aplicación de Android, va a estar formada por un conjunto de elementos básicos de visualización, coloquialmente conocidos como pantallas de aplicación. En Android cada uno de estos elementos, o pantallas, se conoce como Activity. Su función principal es la creación de la interfaz de usuario. Las diferentes actividades creadas serán independientes entre sí. Toda Activity ha de pertenecer a una clase descendiente de Activity.[8]

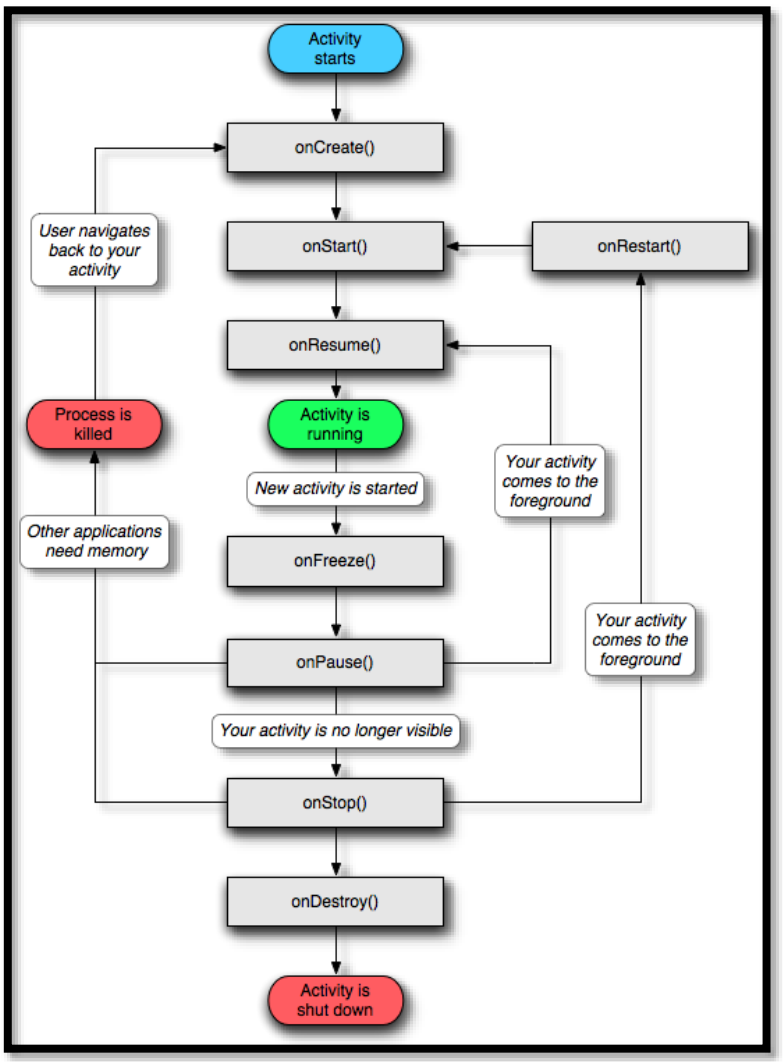

Ilustración 9 - Ciclo de vida de una Actividad

[7] [J13] – Capítulo: Visión general y entorno de desarrollo.
[8] [J13] – Capítulo: Visión general y entorno de desarrollo.

Como se puede apreciar en la figura 9, Android es sensible al ciclo de vida de una Activity, por lo tanto es preciso comprender y manejar los eventos relacionados con el ciclo de vida si se quiere crear aplicaciones estables.

Cada vez que una Activity cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad. Así habrá que atender a los siguientes momentos:

- OnCreate(Bundle): Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado dela actividad (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.
- OnStart(): Nos indica que la actividad está a punto de ser mostrada al usuario.
- OnResume(): Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.
- OnPause(): Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.
- OnStop(): La actividad ya no va a ser visible para el usuario. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.
- OnRestart(): Indica que la actividad va a volver a ser representada después de haber pasado por*onStop()*.
- OnDestroy(): Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al método *finish()*. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

Como se ha visto, el ciclo de vida de una aplicación Android es bastante diferente al ciclo de vida de una aplicación en otros sistemas operativos. La mayor diferencia es que, en Android el ciclo de vida es controlado por el sistema, en lugar de ser controlado directamente por el usuario.

3.3.4 Service

Un servicio es un proceso que se ejecuta “detrás”, sin la necesidad de una interacción con el usuario. Es algo parecido a un demonio en Unix o a un servicio de Windows.

En Android disponemos de dos tipos de servicios: servicios locales, que pueden ser utilizados por aplicaciones del mismo terminal y servicios remotos, que pueden ser utilizados desde otros terminales. [9]

Como se ha explicado, las funciones de estos servicios son diferentes, y por tanto, también sus ciclos de vida.

[9] [J13] – Capítulo: Visión general y entorno de desarrollo.

También comentar y como se podrá ver en la figura 10 un servicio tiene un ciclo de vida diferente al de una Activity, y habrá que tratarlo y entenderlo, al igual que sucedía con las Actividades si lo que se pretende es crear aplicaciones estables y aprovechar todos los recursos que estos nos ofrecen.

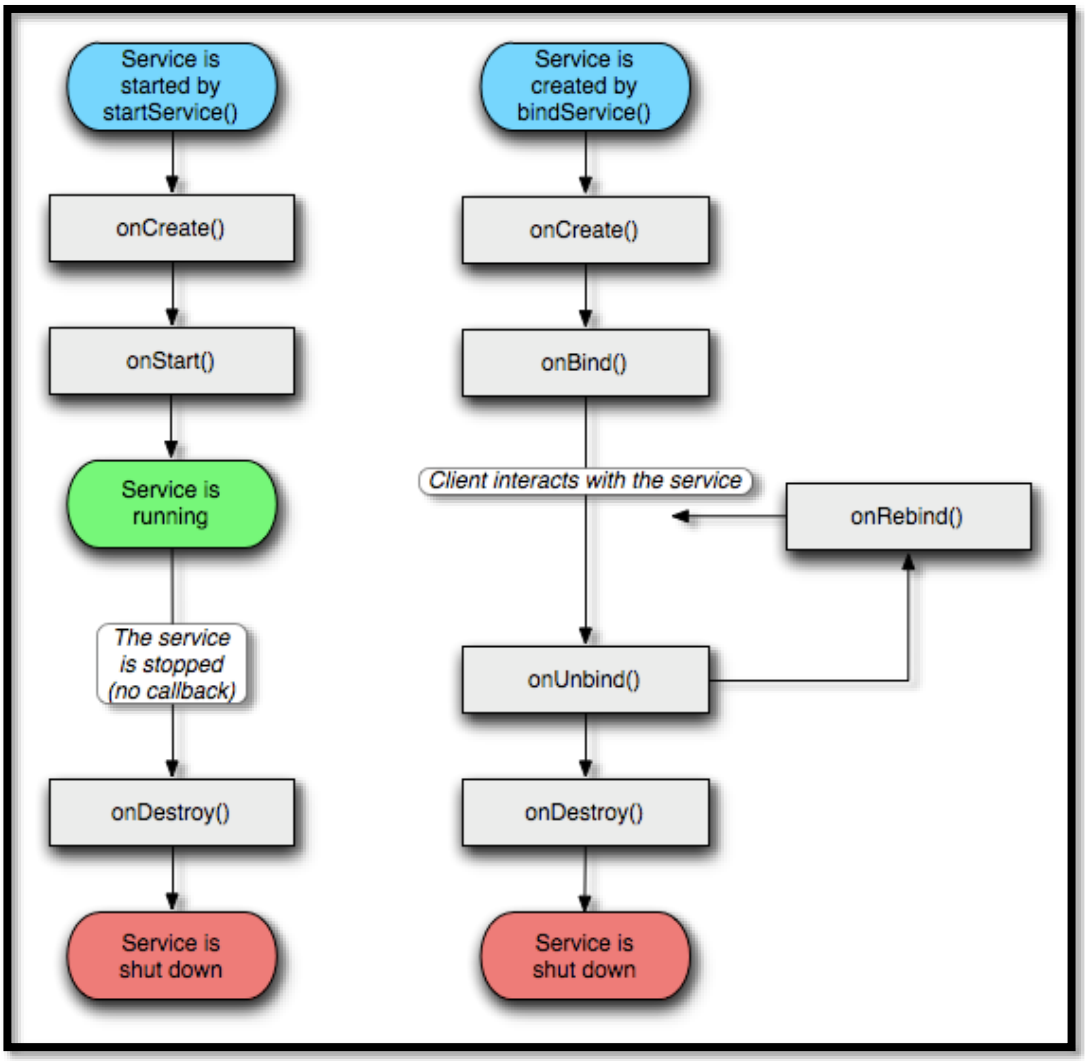

Ilustración 10 - Ciclo de vida de un Servicio

Como se puede apreciar en la figura 10 si el servicio es iniciado mediante startService() el sistema comenzará creándolo y llamando a su método onCreate(). A continuación llamará a su método onStartCommand(Intent intent, int flags, int startId) con los argumentos proporcionados por el cliente. El servicio continuará en ejecución hasta que sea invocado el método stopService() o stopSelf().

Podemos también utilizar bindService(Intent servicio, ServiceConnection conexion, int flags) para obtener una conexión persistente con un servicio. Si dicho servicio no está en ejecución, será creado (siempre que el *flag* BIND_AUTO_CREATE esté activo), llamándose al método onCreate(), pero no se llamará a onStartCommand(). En su lugar se llamará al método onBind(Intent intencion) que ha de devolver al cliente un objeto IBinder a través del

cual se podrá establecer una comunicación entre cliente y servicio. Esta comunicación se establece por medio de una interfaz escrita en AIDL, que permite el intercambio de objetos entre aplicaciones que corren en procesos separados. El servicio permanecerá en ejecución tanto tiempo como la conexión esté establecida, independientemente de que se mantenga o no la referencia al objeto IBinder.

3.3.5 Frament

Un Fragment no puede considerarse ni un control ni un contenedor, aunque se parecería más a los segundo. Un Fragement podría definirse como una porción de la interfaz de usuario que puede añadirse o eliminarse de una interfaz de forma independiente al resto de elementos de la actividad, y que por supuesto puede reutilizarse en otras actividades.

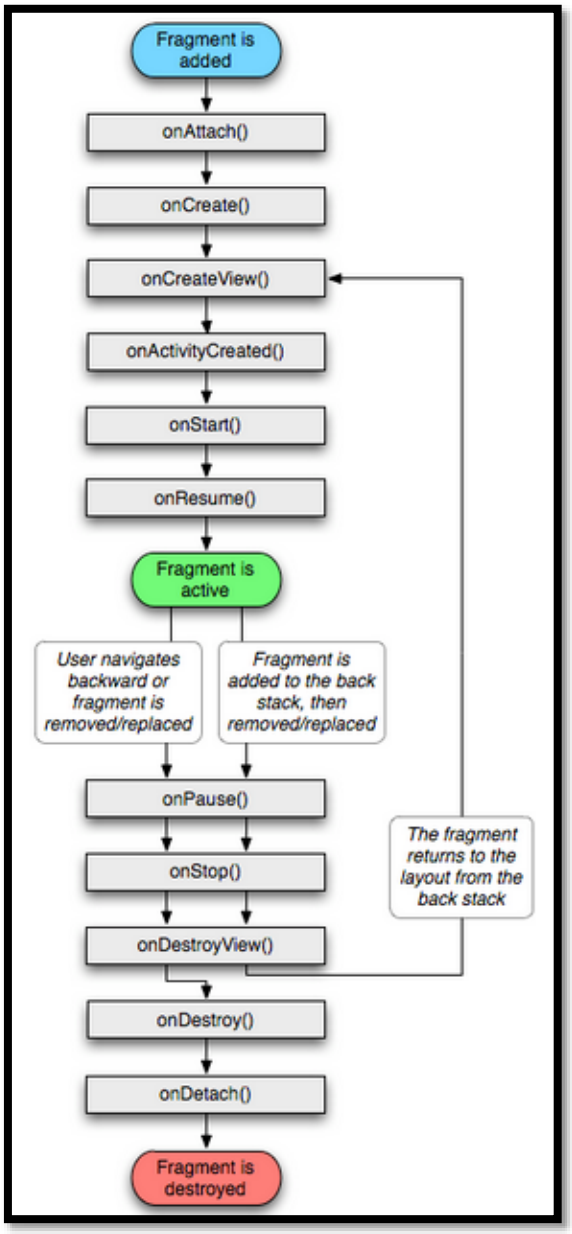

Ilustración 11 - Ciclo de vida de un Fragment

Un Fragment debe de estar siempre embebido en una Activity, por lo que el ciclo de vida de un fragmento está ligado al de la actividad.

Para poder crear un fragmento es necesario crear una subclase de Fragment, y se recomienda utilizar las siguientes funciones que se pueden apreciar en la figura 11:

- OnCreate(): El sistema llama a esta función cuando se crea el Fragment.
- OnCreateView(): El sistema llama a esta función cuando se dibuja por primera vez el Fragment en la interfaz de usuario.
- OnPause(): El sistema llama a esta función cuando el usuario deja de utilizar el Fragment. Esto no implica que éste sea destruido.

3.3.6 Intent

Un Intent representa la voluntad de realizar alguna acción, como realizar una llamada de teléfono, visualizar una página web. Se utiliza cada vez que se quiera:

- Lanzar una actividad.
- Lanzar un servicio.
- Lanzar un anuncio del tipo broadcast.
- Comunicarnos con un servicio.

También se utilizarán los Intent para el intercambio de información entre los componentes.[10]

3.3.7 Gráficos

Android nos proporciona a través de su API gráfica una potente y variada colección de funciones que pueden cubrir prácticamente cualquier necesidad gráfica de una aplicación.

Dentro de las múltiples clases que se nos ofrecen para este tipo de tareas solo serán descritas a continuación aquellas que han sido utilizadas para este proyecto.

3.3.7.1 Canvas

La clase Canvas representa una superficie donde podemos dibujar. Dispone de una serie de métodos que nos permiten representar líneas, círculos, texto,… Para dibujar en un Canvas necesitamos un pincel (Paint) donde definiremos el color, grosor de trazo, transparencia,…. También podremos definir una matriz de 3x3 (Matrix) que nos permitirá transformar coordenada aplicando una translación escala o rotación.[11]

3.3.7.2 Paint

Como ya se ha mencionado, la mayoría de los métodos de la clase Canvas utilizan un parámetro de tipo Paint. Esta clase nos permite definir el color, estilo o grosor de trazado de un gráfico vectorial.[12]

[10] [J13] – Capítulo: Visión general y entorno de desarrollo.
[11] [J13] – Capítulo: Gráficos en Android.
[12] [j13] – Capítulo: Gráficos en Adnroid.

3.3.7.3 Color

Android representa los colores utilizando enteros de 32 bits. Estos bits son divididos en 4 campos de 8 bits: alfa, rojo, verde y azul (ARGB, usando las iniciales en inglés). Al estar formado cada componente por 8 bits, podrá tomar 256 valores diferentes.

Los componentes rojo, verde y azul son utilizados para definir el color, mientras que el componente alfa define el grado de transparencia con respecto al fondo. Un valor de 255 significa un color opaco y a medida que vayamos reduciendo el valor el dibujo se irá haciendo transparente.

3.3.7.4 Drawable

La clase Drawable es una abstracción que representa "algo que puede ser dibujado". Esta clase se extiende para definir gran variedad de objetos gráficos más específicos. Muchos de ellos pueden ser definidos como recursos usando ficheros XML. [13]

3.3.8 Adapter

Un Adapter en Android actúa como puente entre un AdapterView y los datos de una Vista (View). El adaptador permite el acceso a los elementos e datos, éste también es responsable de crear una vista para cada elemento en la colección de datos.

3.3.9 Handler

Cuando se lanza un programa en Android, la ejecución se lleva a cabo en el llamado UI Thread, que es el hilo principal de la aplicación y el que se encarga de actualizar la interfaz de usuario. Los Handlers implementan un mecanismo de paso de mensajes entre threads de forma que nos permiten enviar mensajes desde nuestro hilo al UI Thread.

3.3.10 AsyncTask

Como se ha comentado, en Android es muy frecuente lanzar nuevos hilos. Tendremos que hacerlo siempre que exista la posibilidad de que una tarea pueda bloquear el hilo de la interfaz de usuario. Esto suele ocurrir en cálculos complejos o en accesos a la red. Una tarea asíncrona se define por un cálculo que se ejecuta en un hilo secundario y cuyo resultado queremos que se publique en el hilo de la interfaz de usuario.

3.3.11 Camera

La clase Camera se utiliza para configurar los ajustes en la captura de una imagen, iniciar, detener vista previa, tomar fotos, y recuperar los marcos para la codificación de vídeo. Esta clase es un cliente para el servicio de la cámara, que gestiona el hardware real de la cámara.

[13] [J13] – Capítulo: Gráficos en Android.

3.4 Localización

La plataforma Android dispone de un interesante sistema de posicionamiento que combina varias tecnologías.

- Sistema de localización global basado en GPS. Este sistema solo funciona si disponemos de visibilidad directa de los satélites.
- Sistema de localización basado en la información recibida de las torres de telefonía móvil y de puntos de acceso Wi-Fi. Funciona en el interior de los edificios.

3.5 Eventos

Android captura los distintos eventos de usuario de forma homogénea y se los pasa a la clase encargada de recogerlos. Por lo general va a ser un objeto tipo View el que recogerá estos eventos por medio de dos técnicas alternativas. Los escuchadores de eventos (Event Listener) y los manejadores de eventos (Event Handler).

3.5.1 Event Listener

Es una interfaz de la clase View que contiene un método callback que ha de ser registrado. Cada escuchador de eventos tiene solo un método callback, que será llamado por Android cuando se produzca la acción correspondiente.

3.5.2 Envent Handler

Permiten, si estas creando un descendiente de la clase View, disponer de varios métodos callback directamente como manejadores de eventos por defecto. Esta es la forma más sencilla, ya que no hace falta usar una interfaz, ni registrar el método callback.

3.6 Pantalla multi-touch

Las pantallas táctiles más modernas tienen la posibilidad de indicar la posición de varios punteros sobre la pantalla a un mismo tiempo.

3.7 Java

Se ha decidido utilizar este lenguaje de programación por varias razones. En primer lugar porque se había elegido previamente el Sistema Operativo Android y es un lenguaje nativo de este. Además, parece que Google ha decidido que era el mejor lenguaje para su SO, o sea, que quien soy yo para decir lo contrario.

Pero además de por esta razón fundamental, me siento cómodo programando en Java gracias a la relativa experiencia alcanzada durante estos años cursando el Grado de Informática.

Por otro lado, ha sido seleccionado este lenguaje, por las miles de librerías que nos ofrece, así como por la numerosa documentación que puede encontrarse.

Otra razón ha sido por las completas herramientas libres que se ofrecen y que están orientadas a dicho lenguaje. Eclipse ADT y ahora Android Studio son un ejemplo.

3.8 Android Studio

Este proyecto ha sido desarrollado apoyándose en una herramienta especializada para el desarrollo de programas Android, es decir, con la ayuda de un IDE ("Integrated Devolopment Environments"). Podría haber sido programado directamente mediante un editor de textos, pero este tipo de herramientas facilitan la programación y el desarrollo de programas Android facilitando en gran medida la tarea del desarrollador, ya que integran entre otros aspectos un sistema de ayuda para la construcción de interfaces gráficas de usuario lo cual agiliza mucho el trabajo.

El IDE seleccionado ha sido la herramienta Android Studio.

Es un entorno de desarrollo integrado (IDE) para la plataforma Android. Fue anunciado por Ellie Powers el 16 de mayo de 2013. Android Studio está disponible para desarrolladores de forma gratuita. Basado en IntelliJ IDEA de JetBrains, está diseñado específicamente para desarrollar para Android. Está disponible para descargar para Windows, Mac OS X y Linux.

3.8.1 Principales características de Android Studio

- Renderización en tiempo real.
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Soporte para construcción basada en Grandle.
- Refactorización específica de Android y arreglos rápidos.
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones y otros problemas.
- Plantillas para crear diseños comunes de Android y otros componentes.
- Soporte para programa aplicaciones para Android Wear.

4 Análisis

En este apartado se realizará el análisis de la solución.

El proceso de reunión de requisitos y posterior elaboración de los casos de uso se intensifica y se centra especialmente en el software. Dentro del proceso de análisis es fundamental que a través de una colección de requerimientos funcionales y no funcionales, el desarrollador del software comprenda completamente la naturaleza de los componentes que deben implementarse para desarrollar la aplicación, la función requerida, comportamiento, rendimiento e interconexión.

Por tanto, el propósito del análisis de un problema es ayudar al programador a conseguir una cierta comprensión de la naturaleza del problema. El problema debe estar bien definido si se desea llegar a una solución satisfactoria. Ya que una buena definición del problema es uno de los requisitos más importantes para llegar a una solución eficaz.

4.1 Requerimientos

La definición formal nos dice:

- Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.
- Una condición o capacidad que debe estar presente en un sistema o componentes de un sistema para satisfacer un contrato, estándar, especificación u otro documento formal.
- Una representación documentada de una condición o capacidad como en el punto 1 o en el 2.

Los encontramos en formas variadas como lista de pedido, casos de uso, historias, escenarios de negocio, reglas de negocio, condiciones de sistema entre otras.

Pero lo que es muy cierto es que los requerimientos deberán de ser claros, descritos o bosquejados de manera atómica además de saber cómo serán probados, por dichas razones hay que documentarlos y colocarles un identificador único tal que podamos transmitirlos, implementarlos, verificarlos y obtener las aprobaciones correspondientes.

En primer lugar, comentar que se han obtenido muchos más requisitos de los finalmente implementados. Como ya se ha mencionado, se han priorizado algunos para la entrega de este proyecto, dejando otros para una posible y posterior ampliación de la aplicación una vez finalizado el Proyecto de Fin de Grado y continuando en colaboración con la Facultad de Historia de la UB. A continuación, únicamente serán descritos aquellos requisitos que han sido implementados para la entrega de este proyecto.

4.1.1 Desde el punto de vista funcional

Un requisito funcional define una función del sistema de software o sus componentes.

- Poder crear un proyecto, cargarlo o eliminarlo.
- Que un proyecto venga definido por su nombre y el autor. Aunque no sea necesario introducir el autor para poder crearlo.
- Que la creación de un proyecto no suponga la creación de una nueva vista.
- Crear una vista que permita crear Unidades Estratigráficas (Ues) y sus relaciones.
- Que pueda navegarse entre las unidades estratigráficas estructuradas como padres e hijos con un simple clic.
- Que la creación de Ues pueda realizarse solo introduciendo el nombre o identificador de la Ue en cuestión.
- Que el identificador pueda contener letras o números o letras y números.
- Poder caracterizar las Ues a través de sus dataciones, tipo de unidad estratigráfica, descripción y por supuesto, su identificador o nombre.
- Que la datación presente dos campos para poder establecer un periodo, en el caso de ser una datación relativa.
- Que las dataciones pueda incluir letras y números.
- Que pueda sacarse una foto y asociarse a una Ue en concreto.
- Que pueda recogerse un punto gps y asociarse a una Ue en concreto.
- Que pueda buscarse una Ue a partir de su nombre o identificador.

- Que puedas modificar el nombre o identificador de una Ue y esta siga conservando todos sus datos previamente introducidos y lógicamente sin que se produzcan inconsistencias en el proyecto.
- Que puedas crear una relación con una Ue que no se ha creado previamente. Es decir, crear la Ue con la que se quiere relacionar y establecer dicha relación todo en un mismo momento.
- Que puedan eliminarse relaciones sin que se eliminen las Ues.
- Que al eliminarse una relación, la Ue hijo y sus descendientes pasen a depender del nodo raíz.
- Control de posibles errores a la hora de introducir dichas relaciones desde un punto de vista de anterioridad y posterioridad.
- Que se informe al usuario de la existencia de errores. Por ejemplo mostrándole el conjunto de Ues que crearía el ciclo y por tanto un posible anacronismo.
- La existencia de una vista, a modo de Diario de Excavación, que muestre el conjunto de todas las Ues documentadas, así como sus datos fundamentales (identificador, fecha de creación, tipo (estrato, estructura, negativa), descripción, miniatura de la foto, fecha de creación de la foto, nombre de la foto y el punto gps)
- Que las miniaturas de las fotos almacenadas y mostradas pueda ser vista en pantalla completa con un simple clic.
- Que pueda realizarse una búsqueda de una Ue concreta, dentro de este espacio diseñado a modo de Diario de Excavación.
- Crear una vista, a modo de Cronoestratigraf[14], donde se agrupen las Ues a partir de un criterio basado en la temporalidad de estas. Es decir, agrupar aquellas Ues que supuestamente hayan coexistido en el tiempo aunque no tengan una relación de contacto entre ellas.
- Que en esta vista se pueda seleccionar una Ue concreta con un simple clic y se marquen con diferentes colores, la Ue, sus relaciones directas y aquellas que le son indirectas. (Azul para la Ue seleccionada, verde para sus relaciones directas y rojo para sus relaciones indirectas).
- Poder realizar una búsqueda de una Ue concreta en dicha vista.
- Creación de una vista que represente gráficamente a modo de grafo dirigido las Ues almacenadas y sus relaciones.
- Que pueda realizarse zoom sobre el grafo.
- Que pueda realizarse una búsqueda de una Ue en concreto en dicha vista.
- Que se eliminen para la vista en cuestión las relaciones redundantes.
- Que se pueda navegar entre las diferentes vistas conservando en todo momento el foco centrado en una Ue en concreto. Apareciendo en todo momento remarcada con un color diferente.
- Mostrar información acerca de cómo funciona la aplicación a modo de instrucciones.

[14] Es una representación gráfica de las UEs agrupadas a partir de su contemporaneidad y no siguiendo sus relaciones físicas. Será una vista similar a la existente en el programa Stratigraf. Programa diseñado, hace unos años, por investigadores de la Facultad de Historia de la UB.

4.1.2 Desde el punto de vista no funcional

Un requisito no funcional o atributo de calidad es, en la ingeniería de software un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos.

- Implementación de la aplicación únicamente para su uso en móviles.
- Almacenar toda la información en una Base de Datos local.
- Crear carpetas específicas para las fotografías almacenadas.
- Procurar seguir el modelo de diseño MVC.
- Seguir las pautas aprendidas durante la asignatura de Factores Humanos relacionadas con la usabilidad, accesibilidad, robustez, estabilidad, entre otros. Con la intención de conseguir fundamentalmente una aplicación intuitiva, rápida y estable.
- Desarrollo de una interfaz con un diseño moderno utilizando algún recurso novedoso.

4.2 Casos de Uso

Los conceptos básicos de un diagrama de casos de uso serán:

- **Actor:** que es una representación de una entidad externa en lo que tiene que ver con el software en tiempo de ejecución. Las entidades externas en cuestión pueden ser personas, máquinas, otras máquinas, otros sistemas de software o, cuando algún caso de uso debe iniciarse automáticamente en una fecha y hora determinada, un actor ficticio que se puede llamar reloj, tiempo o algo por el estilo. Cada actor tiene un papel para cada caso de uso en los que interviene; un papel es primario si el actor arranca el caso de uso.

- **Caso de uso, o comportamiento:** son procesos autónomos iniciados por un actor y/o por otro caso de uso. Representan funciones ofrecidas por el software e identifican las entradas y las salidas. Un caso de uso debe dar siempre un resultado definido. Se supone que los actores no llevan a cabo ninguna secuencia de caso de uso rígida, sino que invocan cada caso de uso en momentos determinados por curso de su trabajo con la única limitación de las relaciones de prerrequisito entre casos de uso.

- **Asociación entre actores y casos de uso**: es siempre binaria. El significado de una asociación entre un actor y un caso de uso es que las instancias del actor interactúan con el software durante las ejecuciones del caso de uso.

Los requisitos presentados, adquiridos durante la fase correspondiente, dan lugar a los siguientes casos de uso, que se muestran en la figura 12.

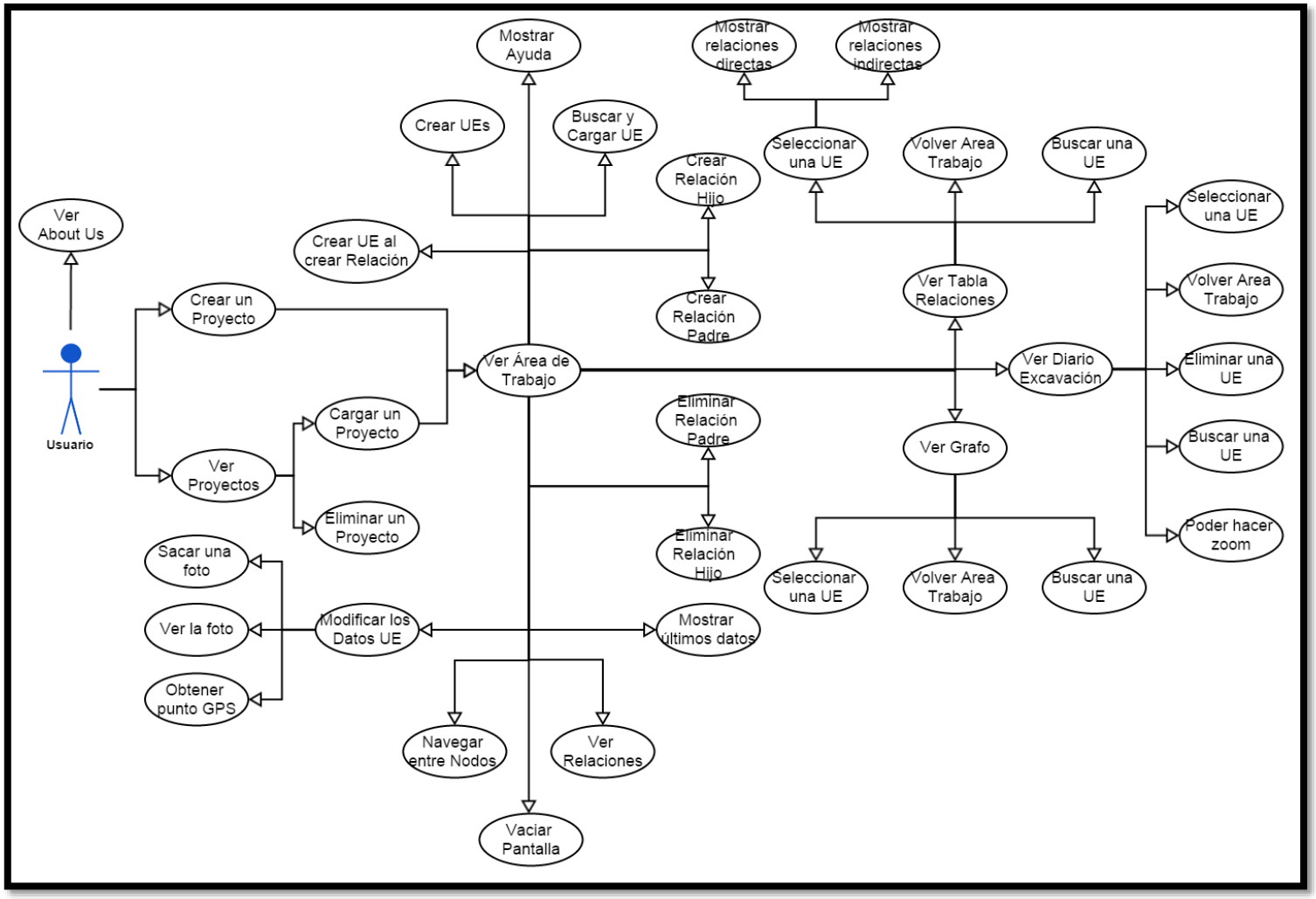

Ilustración 12 - Diagrama de Casos de Uso

Como se puede apreciar en la figura 12, el diagrama de casos de uso identifica los comportamientos ejecutantes, llamados casos de uso, de un clasificador generalmente complejo (por ejemplo, un software) y especifica con qué usuarios y otras entidades exteriores tienen interacción (en el sentido que reciben información o los dan o son puestos en marcha por ellas) y también ciertas relaciones entre estos comportamientos.

4.2.1 Casos de uso textuales

La especificación textual de los casos de uso complementa el diagrama de casos de uso.

Consiste en describir el proceso de cada uno de ellos por separado. Esta descripción se hace en lenguaje natural porque es necesario que la puedan revisar los representantes del cliente, que no es seguro que entiendan notaciones más formales, pero en una forma normalizada que garantice que los requisitos funcionales de proceso sean lo suficientemente precisos para que se pueda desarrollar un software a partir de ellos.

A continuación se presentarán los diferentes casos de usos textuales contemplados para esta aplicación.

4.2.1.1 UC1 - Ver información sobre los creadores del código

- Breve descripción:
  - Este caso de uso permitirá al actor ver información referente a los autores del proyecto.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona el botón “About us”.
- Flujo básico:
  1. El actor pulsará el botón que permite ver la información mencionada.
  2. Se abrirá un dialogo que muestra la información sobre los autores de la aplicación.
  3. El actor podrá seleccionar el botón “Aceptar” que cerrará el dialogo y volverá a la vista del “Menú Principal”.
- Flujo alternativo:
  - Sin flujo alternativo.
- Precondiciones:
  - Sin precondiciones.
- Pos-condiciones:
  - Que el actor pueda ver satisfactoriamente la información sobre los creadores de la aplicación.

4.2.1.2 UC2 - Crear un Proyecto

- Breve descripción:
  - Este caso de uso permitirá al actor crear un nuevo proyecto.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona el botón “Crear un nuevo proyecto”.
- Flujo básico:
  1. El actor pulsará el botón que permite crear un nuevo proyecto.
  2. Se abrirá a un dialogo donde el actor:
  3. Tendrá que introducir el nombre del proyecto.
  4. Podrá introducir el autor del proyecto.
  5. Clicar en el botón “Aceptar”:
  6. Se creará un nuevo proyecto almacenándolo en la base de datos correspondiente.
  7. Se pasará a la vista “Área de Trabajo”.
- Flujo alternativo:
  - Si tras el clicar el botón “Aceptar” de la acción del flujo básico 5, el actor, no ha realizado la acción del flujo básico 3, se cerrará el diálogo, se lanzará un Toast indicando al actor que no se realizarán acciones y se vuelve a la vista “Menú Principal”.
  - Si clica en el botón “Cancelar” en cualquier momento, se cerrará el diálogo y se volverá a la vista “Menú Principal”.

    - Si durante la acción 3 del flujo básico, introduce un nombre de un proyecto que ya existe previamente, tras pulsar el botón “Aceptar” de la acción 5 del flujo básico, se cerrará el diálogo y se lanzará un Toast avisando al actor que no se puede realizar la acción de crear un nuevo proyecto ya que le nombre introducido coincide con el de un proyecto ya existente.
- Precondiciones:
    - Sin precondiciones.
- Pos-condiciones:
    - Que tras realizar la acción el actor haya creado el proyecto satisfactoriamente y consiga pasar a la avista “Área de Trabajo”.

4.2.1.3 UC3 - Ver los proyectos

- Breve descripción:
    - Este caso de uso permitirá al actor ver los proyectos guardados previamente.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor selecciona “Cargar un proyecto”.
- Flujo básico:
    1. El actor pulsará el botón que permite cargar un proyecto guardado:
    2. Se abrirá una nueva vista con todos los proyectos almacenados.
    3. El actor podrá clicar en el botón “Aceptar” cerrando el dialogo.
- Flujo alternativo:
    - Sin flujo alternativo.
- Precondiciones:
    - Que previamente se haya creado algún proyecto.
- Pos-condiciones:
    - Que el actor sea capaz de ver los proyectos satisfactoriamente.

4.2.1.4 UC4 - Cargar un Proyecto

- Breve descripción:
    - Este caso de uso permitirá al actor cargar un proyecto guardado previamente.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor selecciona “Cargar un proyecto”.
- Flujo básico:
    1. El actor pulsará el botón que permite cargar un proyecto guardado previamente.
    2. Se abrirá una nueva vista con todos los proyectos almacenados.
    3. Seleccionar un proyecto con un simple clic.
    4. Se cargará el proyecto seleccionado.
    5. Se pasa a la vista “Área de Trabajo”.
- Flujo alternativo:
    - Si el actor en algún momento clica el botón de “back”, se cerrará la vista, no se cargará ningún proyecto y se volverá a la vista “Menú Principal”.

- Precondiciones:
  - Que se haya creado algún proyecto anteriormente.
- Pos-condiciones:
  - Que el actor pueda cargar satisfactoriamente un proyecto y pase a la vista “Área de trabajo”.

4.2.1.5 UC5 - Eliminar un Proyecto

- Breve descripción:
  - Este caso de uso permitirá al usuario cargar un proyecto guardado previamente.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona “Cargar un proyecto”.
- Flujo básico:
  1. El actor pulsará el botón que permite cargar un proyecto guardado previamente.
  2. Se abrirá una nueva vista con todos los proyectos almacenados.
  3. Realizar un clic largo sobre uno de los proyectos.
  4. Se eliminará el proyecto seleccionado.
- Flujo alternativo:
  - Que el actor en algún momento pulse el botón de “back”, se cerrará la vista, no se realizará ninguna acción y se volverá a la vista “Menú Principal”.
- Precondiciones:
  - Que se haya creado algún proyecto previamente.
- Pos-condiciones:
  - Que el actor haya eliminado satisfactoriamente el proyecto.

4.2.1.6 UC6 - Ver el Área de Trabajo

- Breve descripción:
  - Este caso de uso permitirá al actor ver la vista “Área de Trabajo” donde se pueden crear, borrar ues, relaciones etc.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona “Crear un proyecto” o “Cargar un proyecto”.
- Flujo básico:
  1. El actor pulsará el botón que permite cargar un proyecto guardado previamente.
  2. Se abrirá una nueva vista con todos los proyectos almacenados.
  3. Seleccionar un proyecto con un simple clic.
  4. Se cargará el proyecto seleccionado.
  5. Se pasa a la vista “Área de Trabajo”.

  A. El actor pulsará el botón que permite crear un nuevo proyecto.
  B. Se lanzará un diálogo.

C. El actor introduce el nombre del proyecto.
D. El actor clica sobre el botón “Aceptar”.
E. Se cierra el diálogo.
F. Se pasa a la vista “Área de Trabajo”.

- Flujo alternativo:
  - Si el usuario clica en el botón “back” en algún momento, se cerrará la vista, no se realizará ninguna acción y se volverá a la vista “Menú Principal”.
  - Si el actor no ha realizado la acción C del flujo básico, tras la acción D del flujo básico, se cerrará el diálogo, y se lanzará un Toast avisando al actor que no se creará ningún proyecto ya que no ha introducido su nombre.
  - Si en algún momento el actor pulsa el botón “Cancelar” tras la acción B del flujo básico, se cerrará el diálogo y no se realizará ninguna acción, únicamente se volverá a la vista “Menú Principal”.
  - Si durante la acción C del flujo básico, introduce un nombre de un proyecto que ya existe previamente, tras pulsar el botón “Aceptar” de la acción D del flujo básico, se cerrará el diálogo y se lanzará un Toast avisando al actor que no se puede realizar la acción de crear un nuevo proyecto ya que le nombre introducido coincide con el de un proyecto ya existente.
- Precondiciones:
  - En el caso de querer ir a la vista “Área de Trabajo” a través del flujo básico de números, que exista algún proyecto creado anteriormente.
  - Ningún flujo alternativo para el flujo básico de números.
- Pos-condiciones:
  - Que el usuario pueda ver la vista “Área de Trabajo” satisfactoriamente.

4.2.1.7 UC7 - Mostrar Ayuda

- Breve descripción:
  - Este caso de uso permitirá al actor ver instrucciones sobre el funcionamiento.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona mostrar instrucciones.
- Flujo básico:
  1. El actor pulsará el botón que permite ver la información mencionada.
  2. Se abrirá un dialogo que muestra las instrucciones.
  3. El actor seleccionará el botón “Aceptar”.
  4. Se cierra el dialogo.
  5. Se vuelve a la vista “Área de Trabajo”.
- Flujo alternativo:
  - Sin flujo alternativo.
- Precondiciones:
  - Sin precondiciones.
- Pos-condiciones:
  - Que el actor pueda ver las instrucciones satisfactoriamente.

4.2.1.8 UC8 - Crear una UE

- Breve descripción:
  - Este caso de uso permitirá al usuario crear una nueva Unidad Estratigráfica.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona crear una nueva UE.
- Flujo básico:
  1. El actor pulsará el botón que permite crear una nueva UE.
  2. Se abrirá un dialogo donde el actor:
  3. Introducir el nombre de la UE.
  4. El actor clica en el botón “Aceptar”.
  5. Se creará una nueva UE almacenándola en la base de datos correspondiente.
  6. Se pasará a la vista “Área de Trabajo”.
  7. Se creará en la zona correspondiente el botón que representa a dicha UE.
- Flujo alternativo:
  - Si el actor tras la acción 2 del flujo básico, clica sobre el botón “Cancelar” se cerrará el diálogo, no se realizará ninguna acción y se volverá a la vista “Área de Trabajo”.
  - Si durante la acción 3 del flujo básico, el actor no ha introducido ningún nombre para la UE, tras la acción 4 del flujo básico se cerrará el diálogo y se lanzará un Toast avisando al actor que no se realizará ninguna acción ya que no ha introducido el nombre de la UE que pretendía crear.
  - Si durante la acción 3 del flujo básico, el actor ha introducido el nombre de una UE que ya existía previamente, tras la acción 4 del flujo básico se cerrará el diálogo y se lanzará un Toast avisando al actor de que no se puede realizar la acción ya que ha introducido como nombre de la nueva UE, uno que ya existía previamente y que si lo desea que vuelva a intentarlo con otro nombre diferente.
- Precondiciones:
  - Sin precondiciones.
- Pos-condiciones:
  - Que el actor sea capaz de crear la UE satisfactoriamente.

4.2.1.9 UC9 - Buscar y cargar una UE

- Breve descripción:
  - Este caso de uso permitirá al actor buscar y cargar una UE en la vista.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona buscar una UE:
- Flujo básico:
  1. El actor pulsará el botón que permite buscar y cargar una UE:
  2. Se abrirá un diálogo.
  3. El actor introduce el nombre de la UE a buscar y cargar en el campo correspondiente del diálogo.
  4. Clica el botón “Aceptar” del diálogo.

5. Se cierra el dialogo.
6. Se cargará la UE buscada en la vista creando el botón correspondiente que la representa.

- Flujo alternativo:
  - Si la UE no existe tras realizar la acción del flujo básico 4, se cerrará el dialogo y se lanzará un Toast avisando al actor de que dicha UE no existe.
  - Si clica el botón aceptar sin haber introducido el nombre se cerrará el dialogo y se lanzará un Toast avisando al actor de que no se realizará ninguna acción ya que no ha introducido un nombre para la UE que pretendía crear.
  - Si el actor clica el botón "Cancelar" del diálogo de la acción del flujo básico 2, este se cerrará y se volverá a mostrar únicamente la vista "Área de Trabajo".
- Precondiciones:
  - Para buscar y cargar una UE tiene que haber alguna UE guardada con anterioridad.
- Pos-condiciones:
  - Que el actor pueda buscar, cargar y ver la UE en cuestión perfectamente.

4.2.1.10 UC10 - Modificar los datos de una UE

- Breve descripción:
  - Este caso de uso permitirá al actor modificar el nombre y los valores que caracterizan a la UE en cuestión.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor clica sobre el botón que representa a la UE a la que se le pretenden modificar sus valores.
- Flujo básico:
  1. El actor pulsará el botón de la UE que pretende modificar.
  2. Se abrirá una nueva vista donde se le permitirá modificar el nombre, el tipo, dataciones I y II, descripción, se le permitirá sacar otro punto de GPS y otra foto.
  3. El actor clica sobre el botón aceptar que presenta la vista.
  4. Se lanzará un Alert que le pregunta si realmente quiere modificar los datos de la UE en cuestión.
  5. El usuario clica sobre el botón "Aceptar" de dicho Dialog.
- Flujo alternativo:
  - Si a la hora de modificarle el nombre, el actor introduce el nombre de una UE que ya existe, no se podrá realizar la acción y se le notificará por medio de un Toast.
- Precondiciones:
  - Que existe alguna UE sobre la cual modificar sus datos.
- Pos-condiciones:
  - Que el actor pueda modificar los datos de la UE seleccionada satisfactoriamente.

4.2.1.11 UC11 - Vaciar la pantalla

- Breve descripción:

    - Este caso de uso permitirá al actor limpiar la vista de botones que representan las diferentes UEs y sus relaciones de la vista correspondiente al “Área de Trabajo”.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor selecciona vaciar la pantalla.
- Flujo básico:
    1. El actor pulsará el botón que permite vaciar la pantalla:
    2. Se vaciará la vista.
- Flujo alternativo:
    - Sin flujo alternativo.
- Precondiciones:
    - No hay precondiciones para este caso de uso.
- Pos-condiciones:
    - Que el actor consiga vaciar la pantalla satisfactoriamente.

4.2.1.12 UC12 - Crear una relación de anterioridad

- Breve descripción:
    - Este caso de uso permitirá al actor asignar una nueva relación de anterioridad a la UE cargada previamente.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor selecciona crear una nueva relación de anterioridad.
- Flujo básico:
    1. El actor pulsará el botón que permite crear una nueva relación padre.
    2. Se lanza un diálogo.
    3. El actor introduce un nombre para la UE con la que quiere relacionar la UE cargada previamente.
    4. El actor pulsa en el botón “Aceptar” del diálogo.
    5. Se cierra el diálogo.
    6. Se crea el botón que representará a la UE con la que se pretende relacionar a la UE previamente cargada.
    7. Se guarda la relación en la base de datos correspondiente.
- Flujo alternativo:
    - Si el actor durante la acción 3 del flujo básico, no introduce ningún nombre, tras la acción 4 del flujo básico se cerrará el diálogo y se lanzará un Toast indicándole al actor que no se realizará ninguna acción ya que no ha introducido un nombre.
    - Si tras realizar la acción 4 del flujo básico, la relación que pretende establecer crea un ciclo, se cerrará el diálogo y se lanzará un Toast indicándole al actor que no se puede realizar la acción ya que se crea un ciclo. Se lanza otro Toast indicándole al actor las UEs que generan ese ciclo al introducir esa nueva relación.

- Si tras la acción 2 del flujo básico, el actor pulsa el botón “Cancelar”, se cerrará el diálogo, no se realizará ninguna acción y se volverá a la vista “Área de Trabajo”.

- Precondiciones:
  - Que la relación que se pretende establecer no cree un ciclo.
- Pos-condiciones:
  - Que el actor consiga establecer la relación satisfactoriamente.

4.2.1.13 UC13 - Crear una relación de posterioridad

- Breve descripción:
  - Este caso de uso permitirá al actor asignar una nueva relación de posterioridad a la UE cargada previamente.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona crear una nueva relación de posterioridad.
- Flujo básico:
  1. El actor pulsará el botón que permite crear una nueva relación padre.
  2. Se lanza un diálogo.
  3. El actor introduce un nombre para la UE con la que quiere relacionar la UE cargada previamente.
  4. El actor pulsa en el botón “Aceptar” del diálogo.
  5. Se cierra el diálogo.
  6. Se crea el botón que representará a la UE con la que se pretende relacionar a la UE previamente cargada.
  7. Se guarda la relación en la base de datos correspondiente.
- Flujo alternativo:
  - Si el actor durante la acción 3 del flujo básico, no introduce ningún nombre, tras la acción 4 del flujo básico se cerrará el diálogo y se lanzará un Toast indicándole al actor que no se realizará ninguna acción ya que no ha introducido un nombre.
  - Si tras realizar la acción 4 del flujo básico, la relación que pretende establecer crea un ciclo, se cerrará el diálogo y se lanzará un Toast indicándole al actor que no se puede realizar la acción ya que se crea un ciclo. Se lanza otro Toast indicándole al actor las UEs que generan ese ciclo al introducir esa nueva relación.
  - Si tras la acción 2 del flujo básico, el actor pulsa el botón “Cancelar”, se cerrará el diálogo, no se realizará ninguna acción y se volverá a la vista “Área de Trabajo”.
- Precondiciones:
  - Que la relación que se pretende establecer no cree un ciclo.
- Pos-condiciones:
  - Que el actor consiga establecer la relación satisfactoriamente.

4.2.1.14 UC14 - Eliminar una relación de anterioridad

- Breve descripción:

    - Este caso de uso permitirá al actor eliminar una relación de anterioridad existente, de la UE previamente cargada.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor hace un clic largo sobre uno de los botones que representan la relación con la UE cargada previamente.
- Flujo básico:
    8. El actor hará un clic largo sobre el botón que representa la relación de anterioridad.
    9. Se lanza un diálogo.
    10. El actor confirma que quiere eliminar dicha relación clicando sobre el botón "Aceptar".
    11. Se elimina dicha relación.
    12. Se cierra el diálogo.
    13. Se crea el botón que representará a la UE con la que se pretende relacionar a la UE previamente cargada.
    14. Se guarda la relación en la base de datos correspondiente.
- Flujo alternativo:
    - Sin flujo alternativo.
- Precondiciones:
    - Que exista una relación para poder ser eliminada.
- Pos-condiciones:
    - Que el actor consiga eliminar la relación satisfactoriamente.

4.2.1.15 UC15 - Eliminar una relación de posterioridad

- Breve descripción:
    - Este caso de uso permitirá al actor eliminar una relación de posterioridad existente, de la UE previamente cargada.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor hace un clic largo sobre uno de los botones que representan la relación con la UE cargada previamente.
- Flujo básico:
    15. El actor hará un clic largo sobre el botón que representa la relación de anterioridad.
    16. Se lanza un diálogo.
    17. El actor confirma que quiere eliminar dicha relación clicando sobre el botón "Aceptar".
    18. Se elimina dicha relación.
    19. Se cierra el diálogo.
    20. Se crea el botón que representará a la UE con la que se pretende relacionar a la UE previamente cargada.
    21. Se guarda la relación en la base de datos correspondiente.
- Flujo alternativo:
    - Sin flujo alternativo.

- Precondiciones:
  - Que exista una relación para poder ser eliminada.
- Pos-condiciones:
  - Que el actor consiga eliminar la relación satisfactoriamente.

4.2.1.16 UC16 - Ver relaciones

- Breve descripción:
  - Este caso de uso permitirá al actor ver en la vista Área de Trabajo las relaciones que tiene una UE cargada previamente.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor carga una UE en la vista "Área de Trabajo".
- Flujo básico:
  1. El actor cargará una UE a través de:
     a. Crear una UE (UC8).
     b. Buscar y cargar una UE (UC9).
     c. Visitar la vista "Grafo" seleccionar una UE y volver a la vista "Área de Trabajo".
     d. Visitar la vista "Diario de Excavación", seleccionar una UE y volver a la vista "Área de Trabajo".
     e. Visitar la vista "Tabla", seleccionar una UE y volver a la vista "Área de Trabajo".
  2. Se le muestran las relaciones de la UE cargada en la vista Área de Trabajo.
- Flujo alternativo:
  - Que la UE cargada no tenga ninguna relación.
- Precondiciones:
  - Que la UE cargada tenga alguna relación guardada previamente.
- Pos-condiciones:
  - Que el actor consiga ver las relaciones de la UE cargada satisfactoriamente.

4.2.1.17 UC17 - Navegar entre las UEs

- Breve descripción:
  - Este caso de uso permitirá al actor ver en la vista Área de Trabajo las relaciones que tiene una UE cargada previamente.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor clica sobre alguno de los botones que representan las UEs en la vista Área de Trabajo.
- Flujo básico:
  1. El actor clica sobre uno de los botones que representan a un padre o a un hijo en la vista Área de Trabajo.
  2. El botón que representa a la UE cargada pasará a representar a la UE padre o hijo sobre la cual el actor ha clicado.

3. Se muestran las relaciones de esa nueva UE cargada.

- Flujo alternativo:
  - Que el actor clique sobre el botón “back”, saliendo de la vista Área de Trabajo y volviendo a la vista Menú Principal.
  - Que la UE cargada no tenga ninguna relación guardada con antelación, lo cual, de no ser así, impedirá la navegación entre UEs.
- Precondiciones:
  - Que la UE cargada tenga alguna relación guardada previamente.
- Pos-condiciones:
  - Que el actor pueda navegar entre las UEs de manera satisfactoria.

4.2.1.18 UC18 - Ver la vista Tabla

- Breve descripción:
  - Este caso de uso permitirá al actor ver en la vista Tabla.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona la opción ver tabla.
- Flujo básico:
  1. El actor clica sobre el botón que le permite entrar en la vista Tabla.
  2. Se le muestra vista Tabla.
- Flujo alternativo:
  - Si el actor tenía en la vista “Área de Trabajo” alguna UE cargada, en la vista “Tabla” se cambiará el color de fondo del botón que representa a la UE previamente cargada en la vista “Área de Trabajo”, también se cambiará el color de su letra, también se cambiará a color verde de fondo de las UEs que representan sus relaciones directas, y a color rojo el fondo de las UEs que representan sus relaciones indirectas.
- Precondiciones:
  - UC6.
  - Que exista alguna UE creada con anterioridad.
- Pos-condiciones:
  - Que el actor pueda ver la vista Tabla satisfactoriamente.

4.2.1.19 UC19 - Seleccionar una UE en la vista Tabla

- Breve descripción:
  - Este caso de uso permitirá al actor seleccionar una UE de las representadas en la vista “Tabla”.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor hace un simple clic sobre uno de los botones que representan las UEs almacenadas.
- Flujo básico:
  1. El actor clica sobre el botón que representa a una de las UEs almacenadas.
  2. Se cambia el color de fondo del botón.

3. Se cambia el color de letra del botón seleccionado.
4. Se cambia a verde el fondo de los botones que representan las relaciones directas de esa UE seleccionada.
5. Se cambia a rojo el fondo de los botones que representan las relaciones indirectas de esa UE seleccionada.

- Flujo alternativo:
  - Que el usuario clique sobre el botón “back”, volviendo a la vista “Área de Trabajo”.
- Precondiciones:
  - Que exista alguna UE creada con anterioridad.
- Pos-condiciones:
  - Que el actor pueda marcar/cargar satisfactoriamente una UE.

4.2.1.20 UC20 - Buscar una UE en la vista Tabla

- Breve descripción:
  - Este caso de uso permitirá al actor buscar una UE de las representadas en la vista “Tabla”.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor clica en el botón que permite iniciar la búsqueda.
- Flujo básico:
  1. El actor clica sobre el botón que permite iniciar la vista en el Action Bar.
  2. Se lanza un diálogo.
  3. El actor introduce el nombre de la UE que pretende buscar.
  4. El actor clica sobre el botón “Aceptar” del diálogo.
  5. Se cierra el diálogo.
  6. Se cambia el color de fondo del botón que representa a la UE buscada.
  7. Se cambia el color de letra del botón que representa a la UE buscada.
  8. Se cambia a verde el fondo de los botones que representan las relaciones directas de esa UE buscada.
  9. Se cambia a rojo el fondo de los botones que representan las relaciones indirectas de esa UE buscada.
- Flujo alternativo:
  - Que el usuario clique sobre el botón “Cancelar” del diálogo en cualquier momento posterior a la acción 2 del flujo básico.
  - Si durante la acción 3 del flujo básico, el usuario introduzca el nombre de una Ue que no existe, tras la acción 4 del flujo básico, se cerrará el diálogo y se lanzará un Toast avisando al actor que no se puede realizar la acción ya que el nombre de la UE introducida no existe.
- Precondiciones:
  - Que exista alguna UE creada con anterioridad que coincida con el nombre que introduce el actor en la acción 3 del flujo básico.
- Pos-condiciones:
  - Que el actor pueda buscar/cargar satisfactoriamente la UE buscada.

4.2.1.21 UC21 - Volver a la vista Área de Trabajo desde la vista Tabla

- Breve descripción:
  - Este caso de uso permitirá al actor volver a la vista “Área de Trabajo” desde la vista “Tabla”.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor clica sobre el botón “back”.
- Flujo básico:
  1. El actor clica sobre el botón “back” estando en la vista “Tabla”.
  2. Se cierra la vista “Tabla”.
  3. Se carga la vista “Área de Trabajo”.
- Flujo alternativo:
  - Si el actor tiene alguna UE cargada en la vista “Tabla”, tras la acción 3 del flujo básico, se crearán en la vista “Área de Trabajo” los botones pertinentes que representen a la UE seleccionada y sus relaciones.
- Precondiciones:
  - Que el actor se encuentre en la vista “Tabla”.
- Pos-condiciones:
  - Que el actor pueda volver a la vista “Área de Trabajo” desde la vista “Tabla” satisfactoriamente.

4.2.1.22 UC22 - Ver la vista Diario de Excavación

- Breve descripción:
  - Este caso de uso permitirá al actor ver en la vista “Diario de Excavación”.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona la opción ver la vista “Diario de Excavación”.
- Flujo básico:
  1. El actor clica sobre el botón que le permite entrar en la vista “Diario de Excavación”.
  2. Se le muestra vista “Diario de Excavación”.
- Flujo alternativo:
  - Si el actor tiene alguna UE cargada en la vista “Área de Trabajo” cuando se cargue la vista “Diario de Excavación”, se cambiará el color de fondo y de letra del ítem que representa a la UE previamente cargada en la vista “Área de Trabajo”.
- Precondiciones:
  - UC6.
  - Que exista alguna UE creada con anterioridad.
- Pos-condiciones:
  - Que el actor pueda ver la vista “Diario de Excavación” satisfactoriamente.

4.2.1.23 UC23 - Seleccionar una UE en la vista Diario de Excavación

- Breve descripción:
  - Este caso de uso permitirá al actor seleccionar una UE de las representadas en la vista “Diario de Excavación”.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor hace un simple clic sobre uno de los botones que representan las UEs almacenadas.
- Flujo básico:
  1. El actor clica sobre el botón que representa a una de las UEs almacenadas.
  2. Se cambia el color de fondo del botón.
  3. Se cambia el color de letra del botón seleccionado.
- Flujo alternativo:
  - Que el usuario clique sobre el botón “back”, volviendo a la vista “Área de Trabajo”.
- Precondiciones:
  - Que exista alguna UE creada con anterioridad.
- Pos-condiciones:
  - Que el actor pueda marcar/cargar satisfactoriamente una UE.

4.2.1.24 UC24 - Buscar una UE en la vista Diario de Excavación

- Breve descripción:
  - Este caso de uso permitirá al actor buscar una UE de las representadas en la vista “Diario de Excavación”.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor clica en el botón que permite iniciar la búsqueda.
- Flujo básico:
  1. El actor clica sobre el botón que permite iniciar la vista en el Action Bar.
  2. Se lanza un diálogo.
  3. El actor introduce el nombre de la UE que pretende buscar.
  4. El actor clica sobre el botón “Aceptar” del diálogo.
  5. Se cierra el diálogo.
  6. Se cambia el color de fondo del botón que representa a la UE buscada.
  7. Se cambia el color de letra del botón que representa a la UE buscada.
- Flujo alternativo:
  - Que el usuario clique sobre el botón “Cancelar” del diálogo en cualquier momento posterior a la acción 2 del flujo básico.
  - Si durante la acción 3 del flujo básico, el usuario introduzca el nombre de una Ue que no existe, tras la acción 4 del flujo básico, se cerrará el diálogo y se lanzará un Toast avisando al actor que no se puede realizar la acción ya que el nombre de la UE introducida no existe.
- Precondiciones:

- Que exista alguna UE creada con anterioridad que coincida con el nombre que introduce el actor en la acción 3 del flujo básico.
- Pos-condiciones:
  - Que el actor pueda buscar/cargar satisfactoriamente la UE buscada.

4.2.1.25 UC25 - Volver a la vista Área de Trabajo desde Diario de Excavación

- Breve descripción:
  - Este caso de uso permitirá al actor volver a la vista "Área de Trabajo" desde la vista "Diario de Excavación".
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor clica sobre el botón "back".
- Flujo básico:
  1. El actor clica sobre el botón "back" estando en la vista "Diario de Excavación".
  2. Se cierra la vista "Diario de Excavación".
  3. Se carga la vista "Área de Trabajo".
- Flujo alternativo:
  - Si el actor tiene alguna UE cargada en la vista "Diario de Exavación", tras la acción 3 del flujo básico, se crearán en la vista "Área de Trabajo" los botones pertinentes que representen a la UE seleccionada y sus relaciones.
- Precondiciones:
  - Que el actor se encuentre en la vista "Diario de Excavación".
- Pos-condiciones:
  - Que el actor pueda volver a la vista "Área de Trabajo" desde la vista "Diario de Excavación" satisfactoriamente.

4.2.1.26 UC26 - Ver la vista Grafo

- Breve descripción:
  - Este caso de uso permitirá al actor ver en la vista "Grafo".
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor selecciona la opción ver la vista "Grafo".
- Flujo básico:
  1. El actor clica sobre el botón que le permite entrar en la vista "Grafo".
  2. Se le muestra vista "Grafo".
- Flujo alternativo:
  - Si el actor tiene alguna UE cargada en la vista "Área de Trabajo" cuando se cargue la vista "Grafo", se cambiará el color de fondo y de letra del ítem que representa a la UE previamente cargada en la vista "Área de Trabajo".
- Precondiciones:
  - UC6.
  - Que exista alguna UE creada con anterioridad.
- Pos-condiciones:
  - Que el actor pueda ver la vista "Grafo" satisfactoriamente.

4.2.1.27 UC27 - Seleccionar una UE en la vista Grafo

- Breve descripción:
  - Este caso de uso permitirá al actor seleccionar una UE de las representadas en la vista “Grafo”.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor hace un simple clic sobre uno de los botones que representan las UEs almacenadas.
- Flujo básico:
  4. El actor clica sobre el botón que representa a una de las UEs almacenadas.
  5. Se cambia el color de fondo del botón.
  6. Se cambia el color de letra del botón seleccionado.
- Flujo alternativo:
  - Que el usuario clique sobre el botón “back”, volviendo a la vista “Área de Trabajo”.
- Precondiciones:
  - Que exista alguna UE creada con anterioridad.
- Pos-condiciones:
  - Que el actor pueda marcar/cargar satisfactoriamente una UE.

4.2.1.28 UC28 - Buscar una UE en la vista Grafo

- Breve descripción:
  - Este caso de uso permitirá al actor buscar una UE de las representadas en la vista “Grafo”.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor clica en el botón que permite iniciar la búsqueda.
- Flujo básico:
  1. El actor clica sobre el botón que permite iniciar la vista en el ActionBar.
  2. Se lanza un diálogo.
  3. El actor introduce el nombre de la UE que pretende buscar.
  4. El actor clica sobre el botón “Aceptar” del diálogo.
  5. Se cierra el diálogo.
  6. Se cambia el color de fondo del botón que representa a la UE buscada.
  7. Se cambia el color de letra del botón que representa a la UE buscada.
- Flujo alternativo:
  - Que el usuario clique sobre el botón “Cancelar” del diálogo en cualquier momento posterior a la acción 2 del flujo básico.
  - Si durante la acción 3 del flujo básico, el usuario introduzca el nombre de una Ue que no existe, tras la acción 4 del flujo básico, se cerrará el diálogo y se lanzará un Toast avisando al actor que no se puede realizar la acción ya que el nombre de la UE introducida no existe.

- Precondiciones:
    - Que exista alguna UE creada con anterioridad que coincida con el nombre que introduce el actor en la acción 3 del flujo básico.
- Pos-condiciones:
    - Que el actor pueda buscar/cargar satisfactoriamente la UE buscada.

4.2.1.29 UC29 - Volver a la vista Área de Trabajo desde la vista Grafo

- Breve descripción:
    - Este caso de uso permitirá al actor volver a la vista "Área de Trabajo" desde la vista "Grafo".
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor clica sobre el botón "back".
- Flujo básico:
    1. El actor clica sobre el botón "back" estando en la vista "Grafo".
    2. Se cierra la vista "Grafo".
    3. Se carga la vista "Área de Trabajo".
- Flujo alternativo:
    - Si el actor tiene alguna UE cargada en la vista "Grafo", tras la acción 3 del flujo básico, se crearán en la vista "Área de Trabajo" los botones pertinentes que representen a la UE seleccionada y sus relaciones.
- Precondiciones:
    - Que el actor se encuentre en la vista "Grafo".
- Pos-condiciones:
    - Que el actor pueda volver a la vista "Área de Trabajo" desde la vista "Grafo" satisfactoriamente.

4.2.1.30 UC30 - Mostrar los datos de la última UE y relación guardados

- Breve descripción:
    - Este caso de uso permitirá al actor ver cuáles han sido la última UE y relación introducidas en la base de datos.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor selecciona ver la última UE y relación introducida.
- Flujo básico:
    1. El actor selecciona la opción ver última UE y relación introducida.
    2. Se lanza un AlertDialog.
- Flujo alternativo:
    - Sin flujo alternativo.
- Precondiciones:
    - Que exista alguna UE almacenada previamente.
- Pos-condiciones:
    - Que el actor pueda ver la información correspondiente satisfactoriamente.

4.2.1.31 UC31 - Sacar una foto

- Breve descripción:
  - Este caso de uso permitirá al actor sacar una foto por medio de la aplicación.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor clica sobre el botón que permite conectar con la cámara del dispositivo.
- Flujo básico:
  1. El actor selecciona la opción de sacar una foto.
  2. Se activa la cámara del dispositivo.
  3. El usuario saca la foto.
  4. Comprueba que le gusta la foto.
  5. Guarda la foto.
- Flujo alternativo:
  - Sin flujo alternativo.
- Precondiciones:
  - Que exista alguna UE almacenada para que pueda relacionarse con la foto que se pretende sacar.
- Pos-condiciones:
  - Que el actor pueda sacar la foto satisfactoriamente.

4.2.1.32 UC32 - Obtener un punto GPS

- Breve descripción:
  - Este caso de uso permitirá al actor obtener un punto GPS.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor intenta sacar una foto.
- Flujo básico:
  1. El actor selecciona la opción sacar una foto.
  2. Se almacena el punto automáticamente.
  3. Se le muestra un mensaje indicándole que se ha almacenado correctamente.
- Flujo alternativo:
  - Si el usuario no tiene activado la localización o los datos en su dispositivo se le avisará mediante un Alert y se le brindará la posibilidad de activarlo.
  - Si no se ha podido localizar el punto después de haber transcurrido un tiempo prudencial.
- Precondiciones:
  - Que exista alguna UE almacenada previamente.
- Pos-condiciones:
  - Que el actor pueda obtener el punto GPS satisfactoriamente.

4.2.1.33 UC33 - Ver una foto en pantalla completa

- Breve descripción:
  - Este caso de uso permitirá al actor ver una foto en pantalla completa.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el clica sobre una imagen en miniatura.
- Flujo básico:
  1. El actor selecciona la imagen en miniatura.
  2. Se abre una nueva vista con la imagen en pantalla completa.
- Flujo alternativo:
  - Sin flujo alternativo.
- Precondiciones:
  - Que exista alguna UE almacenada previamente con una foto asociada.
- Pos-condiciones:
  - Que el actor pueda ver la foto en pantalla completa satisfactoriamente.

4.2.1.34 UC34 - Eliminar una UE

- Breve descripción:
  - Este caso de uso permitirá al actor eliminar una UE almacenada previamente.
- Actores:
  - Usuario
- Flujo de eventos:
  - El caso de uso se inicia cuando el actor hace un clic largo sobre una de las UEs del Diario de Excavación.
- Flujo básico:
  1. El actor selecciona con un clic largo una de las UEs del Diario de Excavación.
  2. Se lanza un Alert que le pide la confirmación al actor de que realmente desea eliminar dicha UE.
  3. El actor clica sobre el botón “Aceptar”.
  4. Se actualiza la vista del Diario de Excavación.
- Flujo alternativo:
  - Que el actor tras la acción 2 del flujo básico clique en el botón “Cancelar”. No se realizará ninguna acción y se cerrará el Alert.
- Precondiciones:
  - Que exista alguna UE almacenada previamente.
  - Que el actor se encuentre en el Diario de Exacavación.
- Pos-condiciones:
  - Que el actor pueda eliminar correctamente la UE.

4.2.1.35 UC35 - Hacer zoom sobre la vista Grafo

- Breve descripción:
  - Este caso de uso permitirá al actor hacer zoom sobre la vista que representa gráficamente las UEs y sus relaciones.
- Actores:
  - Usuario

- Flujo de eventos:
    - El caso de uso se inicia cuando el actor toca con dos dedos y los desplaza por la pantalla siguiendo el uso lógico de este tipo de dispositivos para obtener el zoom.
- Flujo básico:
    1. El actor coloca los dos dedos sobre la pantalla.
    2. Separa los dedos para aumentar o los acerca para disminuir la escala.
- Flujo alternativo:
    - Sin flujo alternativo.
- Precondiciones:
    - Que exista alguna UE almacenada para que se aprecien los efectos del zoom.
- Pos-condiciones:
    - Que el actor pueda realizar el zoom sobre la vista mencionada con satisfacción.

4.2.1.36 UC36 - Mostrar relaciones directas en el Cronoestratigraf

- Breve descripción:
    - Este caso de uso permitirá al actor visualizar las relaciones directas de una UE seleccionada en el Cronoestratigraf remarcadas con un color verde.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor carga una de las UEs representadas en dicha vista.
- Flujo básico:
    1. El actor carga una UE en dicha vista.
    2. Se le muestran las relaciones directas.
- Flujo alternativo:
    - Sin flujo alternativo.
- Precondiciones:
    - Que exista alguna relación directa con la UE cargada por el usuario.
- Pos-condiciones:
    - Que el actor pueda visualizar las relaciones directas con normalidad.

4.2.1.37 UC37 - Mostrar relaciones indirectas en el Cronoestratigraf

- Breve descripción:
    - Este caso de uso permitirá al actor visualizar las relaciones indirectas de una UE seleccionada en el Cronoestratigraf remarcadas con un color rojo.
- Actores:
    - Usuario
- Flujo de eventos:
    - El caso de uso se inicia cuando el actor carga una de las UEs representadas en dicha vista.
- Flujo básico:
    1. El actor carga una UE en dicha vista.
    2. Se le muestran las relaciones indirectas.
- Flujo alternativo:

    - Sin flujo alternativo.
- Precondiciones:
    - Que exista alguna relación indirecta con la UE cargada por el usuario.
- Pos-condiciones:
    - Que el actor pueda visualizar las relaciones indirectas con normalidad.

5 Diseño de la aplicación

Una vez que se han establecido los requisitos del software, el diseño es la primera de tres actividades técnicas: diseño, implementación y pruebas. Cada actividad transforma la información de forma que al final se obtiene un software validado.

Sin diseño del software nos arriesgamos a construir un sistema inestable, un sistema que falle cuando se realicen pequeños cambios, un sistema que sea difícil de probar, un sistema cuya calidad no pueda ser evaluada hasta más adelante, cuando quede poco tiempo y ya se haya gastado mucho dinero.

En esta fase se alcanza con mayor precisión una solución óptima de la aplicación, teniendo en cuenta los recursos físicos del sistema (tipo de dispositivo, comunicaciones, etc…) y los recursos lógicos (sistema operativo, bases de datos, etc…).

En este apartado se comentará:

- En primer lugar, cuál ha sido el modelo o patrón elegido que estructurará la aplicación.
- Cuáles han sido las clases que han sido necesarias implementar a partir de los requisitos obtenidos. Y como se agrupan estas, a partir de sus funcionalidades, siguiendo el patrón presentado en el punto primero de este apartado.
- El diseño de la interfaz.
- Que algoritmos se han implementado.

5.1 Patrón Modelo-Vista-Controlador

Como su nombre indica no hay que dejar de mencionar que para la implementación de este proyecto se ha intentado seguir el patrón MVC.

El Modelo Vista Controlador es un patrón que divide nuestra aplicación en tres niveles distintos, uno que representa a la interfaz gráfica (Vista), otro que representa el tratamiento de datos (Modelo) y otro que se encarga de toda la lógica que se tiene que llevar a cabo por la aplicación (Controlador). Esto se hace para permitir una mayor portabilidad de una aplicación, e incluso facilitar su mantenimiento.

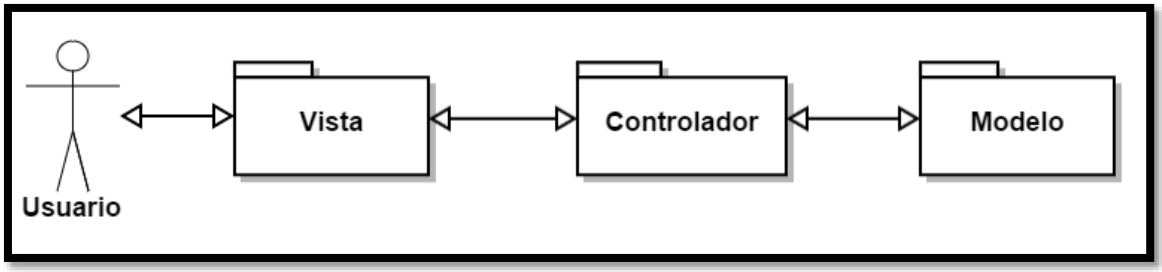

Ilustración 13 - Esquema MVC

Como vemos en la figura 13, la forma en que debe trabajar este patrón es conectando la Vista, que es la parte visible para el usuario de la aplicación, con un Controlador, que recibirá los datos de la Vista para tratarlos, o se los mandará a la Vista para mostrarlos al usuario. A su vez, el Controlador será el que conecte con la capa Modelo, para poder guardar los datos recibidos desde la Vista, o para solicitar los datos que queremos enviar a la Vista.

En Android tenemos un MVC un tanto peculiar, pues nos encontramos con las Vistas, que creamos en XML, y cada una de estas vistas tiene asociada un Activity que la gestiona, sin embargo si tenemos un aplicación con 4 Activities distintas, y todas necesitan acceder a los datos no debemos acceder desde cada una de las Acitivity al Modelo, aunque pertenecen al nivel del Controlador. En lugar de esto las Activities que necesiten conectar al Modelo van a pasar por un Controlador intermedio.

En Android existe una clase que es la encargada de la gestión de toda la aplicación, si vamos al AndroidManifest.xml podremos ver que cuando hemos creado nuevas Activities las hemos ido introduciendo dentro de una etiqueta “application”:

```
1  <application
2      android:allowBackup="true"
3      android:icon="@drawable/ic_launcher"
4      android:label="@string/app_name"
5      android:theme="@style/AppTheme" >
6      <activity
7          android:name="com.proyectosimio.proyecto.Main"
8          android:label="@string/app_name" >
9          <intent-filter>
10             <action android:name="android.intent.action.MAIN" />
11
12             <category android:name="android.intent.category.LAUNCHER" />
13         </intent-filter>
14     </activity>
15 </application>
```

Ilustración 14 - Ejemplo de un fragmento de un Manifest

Pues bien, en Java también podemos hacer referencia a una clase Application, que será la que controlará las Activities, de manera que todo el tráfico entre el Modelo y las Activities pasarán a través de esta clase. Por lo tanto el esquema real del MVC en Android quedaría como muestra la figura 15.

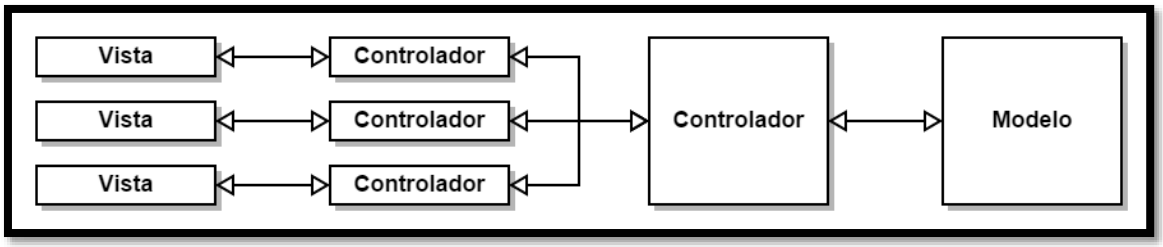

Ilustración 15 - Android MVC

Por lo tanto, antes de nada lo que habrá que hacer es definir la clase Application, que se utilizará, como se ha comentado, para canalizar todo el tráfico entre las Activities y el Modelo. Para conseguir esto, dicha clase tendrá que extender de Application y en el mánifest.xml tendremos que indicar que esta será la clase que va a gestionar la aplicación. Para ello será necesario definir el atributo Android:Name de la etiqueta Application.

5.2 Estructura planteada para AppQueology 2.0

Así, de este modo, la aplicación sigue la estructura que se describe en la figura 16.

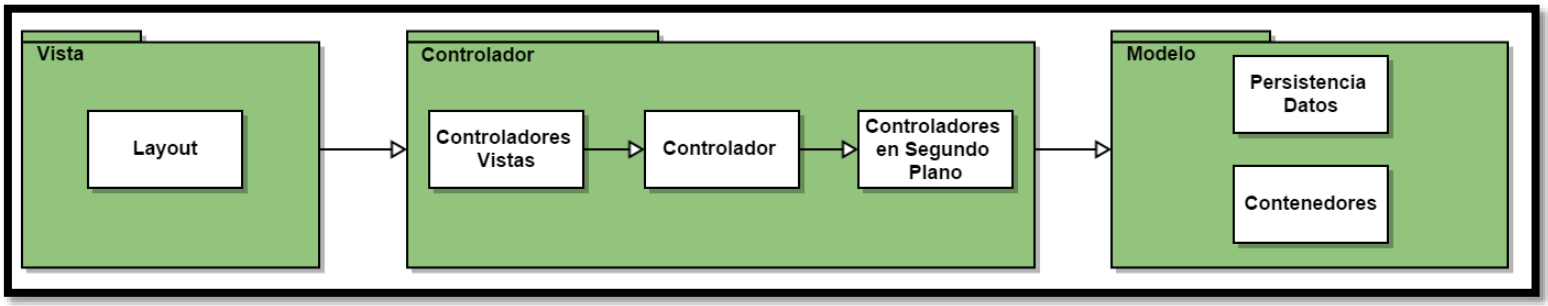

Ilustración 16 - Estructura de la aplicación AppQueology 2.0

Como nos muestra la figura 16, vemos que dispone de tres Packages fundamentales y que corresponden con el modelo en cuestión.

- Package Vista: que albergará el conjunto de los Layouts.
- El Package Controlador: que contendrá los controladores de los diferentes Layouts, el controlador propiamente dicho y una serie de controladores que trabajan en segundo plano para conseguir una aplicación más fluida.
- Y por último, el Package Modelo: que tendrá aquellos componentes que representarán a los diferentes elementos que son necesarios para un correcto funcionamiento de la aplicación.

Siguiendo este patrón, para cada una de los Packages, encontraremos las siguientes clases:

5.2.1 Package Vista

Si queremos combinar varios elementos del tipo vista tendremos que utilizar un objeto del tipo Layout. Un Layout es un contenedor de una o más vistas y controla su comportamiento y posición. Hay que destacar que un Layout puede contener a otro Layout y que es un descendiente de la clase View.

Así el conjunto de Layouts implementados para este proyecto y almacenado en el package Vista son los que pueden apreciarse en la figura 17.

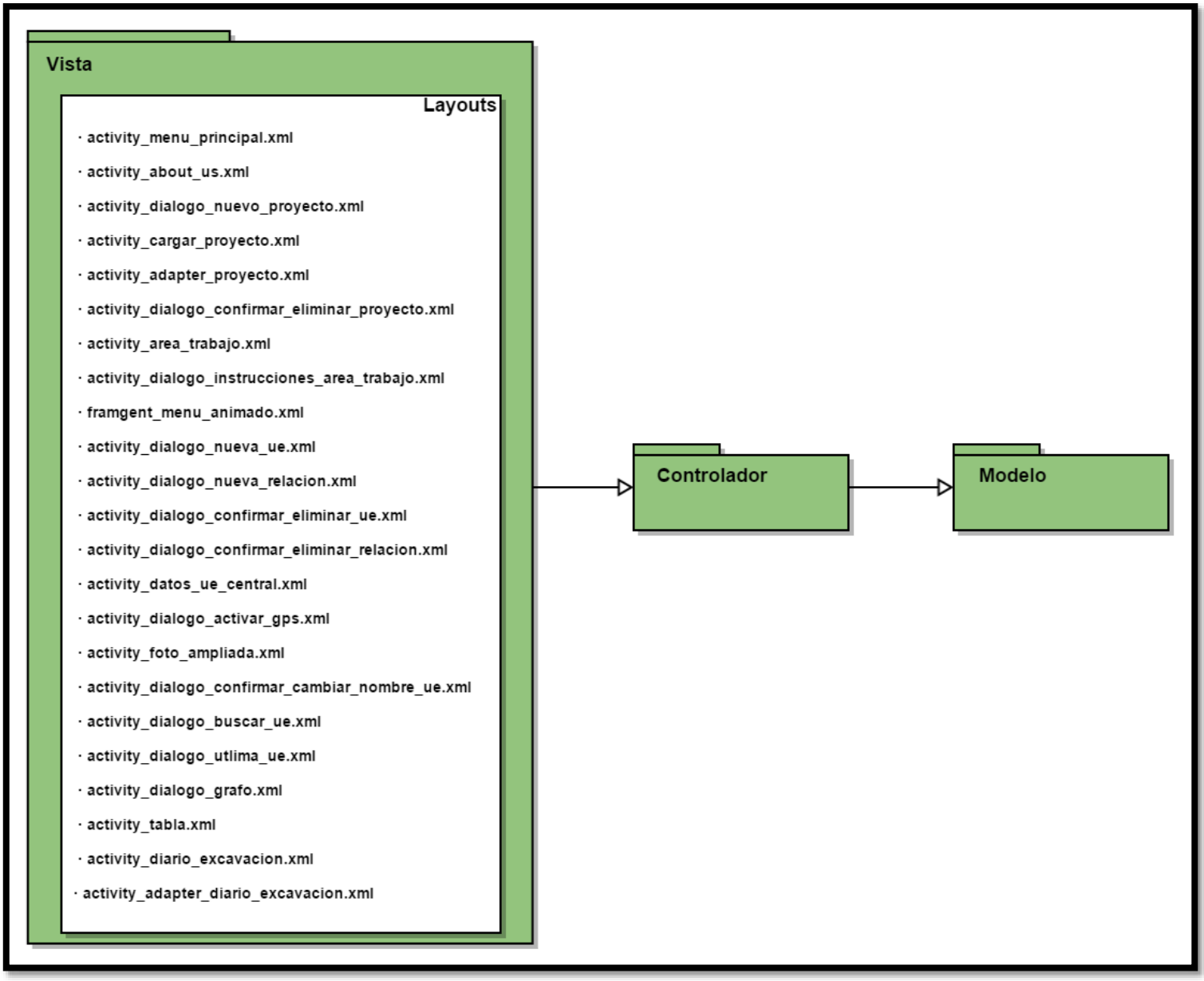

Ilustración 17 - Conjunto de Clases (Layouts) dentro del Package Vista.

Como ha podido apreciarse en la figura 17, el Package Vista de nuestra aplicación, está compuesto de 22 actividades y un fragmento, que serán descritos a continuación para una mejor comprensión de estos.

- **activity_menu_principal.xml:** Este Layout será el encargado de mostrar los elementos necesarios para que el usuario pueda crear un nuevo proyecto y ver los ya almacenados. Un proyecto correspondería con una excavación o un sector individualizado de una excavación; y albergará todos los datos referentes a las UEs y relaciones de dicha excavación o sector. Así esta clase le ofrecerá al usuario la posibilidad de realizar las siguientes acciones:
    - Consultar información sobre los desarrolladores de la aplicación - (UC1).
    - Crear un nuevo proyecto - (UC2).
    - Enlazar con la clase CargarProyecto.java y su Layout correspondiente "activity_cargar_proyecto.sml" - (UC3).
- **activity_dialogo_about_us.xml:** Este Layout será el encargado de mostrar información al usuario acerca de los desarrolladores.

- **activity_dialogo_nuevo_proyecto.xml:** Este Layout permite al usuario introducir los datos referentes a un Nuevo Proyecto para que este sea creado y posteriormente almacenado en la base de datos correspondiente.

- **activity_cargar_proyecto.xml:** Este Layout permite al usuario ver los proyectos almacenados en la base de datos correspondiente. Así le permitirá al usuario realizar las siguientes acciones:
    - Mostrar los proyectos guardados (UC3).
    - Permitir seleccionar y cargar uno de los proyectos (UC4).
    - Borrar un proyecto (UC5).

- **activity_adapter_proyecto.xml:** Este Layout será el encargado de dar aspecto a cada uno de los proyectos que se cargarán en el Layout activity_cargar_proyecto.xml.

- **activity_dialogo_confirmar_eliminar_proyecto.xml:** Este Layout permitirá al usuario, como su nombre indica, confirmar que desea eliminar un proyecto, o cancelar dicha acción.

- **activity_area_trabajo.xml:** Este Layout será el área de trabajo principal de la aplicación, donde se podrán realizar la mayoría de las acciones que se esperan de este proyecto (UC6). Así esta clase permitirá al usuario realizar las siguientes acciones:
    - Consultar las instrucciones de uso de la aplicación – (UC7).
    - Crear una nueva UE - (UC8).
    - Buscar y cargar una UE – (UC9).
    - Asignar relaciones de anterioridad y posterioridad sobre una UE - (UC12 y UC13).
    - Eliminar relaciones (UC14 y UC15).
    - Ver el grafo que representa las UEs y sus relaciones - (UC26).
    - Ver el Diario de Excavación (UC22).
    - Ver el Cronoestratigraf (Tabla) – (UC18).
    - Vaciar la pantalla de trabajo – (UC11).
    - Ver los datos de la última UE y relación introducidas – (UC30).

- Navegar entre las UEs – (UC17).
- Modificar los valores de una UE – (UC10).

- **activity_dialogo_instrucciones_area_trabajo.xml:** Este Layout permite mostrar al usuario que es lo que hace cada uno de los botones que componen la vista activity_area_trabajo.xml.

- **activity_dialogo_nueva_ue.xml:** Este Layout permite al usuario introducir el nombre de la nueva UE que se pretende crear.

- **activity_dialogo_nueva_relacion.xml:** Este Layout permite al usuario introducir el nombre de la UE con la que se pretende relacionar.

- **activity_dialogo_confirmar_eliminar_relacion.xml:** Este Layout permite al usuario, como su nombre indica, confirmar que desea eliminar la relación seleccionada.

- **activity_dialogo_confirmar_cambiar_nombre_ue.xml:** Este Layout permite al usuario, confirmar que desea cambiar el nombre de la UE o, por el contrario, cancelar la operación.

- **activity_dialogo_buscar_ue.xml:** Este Layout permite al usuario cargar una UE, previamente almacenada, en el Area de Trabajo.

- **activity_dialogo_ultima_ue.xml:** Este Layout permite al usuario ver cuál ha sido la última UE guardad en la base de datos correspondiente, así como su fecha de creación. Y también la última relación creada.

- **fragment_menu_animado.xml:** Este Layout mostrará al usuario un botón que una vez activado desplegará y contraerá una serie de botones. Dichos botones ofrecerán las siguientes acciones:
  - Ver las instrucciones de uso de los botones del Layout activity_area_trabajo.xml.
  - Ver la última UE y relación almacenada en las bases de datos correspondientes.
  - Vaciar la pantalla de trabajo de la vista activity_area_trabajo.xml.
  - Ver la vista activity_tabla.xml.
  - Ver la vista activity_grafo.xml.
  - Ver la vista activity_diario_excavacion.xml.

- **activity_tabla.xml:** Este Layout permitirá ver las UEs organizadas en función de que sean contemporáneas y no por sus relaciones de contacto. Además mostrará las relaciones directas e indirectas de una UE. A su vez permitirá seleccionar y buscar una UE determinada en dicha vista.

- **activity_grafo.xml:** Este Layout permitirá representar gráficamente, mediante un grafo dirigido, las UEs almacenadas, así como todas sus relaciones. Así como seleccionar y buscar una UE determinada dentro de dicha vista. (UC26).

- **activity_diario_excavacion.xml:** Este Layout permitirá ver la relación de todas las UEs almacenadas en la base de datos correspondiente para un proyecto dado. Así como todos los datos almacenados para cada una de ellas. También permitirá buscar y seleccionar una UE determinada dentro de dicha vista.

- **activity_adapter_diario_excavacion.xml:** Este Layout será el encargado de dar aspecto a cada uno de los elementos que se cargarán en el Layout activity_diario_excavación.xml.

- **activity_dialogo_confirmar_eliminar_ue.xml:** Este Layout permitirá al usuario confirmar que desea eliminar una UE en la vista activity_diario_excavación.xml, o, por el contrario, cancelar la operación.

- **activity_datos_ue_central.xml:** Este Layout permitirá al usuario introducir datos sobre la UE cargada en la vista activity_area_trabajo.xml. Así como sacar una foto y asociarla a dicha UE cargada. Y también capturar un punto GPS y relacionarlo con la UE mencionada. (UC10, UC31, UC32)

- **activity_activar_gps.xml:** Este Layout permitirá que el usuario confirme si desea activar el GPS. Esta vista será activada en el momento en el que el usuario trate de sacar una foto y no tenga dicho servicio activo.

- **activity_foto_ampliada.xml:** Este Layour permitirá ver en pantalla completa la foto en miniatura que se muestra tanto en el Layout activity_datos_ue_central.xml como en activity_diario_excavación.xml (UC33).

5.2.2 Package Controlador

En este apartado se presentarán cada una de las clases que componen el Package Controlador.

Se ha decido presentar la información separada en tres apartados que harán referencia a los tres tipos de controladores contemplados para la realización de esta aplicación y que pueden apreciarse en la figura 18.

- En primer lugar serán tratados los controladores relacionados con los diferentes Layouts.
- Posteriormente se hablará del Controlador propiamente dicho.
- Y por último, aquellos que trabajan en segundo plano y que son necesarios para conseguir una aplicación más ligera y rápida.

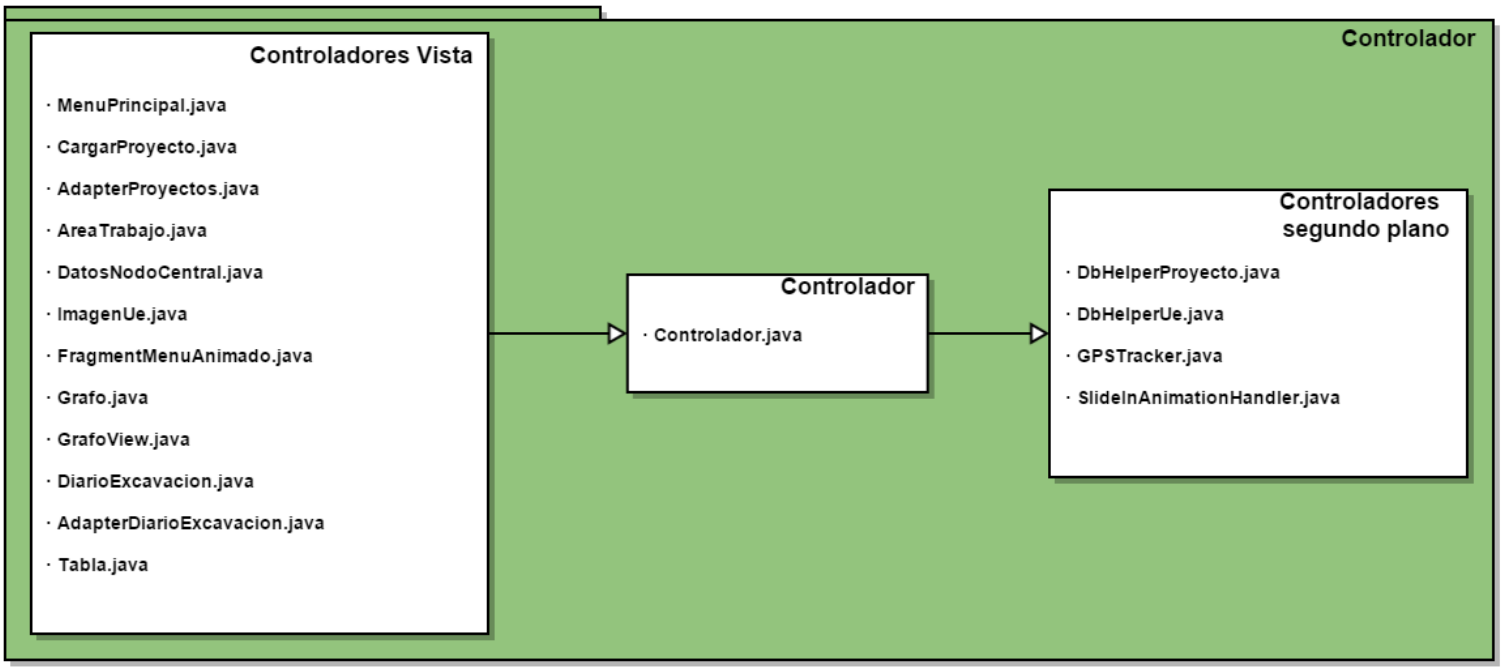

Ilustración 18 - Diagrama que muestra las diferentes clases que conformar el Package Controlador del patrón presentado.

Como ha podido apreciarse en la figura 18:

- Para controlar los diferentes Layouts se han implementado 11 clases controladoras de estos y un fragment.java.
- El Controlador, como tal, se coloca en el centro de la aplicación y es el que actúa como nexo entre los demás puntos de la aplicación.
- Y para los controladores en segundo plano ha sido necesario crear tres nuevas clases.

Todas ellas serán descritas, en los siguientes apartados, para una mejor comprensión de la estructura de la apliación.

5.2.2.1 Controladores de las diferentes vistas

A continuación serán citadas y descritas cada una de las clases .java que controlan los diferentes Layouts mencionados anteriormente:

- **MenuPrincipal.java:** Esta clase es la encargada de gestionar las diferentes acciones que posibilita el Layout acitivity_menu_principal.xml.

- **CargarProyecto.java:** Esta clase es la encargada de gestionar las diferentes acciones que posibilita el Layout activity_cargar_proyecto.xml.

- **AdapterProyectos.java:** Esta clase es la encargada de gestionar cada uno de los Layouts activity_adapter_proyecto.xml que representarán a cada uno de los proyectos almacenados.

- **AreaTrabajo.java:** Esta clase es la encargada de gestionar las deferentes acciones que posibilita el Layout activity_area_trabajo.xml.

- **DatosNodoCentral.java:** Esta clase es la encargada de gestionar las diferentes acciones que ofrece el Layout activity_datos_nodo_central.xml.

- **ImagenUe.java:** Esta clase es la encargada de gestionar las diferentes acciones que ofrece el Layout activity_foto_ampliada.xml.

- **FragmentMenuAnimado.java:** Esta clase es la encargada de gestionar las diferentes acciones que ofrece el Layout fragment_menu_animado.xml. Y permitirá configurar dicho menú agregándole colores, cambiándole su aspecto, su posición, así como capturando los eventos necesarios para su correcto funcionamiento.

- **Grafo.java:** Esta clase es la encargada de crear los elementos necesarios para que se obtenga el resultado esperado en el Layout activity_grafo.java.

- **GrafoView.java:** Esta clase es la encargada de dibujar el grafo dirigido que representa las diferentes UEs y sus relaciones en el Layout activity_grafo.xml así como gestionar las diferentes acciones que dicho Layout ofrece.

- **DiarioExcavacion.java:**Esta clase es la encargada de gestionar las diferentes acciones que ofrece el Layout activity_diario_excavación.xml.

- **AdapterDiarioExcavacion.java:** Esta clase es la encargada de gestionar cada uno de los Layouts adapter_diario_excavacion.xml que representarán a las diferentes UEs almacenadas.

- **Tabla.java:** Esta clase es la encargada de gestionar las diferentes acciones que ofrece el Layout activity_tabla.xml.

5.2.2.2 El controlador propiamente dicho

Esta clase es la que se encargará de hacer de controlador central de la aplicación. La clase que recogerá todas las peticiones de los diferentes controladores de los Layouts, las procesará y devolverá el resultado esperado.

5.2.2.3 Controladores en segundo plano

En este apartado se describirán las diferentes clases que se utilizan para gestionar los diferentes recursos necesitados para el correcto funcionamiento de la aplicación y que funcionan en segundo plano.

- **DbHelperProyecto.java:** Esta clase es la encargada de gestionar todas las acciones que ofrece la Base de datos Proyectos.

- **DbHelperUe.java:** Esta clase es la encargada de gestionar todas las acciones que ofrece la Base de datos Ues.

- **GPSTracker.java:** Esta clase es la encargada de obtener el punto GPS para ser asociado a una UE determinada.

- **SlideInAnimationHandler.java:** Esta clase es la encargada de gestionar las animaciones del menú flotante que presenta el Layout fragment_menu_animado.xml.

5.2.3 Package Modelo

En este apartado se describirán cada una de las clases que se encuentran dentro del package Modelo del patrón presentado y que pueden apreciarse en la figura 19.

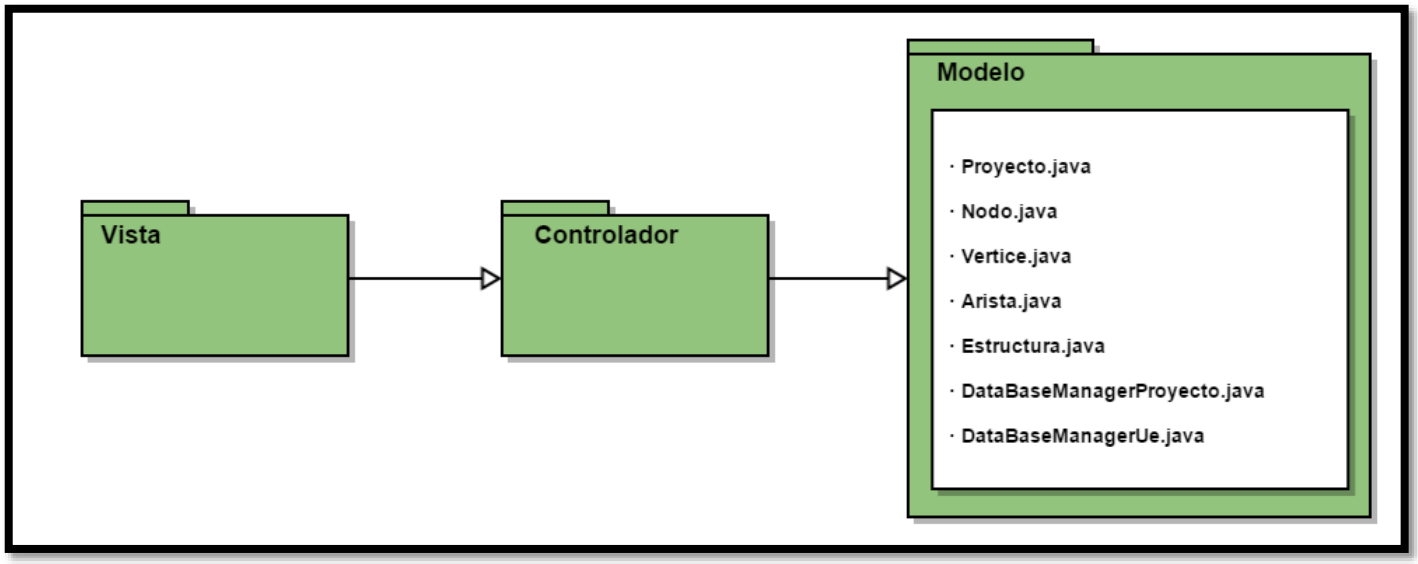

Ilustración 19 - Clases que componen el package Modelo.

Como se ha podido apreciar en la figura 19 ha sido necesario implementar 7 clases que compondrán el Package que nos ocupa y que serán descritos a continuación para una mejor comprensión de estos:

- **Proyecto.java:** Esta clase es la representará a un proyecto dentro de la aplicación.

- **Nodo.java:** Esta clase la que representará a una UE dentro de la aplicación.

- **Vertice.java:** Esta clase es la que representará a una UE gráficamente hablando.

- **Arista.java:** Esta clase es la que representará a una relación entre UEs gráficamente hablando.

- **Estructura.java:** Esta clase será la que representará al conjunto de UEs y sus relaciones.

- **DataBaseManagerProyecto.java:** Esta clase es la que define los diferentes parámetros que configuran la base de datos Proyectos. Por ejemplo, en ella se definirán el conjunto de tablas que formarán la base datos y la estructura que seguirán estas.

- **DataBaseManagerUe.java:** Esta clase es la que define los diferentes parámetros de la base de datos UE. Por ejemplo, en ella se definirán el conjunto de tablas que formarán la base datos y la estructura que seguirán estas.

5.3 Diagrama de clases

El diagrama de clases sirve para definir clasificadores en general, es decir, sea cual sea su estereotipo, pero es cierto que para determinados estereotipos se suelen utilizar otros diagramas. También se describen relaciones de diversas formas entre los clasificadores, que en muchos casos tienen un reflejo en cuanto a las instancias respectivas: relaciones de generalización, especialización, dependencias y un nuevo tipo de realizaciones, las asociaciones, y también se pueden definir instancias particulares de clasificadores definidos en el mismo diagrama o no y describir relaciones entre ellas.

Al igual que en la programación orientada a objetos, en el UML los conceptos de clase y de objeto son también importantes de cara a la modelización de aplicaciones, pero hay que señalar que se consideran así desde un punto de vista independiente de todo lenguaje de programación concreto.

Dentro del proceso de desarrollo de aplicaciones hay algunas etapas pero no todavía código, ahora bien, los requisitos, que representan las necesidades de información por parte de los futuros usuarios de la aplicación, por su naturaleza son independientes de todo lenguaje de programación.

Y cuando se modeliza el futuro código éste puede ser escrito en diferentes lenguajes orientados a objeto, por lo tanto, al menos cuando se quiera flexibilidad en este aspecto, conviene que no haya que recurrir a una notación distinta para cada uno, aunque obviamente conviene que en los diagramas afectados se respeten las restricciones en cuanto a nombres de los lenguajes en los que se puedan tener que programar, sea manualmente o con generación automatizada de código.

En el UML la clase es un estereotipo del clasificador, la mayoría de clases son instanciables y sus instancias se denominan objetos. Una clase puede ser abstracta, en el mismo sentido que un clasificador en general. El hecho de que un clasificador sea abstracto se puede señalar o bien poniendo su nombre en letra cursiva, o bien con la propiedad abstract. Ambas opciones valen también para una clase. [15]

Un objeto puede pertenecer a más de una clase, pero sólo de una de ellas se pueden crear objetos directamente: el resto de clases representan sólo papeles que puede desempeñar el objeto.

[15] [DG01] Capítulos: La utilización de las clases y La definición de las clases.

Un aspecto característico de los objetos es que tienen identidad, lo que significa que dos objetos creados por separado siempre se consideran diferentes aunque tengan los mismos valores a las características estructurales. En cambio las instancias de según qué clasificadores no tienen identidad y, entonces, dos instancias con los mismos valores a las características estructurales se consideran la misma.

Los objetos de las clases llamadas activas, u objetos activos tienen un hilo de control y comienzan la ejecución de su comportamiento en cuanto son creados, mientras que los comportamientos de los objetos del resto de clase, las clases pasivas (objetos pasivos), sólo se ejecutan dentro del hilo de control de otro objeto y deben ser puesto en marcha desde los comportamiento del mismo.

Antes de seguir con la descripción de los elementos que componen una clase serán presentadas, en la figura 20, el diagrama de clases que ha estructurado la aplicación.

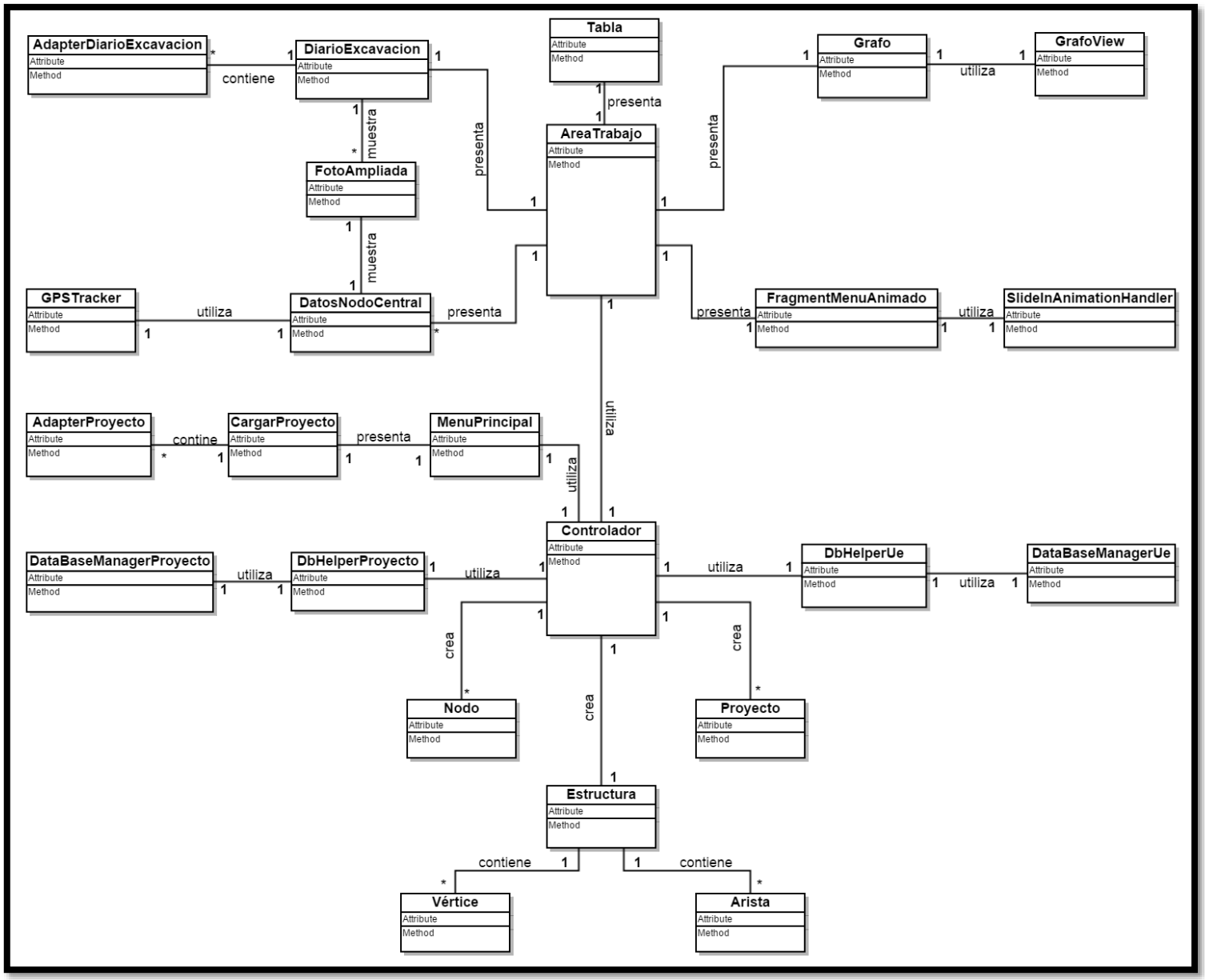

Ilustración 20 - Diagramas de clases de la aplicación AppQueology 2.0

Se ha decidido introducir aquí la figura 20 para una mejor comprensión del código, pero el real Diagrama de Clases con sus atributos y métodos será incluido en el Apéndice ya que por sus dimensiones no puede integrarse dentro de un texto.

Como clasificadores que son las clases tienen características estructurales y de comportamiento, que tienen nombre y definiciones específicos:

- Las características estructurales se denominan atributos. Todo atributo tiene asignado un tipo y puede tener una lista formada por un valor o más de este tipo, de acuerdo con la multiplicidad correspondiente que ha de pertenecer a este tipo y pueden variar a lo largo del tiempo. Un atributo de clase esta lista de valores es única para toda la clase, mientras que en el caso de un atributo de objeto, el más típico y frecuente, varía de un objeto a otro. Los valores de los diferentes atributos de un objeto constituyen el estado del objeto.

- Las características de comportamiento se denominan operaciones. En UML existe el término método, que se aplica a la descripción de la implementación de una operación. Cada operación tiene una firma que describe los eventuales parámetros y un valor de retorno. Los parámetros tienen nombre, tipo, multiplicidad y dirección (que indica si el parámetro es de entrada, de salida o de entrada y salida a la vez). Una operación de clase no tiene objeto de contexto sino clasificador de contexto, que es su clase.

5.4 Diagrama general de las vistas de la aplicación

La interfaz de una aplicación es como la ropa que viste para salir a la calle. Es también la capa que hay entre el usuario y el corazón funcional de la aplicación, el lugar donde nacen las interacciones.

En mayor medida está compuesta por botones, gráficos, iconos y fondos, que tiene una apariencia visual diferente en cada uno de los sistemas operativos, ya que Android, iOS y Windows Phone tiene su propia forma de entender el diseño.

En este punto se ha de interpretar la personalidad de cada sistema operativo, aportando su propia visión y estilo de diseño, para conseguir aplicaciones que, además de ser fáciles de usar, sean distintas a las demás y tengan coherencia visual con la plataforma que las acoge.

El diseño en Android está basado en una pulcritud brillante en la composición de la interfaz.

Cada gráfico, botón y texto está acompañado por la idea de limpieza visual pero, a la vez, deslumbra con pequeños detalles.

Así, para este trabajo, se ha optado por la siguiente estructura para la interfaz de la aplicación y que puede apreciarse en la figura 21.

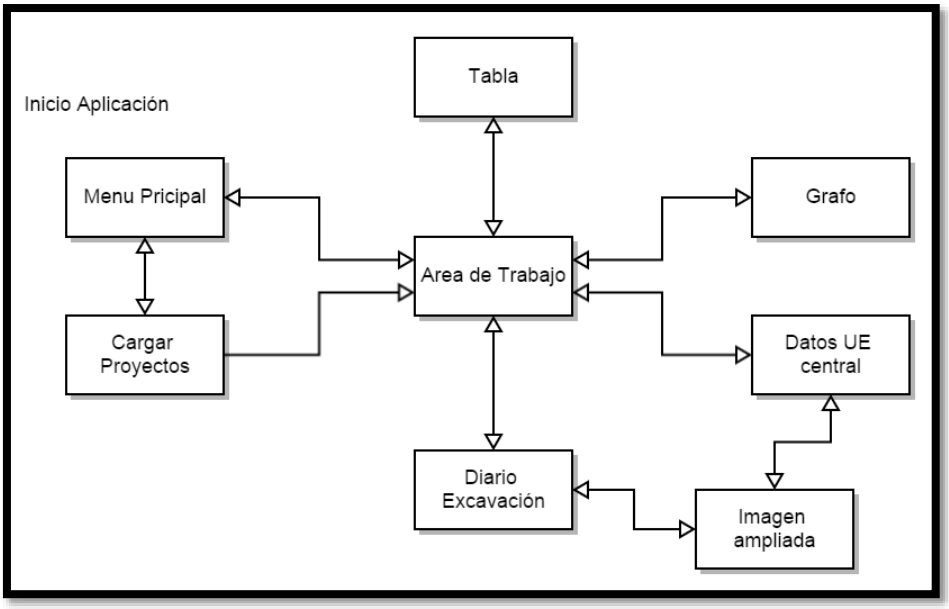

Ilustración 21 - Diagrama de flujo de las vistas de la aplicación.

Como puede apreciarse en la figura 21, la aplicación inicia con la vista Menú Principal que corresponde con el Layout activity_menu_principal.xml. Y como la vista Área de Trabajo, correspondiente con el Layout activity_area_trabajo.xml, actúa como centro de la aplicación y como nexo entre las demás vistas. Por otro lado a continuación se muestran, en la figura 22, un diagrama con las capturas de las principales vistas que presenta la aplicación para una mejor comprensión del funcionamiento de la aplicación:

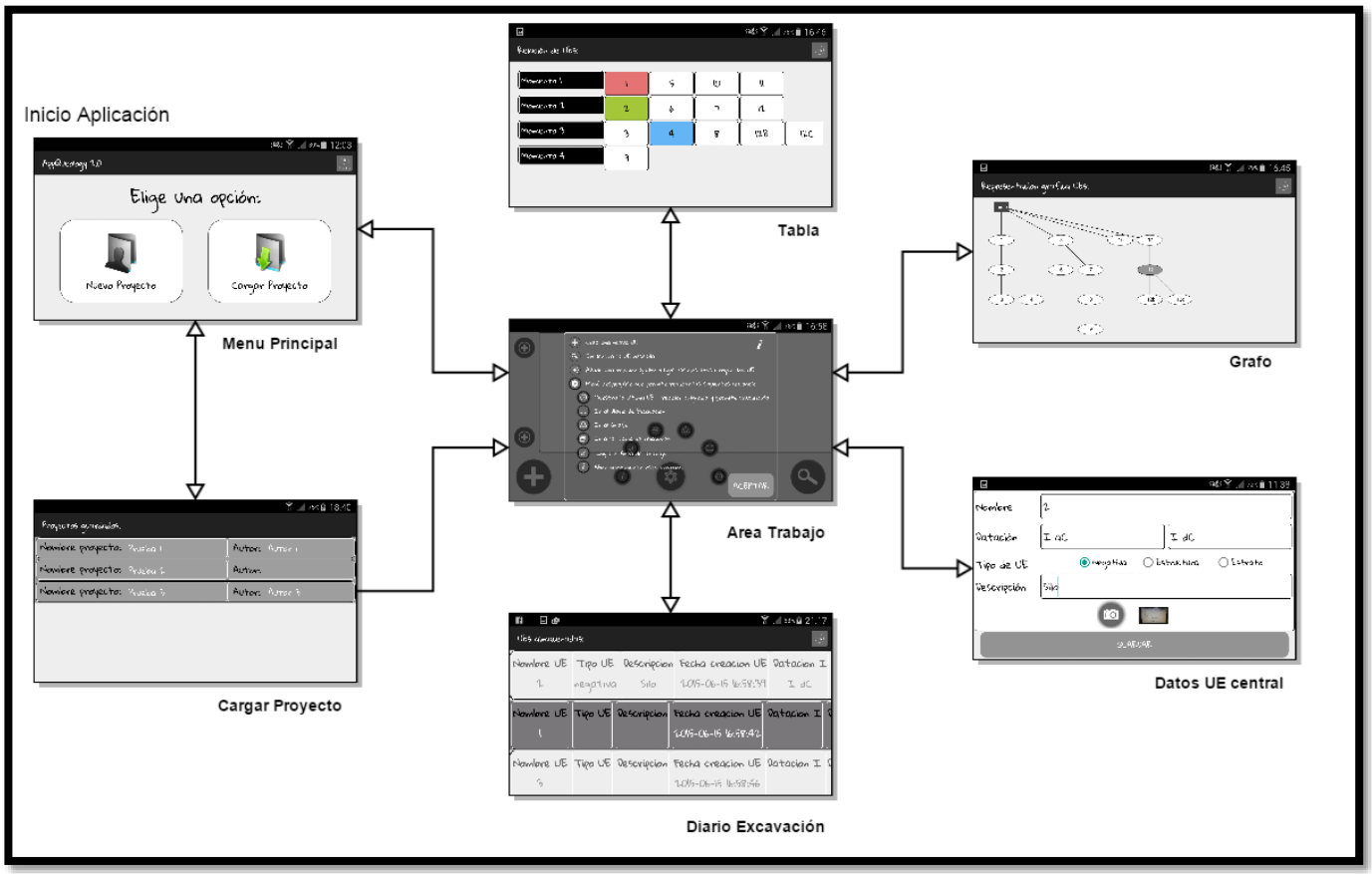

Ilustración 22 - Diagrama de flujo de las vistas de la aplicación.

En la figura 22 puede apreciarse una diferencia respecto a la figura 21 y es que no aparece la vista Imagen Ampliada, esto se debe a que se ha entendido que no tiene mucho sentido mostrar una captura que únicamente presentara una imagen ampliada y nada más, que es lo que muestra dicho Layout. Por lo que se ha optado por omitirla para una mejor comprensión de la estructura.

5.5 Diagrama de la base de datos

Una base de datos correctamente diseñada permite obtener acceso a información exacta y actualizada. El proceso de diseño de una base de datos se guía por algunos principios.

El primero de ellos es que se debe evitar la información duplicada, o lo que es lo mismo, los datos redundantes, porque malgastan el espacio y aumentan la probabilidad de que se produzcan errores e incoherencias.

El segundo principio es que es importante que la información sea correcta y completa. Si la base de datos contiene información incorrecta, los informes que recogen información de la base de datos contendrán también información incorrecta y, por tanto las decisiones que tome a partir de esos informes estarán mal fundamentadas.[16]

Un buen diseño de base de datos es, por tanto, aquél que:

- Divide la información en tablas basadas en temas para reducir los datos redundantes.
- Ayuda a garantizar la exactitud e integridad de la información.
- Satisface las necesidades de procesamiento de los datos y de generación de informes.

Así para la realización de este proyecto se han creado diferentes bases de datos SQLite, que se almacenan de forma local en el dispositivo.

Se ha realizado el diseño de este modo pensando en la posibilidad de que en un futuro, tras la entrega de este proyecto, se implemente la exportación de la base de datos.

Y para no tener que exportar una base de datos con toda la información de las UEs almacenadas de todos los proyectos, se han creado una base de datos que almacena los proyectos y una base de datos de UEs y relaciones para cada proyecto.

Así en la figura 23, presentada a continuación, podrá apreciarse la estructura de las bases de datos implementadas.

[16] [AHS06] – Capítulo: Introducción.

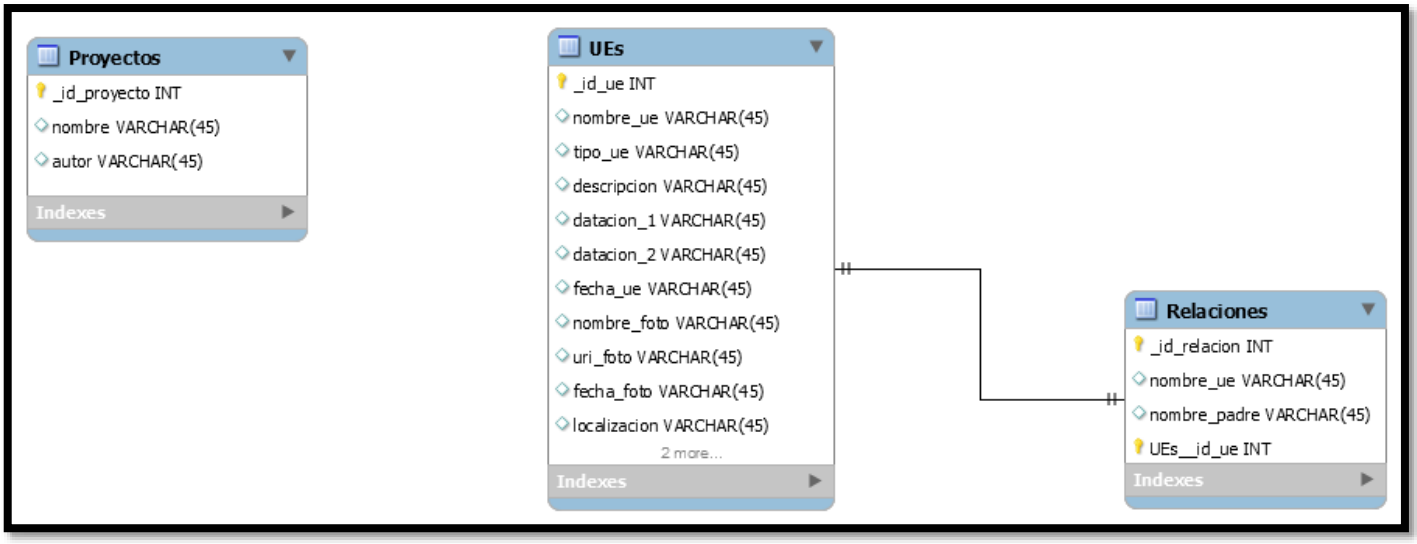

Ilustración 23 - Diagrama de las Bases de datos implementadas

Como puede apreciarse en la figura 23, la Tabla Proyectos presenta los siguientes atributos:

- La id o identificador del Proyecto en cuestión.
- Su nombre.
- Y el autor del mismo.

También como la tabla UEs presenta las siguientes columnas:

- Id o identificador de la UE.
- Nombre de la UE.
- Tipo de UE (estructura, negativa o estrato).
- Una descripción de la UE.
- Dos campos para la datación.
- Una fecha de creación de cada una de las UEs almacenadas.
- Un campo destinado al nombre de la foto que puede asociarse, como ha podido verse en puntos anteriores de esta memoria, a una UE determinada.
- El uri de la foto que indicará el lugar donde se encuentra almacenada.
- Un campo destinado a la fecha de creación de la foto en cuestión.
- Y por último, otro destinado a albergar un punto del GPS con sus componentes latitud y latitud.

La figura 23 también nos permite ver como la tabla Relaciones contempla los siguientes aspectos:

- Id o identificador de cada una de las Relaciones almacenadas.
- Un nombre que corresponde con la UE que se encontrará en uno de los extremos de la relación. Y que se situará en una posición de anterioridad respecto a la que le sigue.
- Otro nombre que representará al otro extremo de la relación y que jugará el papel de hijo o de Unidad Estratigráfica posterior.

Se han definido como atributos “UNIQUE” y “NOT NULL” tanto el nombre del proyecto, en la tabla Proyectos de la base de datos Proyectos, como el nombre de la UE en la tabla UEs de la base de datos con el nombre del proyecto en cuestión.

También se ha establecido en la tabla UEs la fecha de la UE y la de la foto como “DATETIME DEFAULT CURRENT_TIMESTAMP”, de este modo se recogerán automáticamente los valores correspondientes.

La fecha de la foto se utilizará también como nombre de la foto precedida de PIC_.

Los aspectos más concretos referentes al funcionamiento interno de la base de datos serán tratados en puntos posteriores, cuando se analicen las diferentes clases que componen el proyecto.

5.6 Algoritmos implementados

En matemáticas, lógica, ciencias de la computación y disciplinas relacionadas, un algoritmo es un conjunto prescito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución.

Así en este apartado se describirán los algoritmos implementados más importantes.

5.6.1 BFS (Breadth First Search)

Este algoritmo permite recorrer en anchura el objeto Estructura anteriormente descrito. Ha sido implementado en un momento inicial del desarrollo de la aplicación para recorrer la estructura mencionada. Pero finalmente ha tenido que ser substituido por los algoritmos descritos a partir del punto 5.6.3. Se ha decido conservar dentro del código pensando en que podría ser útil en un momento posterior si es que se continúa ampliando la aplicación una vez entregado el Proyecto que nos ocupa.

A continuación serán descritos en pseudocódigo los puntos fundamentales que estructuran este algoritmo.

- Se marcan todos los nodos como no visitados.
- Se crea un objeto del tipo LinkedList (L).
- Se obtiene el primer elemento de la lista de vértices.
- Se agrega a L
- Se recorre L mientras no se encuentre vacía:
    - Se saca el primer elemento de L (Ln)
    - Se marca como visitado.
    - Se elimina Ln de L.
    - Se crea L2.
    - Se agrega a L2 los descendientes de Ln.
    - Se recorren los elementos de L2.
        - Se obtiene el primer elemento (L2n) de L2.
        - Si no se ha visitado:
            - Se agrega a L.

- Se marca como visitado en L2.

5.6.2 DFS (Depth First Search)

Este algoritmo permite recorrer en profundidad el objeto Estructura descrito en puntos anteriores. Ha sido implementado en un momento inicial del desarrollo de la aplicación para recorrer la estructura mencionada. Pero finalmente ha tenido que ser substituido por los algoritmos descritos a partir del punto 5.6.3. Se ha decido conservar dentro del código pensando en que podría ser útil en un momento posterior si es que se continúa ampliando la aplicación una vez entregado el Proyecto que nos ocupa.

A continuación serán descritos en pseudocódigo los puntos fundamentales que estructuran este algoritmo.

- Se marcan todos los vértices como no visitados.
- Se crea un Stack (S).
- Se agrega a S el vértice inicial.
- Mientras S no se encuentre vacío:
    - Se recupera el primer elemento (Sn) de S
    - Se recupera un hijo no visitado de Sn
    - Si Sn_hijo es diferente de null:
        - Sn_hijo se marca como visitado.
        - Sn_hijo agrega a S.
    - Si Sn_hijo es igual a null:
        - Se saca otro elemento de S.

5.6.3 Algorimto esCiclo

Este algoritmo comprueba si al crear una nueva relación se genera un ciclo.

- Se marcan todos los nodos como no visitados.
- Se busca el padre (w) del nuevo nodo que se quiere insertar (v) y se almacena en un LinkedList (L1).
- Se recorre L1:
    - Si el elemento (L1n) no se encuentra visitado:
        - Se almacena en un Array (A). Dicho Array (A) será utilizado para mostrar los nodos implicados en el ciclo, en el caso de producirse este.
            - Si “L1n” es igual a “v”:
                - Se considera que “v” genera un ciclo.
            - Si “L1n” y “v” son diferentes:
                - Se crea un nuevo LinkedList (L2) con el padre/s de “L1n”
                - Se recorre “L2”. Cada elemento de L2 se almacena al final de “L1”
    - Si el elemento se encuentra visitado:
        - No se realiza ninguna acción. Se pasa al siguiente nodo de “L1”.

5.6.4 Algoritmo para la eliminación de relaciones redundantes

Este algoritmo se encarga de eliminar las relaciones redundantes del objeto Estructura descrito en puntos anteriores.

Primera parte: (Variante DFS)

- Se marcan todos los nodos como no visitados.
- Se crea un Stack (pila -> último en entrar, primero en salir).
- Se introduce el nodo raíz en la pila.
- Mientras la pila no se encuentre vacía hacer:
    - Se trata el último nodo introducido en la pila.
    - Se buscan los hijos del nodo a tratar no visitados desde el nodo a tratar:
        - Si tiene hijos:
            - Se introducen en la pila.
            - Se marca como visitado desde el nodo a tratar.
        - Si no tiene:
            - Se saca el elemento a tratar de la pila.

Segunda parte: (Eliminación de las aristas redundantes)

Al haber marcado cada nodo (v) como visitado desde un nodo en concreto (w), se han creado cada uno de los caminos para llegar a dicho nodo (v).

Ahora es ir comparando los caminos para ver si existe alguno considerado como redundante.

Ejemplo:

Camino A: [nodo_1, nodo_2, nodo_3, nodo_4, nodo_5]

Camino B: [nodo_1, nodo_5]

Camino C: [nodo_1, nodo_6, nodo_5]

A continuación se analizarán dos casos diferentes siguiendo el algoritmo en cuestión. Su representación gráfica puede observarse en la figura 24:

- En primer lugar, se analizará el **"Camino B"** que será considerado como **redundante**.

- En segundo lugar, se tratará el **"Camino C"** que **no** será considerado como **redundante**.

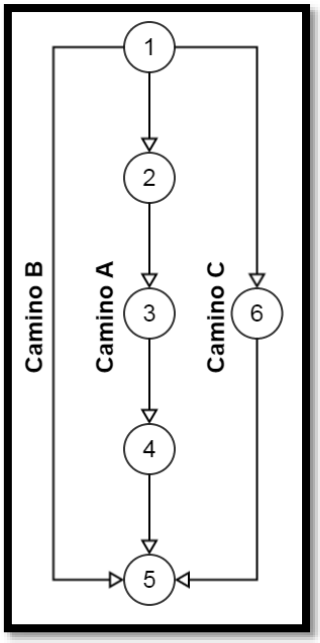

Ilustración 24 - Ejemplo de redundancia.

Como puede apreciarse en la figura 24, suponiendo que nos encontramos analizando el nodo_5 (común en los tres caminos).

- Vemos si el inmediatamente anterior al 5 en el "Camino B" (el 1) se encuentra en el "Camino A". Como es así se considera el "Camino B" como redundante.

- Vemos si el inmediatamente anterior al 5 en el "Camino C" (el 6) se encuentra en el "Camino A". Como esto no es así no se considera el "Camino C" como redundante.

5.6.5 Algoritmo para calcular la altura del vértice dentro de la Estructura

Este algoritmo permitirá saber a qué altura se encuentra el vértice recibido por parámetro (v) dentro de la Estructura que representa las UEs y sus relaciones.

- Si v es igual a v_inicio:
  - Return 0.
- Else:
  - Se crea un contador e inicializa a 0.
  - Se marcan todos los vértices como no visitados.
  - Se crea un Stack (S)
  - Se agrega v_inicio a S.
  - Se marca como visitado v_inicio.
  - Se recorre el stack:
    - Se saca un elemento (Sn)
    - Se obtiene un hijo no visitado (Sn_hijo) de (Sn).
    - Si Sn_hijo diferente de null:
      - Si v es igual a Sn_hijo:
        - Se incrementa el contador.
        - Return contador.
      - Si no son iguales:
        - Se marca como visitado Sn_hijo.
        - Se agrega Sn_hijo a S.
    - Sino es igual a null:
      - Se saca un elemento de S.
      - Contador --
  - Return contador.

5.6.6 Algoritmo para calcular Momentos

Este algoritmo se utiliza en la clase Controlador.java y servirá para mostrar las UEs organizadas en función de si son contemporáneas (Momento) y no por sus relaciones físicas.

A continuación se mostrará, en la figura 25, un ejemplo de un posible caso real:

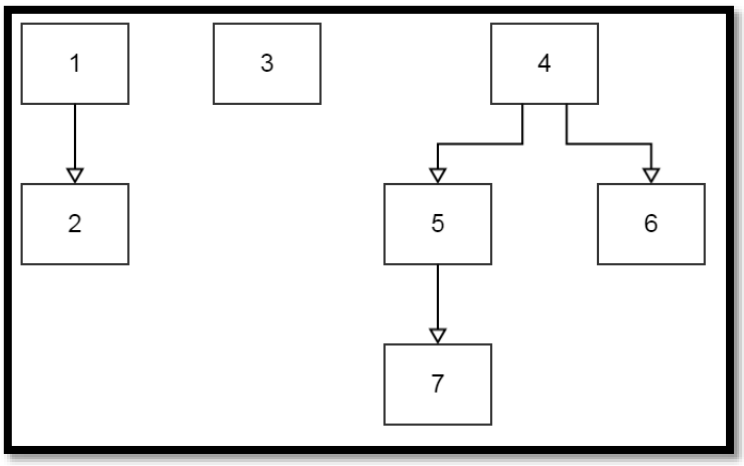

Ilustración 25 - Ejemplo de momentos.

Según este ejemplo de la figura 25, existirían los siguientes momentos:

- Momento 1: UE1, UE3 y UE4.
- Momento 2: UE2, UE5 y UE6.
- Momento 3: UE7.

Según esto se ha implementado el siguiente algoritmo:

- Se crea un ArrayList<ArrayList> momentos.
- Se marcan todos los vértices como no visitados.
- Se recorren los vértices:
    - Se marca como visitado el vértice n.
    - Se calcula el momento del vértice dentro de la Estructura.
    - Si no existe un ArrayList<String> para ese grado.
        - Se crea.
        - Se agrega dicho vértice a ese ArrayList.
        - Se agrega ese ArrayList a momentos.
    - Si ya existe:
        - Se agrega al ArrayList correspondiente de momentos.
- Return momentos.

5.6.7 Algoritmo calcular dimensiones View

Este algoritmo es utilizado en la clase GrafoView.java y se encarga de calcular las dimensiones que deberá tener la View para que albergue toda la figura dibujada en el Canvas y que representa al conjunto de UEs y relaciones.

Así el algoritmo será el siguiente:

- Se obtienen los momentos.

Para calcular el ancho:

- Se comprueba cual es el ArrayList de los momentos de mayor longitud.
- Return el mayor tamaño.

Para calcualar el alto:

- Se calcula el número de ArrayList que configuran el ArrayList<ArrayList> momentos.
- Return tamaño.

El ancho será = (500 * ancho máximo) ^ ancho máximo.
El alto será = 300 por alto_máximo.

5.6.8 Algoritmo buscar una UE en la vista activity_grafo.xml

Este algoritmo se utiliza en la clase Grafo.java y sirve para a medida que se dibuja la figura que representa gráficamente el conjunto de UEs y sus relaciones encontrar la UE cargada por el usuario en otra vista. También se utilizará en la misma clase para remarcar la UE seleccionada por el usuario.

- Mediante el método onTouch() se obtiene la X y la Y que representan los puntos donde ha sido tocada la pantalla.
- Posteriormente cuando se está dibujando la figura en el método onDraw()
    - Se comprueba si el Rectf que se va a dibujar contiene dichos puntos.
    - Si los contiene:
        - Se le modifica su aspecto.

5.6.9 Algoritmo para eliminar UEs

Este algoritmo se utiliza en la clase AdapterDiarioExcavacion.java y sirve para eliminar UEs de la base de datos.

- Se elimina de la base de datos la UE en cuestión.
- Si se produce un error en la eliminación:
    - Se lanza un Toast avisando en el usuario de que se ha producido una inconsistencia en el proyecto.
- Si no se produce un error:
    - Se actualiza la vista.
    - Se obtienen todas las relaciones.
    - Se recorren las relaciones:
        - Se eliminan las relaciones de la base de datos donde aparezca dicha UE eliminada.
        - Si se produce un error en la eliminación de la relación:
            - Se lanza un Toast avisando al usuario de que se ha producido una inconsistencia en el proyecto.

5.6.10 Algoritmo para eliminar relaciones

Este algoritmo se utiliza en la clase AreaTrabajo.java y sirve para eliminar relaciones.

- Se obtienen todas las relaciones.
- Se recorren las relaciones.
- Se comprueba si alguna relación es la eliminada.
- Si lo es:
    - Se elimina de la base de datos correspondiente.
    - Si no se produce un error:
        - Se actualiza la vista.
        - Se vuelve a recorrer las relaciones.
        - Si algún nodo no depende de ningún otro, tras haber realizado la eliminación, pasa a depender del nodo raíz.
    - Si se produce un error:
        - Se lanza un Toast avisando al usuario de que hay inconsistencias en el proyecto.

5.6.11 Algoritmo para dibujar los vértices del grafo

Este algoritmo, como su nombre indica, se utiliza para dibujar las diferentes UEs y sus relaciones en el Canvas de la vista acitivity_grafo.xml.

- Se marcan todos los vértices como no visitados.
- Se crea un Stack (S).
- Mientras no se encuentre vació S:
    - Se trata Sn.
    - Se recupera un hijo no visitado (v) de Sn.
    - Si v diferente de null:
        - Se crea un RectF (R) con un factor de ancho y alto para su posición.
        - Se agrega R a un ArrayList<RectF> (Array).
        - Se incrementa el factor alto + 300.
        - Se agrega v a S.
        - Boolean (control) igual a true.
    - Si v es igual a null:
        - Factor alto – 300.
        - Si control:
            - Factor ancho + 300.
            - control igual a false.
        - Se saca un elemento de S.
- Return array.

6. Implementación

Una implementación o implantación es la realización de una aplicación, o la ejecución de un plan, idea, modelo científico, diseño, especificación, estándar, algoritmo o política…

En ciencias de la computación, una implementación es la realización de una especificación técnica o algoritmos como un programa, componente software, u otro sistema de cómputo.

Por tanto, en esta fase del trabajo se ha obtenido el código de la aplicación a partir de la documentación obtenida durante las fases de análisis y diseño.

Así en este punto, se mostrarán los aspectos más importantes de la implementación para cada una de las clases desarrolladas. Con ello se pretende dar a conocer al lector como se han conseguido cumplir o satisfacer los requisitos obtenidos en la fase correspondiente. Y que pasos han sido necesarios realizar para la consecución de los resultados esperados.

6.1 Aspectos introductorios

Como se ha comentado, para el desarrollo de esta aplicación se ha utilizado el patrón Modelo-Vista-Controlador. Debido a esto, se ha optado por presentar las diferentes clases implementadas siguiendo este modelo.

Pero antes de empezar con la descripción mencionar que en todas las clases se han declarado como privados, tanto los atributos de las diferentes clases, como los métodos que únicamente son utilizados dentro de la clase.

También se ha procurado encapsular en diferentes funciones el código para un mejor mantenimiento de este. Así por ejemplo, todas las clases constarán de un método "inicialización" donde, como su nombre indica, se inicializarán las variables o elementos necesarios.

También se han ordenado los métodos por categorías para una mejor comprensión del código. Un ejemplo de esto sería el que se puede apreciar en la figura 26.

```
import ...

public class MenuPrincipal extends ActionBarActivity {

    /*-------------------------------------------------------------------------------------------*/

    Atributos de la clase

    /*-------------------------------------------------------------------------------------------*/

    Constructor de la Activity e inicializacion

    /*-------------------------------------------------------------------------------------------*/

    Capturando eventos

    /*-------------------------------------------------------------------------------------------*/

    Alerts

    /*-------------------------------------------------------------------------------------------*/
}
```

Ilustración 26 - Ejemplo de la clase MenuPrincipal.java

Como se puede apreciar en la figura 26, se muestra un apartado referente a los Atributos de la clase, otro que corresponde al constructor de la actividad y los métodos necesarios para inicializar la clase. Otro destinado a la captura de los eventos que dicha clase permite y relacionados tanto con su ciclo de vida como con los diferentes elementos que componen su Layout. Y por último, uno destinado a albergar los métodos que tratan los Alerts que dicha clase contiene, así como la personalización de la apariencia de dichos Dialogs.

Por último, mencionar, que se ha intentado que todo el proyecto presente la misma apariencia para que mantenga una estética coherente y sea entendido como un todo.

6.2 Clases

Una clase es una plantilla de la que se crean los objetos individuales.

Un objeto es la entidad existente en la memoria del ordenador que tiene unas propiedades (atributos o datos sobre sí mismo almacenados por el objeto) y unas operaciones disponibles específicas (métodos).

Así en este apartado se describirán las diferentes clases implementadas para este proyecto. Se empezarán con los Layouts y sus clases controladoras correspondientes, para posteriormente pasar a las clases comprendidas en el package Controlador. Y por último, serán tratadas aquellas relacionadas con el Modelo de la aplicación.

6.2.1 Layouts y controladores de dichos Layouts

A continuación, y como se ha comentado se comentarán los diferentes Layouts xml y sus correspondientes controladores .java:

6.2.1.1 MenuPrincipal.java – activity_menu_principal.xml

El aspecto de su Layout, anteriormente mencionado puede apreciarse en la figura 27.

Ilustración 27 - Captura de la activity_menu_principal.xml

Como se puede apreciar, en la figura 27, este Layout nos presenta tres botones que permitirán al usuario:

- Consultar información sobre los desarrolladores de la aplicación - (UC1).
- Crear un nuevo proyecto - (UC2).
- Enlazar con la clase CargarProyecto.java y su Layout correspondiente "activity_cargar_proyecto.sml" - (UC3).

MenuPrincipal.java, extiende de la clase ActionBarActivity.

Para versiones antiguas del Sistema Operativo, Google había maltratado un poco este componente, ya que no lo había incluido en la librería de compatibilidad Android-suport, lo que

hacía que no fuera compatible con versiones de Android anteriores a la 3.0. Esto obligaba a recurrir a librerías externas como ActionBarSherlock para hacer nuestra aplicación compatible con cualquier versión de Android. Pero, afortunadamente, a partir de julio del 2013, ya se puede ver debutar al ActionBar de serie en la librería de compatibilidad de Android.

Para que el ActionBar presente este aspecto ha sido necesario personalizarlo. Por ejemplo, eliminando la sombra que por defecto se nos muestra bajo la barra. También, entre otras cosas, cambiarle el tipo de letra, el color, etc. Con esto aprovecho para mencionar que para todo el proyecto se ha utilizado el tipo de letra "gloriahalleluja.ttf".

Para poder usar, en un proyecto de Android Studio, un tipo de letra que no viene predefinido por defecto, es necesario crear la carpeta "assets", de no existir, e introducir ahí el archivo con extensión .ttf correspondiente.

En este componente, como puede verse en la ilustración 25, se ha introducido un botón que permite mostrar información acerca de los desarrolladores de la aplicación.

Para poder introducir este botón vía código, ha sido necesario utilizar el método onCreateOptionsMenu(Menu menu). En él se le indica a la barra, que debe inflarse con el contenido del archivo menú_main.xml almacenado en el recurso "menu".

A continuación, en la figura 28, se muestra el código utilizado en este xml y que servirá a modo de ejemplo para explicaciones futuras en los que aparezca este tipo de elemento.

```
1  <menu
2      xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      tools:context=".MainActivity">
6
7      <item
8          android:id="@+id/action_info"
9          android:icon="@drawable/about"
10         android:title="Informacion"
11         android:orderInCategory="100"
12         app:showAsAction="ifRoom"/>
13
14 </menu>
15
```

Ilustración 28 - menu_main.xml

Únicamente destacar, de los datos que presenta la figura 28, que para que aparezca el icono directamente en el ActionBar hay que introducir la siguiente línea: app:showAsAction="ifRoom".

Para capturar el evento correspondiente al botón que nos ocupa, ha sido necesario sobrescribir el método onOptionsItemSelected(MenuItem ítem) e utilizar un onClickListener.

En el momento en el que el usuario utilice este botón se lanzará un Alert que permitirá mostrar la información correspondiente.

Para personalizar este tipo de AlertDialog ha sido necesario diseñar el Layout “activity_dialogo_about_us.xml”, que permite que el Alert presente el aspecto que se puede apreciar en la figura 29:

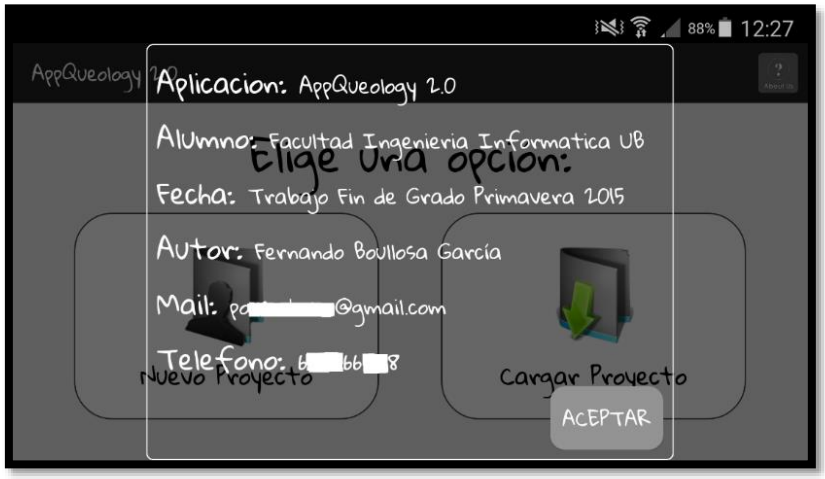

Ilustración 29 - Captura del AlertDialog "activity_dialogo_about_us".

Como puede apreciarse en la figura 29, presenta un fondo transparente. Para conseguir este efecto se ha utilizado el tipo de Color Transparent que ya trae definido la clase Color por defecto.

También puede observarse como se ha personalizado el botón y el contorno del LinearLayout con bordes redondeados. Para ello se ha predefinido en el recurso “drawable” una serie de “drawables resources files”.

A continuación, en la figura 30, se mostrará el código de uno de los xml implementados para comprender mejor lo que se está explicando y también a modo de ejemplo para explicaciones futuras:

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <shape
4      xmlns:android="http://schemas.android.com/apk/res/android"
5      android:shape="rectangle">
6
7      <stroke
8          android:width="1dp"
9          android:color="#ffffff" />
10
11     <corners
12         android:radius="5dp" />
13
14 </shape>
```

Ilustración 30 - Codigo de ejemplo de un Drawable resource file.

Como puede apreciarse en la figura 30, se tratan aspectos como el color, el stroke (el borde) y el radious corners (grado de curvatura de las esquinas)” entre otras cosas.

Además de lo anteriormente comentado, el Alert presenta un botón que permite cerrar el Dialog cuando el usuario lo desee. Para todos los diálogos de la aplicación se ha anulado el botón "back" del dispositivo, obligando al usuario a utilizar los botones que presente el Alert en cuestión. El evento de dicho botón ha sido capturado con un onClickListener.

Como se observaba en la ilustración 25, el Layout nos ofrece también un botón "Crear Proyecto" que, como su nombre indica, permite al usuario crear un nuevo proyecto. Para captura su evento correspondiente se ha utilizado un onClickListener.

Si el usuario lo acciona se lanzará un AlertDialog que presenta el aspecto que puede apreciarse en la figura 31.

Ilustración 31 - Captura del AlertDialogo que permite crear un nuevo proyecto.

Para que presente esta apariencia, que puede apreciarse en la figura 31, ha sido necesario diseñar el siguiente Layout: "activity_dialogo_nuevo_proyecto.xml".

Como puede apreciarse presenta dos EditText que permitirán al usuario introducir el nombre del proyecto y su autor. En este punto, comentar que se ha puesto la restricción de que no pueda crearse un proyecto sin que, el usuario, le asigne un nombre previamente. En caso de que el actor accione el botón "Aceptar" sin que haya introducido el nombre del proyecto anteriormente, se cerrará el diálogo y se le informará por medio de un Toast que dicha acción no ha podido realizarse por esta razón.

Si el usuario realiza las acciones pertinentes se creará el proyecto en cuestión y se pasará por medio de un Intent al Layout "activity_area_trabajo.xml" y a su clase java controlador correspondiente "AreaTrabajo.java" – (UC6).

Por el contrario, si el usuario clica sobre el botón "Cargar Proyecto", también por medio de un Intent, se pasará al Layout "activity_cargar_proyecto.xml" y a su clase controladora correspondiente "CargarProyecto.java" - (UC3).

Como clase inicial, MenuPrincipal permanecerá abierta hasta que se finalice la aplicación. Ya que siempre se podrá volver a ella por medio del botón "back" del dispositivo. Con esto, se le permite al usuario cambiar el proyecto con el que pretende trabajar.

6.2.1.2 CargarProyecto.java – activity_cargar_proyecto.xml

CargarProyecto.java extiende de ActionBarActivity.

Se encarga de:

- Solicitarle a la clase Controlador.java todos los proyectos guardados en la base de datos.
- Crear un objeto del tipo AdapterProyectos al que le pasara el ArrayList<Proyecto> con todos los proyectos.
- Añadirá al ListView que contiene su Layout "activity_cargar_proyecto.xml" el objeto del tipo AdapterProyectos.

Únicamente comentar de su Layout, además de lo ya mencionado, que presenta, al ser un ListView, un scroll vertical por defecto.

6.2.1.3 AdapterProyectos.java - activity_adapter_proyecto.xml

Esta clase extiende de ArrayAdapter>Proyecto> y será la encargada de ir creando un ítem para cada uno de los proyectos que recibe su constructor por parámetro.

A través del método getView(int position, View view, ViewGroup parent) se infla el contexto, es decir, su activity padre -> activity_cargar_proyecto.xml con el Layout activity_adapter_proyectos.xml.

Este último xml presenta el aspecto, que puede apreciarse en la figura 32, antes de adaptar su apariencia a la del resto del proyecto, tipo de letra, color,…:

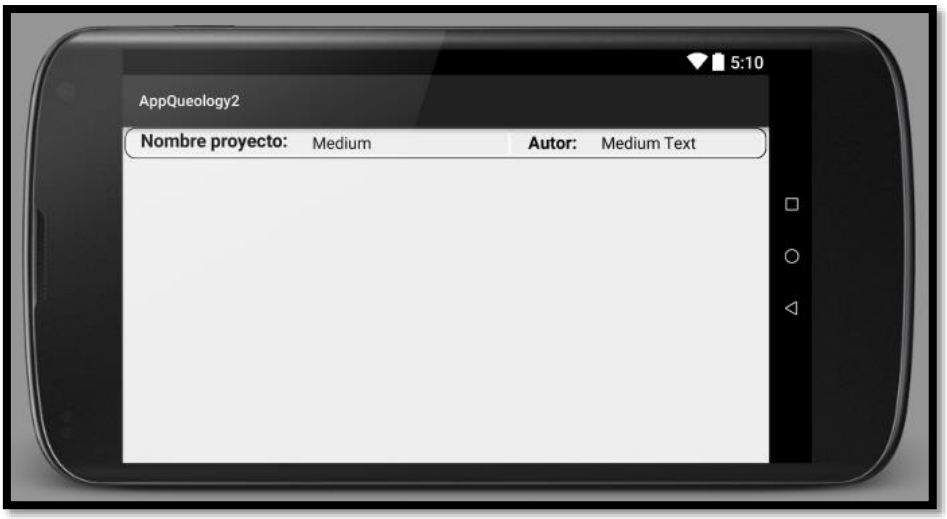

Ilustración 32 - Captura del Layout activity_adapter_proyecto.xml.

La figura 32 nos muestra como dicha vista contempla el nombre y el autor del proyecto. Elementos que, además del identificador del proyecto, constituyen las columnas de la tabla Proyecto de la base de datos correspondiente y descrita en apartados anteriores.

A continuación, en la figura 33, un ejemplo de cómo sería el aspecto del Layout activity_cargar_proyecto.xml con los proyectos cargados:

Ilustración 33 - Captura del Layout activity_cargar_proyecto.xml

También ha sido necesario capturarle a cada uno de los Views que serán añadidos al Layout padre, que pueden apreciarse en la figura 33, y que corresponden con cada una de las filas que lo componen, los eventos correspondientes. Así se utilizará:

- Un onClickListener para capturar el evento que representará a seleccionar y cargar un proyecto.
- Un onLongClickListener para capturar el evento que permitirá seleccionar el proyecto a eliminar.

Si el usuario selecciona y carga un proyecto se cerrará esta Activity y se le enviará por medio de un Intent al Layout "activity_area_trabajo.xml" – (UC6).

En caso de quererse eliminar un determinado proyecto se le pedirá la confirmación de la acción, por medio de un AlertDialog, al usuario. Este Alert inflará el contexto padre con el Layout "activity_dialogo_confiramar_eliminar_proyecto.xml" y presenta el aspecto que puede apreciarse en la figura 34.

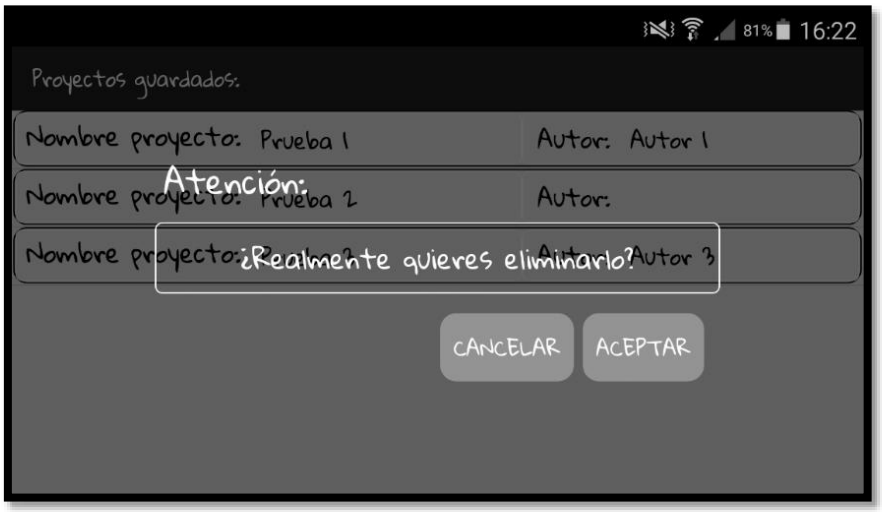

Ilustración 34 - Captura pantalla del AlertDialog que solicita confirmación para eliminar un proyecto seleccionado.

En la figura 34 puede apreciarse como se presentan dos botones que permitirán confirmar la eliminación del proyecto en cuestión por parte del usuario, o por el contrario, cancelar la acción.

En caso de que se produjera algún error durante la eliminación del proyecto se le comunicaría al usuario por medio de un Toast.

6.2.1.4 AreaTrabajo.java – activity_area_trabajo.xml

Antes de empezar con los aspectos de la implementación relacionados con las funcionalidades que ofrece, se explicará cómo se estructura la vista. Así, en primer lugar, comentar que el Layout nos presenta una serie de botones que se le mostrarán o no al usuario en función del momento en el que se encuentre. Es decir, para evitar posibles errores de consistencia se han eliminados botones mientras que el usuario no se encontrara en disposición de poder utilizarlos.

Aquí, en la figura 35, se puede ver un caso de ejemplo de dicha vista Área de Trabajo:

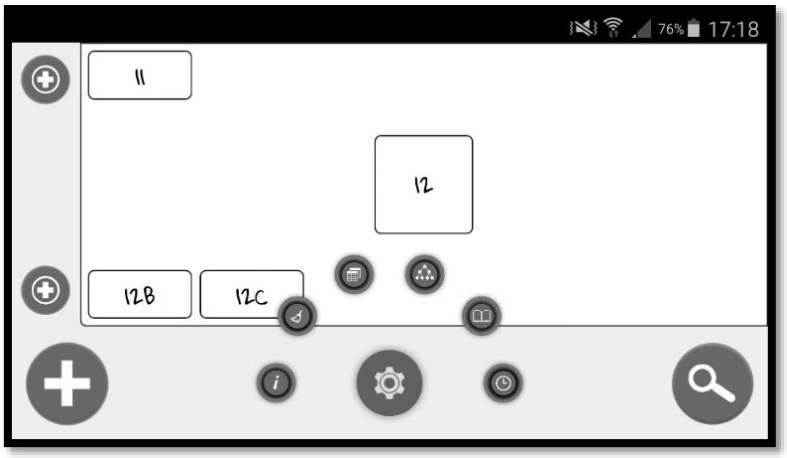

Ilustración 35 - Captura pantalla Layout Area Trabajo

En esta figura 35 se ven una serie de botones que serán descritos a continuación:

	(btn_1) Permite crear una nueva UE.
	(btn_2) Permite buscar una UE ya existente.
	(btn_3) Permite crear una relación de anterioridad o posterioridad.
	(btn_4) Menú desplegable que permite realizar las acciones que aparecen debajo.
	(btn_5) Muestra el grafo con las UEs y sus relaciones.
	(btn_6) Muestra el Diario de Excavación.

	(btn_7) Muestra el Stratigraf.
	(btn_8) Permite vaciar la pantalla.
	(btn_9) Permite mostrar un Alert con las instrucciones de uso.
	(btn_10) Permite mostrar los datos de la última UE y relación almacenadas.

Por otro lado, en esta vista presentada en la figura 36, se diferencian una serie de zonas fundamentales.

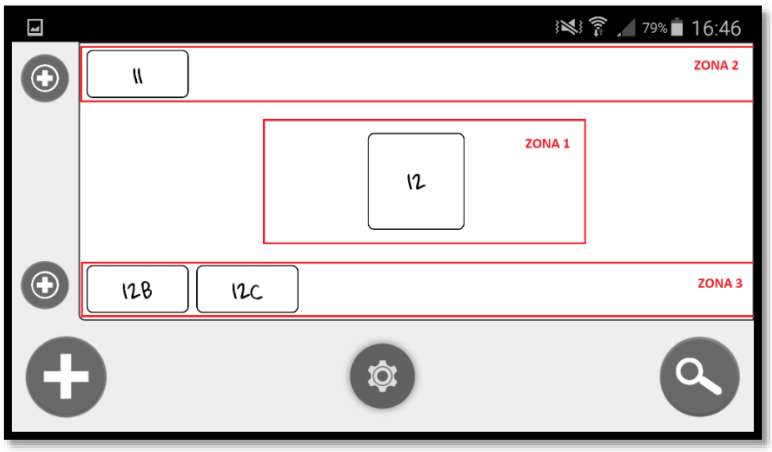

Ilustración 36 - Captura con las distintas zonas del Layout

A continuación serán descritas las diferentes partes del Layout presentados en la figura 36:

En la Zona 1 es donde se cargará la UE con la que se pretende trabajar. Esta UE se puede cargar:

- Creando una nueva UE - (btn_1).
- Buscando y cargando una nueva UE - (btn_2).
- Visitando alguna de las vistas correspondientes a los botones btn_1, btn_2 y btn_3, seleccionando una UE y volviendo por medio del botón “back” del dispositivo, a la vista actual.

Si en esta zona no hay ninguna UE cargada, no se podrán añadir relaciones de anterioridad y posterioridad, por lo que la vista se verá tal y como se presentan en la figura 37.

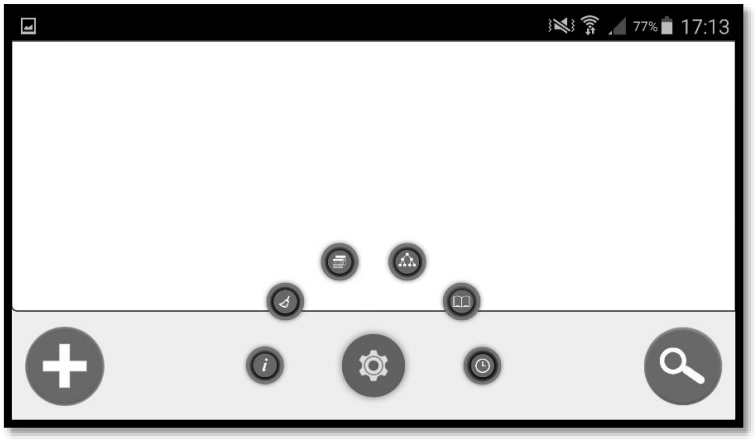

Ilustración 37 - Captura de pantalla del Layout Area Trabajo.

Como se puede observar, en la figura 37, al no aparecer los botones, no se le permite al usuario caer en el error de pulsar el botón y de este modo, se puede evitar la parte correspondiente al control de errores que obligaría una implementación diferente.

En la Zona 2 es donde aparecerán las relaciones de anterioridad que presentará la UE cargada en la Zona 1. Se podrá añadir una nueva relación de anterioridad utilizando el btn_3 que está justo a la izquierda de dicha zona.

La Zona 3 es similar a la 2 y funciona del mismo modo. Lo único que mostrará las relaciones de posterioridad. Se puede añadir una nueva relación de posterioridad ciclando sobre el btn_3 que se encuentra a la izquierda de dicha zona.

A continuación serán descritas por separado cada una de las opciones presentadas anteriormente:

Crear una nueva UE - (UC8):

En primer lugar se captura el evento del botón btn_1 con un onClickListener. Posteriormente se lanza un Alert para permitirle al usuario que introduzca el nombre de la nueva UE que pretende crear. Dicha ventana emergente infla el contexto actual a partir del Layout "activity_dialogo_nueva_ue.xml".

Dicho Dialog presenta el aspecto que puede apreciarse en la figura 38.

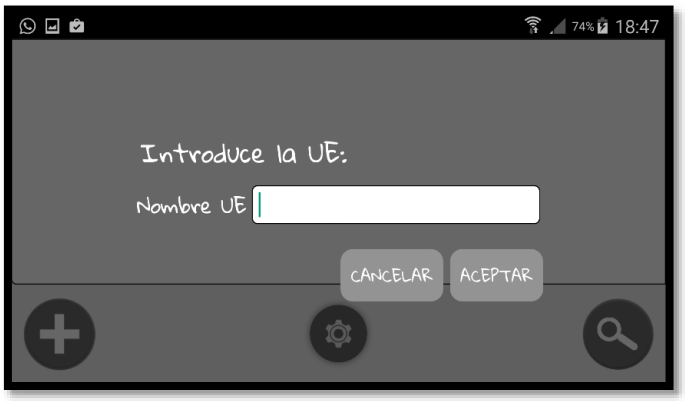

Ilustración 38 - Captura del AlertDialogo crear nueva UE.

Si el usuario clica sobre el botón aceptar, presentado en la figura 38, sin haber introducido previamente un nombre para esa nueva UE se cerrará el menú y se lanzará un Toast avisando al usuario de que no se ha realizado la acción por las razones mencionadas.

Si la tarea se realiza con éxito, en la Zona 1, se crea el botón que representará a la UE cargada y con la que se quiere trabajar.

A este botón se le captura el evento onClick por medio de un onClickListener que permite pasar a un nuevo Layout "activity_datos_ue_central.xml" por medio de un Intent (UC10).

Buscar y cargar una UE – (UC9):

En primer lugar se captura el evento onClick del botón correspondiente (btn_2) por medio de un onClickListenter. Y en el caso de que el usuario lo active se lanzará un Dialog con el aspecto que puede apreciarse en la figura 39:

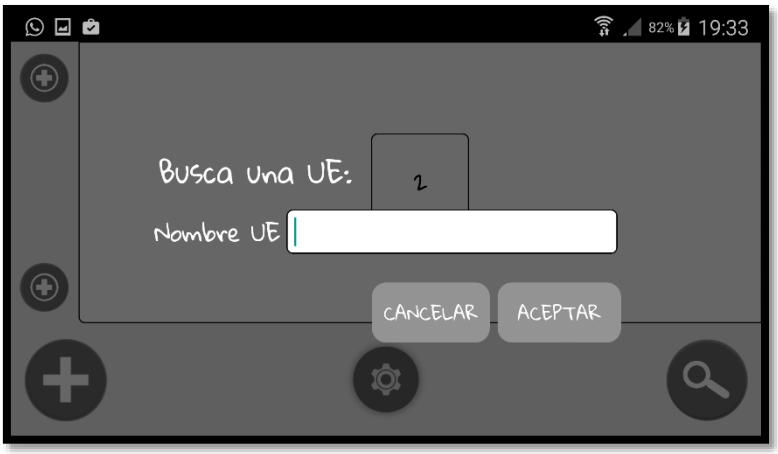

Ilustración 39 - Captura del Alert activity_dialogo_buscar_ue.xml

Este Alert, presentado en la figura 39, puede verse como infla la vista padre con el Layout "activity_dialogo_buscar_ue.xml". Nuevamente se aplica la lógica seguida hasta el momento, si el usuario no introduce el nombre de la UE que pretende buscar y cargar, no se realizará ninguna acción y se le avisará por medio de un Toast.

Por el contrario, si ha introducido el nombre, tras clicar sobre el botón "Aceptar" se comprueba si dicha UE existe. De no ser así, nuevamente no se realizaría ninguna acción y se le comunicaría por medio de un Toast.

Si introdujo el nombre y la UE existe, se limpiará la pantalla y se creará el botón, en la Zona 1, que representará a la UE buscada. Así como todos los botones, en las Zonas 2 y 3, que representen a las diferentes relaciones que tenga dicha UE tras habérselas solicitado a Controldor.java.

Crear una nueva relación – (UC12 y UC13):

Este caso es válido tanto para crear una relación de anterioridad como de posterioridad.

En primer lugar se capturan los eventos onClick por medio de un onClickListener para los botones del tipo btn_3.

En el caso de que el usuario active alguno de los botones se lanzará un Alert que infla el contexto actual con el Layout “activity_dialogo_nueva_ue.xml”. Y, como es lógico, su aspecto será el mismo que el de la ilustración 38. Así como su funcionamiento. Si el usuario no introduce el nombre, no se realizará ninguna acción y se le comunicará por medio de un Toast.

Una vez introducido el nombre de la UE con la que se quiere relacionar la UE central, se comprueba si dicha UE existe previamente. De no ser así, se crea dicha UE y se comprueba si esta nueva relación con la nueva UE en cuestión genera un ciclo. Si esto se produjera se avisaría al usuario por medio de un Toast que no puede realizarse la acción y por qué. Al mismo tiempo que se le muestran los nodos o UEs que intervienen en dicho ciclo.

Si por el contrario dicha UE no genera un ciclo se almacena en la base de datos correspondiente. Al igual que se almacena la nueva relación establecida entre dichas UEs.

Por otro lado, en la vista, se creará, en la zona correspondiente (2 o 3), el botón que representa la relación entre ambas UEs.

Se capturarán dos eventos para dicho botón:

- El evento onClick con un onClickListener. Esto permitirá que la UE seleccionada pase a ser la UE central con la que se pretende trabajar (UC17).
- El evento onLongClick con un onLongClickListener. Esto permitirá eliminar una relación (UC14 y UC15).

Eliminar una relación (UC14 y UC15):

En caso de que el usuario trate de eliminar una relación con un long click, se lanzará un Diálogo pidiéndole la confirmación. Para ello se infla el contexto AreaTrabajo con el Layout “activity_dialogo_confirmar_eliminar_relacion.xml”. Su aspecto es el que puede apreciarse en la figura 40.

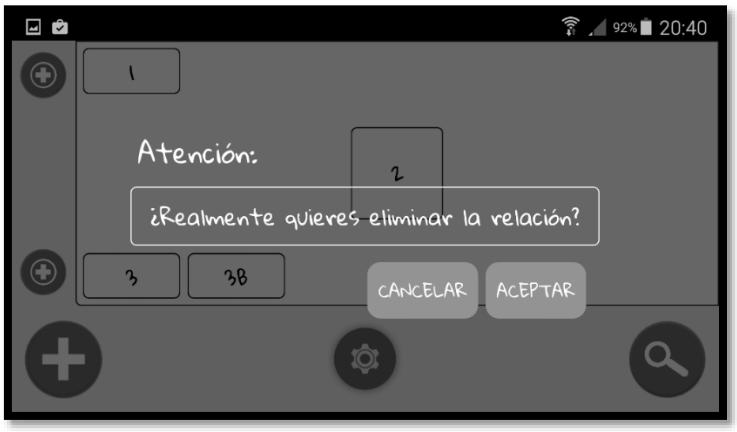

Ilustración 40 - Alert confirmación eliminar relación.

La figura 40 nos muestra como se le presentan dos botones al usuario que permitirán confirmar la eliminación de la relación en cuestión. O por el contrario cancelar la operación.

De producirse algún error durante la tarea, se le comunicará al usuario por medio de un Toast.

Comentar, en este punto, que si se elimina una relación, las UEs descendientes pasarán a depender del nodo raíz.

Vaciar la pantalla de trabajo (UC11):

Esta funcionalidad se ha implementado, aunque no se encontraba en los requisitos, por si el usuario en algún momento necesita limpiar el área de trabajo de botones por cualquier razón.

Para obtener el resultado esperado, en primer lugar se captura el evento onClick del botón en cuestión por medio de un onClickListener. Posteriormente se elimina de la View todos los componentes de las Zonas 1, 2 y 3.

Y por último se ocultan los botones del tipo btn_3 ya que es una acción que el usuario no puede realizar mientras que no se haya cargado una UE en la Zona 1.

Ver los datos de la última UE y relación introducida en la base de datos (UC30):

Para ello es necesario capturar el evento del botón del tipo btn_10 por medio de un onClickListener.

Este botón, en el caso de ser accionado, lanzará un AlertDialog con la información que se podrá ver en la figura 41.

Ilustración 41 - Captura del Alert con la información correspondiente a la última UE y relación creada.

Para que el Dialog presente el aspecto, presentado en la figura 41, ha sido necesario diseñar el Layout “activity_dialogo_ultima_ue.xml”.

Como se puede apreciar se muestra el nombre de la última UE creada y su fecha de creación, así como la última relación asignada y almacenada en la base de datos.

Ver el grafo, el Diario de Excavación y el Cronoestratigraf (Tabla) – (UC18 , UC22 y UC26):

Para conseguir estas tareas se ha seguido el mismo procedimiento. En primer lugar capturar el evento onClick del botón correspondiente (btn_5, btn_6 y btn_7). Posteriormente por medio de un Intent enlazar con la vista apropiada:

- Diario de Excavación: activity_diario_excavación.xml.
- Cronoestratigraf: activity_tabla.xml.
- Grafo: activity_grafo.xml.

Ver las instrucciones de uso de la vista – (UC7):

Al cargarse, por primera vez, la vista Área de Trabajo, se lanza un Alert que infla el contexto padre a partir del Layout activity_dialogo_instrucciones_area_trabajo.xml.

Este Dialog volverá a mostrarse si el usuario activa el botón correspondiente (botón del tipo btn_9). Para ello ha sido necesario capturar su evento onClick a partir de un onClickListener.

Como podrá verse en la ilustración 42, mostrada a continuación, dicho Dialog, presenta un botón "Aceptar" que le permitirá cerrar dicho Alert.

Por otro lado, se ha diseñado con un fondo transparente y se han activado todos los botones en la vista Área de Trabajo, para que el usuario pueda, mientras ve la ayuda, localizar donde se encuentran todos los botones y comprobar que es lo que hace cada uno.

Así su aspecto será el que podrá verse en la figura 42.

Ilustración 42 - Captura del Alert que muestra las instrucciones de uso.

En la figura 42 puede apreciarse un listado de los diferentes botones que ofrece el Layout activity_area_trabajo.xml y que son descritos en el Alert que nos ocupa.

Para que el menú se anime y se abra un segundo después de que se muestre el Alert con las instrucciones de uso, se ha utilizado un Timer y una AsyncTask. Esto se ha implementado de este modo para que el usuario observe el funcionamiento de dicho botón y comprenda mejor como usarlo.

6.2.1.5 *FragmentMenuAnimado.java – fragment_menu_animado.xml*

Como ha podido apreciarse en las ilustraciones anteriores, la vista Área de Trabajo, presenta un botón central (btn_4) que permite mostrar las opciones anteriormente descritas.

Esta clase utilizará el Handler SlideInAnimationHandler.java y la librería (com.oguzdev.circularfloatingactionmenu.library) para animar el menú.

6.2.1.6 *DatosNodoCentral.java – activity_datos_ue_central.xml*

DatosNodoCentral.java será la clase que permitirá modificar los datos de la UE cargada en la Zona 1 del Layout activity_area_trabajo.xml (UC10)

El aspecto que muestra la vista activity_datos_ue_central.xml es el que puede verse en la figura 43.

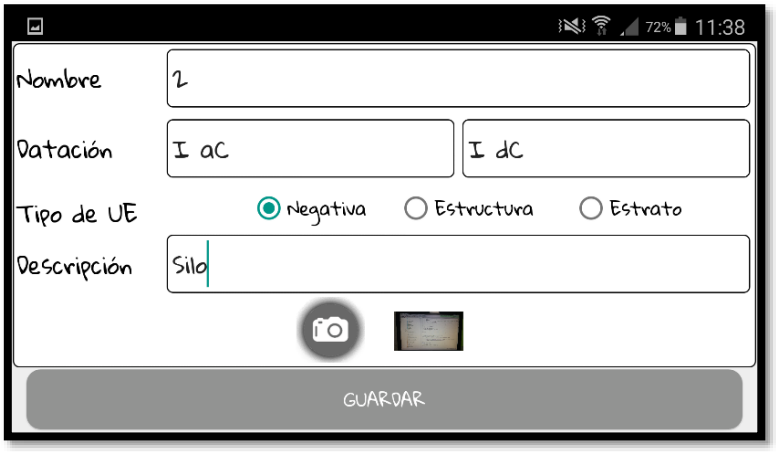

Ilustración 43 - Captura vista activity_datos_ue_central.xml

Como puede apreciarse en la ilustración 43, la vista consta de cuatro EditText que permitirán introducir los datos correspondientes al nombre de la UE, la datación y la descripción de esta. Todos estos elementos permiten la entrada tanto de letras como de números. Y en todos ellos pueden escribirse varias líneas de caracteres. Dichas líneas pueden visualizarse mediante el uso del scroll que presenta cada EditText.

También puede observarse que la datación presenta dos campos ya que era uno de los requisitos obtenidos durante la fase de obtención de estos. Esto se ha hecho ya que en muchas ocasiones no es posible dar una datación absoluta de la UE en cuestión y su datación viene dada por un periodo de tiempo (datación relativa).

Por otro lado, se ha implementado un RadioGroup con tres que hacen referencia a los tres tipos de UEs contemplados (negativa, estrato y estructura y de los que ya se ha hablado en puntos anteriores).

También, desde esta vista, se le permite al usuario sacar una foto de la UE, apareciendo esta posteriormente en miniatura en el propio Layout. Además de permitirle recoger un punto de GPS que se utilice para localizar la posición de la foto y de la UE.

De esta foto en miniatura se ha capturado su evento onClick. Y en caso de ser activado por el usuario se pasará por medio de un Intent al Layout “activity_foto_ampliada.xml”. Es una vista que simplemente muestra en pantalla completa la foto y que por medio del botón “back” del dispositivo permite volver a la vista DatosNodoCentral.java. (UC33)

Este Layout activity_datos_ue_central.xml también presenta un botón que permitirá guardar, en caso de ser correctos, los nuevos datos introducidas por el usuario.

Antes de entrar en más detalles sobre esta clase controladora, decir que ha sido invocada por medio de un startActivityForResoult(), por lo que la Activity padre (AreaTrabajo.java) estará esperando la finalización de esta para realizar una serie de acciones.

Así, tras introducir todos los datos deseados, el usuario podrá clicar sobre el botón “Guardar” del cual se le ha capturado su evento onClick.

Los valores introducidos se enviarán a la Activity AreaTrabajo.java en un objeto del tipo Nodo.

Una vez dentro del método onActivityResoult(), de la clase AreaTrabajo.java, se comprueba si el nombre de la UE es el mismo que el que tenía antes de llamarse a la vista activity_datos_ue.xml.

Si los nombres no son iguales, se comprueba si esa UE existe previamente. Si es así, y ya existe, no se producirá ninguna modificación y se le avisará al usuario por medio de un Toast de que se ha producido dicho error.

Por el contrario, si la UE no existe se lanzará un Alert preguntándole al usuario si está seguro que desea modificar el nombre de la UE. El Alert inflará el contexto padre y presentará el aspecto que podrá verse en la Ilustración 44 que sigue a continuación.

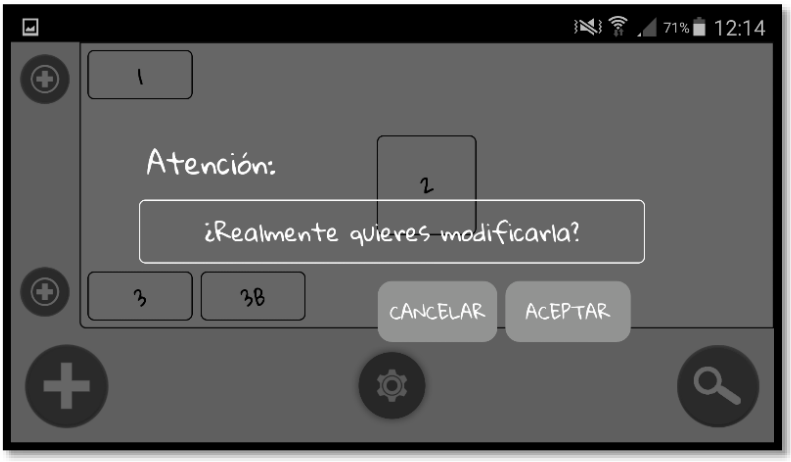

Ilustración 44 - Captura Alert confirmación modificar datos UE

Para que presente este aspecto, mostrado en la figura 44, ha sido necesario diseñar el Layout activity_dialogo_confirmar_cambiar_nombre.xml.

Si el usuario confirma la modificación, se producirán los cambios necesarios en la base de datos (nombre de la UE y de todas las relaciones donde aparezca). Si durante este proceso se produce algún error el usuario será avisado por medio de un Toast.

La foto - (UC31):

Para la realización de esta tarea ha sido necesario habilitar en el AndroidManifest.xml, los permisos correspondientes a la cámara, a la lectura y a la escritura. La intención, en este punto, era guardar las fotos obtenidas en la SD, pero desde Android 4.4 Kikat se ha prohibido a las aplicaciones instaladas por el usuario escribir en la tarjeta microSD. Para ello es necesario ser un usuario de la categoría media_rw.

También ha sido necesario capturar el evento del botón que permite sacar la foto por medio de un onClickListener. Si el usuario activa este botón, mediante un startActivityForResoult() se pasará a la Activity "MediaStore.ACTION_IMAGE_CAPTURE" Será necesario pasarle por parámetro la uri donde se pretende almacenar la foto en cuestión. Esta Activity se encuentra ya programada por defecto en el dispositivo y permite conectar con el software de la cámara.

En el método onActivityResoult() se comprobará si se ha sacado la foto y se almacenará la uri de dicha foto en la base de datos correspondiente. Así como su nombre y la fecha en la que ha sido sacada.

Comentar que el nombre de la foto tendrá el siguiente formato: "PIC_fecha de la foto". Y que la fecha de la foto se saca automáticamente del sistema.

También se cargará en el ImageView que presenta la vista activity_datos_ue_central.xml la foto de la UE almacenada, como se puede ver en la ilustración 43.

GPS - (UC32):

La obtención del punto GPS se realiza automáticamente al sacar la foto de la UE.

En primer lugar se comprueba, antes de sacar la foto, si el móvil tiene habilitado el recurso que permite realizar la localización. Es decir, se comprueba si el dispositivo tiene acceso a internet o al localizador de ubicación.

En caso de encontrarse los servicios deshabilitados se le preguntará al usuario si desea activarlos. Para ello se lanza un Dialog que infla el contexto padre con el Layout "activity_dialogo_activaar_gps.xml". Y presenta el aspecto que puede apreciarse en la figura 45.

Ilustración 45 - Captura del Alert que pregunta si activar el GPS

El usuario ante esta situación, como puede observarse en la figura 45, podrá sacar la foto sin capturar el punto GPS o, por el contrario, clicar sobre el botón “Activar GPS”. Si el usuario opta por esta acción se le enlazará con la vista de configuración del propio dispositivo donde podrá activar este recurso.

Debido a que la obtención no siempre funciona a la primera, por cuestiones de localización o de conexión con los satélites, ha sido necesario forzar al dispositivo a que encuentre un punto. Para ello se ha implementado un TimerTask que cada cierto tiempo, en segundo plano, va solicitando el punto al dispositivo. A su vez, se ha creado un contador que si tras transcurrir un tiempo prudencial no se ha conseguido obtener el punto, se le indique al usuario mediante un Toast que no ha sido posible encontrar su localización.

A continuación, en las figuras 46 y 47 puede apreciarse lo anteriormente descrito.

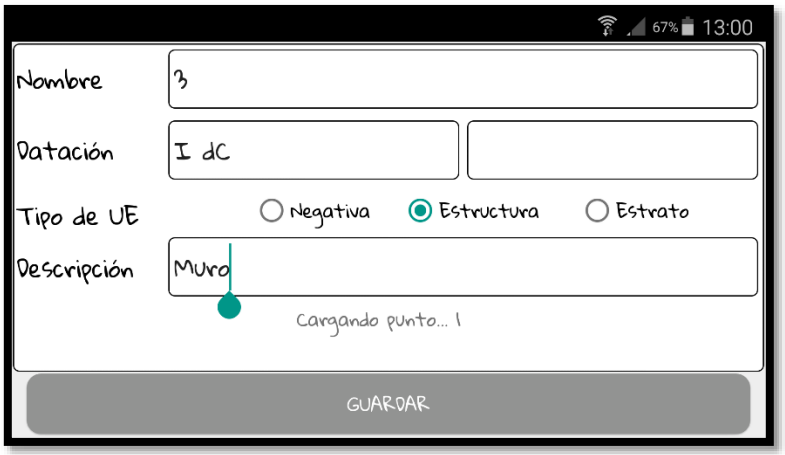

Ilustración 46 - Momento en el que se está capturando el punto GPS

Ilustración 47 - Momento en el que ya se ha capturado el punto GPS

En la figura 46 pude observarse el mensaje que muestra un contador, actualmente con valor 1, que marca el tiempo que se está empleando para obtener el punto. Se ha implementado de este modo siguiendo los conceptos aprendidos en la asignatura de Factores Humanos donde se nos había comentado que si la acción a realizar llevaba más de un cierto tiempo era aconsejable mostrar un mensaje al usuario que le indique que debe esperar. Con la intención de comunicarle que la acción se está realizando pero que tardará un tiempo.

Por otro lado, en la figura 47 puede verse como ya se muestra un mensaje indicándole al usuario que el punto ha sido capturado satisfactoriamente y que ya puede sacar la foto que se asociará a la UE correspondiente.

6.2.1.7 Grafo.java – activity_grafo.xml

Grafo.java extiende de ActionBarActivity. Se le ha añadido un botón que permite buscar una UE dentro del grafo. (UC28)

Para ello ha sido necesario capturar el evento onClick de dicho botón. En caso de que el usuario lo accione es lanzará un Alert que infla el contexto padre mediante el Layout activity_buscar_ue.xml. Será por tanto el mismo Layout que el utilizado en la clase controladora AreaTrabajo.java.

Durante la fase de inicialización de la clase se comprueba si se ha recibido algún parámetro por medio del Intent, ya que esto indicaría que previamente, en la vista Área de Trabajo se ha seleccionado alguna UE y por tanto tiene que ser remarcada también como seleccionada en la vista que nos ocupa. Presentará un objeto del tipo GrafoView (view) que se añadirá a la vista y a la que se le pasarán por parámetro las dimensiones de la pantalla del dispositivo. Activity_grafo.xml presenta dos scrolls (uno vertical y otro horizontal).

Un ejemplo del aspecto que presentaría esta vista es el que puede apreciarse en la figura 48.

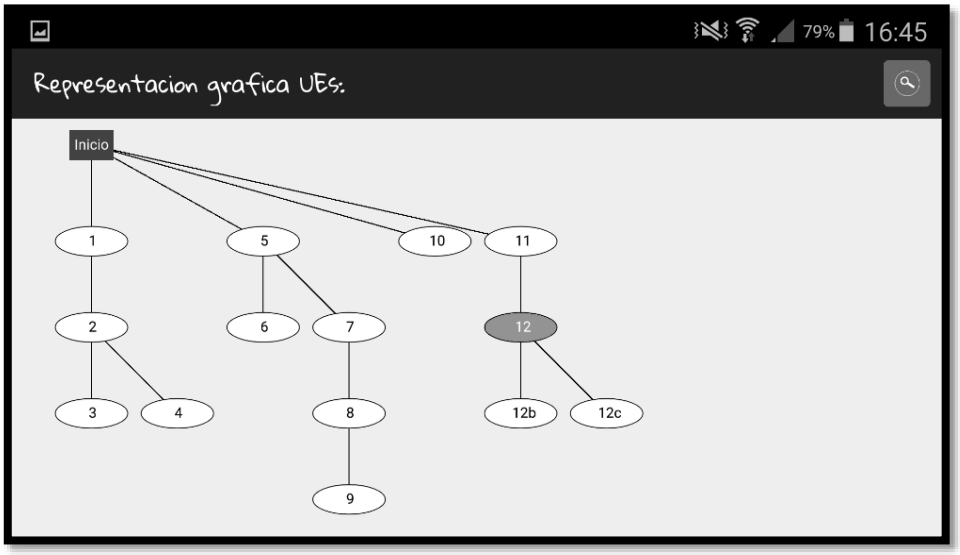

Ilustración 48 - Captura de la vista que muestra graficamente las UEs y sus relaciones

En esta figura 48 puede apreciarse como todos los nodos, que representan a las diferentes UEs, cuelgan del nodo raíz “Inicio”. Este no corresponde conceptualmente con ningún momento contemplado desde un punto de vista arqueológico. Sino que se ha hecho por cuestiones de gestión y puramente por una razón relacionada con el aspecto de la vista.

Por otro lado, puede apreciarse también, en la figura 12, como se encuentra seleccionada la UE 12, con la que se estaría trabajando en ese momento, y como esta aparece resaltada con un color diferente tanto para su fondo como para el color de su letra.

6.2.1.8 GrafoView

Esta clase extiende de View y es la que se encargará de dibujar el grafo.

Ha sido necesario crear una serie de objetos del tipo Paint que almacenarán datos referentes al aspecto de los elementos que se dibujen.

Se ha implementado un zoom. Para ello ha sido necesario definir la escala y las dimensiones del View que albergará el Canvas donde se dibujará el grafo. Esto se ha conseguido sobrescribiendo el método onMeasure() (UC35).

El ancho y alto mencionados se calculan cada vez en el momento de cargar la Clase; en función del número de UEs y relaciones existentes en ese momento.

En el método onDraw se dibujarán las diferentes UEs a partir de objetos del tipo RectF. Y las aristas (relaciones) entre las diferentes UEs a partir del método drawLine(). Los textos introducidos se realizará a partir del método drawText().

Para seleccionar una UE ha sido necesario utilizar el método onTouchEvent() donde se obtendrán las posiciones x e y de dicho contacto con la pantalla. Posteriormente, a la hora de dibujar los diferentes Rectf que representarán a las diferentes UEs, se comprueba si dicho Rectf contiene los puntos x e y. De ser así, se cambiará la apariencia de dicho elemento (UC27).

6.2.1.9 Tabla.java – activity_tabla.xml

Esta clase permitirá mostrar el Cronoestratigraf (Tabla) (UC18). Es decir, mostrar las UEs organizadas en función de su momento de creación y no de su contacto físico. El algoritmo que permite realizar esta operación ha sido ya descrito en apartados anteriores.

Tabla.java extiende de ActionBarActivity.

Al ActionBar que presenta se le ha añadido un botón que permite buscar una UE dentro del Cronoestratigraf (UC20). Si el usuario activa este botón se lanzará un Alert que inflará el contexto padre a partir del Layout acitivity_buscar_ue.xml descrito en puntos anteriores, por lo que no se mostrará su vista a continuación.

Esta vista presentará también dos scrolls, uno vertical y otro horizontal.

Tabla.java también mostrará las relaciones directas e indirectas de una UE seleccionada o cargada previamente por el usuario (UC36 y UC37).

Así durante la fase de inicialización de la clase se comprueba si se ha recibido algún parámetro mediante el Intent que indique que el usuario ha seleccionado una UE en una vista anterior.

De ser así habrá que marcar esa UE en color azul, sus relaciones directas en verde y las indirectas en rojo.

Un ejemplo de cómo sería su aspecto será mostrado a continuación en la figura que nos sigue, la ilustración 49.

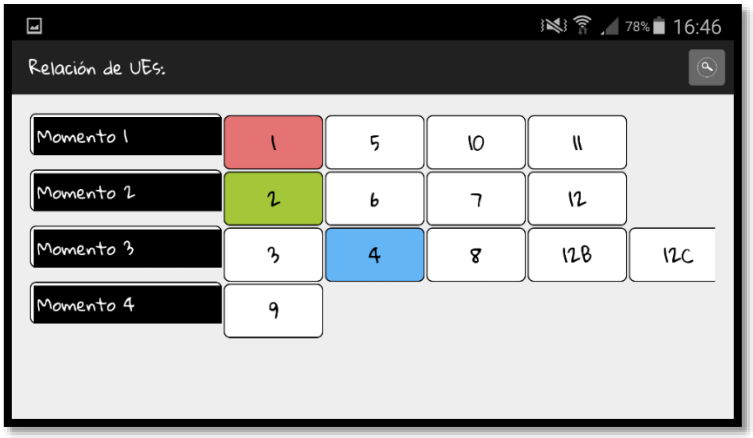

Ilustración 49 - Captura de la vista Stratigraf que muestra las relaciones directas e indirectas de una UE cargada o seleccionada.

La figura 49 nos muestra los diferentes momentos registrados hasta el momento y como se encuentra la UE con nombre 4 seleccionada. Así como las relaciones directas e indirectas de dicha UE, resaltadas en color verde y rojo respectivamente.

En esta ocasión se ha decidido representar los datos por medio de un TableLayout para familiarizarse con este tipo de elemento.

Para ellos es necesario calcular antes de la representación el número de columnas y de filas que son necesarias para dibujar la tabla.

Para que sea más sencillo capturar los eventos, dentro de cada una de las columnas se han introducido botones que representan a las diferentes UEs. Así, de este modo, se permitirá seleccionar una UE por medio de un onClick (UC19).

6.2.1.10 DiarioExcavacion.java – activity_diario_excavacion.xml

Esta clase permitirá al usuario ver un listado con todos los datos de las diferentes UEs almacenadas en la base de datos para un proyecto concreto - (UC22).

Extiende de ActionBarActivity.

Al ActionBar que presenta se le ha añadido un botón que permite buscar una UE dentro del Diario de Excavación (UC24).

Si el usuario activa este botón se lanzará un Alert que inflará el contexto padre a partir del Layout acitivity_buscar_ue.xml descrito en puntos anteriores, por lo que no se mostrará su vista a continuación.

Esta clase contendrá una ListView al que se añadirá un Adapter del tipo AdapterDiarioExcavación.

Un ejemplo de su aspecto sería el presentado, a continuación, en la figura 50 y 51.

Ilustración 50 - Captura parcial del Diario de Excavación.

Ilustración 51 - Captura parcial del Diario de Excavación.

Como se puede ver en las figuras 50 y 51 presentadas anteriormente, la vista, además de tener un scroll vertical por ser una lista, presenta un scroll horizontal para cada uno de los ítems que componen la lista.

Así como en el campo foto se presenta una miniatura de la foto asociada a la Ue correspondiente. Se ha capturado el evento onClick que permite lanzar el Layout activity_foto_ampliada.xml que permitirá ver la imagen en pantalla completa. Con un simple click en el botón “back” del dispositivo, se volverá a la vista que nos ocupa.

6.2.1.11 AdapterDiarioExcavacion.java – activity_adapter_diarioexcavación.xml

Esta clase extiende de ArrayAdapter<Nodo> y será la encargada de ir creando un ítem para cada una de las UEs que recibe su constructor por parámetro.

A través de su método getView() se infla su contexto padre, con el Layout activity_adapter_diario_excavación.xml. Este presenta el aspecto que se puede ver en la figura 52, antes de adaptar su apariencia a la del resto del proyecto:

Ilustración 52 - Captura del Layout activity_adapter_diario_excavacion.xml

Para esta vista presentada en la figura 52, se han capturado dos tipos de eventos para cada uno de los ítems incluidos:

- El evento onClick a través de un onClickListener para seleccionar una UE concreta (UC23)
- El evento onLongClick a través de un onLongClickListener para eliminar una UE de la base de datos (UC34).

Así, como se ha comentado, el usuario podrá seleccionar una UE de las presentadas en la vista. Esto hará que cambie el aspecto del ítem seleccionado como se puede ver en las dos ilustraciones anteriores.

Por otro lado también podrá eliminar una UE haciendo un click largo sobre uno de los ítems. Pero antes de eliminar la UE seleccionada tendrá que dar su confirmación en un Alert que le será presentado. El Dialogo mencionado inflará su contexto padre con el Layout activity_dialogo_confirmar_eliminar_ue.xml y presenta el aspecto presentado en la figura 53 mostrada a continuación.

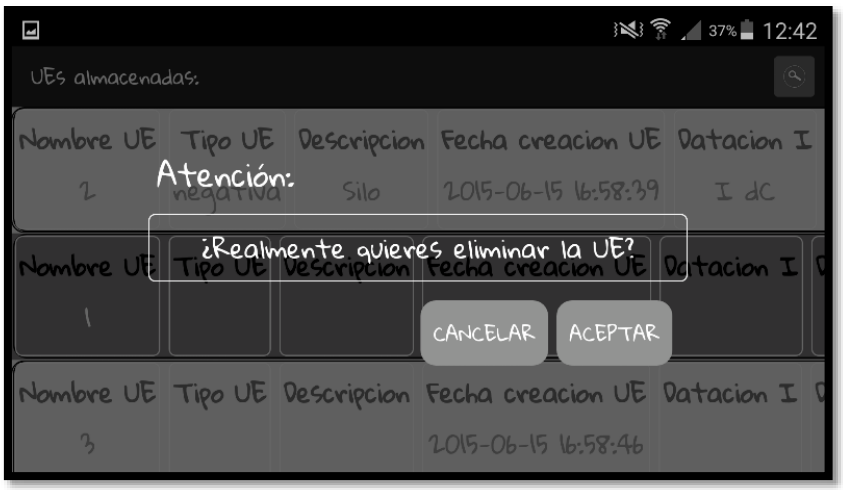

Ilustración 53 - Captura del Alert que se le presenta al usuario para que confirme que desea eliminar la UE seleccionada.

Como puede observarse, en la figura 53, el usuario podrá confirmar la eliminación de la UE correspondiente clicando en el botón aceptar o cancelar tal acción.

Cuando una UE es eliminada sus descendientes pasan a depender del nodo raíz. Y hay que eliminar, además de la UE de la base de datos, todas sus relaciones.

Para conseguir los resultados esperados, y mencionados anteriormente, ha sido necesario capturar el evento onClick de la imagen en miniatura que se presenta en cada uno de los ítems de la vista para permitirle, al usuario, ver la imagen en pantalla completa. Como se había explicado para la clase DatosNodoCentral.java se pasaría por medio de un Intent a la clase ImagenUE.java y a su respectivo Layout activity_foto_ampliada.xml. Esta vista es muy simple, solo contiene un ImageView que ocupa toda la pantalla y donde se mostrará la imagen seleccionada.

Por otro lado, con un simple clic en el botón “back” del dispositivo se volverá a la vista activity_diario_excavación.xml.

6.2.2 Modelo

En este apartado se comentarán aspectos de la implementación relacionados con las clases que componen este campo del patrón MVC.

6.2.2.1 Proyecto.java

Esta clase implementa Serializable. Un objeto serializable es un objeto que se puede convertir en una secuencia de bytes y ser reconstruido posteriormente. Por ejemplo, los objetos comunes como String, Vector o ArrayList implementan ya por defecto a Serializable.

Esta clase será utilizada para caracterizar los diferentes proyectos que sean creados por el usuario.

Un objeto Proyecto contiene los siguientes atributos:

- String id.
- String nombre.
- String autor.

Además de los métodos getters y setters necesarios para su correcto funcionamiento.

6.2.2.2 Nodo.java

Esta clase también implementa Serializable y caracterizará cada una de las UEs creadas por el usuario.

Cada objeto del tipo Nodo contendrá los siguientes atributos:

- Int id.
- String nombre
- String tipo_ue.
- String descripción

- String fecha_creación_ue
- String datación_1
- STring datación_2
- String nombre_foto
- String fecha_foto
- String uri_foto
- String localización

Además de los métodos getters y setters correspondientes.

6.2.2.3 Vertice.java

Esta clase implementa Serializable y permitirá caracterizar cada una de las UEs dentro de una estructura de datos que permitirá organizar estas en función de sus relaciones. Dicha estructura será utilizada en la vista, ya descrita, activity_grafo.xml.

Cada objeto del tipo Vertice contendrá los siguientes atributos:

- Int estado (para saber si ha sido visitado o no).
- Int ciclo (para saber si ha sido visitado cuando se comprueba si una nueva relación genera un ciclo o no).
- Int numero_padres (se utiliza para controlar si ha sido visitado durante la eliminación de aristas redundantes).
- Int numero_hijos (se utiliza para controlar si ha sido visitado durante la eliminación de haristas redundantes).
- Nodo nodo (contendrá todos los datos referentes a la UE que representa dicho vértice).
- ArrayList<String> visitados (se utilizará para saber todos los vértices descendientes de una UE concreta).

Además de todos los métodos getters y setters neceasarios para el correcto funcionamiento de la aplicación.

6.2.2.4 Arista.java

Esta clase también implementa Serializable y permitirá caracterizar cada una de las relaciones de una UE dentro de la estructura de datos que permitirá organizar estas en función de sus relaciones. Dicha estructura será utilizada en la vista, ya descrita, activity_grafo.xml.

Cada objeto del tipo Arista contendrá los siguientes atributos:

- Nodo nodo.
- Vertice v1 (representará a uno de los vértices de la relación).
- Vertice v2 (representará el segundo vértice de la relación).

También contendrá todos los getters y setters necesarios.

6.2.2.5 *Estructura*

Esta clase también implementa Serializable y será la encargada generar una estructura a partir de LinkedList que representará el árbol que será utilizado para su representación gráfica en la vista activity_grafo.xml.

Presenta los siguientes atributos:

- LinkedList lista_vertices (almacenará el conjunto de todos los vértices).
- LinkedList lista_aristas (almacenará el conjunto de todas las aristas).
- Vetice v_inicio.
- Int cv (contador de vértices)
- Int ce (contador de aristas)

Además de una serie de métodos que permitirán:

- Crear una nueva arista y un nuevo vértice.
- Eliminarlos.
- Buscarlos.
- Devolver las aristas descendientes o ascendentes de un vértice.
- Contar el número de vértices padres o hijos de un vértice.
- Devolver todos los vértices adyacentes de un vértice concreto.
- Comprobar si se produce un ciclo al crear una nueva relación.
- Saber en qué nivel del árbol se encuentra un vértice concreto (esto se utilizará, en la vista activity_tabla.xml, para ordenar las UEs en función del momento temporal en el que ha existido dicha UE y no a partir de su relaciones de contacto con otras UEs).
- BFS y DFS (utilizados para recorrer la estructura, serán explicados en puntos posteriores).
- Entre otros.

6.2.2.6 *DataBaseManagerProyecto*

Únicamente destacar que la clase que nos ocupa contendrá un objeto del tipo ConentValues que permitirá albergar el conjunto de datos necesarios para cada una de las operaciones mencionadas anteriormente. Un objeto del tipo ContentValues es una colección del tipo diccionario, donde se almacenarán parejas del tipo clave-valor, donde la clave será el nombre de cada campo y el valor será el dato correspondiente a insertar en dicho campo.

6.2.2.7 *DatabaseManager*

Esta clase es similar a la descrita anteriormente, pero en esta ocasión será la encargada de gestionar las UEs y las relaciones de estas, pero por todo lo demás es exactamente igual por lo que no se entrará en más detalles.

6.2.3 Controladores

En este apartado se explicarán los diferentes controladores utilizados para el conseguir un correcto funcionamiento de la aplicación y que el código de esta se estructure según el patrón Modelo-Vista-Controlador.

6.2.3.1 Controlador

Esta es la clase que ejerce de controlador propiamente dicho. Es la clase principal de esta parte del patrón MVC. Será la clase que se encargará de enlazar la vista con los datos y presentará todos los métodos necesarios para poder llevar a cabo dicho propósito.

6.2.3.2 DbHelperProyecto.java

Esta clase extiende de SQLiteOpenHelper y es la encargada de inicializar, en caso de ser necesario, la base de datos que alberga los diferentes proyectos.

6.2.3.3 DbHelperUE.java

Esta clase extiende de SQLiteOpenHelper y es la encargada de inicializar, en caso de ser necesario, la base de datos que alberga las diferentes UEs y sus relaciones.

6.2.3.4 GPSTracker.java

Es una clase .java que extiende de Service e implementa LocationListener.

Utilizará el método Location Location() para mediante un getSystemService(LOCATION_SERVICE) obtener los puntos longitud y latitud. Pero antes de nada se comprueba si está activado el servicio. En caso contrario se le lanzará un Alert que le pregunta al usuario si quiere activar el servicio.

6.3 Otros aspectos a considerar

Hoy en día, las aplicaciones de cualquier entorno (web, escritorio, móvil) piden la autorización explícita de los usuarios para acceder a determinada información que puede resultar delicada para el usuario.

En Android, para poder utilizar la información o servicios de otras aplicaciones hacemos uso del famoso archivo AndroidManifest.xml, en donde se define un elemento <uses-permission>. Se pueden tener cero o muchos elementos de este tipo, definiéndolos todos como hijos directos del elemento raíz del archivo.

Así para el desarrollo de esta aplicación ha sido necesario dar permisos para la cámara, para el autofocus de esta, para la escritura y lectura de archivos y para el acceso al GPS.

Todos los permisos del sistema empiezan con android.permission y se enumeran en la documentación oficial del SDK para la clase Manifest.permision.

Por otro lado, en el momento en el que el usuario intenta sacar una foto para asociarla a una UE determinada, se comprobará si existe la carpeta correspondiente donde serán almacenadas todas las imágenes de la aplicación. Por tanto ha sido necesario especificar una ruta donde serán almacenadas dichas fotos y, como es lógico, donde se encontrará la carpeta en cuestión. Así esta se localizará en la raíz de la carpeta de imágenes del dispositivo y se llamará “mis fotos”.

En un primer lugar, se ha intentado que las fotos se almacenaran en la tarjeta SD externa. Pero desde la API KitKat no es posible realizar esta acción, por cuestiones de seguridad,

si no se tienen los permisos de usuario necesarios. Debido a esto y entendiendo que no necesariamente todos los móviles disponen de almacenamiento externo, se ha decido que era mejor no complicar este aspecto y que todas las imágenes se almacenaran en el espacio de memoria interna del dispositivo, en detrimento de la cantidad de fotografías que podría manejar la aplicación.

7 Pruebas y resultados

Las pruebas de software son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad el producto a la parte interesada.

Así, en este apartado se presentarán las pruebas realizadas y los resultados obtenidos tras dichas pruebas. Comentar que en un principio, se pretendía que fueran realizadas por los propios investigadores de la Facultad de Historia de la Universidad de Barcelona a partir de excavaciones reales y sus registros arqueológicos, pero finalmente no ha podido ser por cuestiones de tempos y he tenido que realizar yo mismo las pruebas.

7.1 Fase de testeo

Para esta fase de pruebas se han realizado los siguientes test.

7.1.1 Comprobar que función el botón AboutUs

Para esta prueba simplemente se ha clicado sobre dicho botón y se ha comprobado si mostraba la información deseada.

Tras realizar dicha prueba, se obtiene el resultado que se puede ver en la figura 54 y que se muestra a continuación.

Ilustración 54 - Captura del Alert que muestra la información sobre los desarrolladores

Como se puede apreciar en la figura 54, el botón mencionado funciona correctamente ya que se muestran los resultados deseados. Para no divulgar información personal sobre los desarrolladores se ha decidido ocultar los campos más delicados.

7.1.2 Crear un nuevo proyecto

Para esta prueba se ha intentado crear diferentes proyectos.

- Nombre del proyecto y autor con combinaciones de letras y números.
- Nombre del proyecto y autor solo con letras.
- Comprobar si diferencia entre mayúsculas y minúsculas.
- Nombre del proyecto y autor solo con números
- Solo nombre proyecto.
- Solo nombre autor.
- Introducir un nombre de proyecto ya existente a la hora de crear uno nuevo.

Para los cinco primeros puntos, primero serán presentadas las pruebas y finalmente los resultados obtenidos para esas cinco pruebas.

7.1.2.1 Nombre proyecto y autor con letras y números

Para esta prueba se introducirán para el nombre del proyecto y el autor una combinación de letras y números.

Así, se han introducido los datos que se muestran en la figura 55.

Ilustración 55 - Captura crear proyecto con letras y números.

Como se puede apreciar en la figura 55 se han introducido tanto para el nombre como para el autor una combinación de letras y números.

7.1.2.2 Nombre proyecto y autor solo con letras

En esta prueba se utilizarán solo letras para para escribir el nombre y el autor del proyecto.

Así, se han introducido los datos que se pueden ver en la figura 56 y que se presentan a continuación.

Ilustración 56 - Captura crear proyecto solo con letras.

Como se puede apreciar en la figura 56 se han introducido tanto para el nombre, como para el autor, una combinación solo de letras.

7.1.2.3 Comprobar si diferencia entre mayúsculas y minúsculas

Para esta prueba se ha introducido un nombre de un proyecto ya almacenado anteriormente pero esta vez escrito en minúsculas. De este modo, se comprobará si realiza diferencias entre mayúsculas y minúsculas.

Así, se han introducido los datos que se pueden ver en la figura 57 que se presenta a continuación.

Ilustración 57 - Captura crear proyecto en minúsculas.

Como se puede apreciar en la figura 57 se ha introducido un nombre de un proyecto que ya existía previamente (figura 54), pero esta vez se ha escrito en minúsculas.

7.1.3.4 Nombre del proyecto y del autor solo con números

Para esta prueba se han introducido tanto para el nombre del proyecto como del autor una combinación de números.

Así, se han introducido los datos que se pueden ver en la figura 58 que se presenta a continuación.

Ilustración 58 - Captura crear proyecto solo con números.

Como se puede apreciar en la figura 58 se ha introducido tanto para el nombre como el autor una combinación de números.

7.1.3.5 Solo nombre del proyecto

Para esta prueba se ha introducido el nombre del proyecto y nada en el campo del autor.

Así, se han introducido los datos que se pueden ver en la figura 59 que se presenta a continuación.

Ilustración 59 - Captura crear proyecto solo con nombre proyecto.

Como se puede apreciar en la figura 59 se ha introducido solo el nombre del proyecto y no se ha asignado ningún valor al campo autor.

7.1.3.6 Resultados obtenidos para las pruebas realizadas anteriormente

En este apartado se mostrarán los resultados obtenidos para las cinco pruebas descritas anteriormente.

Así, tras realizar estas pruebas se han obtenido los resultados que se pueden apreciar en la figura 60 que se muestra a continuación.

Ilustración 60 - Captura de los resultados obtenidos

Como se puede observar, en la figura 60, se han conseguido crear todos los proyectos de las pruebas anteriormente realizadas.

- La de crear un proyecto con combinaciones de letras y números tanto para el nombre como para el autor.
- Crear un proyecto con combinaciones de letras en el nombre y el autor.
- Crear un proyecto con combinaciones de números en el nombre y en el autor.
- Crear un proyecto solo con el nombre del proyecto.
- Crear un proyecto con un nombre ya creado pero escrito en minúscula.

7.1.3.7 Solo el autor

Para esta prueba se ha introducido solo el autor del proyecto y nada en el campo nombre.

Así, se han introducido los datos que se pueden ver en la figura 61 y que es presentada a continuación.

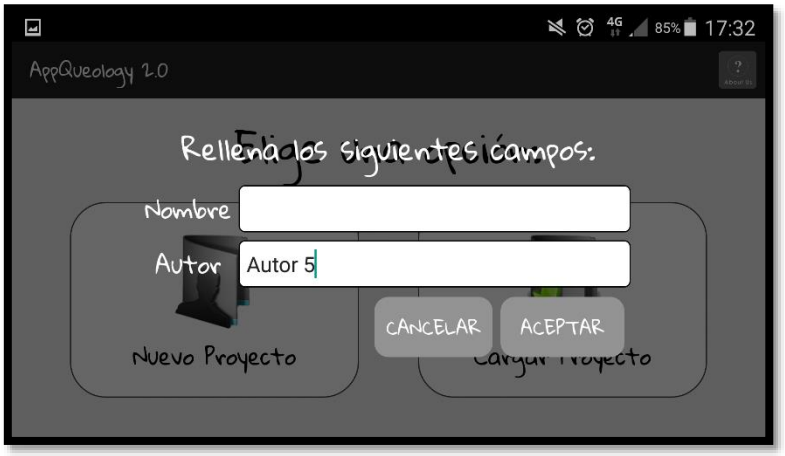

Ilustración 61 - Captura crear proyecto solo con campo autor.

Como se puede apreciar en la figura 61 se han introducido un valor para el campo autor y nada para el correspondiente al nombre del proyecto.

Tras realizar la prueba se ha obtenido el resultado que se puede apreciar en la figura 62 que se muestra a continuación.

Ilustración 62 - Captura del mensaje enviado tras realizar la prueba.

Como se puede observar en la figura 62 tras realizar la prueba se le avisa al usuario, por medio de un Toast, que no se ha podido crear el proyecto ya que no ha introducido un valor en el campo nombre.

7.1.3.8 Introducir un nombre de proyecto ya existente

Para esta prueba se ha introducido en el campo nombre, un valor de un proyecto ya registrado en la base de datos correspondiente. Con la intención de comprobar si se realiza la acción correctamente y se obtienen los resultados esperados.

Así se han introducido los datos que se pueden ver en la figura 63 que se presenta a continuación.

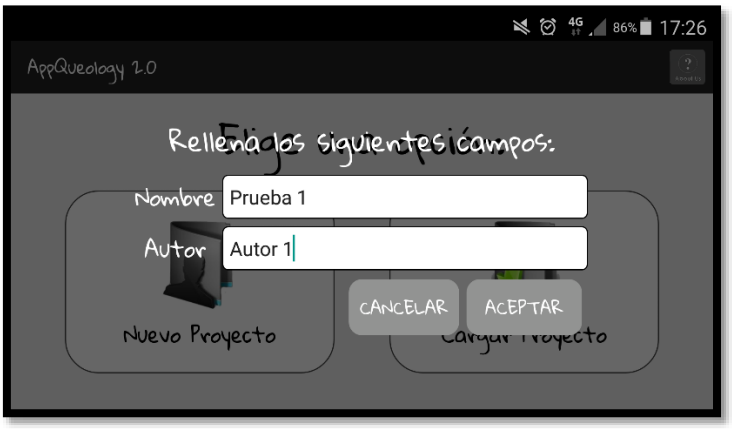

Ilustración 63 - Captura crear proyecto con datos ya almacenados en base datos

Como se puede apreciar en la figura 63 se han introducido datos ya introducidos para otro proyecto (figura 54).

Tras realizar esta prueba se ha obtenido los resultados que se muestran a continuación, en la figura 64.

Ilustración 64 - Captura de los resultados obtenidos tras la realización de la prueba.

Como se puede observar en la figura 64 no se ha podido crear el proyecto. Además ha podido constatarse como se le comunica al usuario por medio de un Toast, que ese nombre ya existe.

7.1.2 Eliminar un proyecto

En este apartado se intentará eliminar un proyecto.

Se ha intentado el eliminar el último proyecto creado de la lista de proyectos.

Tras hacer un clic largo sobre el último proyecto sale el mensaje que se puede observar en la figura 65.

Ilustración 65 - Captura del mensaje que pide la confirmación.

Como se puede observar en la figura 65 se nos pide que confirmemos si realmente queremos eliminar el proyecto.

Tras clicar sobre el botón Aceptar obtenemos el siguiente resultado que se observa en la figura 66.

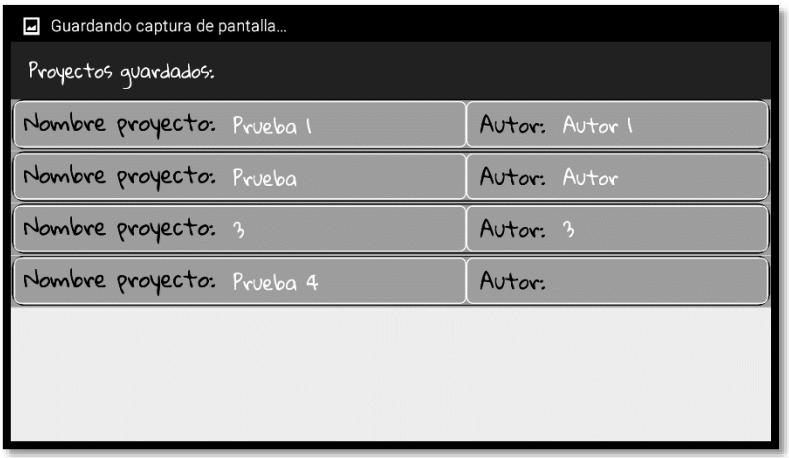

Ilustración 66 - Captura tras eliminar el último proyecto.

Como se puede observar en la figura 66 y comparándola con la figura 60 se observa que ha sido eliminado el último proyecto satisfactoriamente.

7.1.3 Crear una UE

En este apartado se creará una UE. Así se harán las siguientes pruebas.

- Crear una UE solo con letras en su campo Nombre.
- Crear una UE solo con números en su campo Nombre.
- Crear una UE con una combinación de letras y números en su campo Nombre.
- Crear una UE con un nombre que ya existe.
- Crear una UE sin nombre.

7.1.3.1 Crear una UE solo con letras.

En esta prueba se intentará crear una UE solo con letras en su campo Nombre.

Así, tras clicar en el botón correspondiente, que permite introducir el nombre de la UE que se pretende crear, se han introducido los valores que se pueden apreciar en la figura 67 que se muestra a continuación.

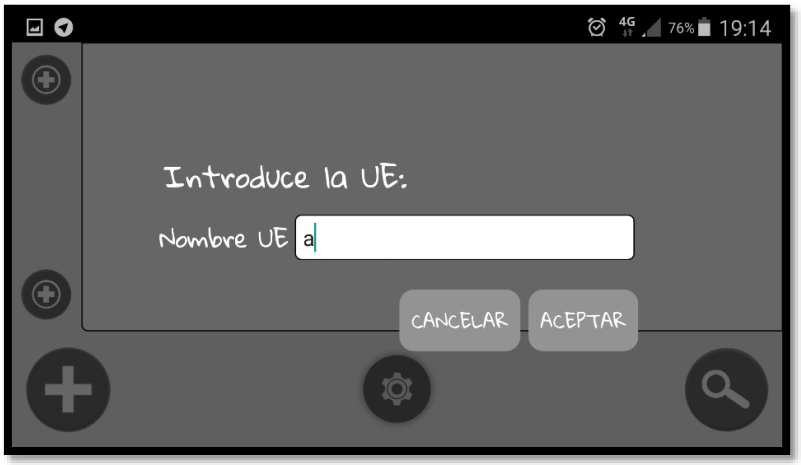

Ilustración 67 - Captura de la prueba en cuestión.

Como se puede observar en la figura 67 se ha introducido la letra “a” como nombre de la UE. Tras clicar sobre el botón Aceptar, se han obtenido los resultados que se muestran en la figura 68.

Ilustración 68 - Captura del resultado de la prueba.

Como se puede observar, en la figura 68, se ha podido crear la UE satisfactoriamente.

7.1.3.2 Crear una UE solo con números

En esta prueba se intentará crear una UE solo con números en su campo Nombre.

Así, tras clicar en el botón correspondiente, que permite introducir el nombre de la UE que se pretende crear, se han introducido los valores que se pueden apreciar en la figura 69 que se muestra a continuación.

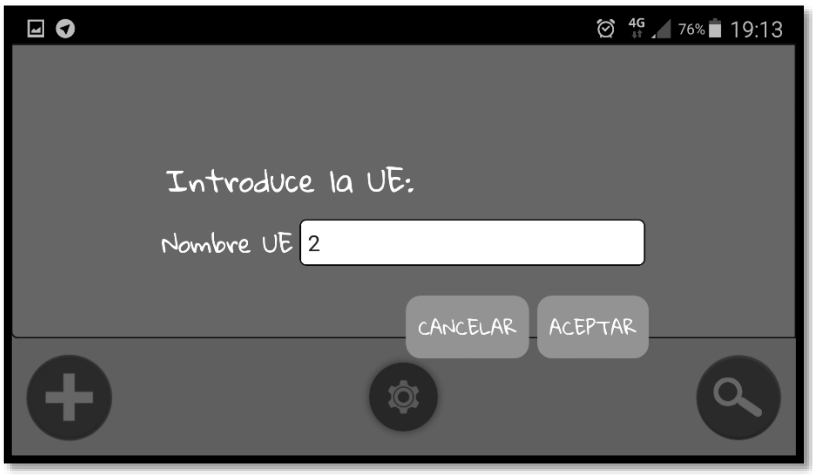

Ilustración 69 - Captura de la prueba en cuestión.

Como se puede observar en la figura 69 se han introducido el número “2” como nombre de la UE. Tras clicar sobre el botón Aceptar se han obtenido los resultados que se muestran en la figura 70.

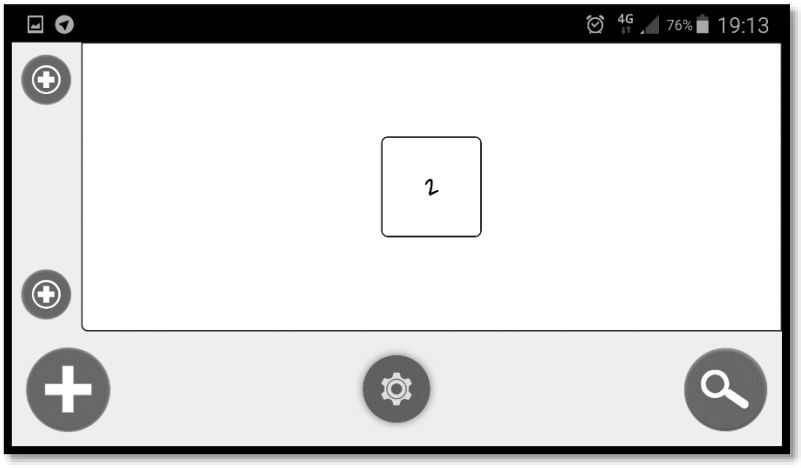

Ilustración 70 - Captura de los resultados obtenidos.

Como se puede observar, en la figura 70, se ha podido crear la UE satisfactoriamente.

7.1.3.3 Crear una UE con combinación de letras y números

En esta prueba se intentará crear una UE con números y letras en su campo Nombre.

Así, tras clicar en el botón correspondiente, que permite introducir el nombre de la UE que se pretende crear, se han introducido los valores que se pueden apreciar en la figura 71 que se muestra a continuación.

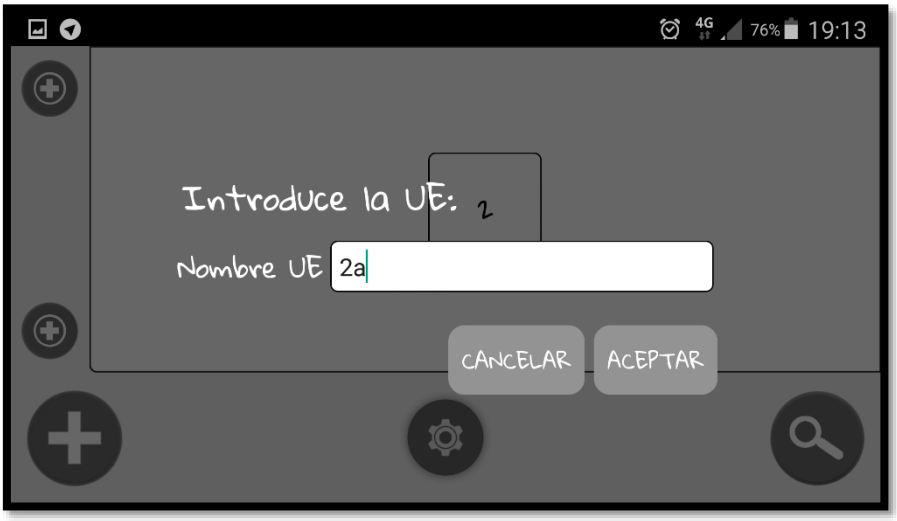

Ilustración 71 - Captura de la prueba en cuestión.

Como se puede observar, en la figura 71, se han introducido la combinación "2a" como nombre de la UE. Tras clicar sobre el botón Aceptar se han obtenido los resultados que se muestran en la figura 72.

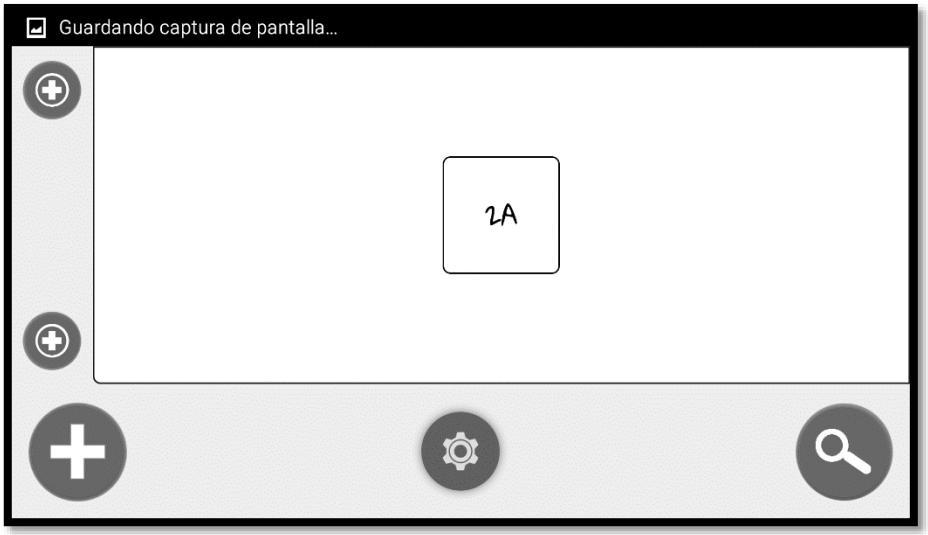

Ilustración 72 - Captura de los resultados obtenidos.

Como se puede observar, en la figura 72, se ha podido crear la UE satisfactoriamente.

7.1.3.4 Crear una UE con mismo nombre

En este apartado la prueba consiste en crear una UE con un nombre de una UE que ya ha sido almacenada previamente.

Así, se intentado crear una UE con nombre "2". Una UE que ya se encuentra en la base de datos.

El resultado obtenido, tras realizar los pasos pertinentes, son los que se pueden apreciar en la figura 73.

Ilustración 73 - Captura de los resultados obtenidos.

Como se puede observar, en la figura 73, no se ha podido crear dicha UE y se ve como se le avisa al usuario por medio de un Toast que ya existe una UE con ese nombre.

7.1.3.5 Crear una UE sin nombre

En esta prueba se intentará crear una UE, dándole al botón Aceptar del Alert, sin haber introducido previamente un nombre para dicha UE.

Los resultados obtenidos pueden observarse en la figura 74.

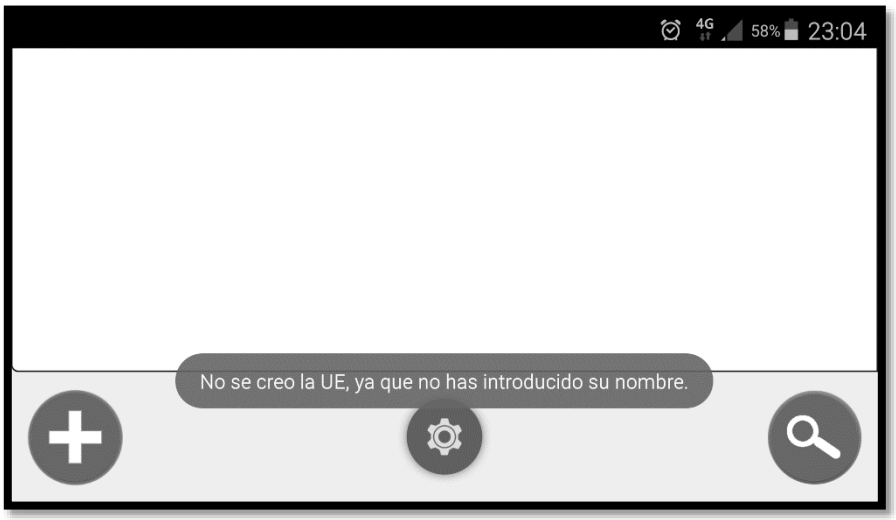

Ilustración 74 - Captura de los resultados obtenidos.

La figura 74 nos muestra como no ha sido posible crear la UE ya que no se ha introducido su nombre.

7.1.4 Crear relaciones

En este apartado se intentarán crear relaciones tanto de anterioridad, como posteriores.

Se ha probado a crear relaciones solo con letras, solo con números y con letras y números. Además de crear más de una relación.

Así tras realizar dichas pruebas se han obtenido los resultados que se observan a continuación en la figura 75.

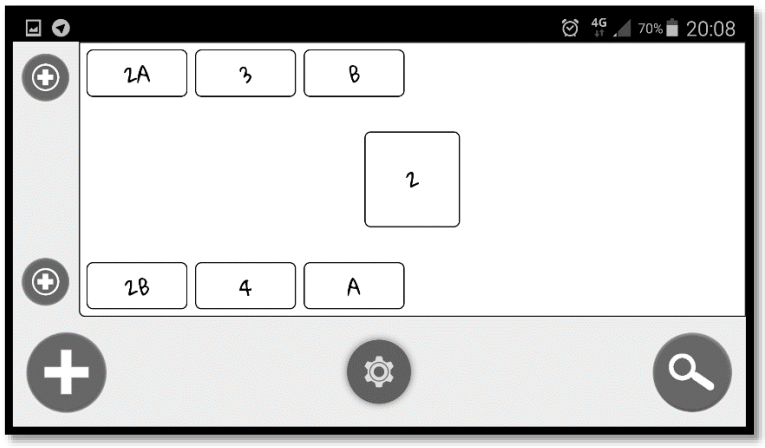

Ilustración 75 - Captura de los resultados obtenidos.

Como se puede apreciar en la figura 75 ha sido posible crear relaciones de anterioridad y posterioridad tanto con solo letras, solo números o una combinación de ambas.

También se ha intentado crear una relación clicando en el botón Aceptar del Alert que permite introducir su nombre, sin previamente haber escrito ningún valor en dicho campo. El resultado puede observarse en la figura 76.

Ilustración 76 - Captura de los resultados obtenidos.

La figura 76 nos muestra como no ha sido posible crear una nueva relación sin haber introducido un nombre previamente.

También se ha comprobado cómo es posible navegar entre las UEs, simplemente clicando sobre ellas.

7.1.5 Eliminar relaciones

En este apartado se eliminarán relaciones tanto de anterioridad como posteriores.

Así tras hacer un clic largo sobre la relación de anterioridad “3” y sobre la posterior “2b” se obtiene el siguiente resultado que se puede apreciar en la figura 77.

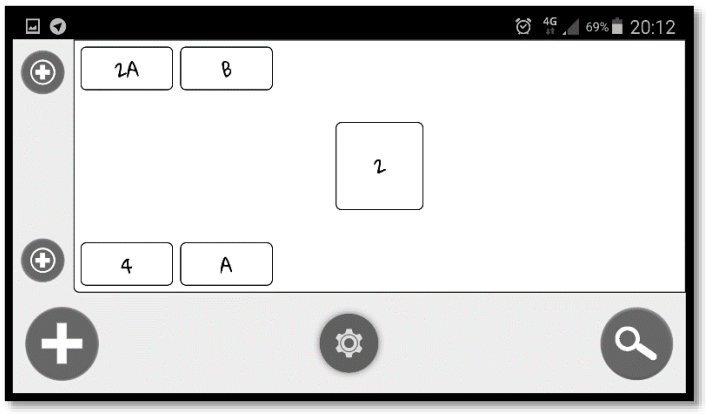

Ilustración 77 - Captura de los resultados de la prueba.

Tras confirmar que se deseaba eliminar las relaciones mencionadas, como se puede apreciar en la figura 77, ha sido posible eliminarlas satisfactoriamente.

También se ha comprobado que al eliminar una relación. La UE que dependía de dicha relación, en caso de no tener más padres, pasa a depender del nodo raíz. Por el contrario, si tenía más padres, seguirá ocupando la misma posición y únicamente se le habrá eliminado dicha relación.

7.1.6 Buscar una UE

En esta prueba se intentará buscar una UE almacenada previamente en la base de datos correspondiente.

Así, en primer lugar, se ha intentado buscar una UE “2b” que no existe en la BBDD. Y el resultado obtenido se puede apreciar en la figura 78.

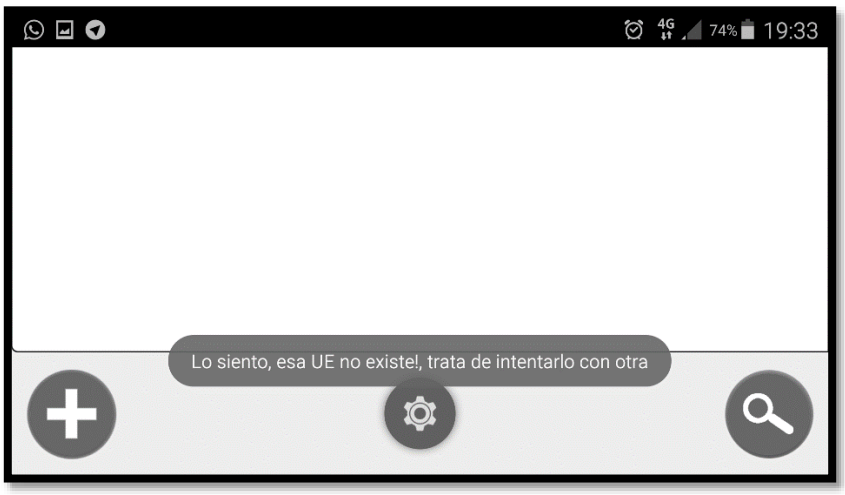

Ilustración 78 - Captura de los resultados obtenidos.

Como se puede apreciar, en la figura 78, no se ha cargado la UE buscada. A su vez, también se puede ver como se le comunica al usuario que dicha UE no existe.

Posteriormente, se ha probado con una UE que sí que se encuentra almacenada en la base de datos correspondiente, como se puede observar en la figura 79.

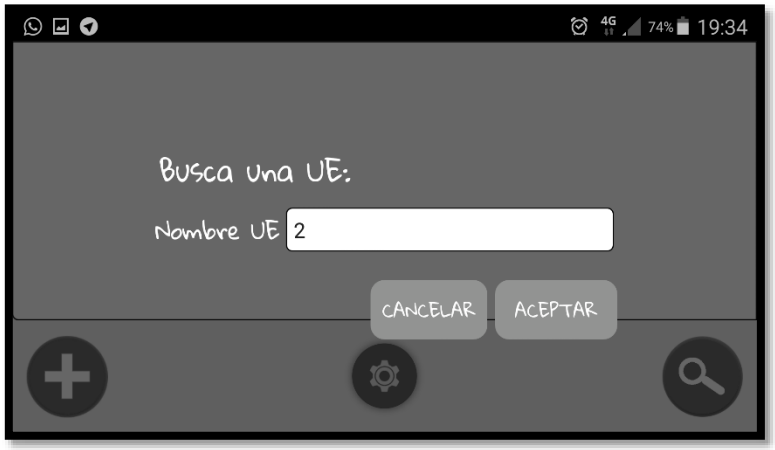

Ilustración 79 - Captura de la búsqueda de la UE "2".

Como se puede observar, en la figura 79, tras clicar el botón correspondiente que permite introducir el nombre de la UE que se pretende buscar, Se ha introducido el valor “2”. Posteriormente, tras seleccionar el botón Aceptar del Alert, se han obtenido los resultados que se pueden ver en la figura 80.

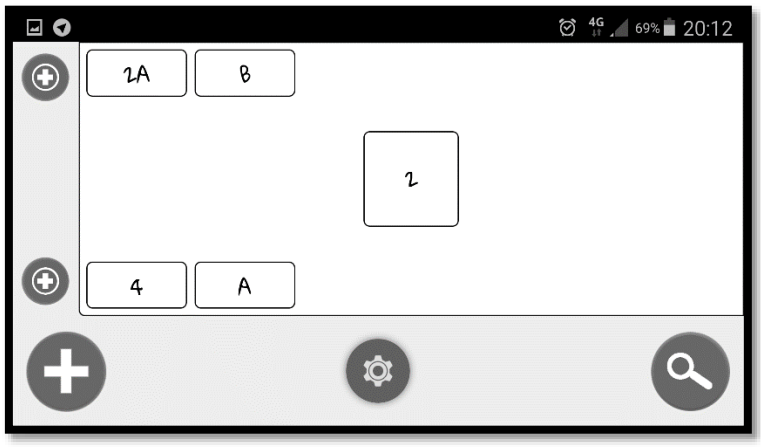

Ilustración 80 - Captura de los resultados obtenidos.

En la figura 80 se puede observar los resultados obtenidos. Se ve como se ha cargado tanto la UE “2” como todas las relaciones que dicha UE tiene.

7.1.7 Comprobar el Menú Flotante

En este apartado se ha comprobado si funcionan todos los botones que ofrece dicho menú y si estos realizan las tareas esperadas. Tras realizar las pruebas ha sido posible constatar que sí que funcionan correctamente. También se ha comprobado que se cierra siempre que haya una UE cargada en la vista.

Así, ha sido posible ver las instrucciones de uso de la vista, como se puede apreciar en la figura 81.

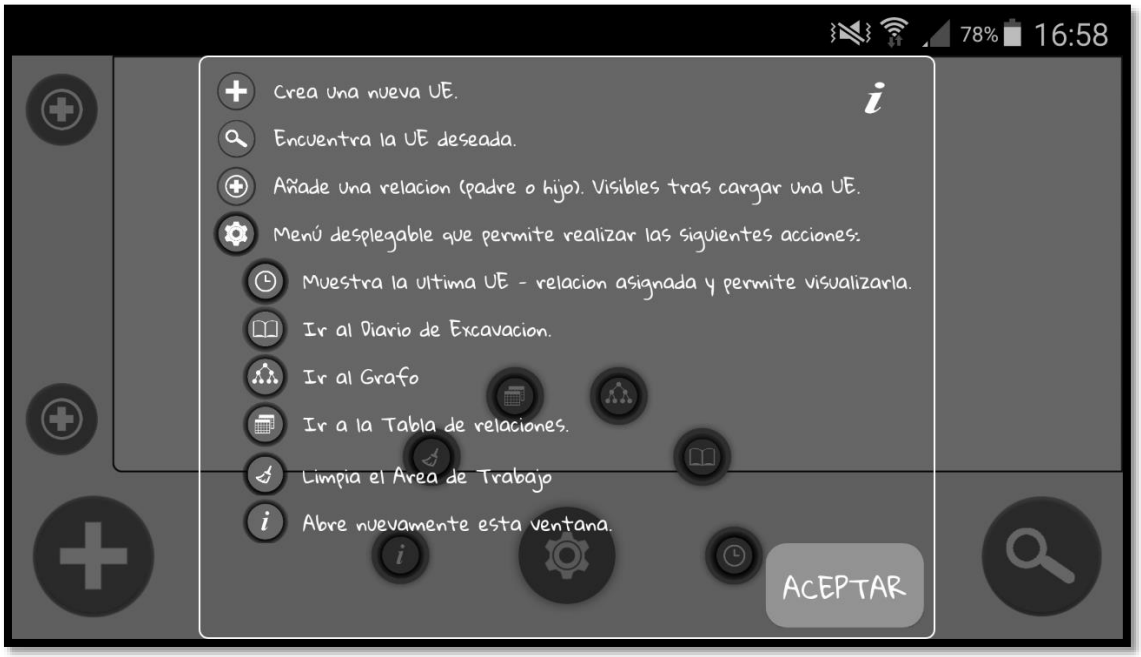

Ilustración 81 - Captura de las instrucciones de uso de la vista.

En la figura 81, se puede ver cómo, tras clicar en el botón correspondiente, se muestran las diferentes funcionalidades de los botones que presenta el menú flotante, así como las de los botones que presenta la vista. Al haberse diseñado el Alert, que muestra la información, con el fondo transparente, es posible ver donde se sitúan los botones mencionados.

También ha sido posible ver la información de la última UE y relación almacenadas. Como se puede apreciar en la figura 82.

Ilustración 82 - Captura de los resultados obtenidos.

En la figura 82 podemos ver cómo tras clicar en el botón apropiado se obtiene la información de la última UE almacenada, su fecha de creación, así como la última relación insertada en la base de datos correspondiente.

Con este caso se comprueba también que es posible crear una UE directamente en el momento que se establece una relación, en caso de no existir esta. Ya que como se puede ver en la figura 81 la última relación creada es la "A" y la última UE insertada, en la base de datos, también es la "A".

También se ha comprobado que tras clicar en el botón que vacía la pantalla ("la escoba") la acción se realizaba satisfactoriamente.

El resto de botones, aquellos que permiten visualizar el Grafo, la Tabla y el Diario de Excavación, funcionan también correctamente.

7.1.8 Agregar datos en la UE cargada

En este apartado se realizarán las siguientes pruebas.

- Guardar una UE sin nombre.
- Caracterizar una UE rellenando los campos correspondientes.
- Cambiar el nombre de una UE.
- Cambiar el nombre de una UE al de una UE que ya existe.
- Asociar un punto GPS y una foto a una UE determinada.

La figura 83 nos ilustra con la vista con la que vamos a trabajar.

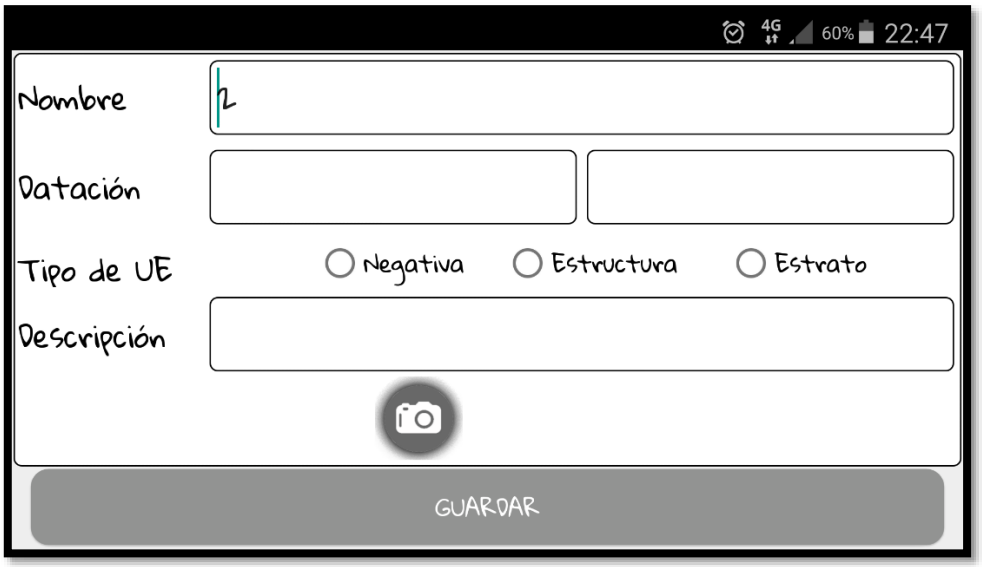

Ilustración 83 - Captura de la vista con la que haremos las pruebas.

La figura 83 nos muestra los campos que pueden rellenarse para caracterizar una UE. Por otro lado, se observa cómo se cargan los valores que la UE tiene almacenados en el momento de abrir la vista. En este caso solo el nombre de la UE.

7.1.8.1 Guardar una UE sin nombre

En esta prueba lo que se ha intentado es borrar el nombre de la UE e intentar guardar los cambios realizados para ver qué pasa.

En la figura 84 se puede apreciar los resultados obtenidos.

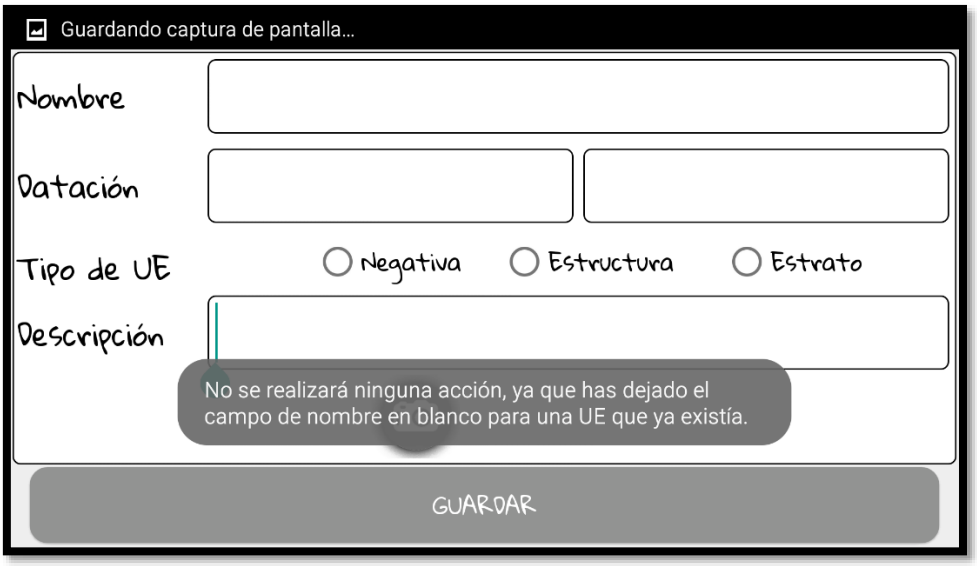

Ilustración 84 - Captura de los resultados obtenidos.

Como nos muestra la figura 84 no ha sido posible realizar la acción. Al mismo tiempo se observa como se le indica al usuario cual es la causa del error.

7.1.8.2 Caracterizar una UE determinada

En este apartado, se rellenarán los campos que ofrece una UE cargada previamente en la parte central de la vista Área de Trabajo, como puede apreciarse en la figura 85.

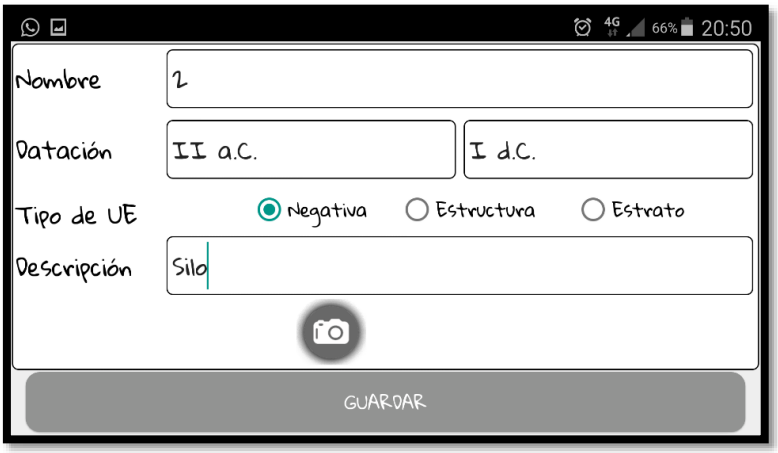

Ilustración 85 - Captura de los resultados obtenidos.

En la figura 85 se nos muestra cómo ha sido posible introducir en los campos de datación y descripción tanto números como letras, como se había solicitado. Como el tipo de UE solo puede tener una opción clicada en cada momento.

Tras clicar sobre el botón guardar y volver a abrir la vista, se vuelven a cargar todos los valores anteriormente introducidos, demostrando que los datos se han guardado correctamente.

7.1.8.3 Cambiar el nombre de la UE

En esta prueba se ha intentado cambiar el nombre de la UE "2" por "444" y guardar los cambios realizados.

En la figura 86 se muestran los resultados obtenidos tras haber clicado en la opción Aceptar del Alert que pide la confirmación por parte del usuario para que sea modificada la UE.

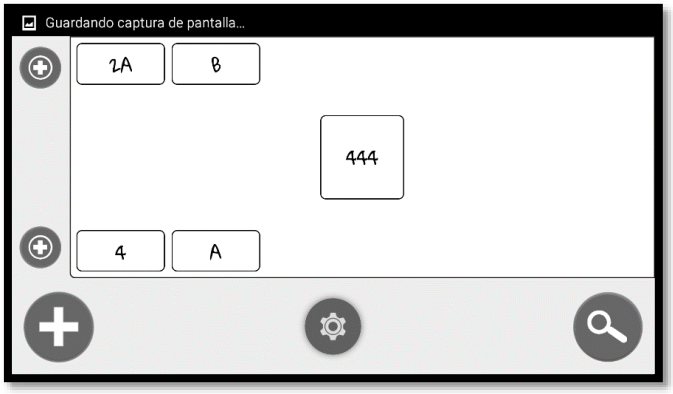

Ilustración 86 - Captura de los resultados obtenidos.

Como se puede apreciar, en la figura 86, ha sido posible realizar los cambios satisfactoriamente.

7.1.8.4 Cambiar el nombre de la UE al de uno que ya existe

En esta prueba se ha cambiado el nombre de la UE al de otra UE que ya existía previamente.

Así en la figura 87 puede apreciarse los resultados obtenidos tras haber intentado cambiar el nombre por el de "3" que es una UE que ya existe almacenada en la BBDD.

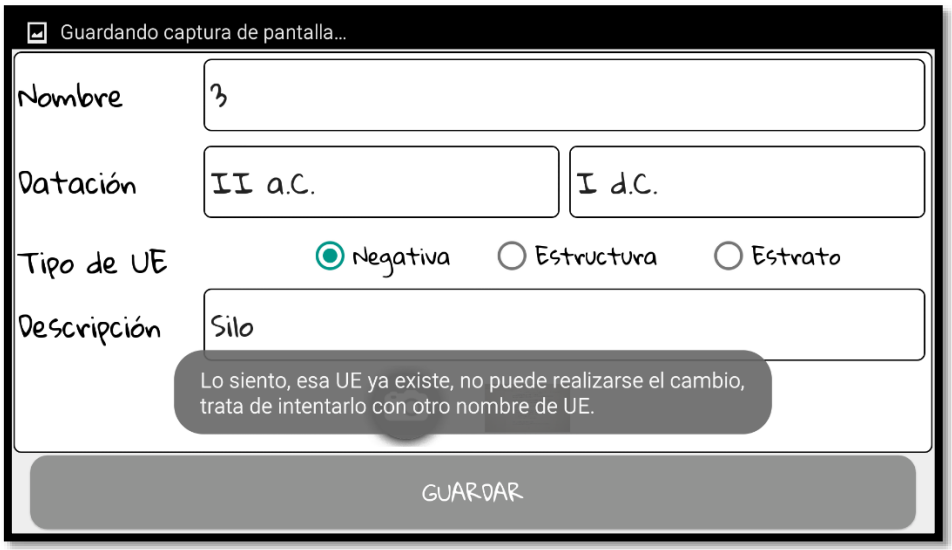

Ilustración 87 - Captura de los resultados obtenidos.

En la figura 87 se puede ver como no ha sido posible realizar el cambio. También como se le indica al usuario cuál ha sido la causa del error.

7.1.8.5 Asociar un punto GPS y una foto

En esta prueba se ha intentado asociar un punto GPS y una foto a una UE determinada.

Para realizar la acción del GPS hay que clicar una vez sobre el botón de la cámara.

Se ha comprobado como en caso de no tener activado GPS en el dispositivo se le avisa al usuario, como puede apreciarse en la figura 88.

Ilustración 88 - Captura del alert que avisa al usuario de que tiene desactivado el GPS.

Como puede apreciarse, en la figura 88, se le posibilita tanto la opción de sacar la foto sin una previa obtención de un punto GPS o por el contrario activar dicho servicio.

Una vez activado, se ha obtenido el siguiente resultado que puede apreciarse en la figura 89.

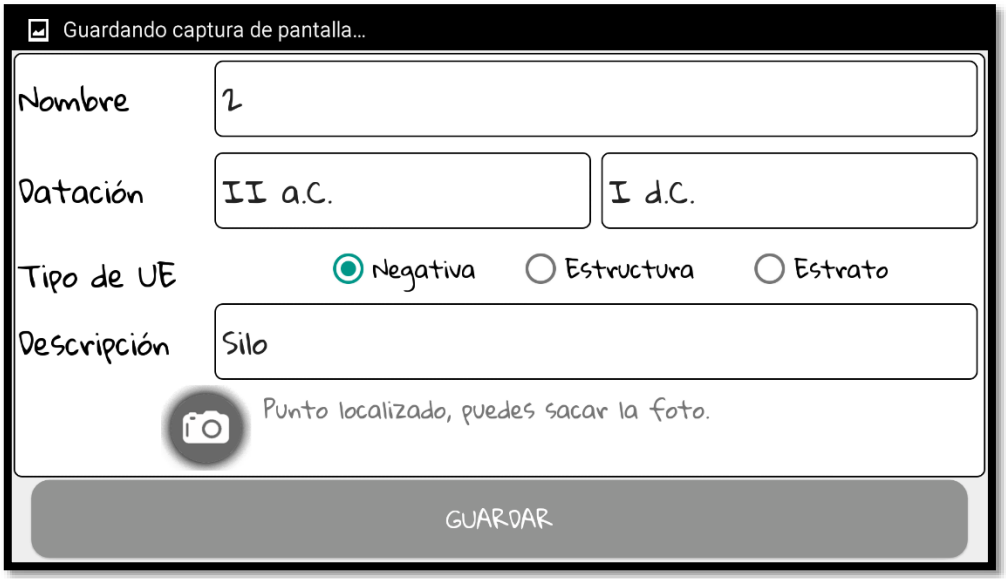

Ilustración 89 - Captura de los resultados obtenidos.

Puede observarse en la figura 89 como ha sido posible asociar un punto GPS a una UE dada. Se ha constatado que probablemente hubiera sido mejor implementar un botón para cada opción (uno para la foto y otro para el GPS). Y que no se realizaran las dos acciones con el mismo botón, ya que, de este modo, no queda del todo claro como funciona la vista.

Por otro lado, se han seguido con las pruebas y se a clicado nuevamente en el botón de la cámara. Y ras sacar la foto se han obtenido los siguientes resultados que se pueden ver en la figura 90.

Ilustración 90 - Captura de los resultados obtenidos.

La figura 90 nos muestra cómo ha sido posible asociar una foto a una UE determinada. También se ha comprobado que una vez cerrada la vista y vuelta a abrir, los datos insertados durante esta prueba, aparecen cargados correctamente.

Se ha comprobado también cómo una vez clicado sobre la foto en miniatura, esta, se abre en pantalla completa. Y como dándole al botón back del dispositivo, se vuelve a la vista que se muestra en la figura 89.

7.1.9 Vista Diario de Excavación

En esta prueba se ha comprobado si los scrolls vertical y horizontal que presenta la vista funcionan correctamente. Y ha dado un resultado positivo.

Posteriormente, se ha comprobado si se cargan correctamente los datos que se había almacenado para cada una de las UEs almacenadas. Y también se ha obtenido un resultado satisfactorio.

Se ha comprobado que si había una UE cargada en la vista Área de Trabajo, al abrir la vista que nos ocupa, dicha UE aparecía resaltada con un color diferente.

Por otro lado, también se ha clicado sobre una de las UEs y se ha comprobado que esta cambia su aspecto para comunicarle al usuario que ha sido seleccionada. Además se ha comprobado que al volver a la vista Área de Trabajo, dicha UE se encontraba cargada en la parte central de la vista.

Así como se ha supervisado si tras clicar en las imágenes almacenadas para cada una de las UEs se abrían estas en pantalla completa. También se ha obtenido el resultado deseado.

También se ha buscado una UE utilizando el botón implementado para tal función y se ha conseguido lo esperado. Que la UE buscada apareciera resaltada entre las otras, con un color diferente.

Por último, se ha eliminado una UE con un clic largo. Tras aceptar en el Alert que pide la confirmación, se ha observado como dicha UE ha sido eliminada. Tras realizar esta acción se ha comprobado que las UEs que dependían de dicha UE han pasado a depender del nodo raíz o inicio.

7.1.10 Vista Grafo

En este apartado se describirán las pruebas realizadas para esta vista.

Se ha comprobado que si se encontraba una UE cargada en la vista Área de Trabajo al abrir la vista que nos ocupa, dicha UE aparecía resaltada con un color diferente.

Por otro lado, también se ha clicado sobre una de las UEs y se ha comprobado que esta cambia su aspecto para comunicarle al usuario que ha sido seleccionada. Además se ha comprobado que al volver a la vista Área de Trabajo, dicha UE se encontraba cargada en la parte central de la vista.

También se ha buscado una UE utilizando el botón implementado para tal función y se ha conseguido lo esperado. Que la UE buscada apareciera resaltada entre las otras, con un color diferente.

También se ha comprobado que funcionaba el zoom como pueden apreciarse en las figuras 91 y 92.

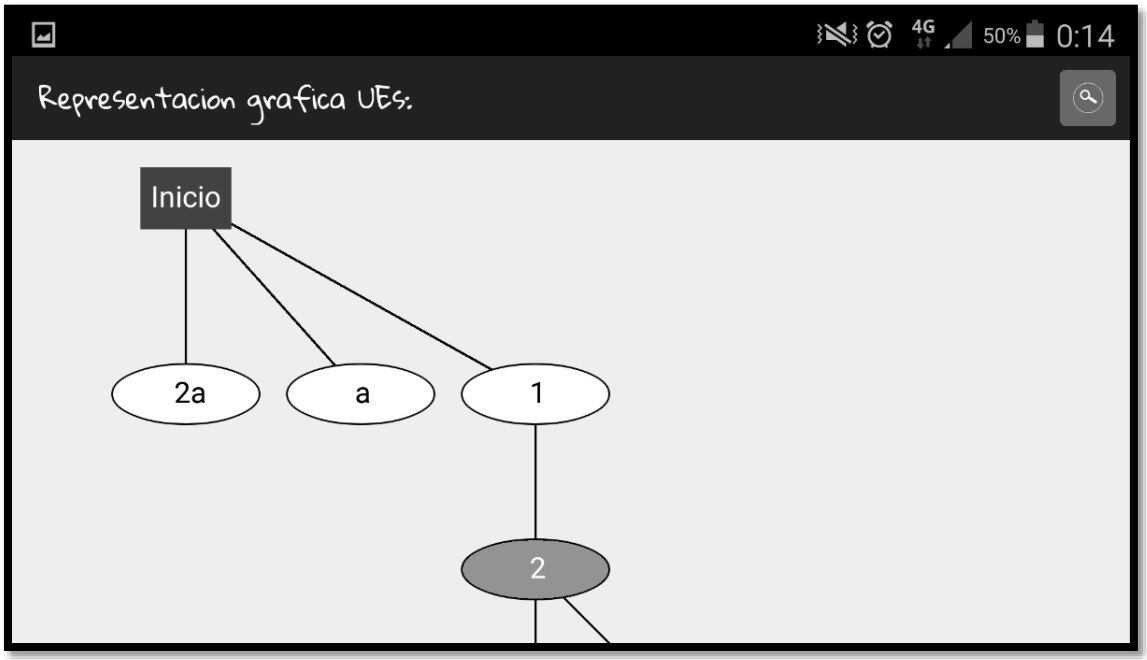

Ilustración 91 - Catura sin haber aplicado zoom.

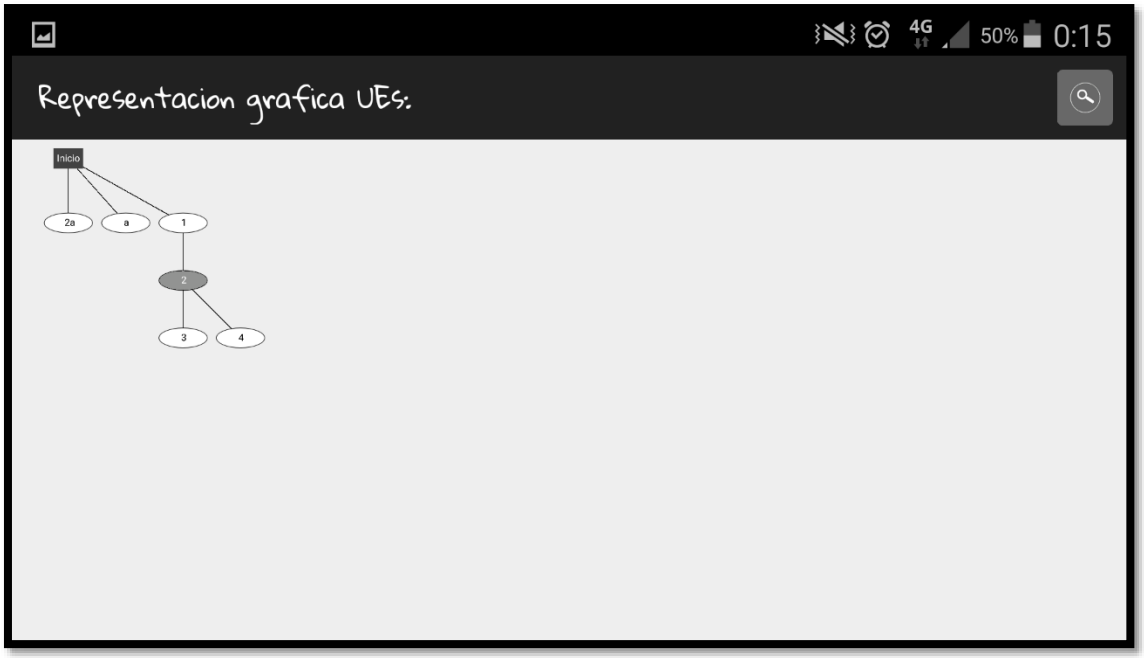

Ilustración 92 - Captura tras haber aplicado zoom out.

Como puede apreciarse en la figura 91 y 92 se ha podido constatar que es posible hacer zoom out y zoom in, pero se ha notado que funciona mejor si el zoom se aplica con los dos dedos dibujando una línea vertical. Mejor que si se realizaba dicha acción en horizontal o diagonal.

7.1.11 Vista Tabla

En esta apartado se comentarán las pruebas realizadas en esta vista.

Se ha comprobado que si se encontraba una UE cargada en la vista Área de Trabajo al abrir la vista que nos ocupa, dicha UE aparecía resaltada con un color diferente.

Por otro lado, también se ha clicado sobre una de las UEs y se ha comprobado que esta cambia su aspecto para comunicarle al usuario que ha sido seleccionada. Además se ha comprobado que al volver a la vista Área de Trabajo, dicha UE se encontraba cargada en la parte central de la vista.

También se ha buscado una UE utilizando el botón implementado para tal función y se ha conseguido lo esperado. Que la UE buscada apareciera resaltada entre las otras, con un color diferente.

También se ha comprobado que tras clicar una UE se mostraban las relaciones directas e indirectas con otro color como se puede apreciar en la figura 93.

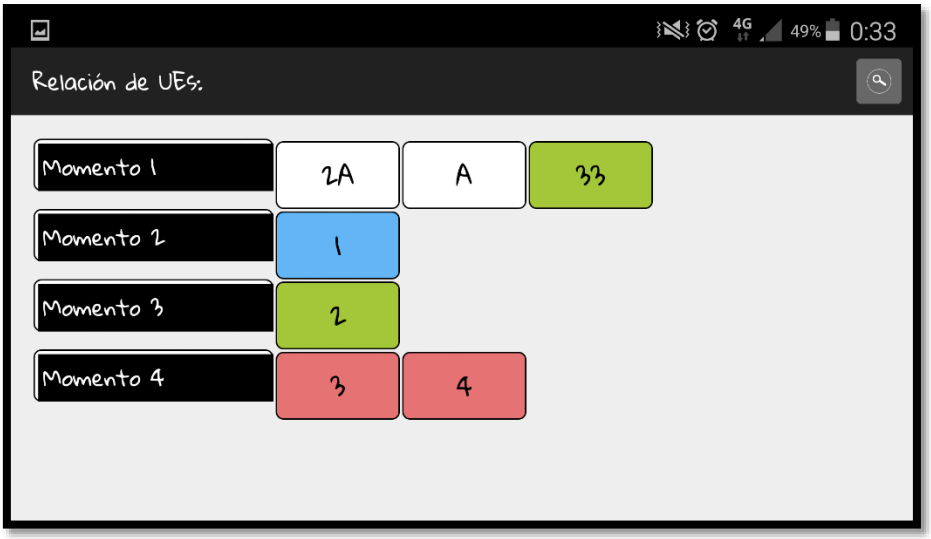

Ilustración 93 - Captura de los resultados obtenidos.

La figura 93 nos muestra cómo tras clicar sobre la UE "1" esta se ha seleccionado y se han mostrado en verde sus relaciones directas y en rojo las indirectas.

7.1.12 Detección de ciclos

En este apartado se comprueba si se detectan bien los ciclos.

Así se ha creado un caso de ejemplo que simula la aparición de un ciclo. El resultado obtenido puede observarse en las figura 94 y 95.

Ilustración 94 - Captura de cómo se informa la usuario de la aparición de un ciclo.

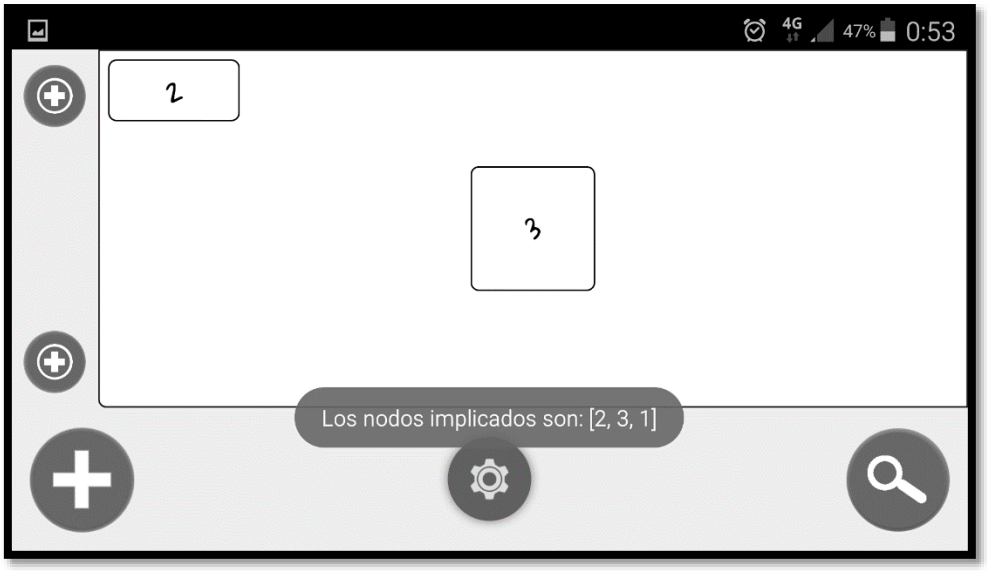

Ilustración 95 - Captura de cómo se informa al usuario de las UEs implicadas en el ciclo.

En la figura 94 se puede ver cómo tras introducir una relación que genera un ciclo se le informa al usuario por medio de un Toast que dicha relación crea un ciclo. Aunque, tras realizar la acción, puede observarse que no se ha creado dicha relación, igual sería más claro si se le indicara en el Toast que la relación no se ha guardado. Ya que sino el usuario podría confundirse y pensar que no se ve la relación introducida o algo parecido.

En la figura 95 se observa como se le indica al usuario las UEs que crean el ciclo para que pueda razonar el por qué y no volver a repetir dicha acción.

7.1.13 Eliminación de redundantes

En este apartado se ha comprobado si se detectan y eliminan correctamente las relaciones redundantes. La primera prueba realizada es la que se puede apreciar en la figura 96.

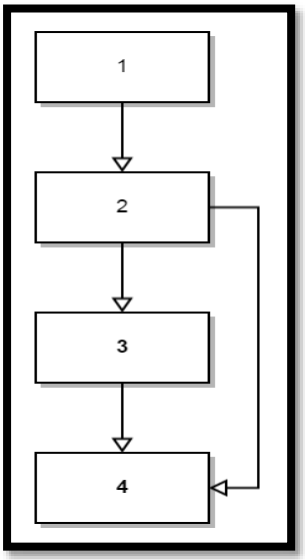

Ilustración 96 - Caso de Prueba 1

Tras realizar esta prueba se ha podido constatar como la relación 2-4 no aparece representada en la vista Grafo.

Por otro lado, también se ha comprobado que, por el contrario, en la vista Área de Trabajo dicha relación eliminada en la vista Grafo, sí que aparece reflejada.

Constatando, por tanto, que la eliminación únicamente se ha realizado en dicha vista y no es que se elimine del registro de datos.

Por otro lado, se ha realizado la siguiente prueba que se puede observar en la figura 97.

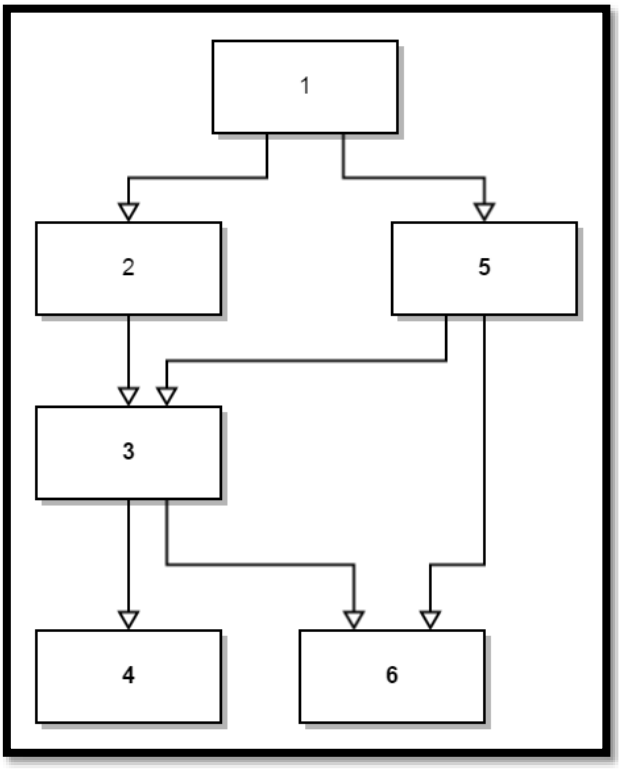

Ilustración 97 - Caso creado para comprobar si se eliminan correctamente las relaciones redundantes.

Tras realizar la prueba sobre el caso presentado en la figura 97 se han obtenido los siguientes resultados que se pueden apreciar en las figuras 98 y 99.

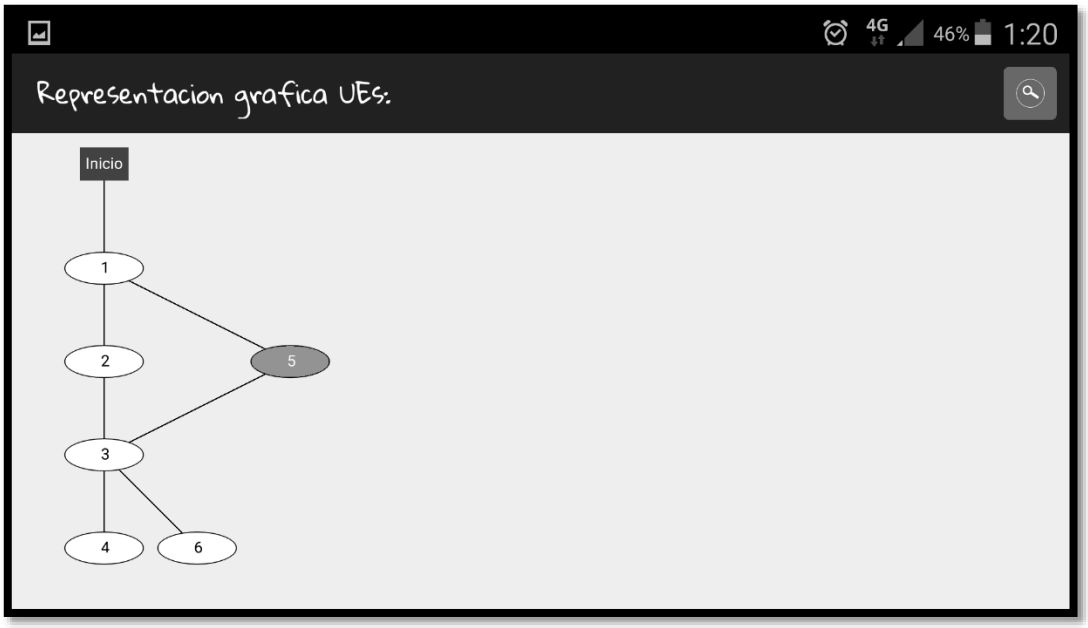

Ilustración 98 - Captura de los resultados obtenidos en la vista Grafo

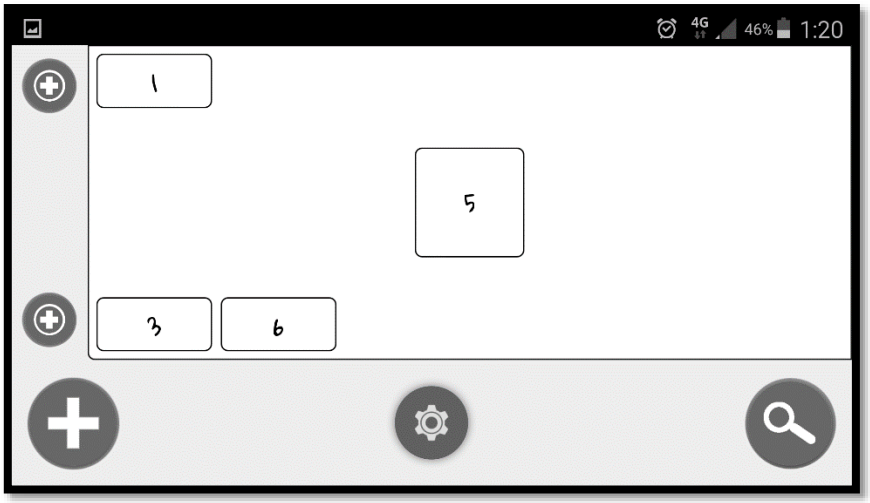

Ilustración 99 - Captura del resultado obtenido en la vista Área de Trabajo.

La figura 98 nos muestra como se ha eliminado correctamente en la vista Grafo la arista redundante que sería la 5-6.

Por otro lado, en la figura 99, se puede ver cómo tras pasar de la vista Grafo a la vista Área de Trabajo la relación sigue existiendo.

7.2 Comentarios

En este apartado se harán una serie de comentarios referentes a las pruebas realizadas.

Me hubiera gustado probar la aplicación con un caso real. Introduciendo todas las UEs y relaciones que pueden darse en una Excavación de Arqueología.

Con ello se conseguiría información referente al rendimiento de la aplicación, a los tiempos de respuestas de la aplicación. Saber si es necesario crear más procesos en segundo plano que realicen los diferentes cálculos y que permitieran hacer la aplicación más ligera. O por el contrario, si esta ya soporta un gran volumen de datos y funciona correctamente.

Pero como se ha comentado al inicio de este apartado de pruebas, un mal entendido ha ocasionado una importante pérdida de tiempo que ha imposibilitado realizar este ejercicio.

Por otro lado comentar que el resto de las pruebas realizadas han sido satisfactorias.

8 Conclusión

Ya que este no ha sido un trabajo de investigación propiamente dicho, sino más bien un proyecto que prende desarrollar una aplicación Android funcional para solventar un problema dado, no se obtendrán unas conclusiones abstraídas a partir de dicho proceso de investigación. Sino que son conclusiones obtenidas más a parir de un proceso de investigación más individual y personal, de adquisición de conocimientos y de afianzamientos de otros.

Es decir, en este apartado se hará un breve resumen de los puntos principales que aborda el proyecto, se comentarán las sensaciones que me ha producido el realizarlo y algunas reflexiones sacadas de su elaboración.

8.1 Conclusiones

AppQuelogy 2.0, la aplicación Android hecha por y para arqueólogos, ha pretendido ser un software que ayude en la gestión y documentación de una excavación arqueológica.

Ha sido desarrollada con la intención de acercar las posibilidades que ofrecen los dispositivos móviles a una disciplina que no ha bebido demasiado de la fuerte explosión tecnológica de los últimos años.

Quince semanas de trabajo han dado lugar a una aplicación que creo que es capaz de dejar en evidencia como un dispositivo de este tipo puede ayudar en los procesos mencionados. Como la Arqueología puede aprovecharse de ello y la necesidad que hay de que esto no se quede aquí y que esta disciplina siga adaptándose a los nuevos tiempos.

El haber programado en Android me ha resultado realmente gratificante, hasta tal punto que, tras la realización de este breve trabajo, creo que es uno de los campos que más me atrae actualmente. He aprendido muchas cosas durante la elaboración de este proyecto, pero también me ha servido para tomar consciencia de que hay muchísimas cosas que aún desconozco.

Por otro lado, los conocimientos adquiridos a lo largo de la implementación y desarrollo de la aplicación no han llegado siempre en el momento esperado o deseado por lo que actualmente si tuviera que realizar otra vez la aplicación probablemente no saldría igual. Por ejemplo he descubierto demasiado tarde los Fragment y por tanto no he podido aprovechar muchas de sus funcionalidades o han sido utilizadas tarde.

Comentar en este punto que, aunque parezca un poco ridículo, lo que me ha causado más dificultades es la parte referente al almacenamiento de las fotos y el trabajo con ellas. Me ha robado muchas horas el no haberme dado cuenta de una cuestión tan simple como es que el tamaño de la foto influía tanto en el almacenamiento como en la relación existente entre esta y el contenedor donde se pretendía visualizar así como con el dispositivo utilizado.

Me ha gustado trabajar con Android Studio, un IDE que no había probado hasta el momento y que me resultó realmente cómodo e incluso mejor que Eclipse, por la cantidad de shortcuts que presenta y el amplio y buen sistema de sugerencias que presenta y ofrece.

Por otro lado este trabajo me ha ayudado a refrescar muchas de las ideas que he aprendido durante los años que he cursado el Grado y que hasta el momento se encontraban inconexas, sin formar un todo. Este proyecto ha sido utilizado para unificar muchos de esos conocimientos y me ha servido para relacionar y comprender el porqué de muchas de las asignaturas impartidas.

También me ha hecho pensar mucho sobre el tema y como muchas materias, como por ejemplo Tratamiento de Imágenes o Visión Artificial podrían ayudar a la Arqueología y como estos campos podrían aplicarse a esta.

Todo esto me ha motivado para, en el futuro, seguir trabajando en este campo y continuar desarrollando software que pueda suponer una mejora para el desempeño de una disciplina que me gusta tanto como es la Arqueología y que desgraciadamente ha sido tan desmantelada en estos últimos años.

9 Trabajo futuro

En este apartado se pretende describir la posible continuidad del trabajo realizado en un momento posterior a la elaboración de este que nos ocupa.

9.1 Continuidad

Como se ha mencionado en diferentes ocasiones a lo largo de esta memoria, este trabajo ha sido realizado en colaboración con la Facultad de Historia de la Universidad de Barcelona. Esta colaboración es fruto de años de trabajos en común que han dado lugar a diferentes proyectos y producciones de diferente ámbito.

De hecho, este proyecto, inicialmente, pretendía continuar el realizado por Joaquim M. Fornaguera Marimon durante el 2004, en colaboración con la Facultad de Historia de dicha Universidad, pero desde un punto de vista más funcional y no tanto desde un enfoque relacionado con la usabilidad.

Por otro lado, este proyecto pretendía también adaptar y actualizar el programa Stratigraf[17] a los nuevos dispositivos móviles. Pero esta labor se ha visto empañada por el simple hecho de tener que adaptar los requisitos y las funcionalidades que debería cumplir la aplicación al marco en que se desarrolla este Proyecto de Fin de Grado.

Debido a que los plazos de dicho proyecto vienen dados, son inamovibles y no adaptables a las necesidades de elaboración del producto final, ha sido necesario, como ya se ha dejado intuir, reducir los objetivos que se tenían inicialmente, a unos que fueran abarcables por un trabajo de estas características.

AppQueology 2.0 ha generado un contexto relativamente propicio para que pueda ser continuada, ampliada y readaptada a los requisitos iniciales que presentaron los investigadores de la Factultad mencionada. Ya que han quedado aspectos pendientes como:

- Diseñar la aplicación para su uso en un dispositivo tipo tablet.
- Que la base de datos no sea local, sino que se encuentre en un servidor que permita el acceso a los datos desde diferentes dispositivos a la vez.
- Que las diferentes bases de datos implementadas soporten datos de tipo vectorial o espacial.
- Desarrollar un programa informático para ordenador desde el que se puedan gestionar los datos obtenidos en campo.
- Que para la vista Tabla se ofrezcan recomendaciones de periodización a partir de las UEs anteriores y posteriores ya datadas.
- Agrupación de estas en periodos históricos.
- Entro otras muchas cosas…

[17] Programa diseñado hace unos años por investigadores de la Facultad de Historia de la Universidad de Barcelona.

El tener avanzado todo este trabajo en la obtención de requisitos, el comprender después de meses de trabajo y de reuniones semanales que es lo que se espera realmente de la aplicación a desarrollar, así como el hecho de haberme dedicado anteriormente de forma profesional a la Arqueología y ser conocedor de las necesidades que hay hoy en día en dicho campo, creo que, como ya se ha mencionado, es el momento propicio para que AppQueology 2.0 no se quede aquí y se continúe ampliando y adaptando, con el objetivo de obtener una aplicación que pueda ser tanto utilizada en excavaciones reales, como dentro del mundo universitario. Con la intención de ayudar en la docencia, en cuestiones relacionadas con aspectos metodológicos, de gestión o de documentación arqueológica.

Por lo que espero que sea posible encontrar los recursos necesarios para que esta colaboración continúe y repercuta satisfactoriamente a ambas partes.

10 Bibliografía

Cuando se elabora una bibliografía se deben describir los documentos consultaos redactando referencias bibliográficas. Se trata de facilitar el acceso futuro a los documentos originales haciendo constar los datos fundamentales de cada documento de manera sencilla pero normalizada.

Así, en este apartado, se recopilará todos los recursos utilizados para la elaboración de un proyecto de estas características. Recogiendo todos los artículos, libros y fuentes necesarias para la creación de este Trabajo de Fin de Grado.

10.1 Bibliografía empleada

Libros

- [AHS06] Abraham Silberschatz, Henry F. Korth, S. Sudarshan. "*Fundamentos de Bases de Datos*". 5ª Edicion. Basauri. McGraw-Hill/Interamericana de España. 2006. ISBN: 0-07-295886-3.

- [CP98] Colin Renfrew, Paul Bahn. "*Arqueología. Teorías, Métodos y Práctica*". 2ª Edición. Madrid. Akal. 1998. ISBN: 84-460-0234-5.

- [DG01] David Arnow, Gerald Weiss. "*Introducción al a programación con JAVA. Un enfoque orientado a objetos*". Madrid. Pearson Educación S.A. 2001. ISBN: 84-7829-033-8.

- [J13] Jesús Tomás Gironés. "*El gran libro de Android*". 3ª Edición. Barcelona.Marcombo.2013. ISBN: 978-84-267-1976-8.

- [L03] Larman, C. "*UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado*". 2ª Edicion. Madrid. Prentice Hall. 2003. ISBN: 8420534382.

Artículos y trabajos de investigación:

- [UAM07] Escuela Politécnica Superior Universidad Autónoma de Madrid. "*Desarrollo del software*". 2007.

- [INE14] Instituto Nacional de Estadística. "*Encuesta sobre Equipamiento y Uso de Tecnologías de Información y Comunicación en los Hogares*". 2014.

- [IAB13] Interactive Advetising Bureau. "*V Estudio Anual IAB Spain Mobile Marketing. Informe de Resultados*". Septiembre 2013.

- [J04] Joaquim M. Fornaguera Marimon. "*Appqueology*". Barcelona. 2004

- [MV04] Museo d' historia de València. "*Recomendaciones Metodológicas para el Trabajo Arqueológico en la ciudad de Valencia*". Enero 2004.

- [R99] Rebeca Blanco Rotea. "*Metodología para el análisis estratigráfico del patrimonio construido y su aplicación en San Fiz de Solovio*". Santiago de Compostela. Septiembre 1999.

Sitios Web:

- Stack Overflow. Disponible en: https://stackoverflow.com

- Universidad Politècnica de València. "*Diploma de Especialización en desarrollo de aplicaciones para Android*". Disponible en :

file:///C:/Users/unodtantos/Desktop/Las%20versiones%20de%20Android%20y%20niveles%20de%20API%20-%20Diploma%20de%20Especializaci%C3%B3n%20en%20desarrollo%20de%20aplicaciones%20para%20Android.html

- Wikipedia. Disponible en: https://es.wikipedia.org/wiki/Wikipedia

11 Apéndice de contenidos

En este apartado se incluirá información que por cuestiones de espacio o de estructura del proyecto o incluso para evitar distracciones del lector, no se ha incluido dentro del texto y era merecedor de una sección aparte e individualizada para tal caso.

11.1 Apéndice

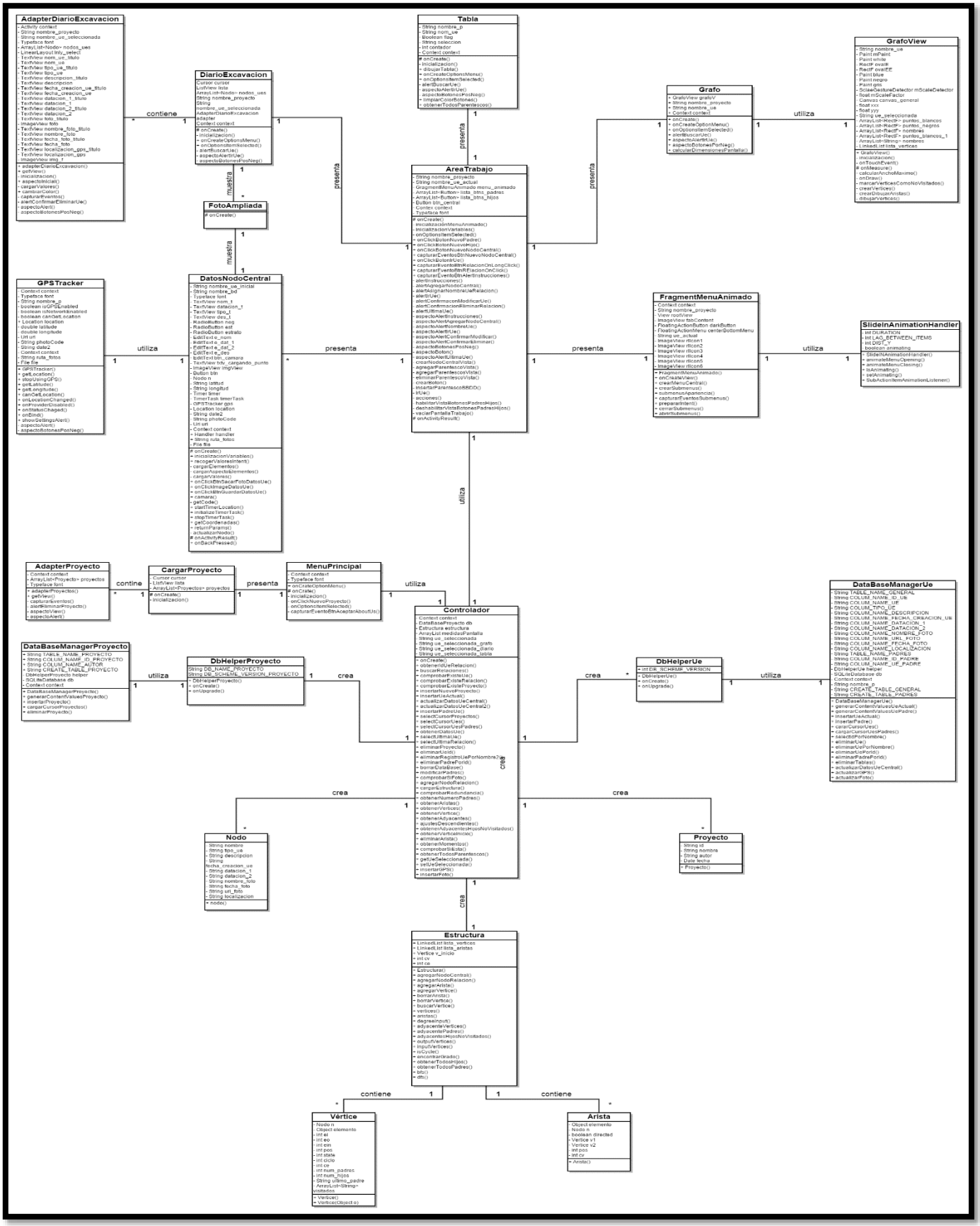

Ilustración 100 - Diagrama de clases