Universidad de las Ciencias Informáticas

Facultad 2 **Dirección de Seguridad Informática**

Aplicación Android de gestión de información de vulnerabilidades para la Dirección de Seguridad Informática

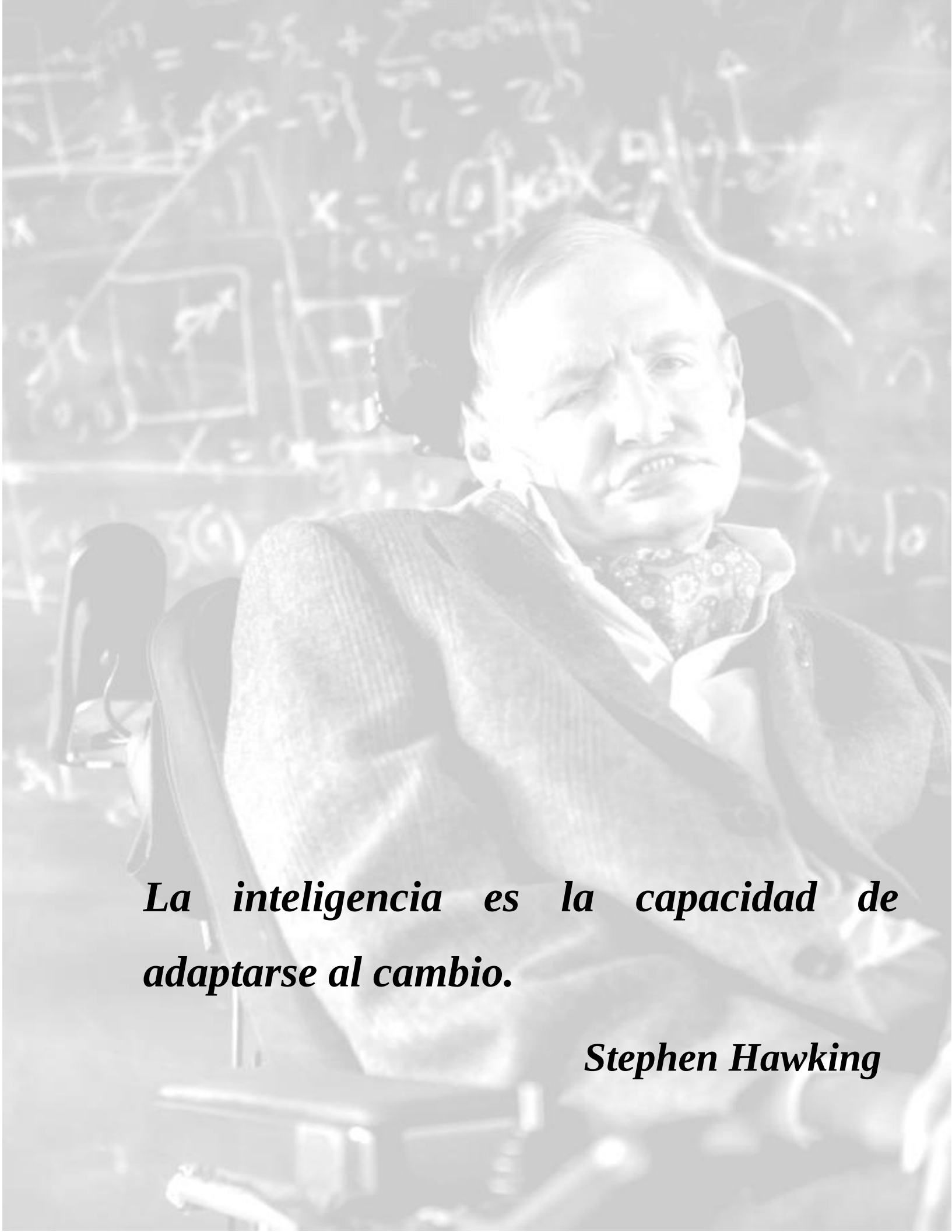
Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Leonardo Aguilera Blanco

Tutores: M. Sc. Henry Raúl González Brito

Ing. Madelín Haro Pérez

La Habana, junio de 2018



La inteligencia es la capacidad de adaptarse al cambio.

Stephen Hawking

Declaración de autoría

Declaro ser autor de la presente tesis que tiene por título: Aplicación Android de gestión de información de vulnerabilidades para la Dirección de Seguridad Informática y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de junio del año 2018.

Leonardo Aguilera Blanco	M. Sc. Henry Raúl González Brito	Ing. Madelín Haro Pérez
Firma del Autor	Firma del Tutor	Firma del Tutor

Datos de contacto

Datos de autor:

Leonardo Aguilera Blanco: (laguilera@estudiantes.uci.cu)

Datos de tutor:

M. Sc. Henry Raúl González Brito: (henryraul@uci.cu) Graduado de Ingeniería Informática por la Universidad de Camagüey y el Instituto Superior Politécnico José Antonio Echeverría en el 2005. Dirige actualmente el Laboratorio Especializado en Pruebas de Seguridad en Aplicaciones Web de la UCI. Es Máster en Gestión de Proyectos Informáticos y ha realizado un número considerable de pruebas de seguridad a las aplicaciones web de la UCI basadas en diferentes tecnologías. Es profesor principal del curso de postgrado de Pruebas de Penetración en Aplicaciones Web que se ha impartido a especialistas de diversas instituciones del país y ha contado con presencia de un estudiante extranjero. Es coordinador del Grupo de Investigación de Redes y Seguridad Informática de la UCI. Editor de un blog especializado en Seguridad en Aplicaciones Web (https://henryraul.wordpress.com). Los resultados de su equipo han sido referenciados por una entidad especializada extranjera (https://openjournalsystems.com/open-journal-systems-ojs-hacking-epidemic-solutions/). Ha impartido conferencias en eventos nacionales e internacionales en Cuba sobre la seguridad en aplicaciones web. Su equipo ha alcanzado reconocimiento institucional por la labor realizada en diversos aspectos de la seguridad en aplicaciones web.

Datos de tutor:

Ing. Madelín Haro Pérez: (mharo@uci.cu) Graduada de Ingeniería Informática en el 2002, en el Instituto Superior Politécnico José Antonio Echeverría. Laboró como Analista B en la Empresa de Servicios Informáticos de Cienfuegos. Desde marzo del 2003 trabaja en la Universidad de Ciencias Informáticas impartiendo asignaturas como Programación I y II, Metodología de la Investigación Científica, Gestión de Software y dirigiendo la Disciplina Práctica Profesional hasta el 2012 donde comienza a impartir asignaturas de esta área de conocimiento. Ha trabajado en proyectos productivos siendo Líder del primer proyecto de exportación de la universidad, con México. Trabajó en la capacitación de los proyectos con Venezuela Registros y Notarías, Identidad y luego lideró este subproyecto en Prisiones. Atendió Adiestramiento en la Dirección de Postgrado. Ha trabajado en el diseño curricular de la carrera Ingeniería en Ciencias Informáticas, específicamente en la disciplina Práctica Profesional. Actualmente trabaja como Profesor Principal en la Facultad 2 recibiendo reconocimientos por su labor en esta área y como mejor Colectivo de año. Ha recibido cursos y publicado en temas pedagógicos y técnicos.

Agradecimientos

Quiero agradecer en primer lugar a mis padres porque con su ejemplo y educación supieron conducirme por el camino correcto.

A mi novia Lily, por asumir una gran carga por este período, por entender mi ausencia y por alentarme en los momentos necesarios.

A toda mi familia en general, por la preocupación que mostraron en esta etapa.

A mis tutores Ing. Madelín Haro Pérez y M. Sc. Henry González Brito que, con sus ideas, sabios consejos y críticas constructivas contribuyeron en gran medida a la realización de este trabajo y a mi formación científica.

A mis compañeros y amigos que siempre estuvieron pendientes de los avances de la tesis, que de una forma u otra han contribuido con mi formación personal y profesional.

A los especialistas de la Dirección de Seguridad Informática y miembros del Grupo de Hacking Ético que me impulsaron en el mundo de la seguridad informática.

A la Universidad de las Ciencias Informáticas en especial a todos los profesores de la Facultad 2 y la antigua Facultad 7 que han aportado extraordinariamente en mis conocimientos.

A todos sinceramente, muchas gracias.

Resumen

Las vulnerabilidades informáticas aumentan cada año y, como consecuencia, la búsqueda de estas consume cada vez más tiempo y esfuerzo por parte de los especialistas en seguridad informática. La Dirección de Seguridad Informática de la Universidad de las Ciencias Informáticas (UCI), consciente de este problema, ha desarrollado un Sistema de Gestión de Vulnerabilidades (SIGEV). Dicho sistema extrae desde repositorios certificados internacionalmente la información pública de las vulnerabilidades y puede suministrarla a otros sistemas que la necesiten o utilicen; todo ello a través de un servicio web.

El objetivo de este trabajo se enmarca en el desarrollo de una aplicación nativa para dispositivos móviles Android 4.1 o superior, que gestione la información de las vulnerabilidades que provee el SIGEV. Esta aplicación garantiza la movilidad de los especialistas por las diferentes áreas donde auditan en la universidad y tener al alcance las vulnerabilidades de las diversas tecnologías desplegadas en la institución de manera organizada y actualizada y sin dependencia de conexión a la red. Un resultado del trabajo es la notificación a los especialistas de las nuevas vulnerabilidades y su actualización automática de forma que se garantice la información completa y confiable en todo momento.

El desarrollo de la aplicación se realizó utilizando la metodología establecida para el proceso de desarrollo de software en la UCI. Se trabaja en Android Studio utilizando Java como lenguaje de programación. Para la base de datos se utiliza la filosofía ORM y el gestor SQLite.

Palabras Clave: Dispositivos móviles Android, seguridad informática, servicios web, vulnerabilidad

Abstract

Computer vulnerabilities increase every year and, as a consequence, the search for these consumes more and more time and effort on the part of computer security specialists. The Computer Security Directorate of the University of Computer Science (UCI, spanish acronym), aware of this problem, has developed a Vulnerability Management System (SIGEV, spanish acronym). This system extracts from publicly certified repositories the public information of the vulnerabilities and can supply it to other systems that need or use it; all through a web service.

The objective of this work is the development of a native application for mobile devices with Android 4.1 or higher, which manages the information of the vulnerabilities provided by the SIGEV. This application guarantees the mobility of specialists in the different areas where they audit at the university and have access to the vulnerabilities of the various technologies deployed in the institution in an organized and updated manner and without dependence on connection to the network. One result of the work is the notification to the specialists of the new vulnerabilities and their automatic updating in order to guarantee complete and reliable information at all times.

The development of the application was made using the methodology established for the software development process in the UCI. You work in Android Studio using Java as a programming language. The ORM philosophy and the SQLite manager are used for the database.

Key Words: Android mobile devices, informatics security, vulnerability, web services.

Índice

Índice de figuras

Índice de tablas

Introducción

En la actualidad las organizaciones están llamadas a cumplir sus objetivos alineando sus negocios al uso de las tecnologías informáticas. Significa que todos sus procesos, sean estratégicos, claves o de soporte, deben realizarse utilizando aplicaciones que a su vez, manejan todo el arsenal de datos, información y conocimientos del lugar. Con el uso de las redes y las necesidades de interoperabilidad entre los sistemas, las personas tienen acceso a dicha información, desde cualquier parte del mundo posibilitando el trabajo en línea, la independencia geográfica, entre otros valores. Sin embargo, también tiene una desventaja importante: la seguridad.

Se entiende por amenaza a todo elemento o acción capaz de atentar contra la seguridad de la información. Las amenazas surgen a partir de la existencia de vulnerabilidades. Una vulnerabilidad es un fallo o debilidad en el diseño, la implementación, el funcionamiento o la gestión de un sistema, que puede ser explotado con la finalidad de violar la política de seguridad del sistema. Son objetivos de ataques que buscan comprometer la disponibilidad, confidencialidad e integridad de la información provocando consecuencias no deseadas en las organizaciones (Castillo, 2011).

En muchas ocasiones las vulnerabilidades se encuentran disponibles en sitios web encargados de recopilar y mantener actualizada la información de las vulnerabilidades para el uso público. Sin embargo, las organizaciones no tienen conocimiento de ellas debido a que los especialistas que se dedican a la realización de pruebas de intrusión, necesitan gestionar esta información pero el proceso de búsqueda de vulnerabilidades consume tiempo. Esto ocurre debido a que existen aproximadamente 100000 vulnerabilidades conocidas y anualmente se descubren más de 5000. Según los datos mostrados en la ilustración 1, proporcionados por la Lista de CVE (Common Vulnerabilities and Exposures), la cifra anual permanece en aumento y solamente en el primer mes del 2018 fueron descubiertas más de 2500 vulnerabilidades.

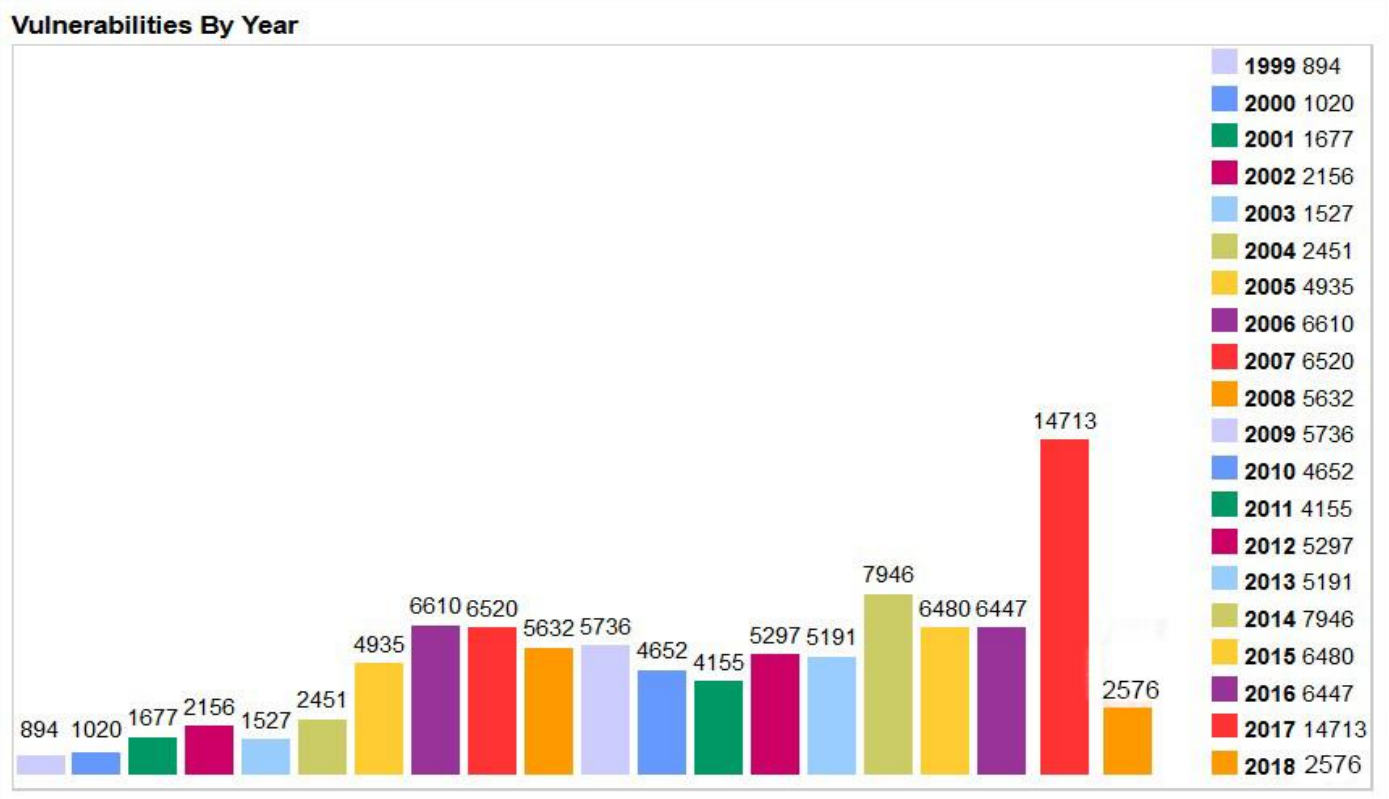


Fig. 1 Cantidad de vulnerabilidades descubiertas por año Fuente: www.cvedetails.com.

Cada instante que no se actualiza o corrige una aplicación vulnerable, la organización está expuesta a ataques que podrían provocar pérdida de datos, robo de información, hurto de identidad y otros problemas que constituyen un peligro serio de seguridad, por lo que es necesario que los tiempos de respuesta ante nuevas vulnerabilidades sea el menos posible.

Según (Suárez, 2015), Cuba ha iniciado un programa de desarrollo de informatización de la sociedad para lograr la soberanía tecnología y, para cumplir ese objetivo, es necesario un incremento considerable del número de aplicaciones en el país para satisfacer las necesidades tecnológicas existentes. Sin embargo, al producirse un aumento de aplicaciones, también se produce un aumento de las amenazas de seguridad, incrementando la posibilidad de que ocurra un ataque que afecte a la economía o la seguridad nacional, por solo citar algunas posibles afectaciones.

El gobierno cubano, consciente de esta debilidad, ha establecido en el Taller Nacional de Informatización y Ciberseguridad 2015, los Lineamientos aprobados por el Partido Comunista de Cuba en el 2016 y ha ratificado por otros medios como la Mesa Redonda (emisión del 25 de noviembre del 2017), entre otros espacios, que este programa de informatización debe estar sustentado sobre un sistema de seguridad de la información que proteja la soberanía tecnológica y asegure el enfrentamiento al uso ilegal de las Tecnologías de la Información y las Comunicaciones (TIC).

El Viceministro de Telecomunicaciones de Cuba, Wilfredo González Vidal, anunció en el 2015 la constitución del Centro de Seguridad del Ciberespacio debido a que entre el 2014 y el 2015 se han notificado agresiones desde más de 170 países a computadoras en Cuba. Dicho Centro busca fortalecer la protección de la infraestructura tecnológica del país y de sus aplicaciones al prevenir y enfrentar riesgos generados por el uso inadecuado de las TIC.

La Universidad de las Ciencias Informáticas (UCI), que tiene como misión producir aplicaciones y servicios informáticos, sirviendo de soporte a la industria cubana del software; está desarrollando un proyecto de investigación titulado “Sistema nacional para el intercambio y gestión de información de amenazas, ataques y vulnerabilidades de seguridad informática” aprobado en el Programa de Prioridad Nacional de Ciencia, Tecnología e Innovación denominado “Informatización de la Sociedad”. Este proyecto busca monitorizar y gestionar los incidentes en los sistemas informáticos del país, para fomentar la cultura basada en la seguridad, minimizar el desconocimiento de lo que acontece en relación a esta materia en otras partes del país y proteger la red de aplicaciones.

La Dirección de Seguridad Informática de la UCI (DSI), como parte del proyecto de investigación antes mencionado y siendo parte activa en hallar soluciones ante las amenazas, vulnerabilidades y ataques, ha desarrollado el Sistema de Gestión de Vulnerabilidades (SIGEV). Su misión es extraer, desde repositorios como el Repositorio Gubernamental de los Estados Unidos de América, el sitio oficial de la lista de CVE y otros reconocidos internacionalmente, la información pública de las vulnerabilidades existentes en tecnologías como Drupal, Wordpress, Jquery y otras. SIGEV está preparado para suministrar esta información a otros sistemas que necesiten esta información, a través de un servicio web.

Un especialista audita normalmente más de una tecnología en la realización de su trabajo. Cada tecnología puede tener un número grande de vulnerabilidades. En este momento, no hay posibilidad de actualizar al trabajador sobre la aparición de nuevas vulnerabilidades y tampoco de que, al obtenerlas, sean organizadas de acuerdo a la tecnología que la genera. La situación trae consigo que a la demora de encontrar las vulnerabilidades se le incremente otro retardo al tener que organizarlas para su uso.

Adicionalmente, la gestión de esta información es engorrosa debido a que solo se encuentra disponible a través de un fichero JSON (JavaScript Object Notation), que dificulta su lectura.

Los especialistas de seguridad informática en la UCI tienen amplia movilidad entre la infraestructura tecnológica y las tecnologías que auditan por lo que necesitan tener al alcance esta información en un dispositivo móvil. Entiéndase esto como un aparato de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, que ha sido diseñado específicamente para una función, pero que puede llevar a cabo otras funciones más generales (Alonso, 2011).

Según (TicWeb, 2017), los dispositivos móviles inteligentes están cambiando la forma en la que la sociedad interactúa entre sí, estos además nos proveen con nuevas aplicaciones que solucionan operaciones que anteriormente exigían una mayor inversión de tiempo.

La Dirección de Seguridad Informática ha seleccionado el sistema operativo Android para el uso de dispositivos móviles para cumplir con la política de informatización de la sociedad de optar por el software libre. Además, los dispositivos móviles con sistema operativo Android son los predominantes en Cuba y en el mundo. De ahí que se identifica como **problema de investigación**: ¿Cómo gestionar la información de vulnerabilidades provenientes del SIGEV, utilizando dispositivos con sistema operativo Android?

Se tiene como **objeto de estudio**: Gestión de vulnerabilidades. Para dar solución al problema se plantea como **objetivo general**: Desarrollar una aplicación Android que permita la gestión de información de vulnerabilidades mediante el servicio web del Sistema de Gestión de Vulnerabilidades de la Dirección de Seguridad Informática; teniendo como **campo de acción**: Gestión de información de vulnerabilidades en dispositivos con sistema operativo Android, determinando como **objetivos específicos**:

- Elaborar la fundamentación teórica de la investigación para la comprensión de la investigación, selección de la metodología e identificación de herramientas y lenguajes necesarios.
- Diseñar la aplicación Android para la identificación de características deseables a implementar.
- Implementar la aplicación Android que permita la gestión de información de vulnerabilidades.
- Validar la aplicación implementada utilizando métodos experimentales y pruebas.

Para dar cumplimiento a los objetivos específicos planteados se proponen las siguientes **tareas de investigación**:

- Elaboración del marco teórico sobre la gestión de vulnerabilidades que fundamente la investigación.
- Elaboración del estado del arte de los diferentes sistemas existentes de gestión de información de vulnerabilidades que permita valorar las características de la aplicación.
- Selección de las tecnologías y herramientas necesarias para el desarrollo de la aplicación Android.
- Identificación de las características que debe tener la aplicación Android.
- Diseño de las características que debe tener la aplicación Android utilizando patrones y estándares recomendados.
- Implementación de la aplicación Android utilizando el diseño realizado y las buenas prácticas de la implementación de desarrollo de aplicaciones Android.
- Aplicación de los métodos de pruebas existentes para la validación de la aplicación Android.

Para dar cumplimiento a las tareas se realizarán los siguientes **métodos científicos**:

- Analítico-Sintético: se utiliza para analizar las teorías, documentos, aplicaciones y conjunto de librerías utilizadas en la implementación de la aplicación para la gestión de la base de datos y la sincronización de esta con el servicio web.
- Modelación: se emplea para crear modelos de datos, arquitectura y otros tipos de diagramas relacionados con la aplicación web y la aplicación a informatizar.

Como **métodos empíricos** se utilizarán:

- Experimentación: se emplea al comprobar, recreando las condiciones de uso de la aplicación a través de pruebas, su funcionamiento y valor para cumplir los requisitos.

La investigación tiene como **aporte práctico** una aplicación de extensión apk capaz de consumir un servicio web para obtener la información de las vulnerabilidades dada una tecnología que el usuario necesite conocer para su gestión. La aplicación almacenará esta información en una base de datos para que pueda ser consultada sin necesidad de una conexión a la red. Además, enviará notificaciones al recibirse nuevas vulnerabilidades, manteniendo a los usuarios actualizados, proporcionándoles una mayor portabilidad y síntesis de la información que necesitan.

El desarrollo está estructurado en tres capítulos que contienen la información siguiente:

Capítulo 1: Fundamentación teórica de la aplicación de gestión de información de vulnerabilidades: Se establecen los principales aspectos teóricos para el objeto de estudio de la investigación. Se realiza el estudio de sistemas de gestión de vulnerabilidades. Se seleccionan herramientas y tecnologías que se emplean en el desarrollo de la aplicación Android, y se caracteriza la metodología para seleccionar aquella que se adecue mejor a las características del producto.

Capítulo 2: Características de la aplicación de gestión de información de vulnerabilidades: Se realiza la modelación del negocio y la identificación y descripción de los requisitos de la aplicación. Se representa la información correspondiente a la etapa del ciclo de vida del software: análisis.

Capítulo 3: Diseño, implementación y validación de la aplicación de gestión de información de vulnerabilidades: Se describe la arquitectura de software y los patrones de diseño que se emplea. Se modela la información relacionada con las etapas de diseño e implementación. Se explica la validación del sistema y los resultados obtenidos.

Capítulo 1. Fundamentación científica de la gestión de información de vulnerabilidades

En el mundo existen sistemas que permiten conocer las vulnerabilidades a través de la web, el estudio de estas aplicaciones y los estándares que utilizan para almacenar esta información permite conocer cómo gestionarla en un dispositivo móvil. Además, se necesita evaluar las herramientas y tecnologías necesarias para el consumo de un servicio web. Un estudio de las metodologías de desarrollo de software permite escoger la apropiada para la obtención de un producto con calidad.

1.1 Gestión de vulnerabilidades

La gestión de vulnerabilidades es un proceso continuo de las tecnologías de la información consistente en la identificación, evaluación y corrección de vulnerabilidades en los sistemas de información y las aplicaciones de una organización (Akamai, 2018).

El proceso de gestión de vulnerabilidades consiste en las siguientes acciones (Akamai, 2018):

- Obtención de un inventario de los activos de las tecnologías de la información de una empresa, lo que incluye servidores, infraestructura de redes, estaciones de trabajo, impresoras y aplicaciones.
- Detección de las vulnerabilidades existentes mediante escáneres de redes, escáneres de vulnerabilidades y software de pruebas de penetración automáticas y determinación de los niveles de riesgo.
- Reparación de sistemas y dispositivos vulnerables y presentación de informes sobre las medidas correctivas adoptadas.

Una característica distintiva en este proceso de acuerdo a (López, 2018) es que la detección de vulnerabilidades mediante los escaneos automatizados arrojan informaciones que no son todas correctas o completas. En la literatura estos resultados se conocen como falsos negativos y falsos positivos. Ambos implican la realización de búsquedas manuales de vulnerabilidades luego de haber utilizado la herramienta automática.

Para la comprobación manual de estas vulnerabilidades es necesario conocer y gestionar la información de las vulnerabilidades que poseen estas tecnologías de la información y determinar su nivel de criticidad.

1.1.1 Estándares utilizados en la gestión de vulnerabilidades

Common Vulnerabilities and Exposures

Common Vulnerabilities and Exposures (CVE) es una lista de vulnerabilidades de seguridad de la información públicamente conocidas. Es el estándar más utilizado, permite identificar cada vulnerabilidad, asignándole un código de identificación único. Se conoce como identificador CVE (CVE-ID) y está formado por las siglas de este diccionario seguidas por el año en que es registrada la vulnerabilidad o exposición y un número arbitrario de cuatro dígitos. Estos tres elementos van separados por un guion resultando un identificador con el siguiente formato: CVE -Año de registro-Número de cuatro cifras asignado a la vulnerabilidad y se permite añadirle más dígitos según fuera necesario (CVE, 2017).

Sin embargo, saber el número de vulnerabilidades no es suficiente para un análisis fiable, no todas ellas tienen el mismo impacto. Por este motivo surge CVSS (*Common Vulnerabilities Scoring System*), creado en 1999 por FIRST, una confederación internacional de equipos de respuesta a incidentes informáticos que gestionan problemas de seguridad y promueven programas de prevención, este estándar permite priorizar vulnerabilidades asignando diferentes puntuaciones (Mell, 2006).

CVSS v3.0

CVSS es un sistema de valoración de vulnerabilidades donde se redactan sus especificaciones en (FIRST, 2017). Creado como un método estándar para determinar y clasificar la criticidad de las vulnerabilidades. Este estándar pertenece a FIRST, una confederación internacional de equipos de respuesta a incidentes informáticos que gestionan problemas de seguridad y promueven programas de prevención.

Las vulnerabilidades representan un riesgo crítico para cualquier organización que opere una red informática, y pueden ser difíciles de categorizar y mitigar. CVSS proporciona una forma de capturar las características principales de una vulnerabilidad y producir una puntuación numérica que refleje su gravedad, así como una representación textual de esa puntuación. El puntaje numérico puede traducirse en una representación cualitativa, como baja, media, alta y crítica, para ayudar a las organizaciones a evaluar y priorizar adecuadamente sus procesos de gestión de vulnerabilidad.

CVSS brinda tres beneficios importantes. En primer lugar, proporciona puntajes de vulnerabilidad estandarizados. Cuando una organización utiliza un algoritmo común para calificar vulnerabilidades en todas las tecnologías, puede aprovechar una única política de administración de vulnerabilidades que define el tiempo máximo permitido para validar y remediar una vulnerabilidad determinada. A continuación, proporciona un marco abierto. Cuando se computa el puntaje ambiental, la vulnerabilidad se vuelve contextual para cada organización y ayuda a comprender mejor el riesgo que representa esta vulnerabilidad.

A continuación se describen las métricas utilizadas por este estándar para establecer las puntuaciones a las vulnerabilidades registradas.

1.1.2 Métricas empleadas por el estándar CVSS v3.0

El estándar CVSS se encarga de estudiar las diversas propiedades de una vulnerabilidad y las agrupa en tres métricas diferentes, métricas base, métricas temporales y métricas de entorno, las métricas base representan las características intrínsecas a la vulnerabilidad, que son constantes en el tiempo y en el entorno del usuario, las métricas temporales representan las características de una vulnerabilidad que pueden cambiar en el tiempo, pero que son constantes en el ambiente de un usuario y por último las métricas de entorno que representan las características de una vulnerabilidad que son relevantes y únicas para el entorno de un usuario en particular.

Tanto las métricas temporales como las de entorno, son métricas que pueden cambiar por lo que son opcionales y no tienen ningún efecto en la evaluación, se tienen en cuenta debido a que representan las condiciones en las que surgen las vulnerabilidades.

Sin embargo, las métricas bases representan las características intrínsecas a la vulnerabilidad, son constantes en el tiempo y en el entorno. Esta incluye métricas de vector de acceso, complejidad de acceso y autenticación, de manera que permiten definir cómo se puede acceder a una vulnerabilidad y si se cumplen las condiciones para ser explotada. La severidad de las tres métricas mide la manera en la que una vulnerabilidad, si se explota, afecta de forma directa a los activos de las tecnologías de la información. Los impactos se determinan de manera independiente, como el grado de pérdida de confidencialidad, integridad y disponibilidad, ya que una vulnerabilidad podría causar pérdida parcial de integridad y disponibilidad, pero tal vez no afecte la confidencialidad.

Los valores de las métricas que se describen a continuación fueron extraídas del documento de especificación CVSS v3.0, disponible en: https://www.first.org/cvss/cvss-v30-specification-v1.8.pdf.

Vector de acceso

El vector de acceso es una métrica que refleja cómo es explotada la vulnerabilidad. Mientras más remoto sea un atacante, mayor es la puntuación de la vulnerabilidad.

Valor de la métrica:

- Local: Una vulnerabilidad explotable solo con acceso local requiere al atacante tener ya sea acceso físico al sistema vulnerable o una cuenta local.
- Red adyacente: Una vulnerabilidad explotable con un acceso de red adyacente requiere al atacante tener acceso ya sea al dominio de difusión o colisión del software vulnerable.
- Red: Una vulnerabilidad explotable con acceso a red significa que el software vulnerable está en la pila de red y el atacante no requiere acceso a la red local o acceso local.

- Físico: Requiere que el atacante físicamente toque o manipule el componente vulnerable.

Complejidad de acceso

La complejidad de acceso es una métrica que mide la complejidad del ataque requerido para explotar la vulnerabilidad una vez el atacante gana acceso al sistema objetivo.

Valor de la métrica:

- Alta: Existen condiciones especiales de acceso.
- Media: Las condiciones de acceso son un tanto especializadas.
- Baja: No existen condiciones de acceso especializado o circunstancias extenuantes.

Privilegio requerido

Privilegio requerido es una métrica que describe el nivel de privilegio que el atacante debe poseer antes de satisfactoriamente explotar la vulnerabilidad. El valor de la métrica es mayor si no se requieren privilegios.

Valor de la métrica:

- Ninguno: El atacante no requiere autenticación antes del ataque ni acceso a la configuración o archivos para realizar el ataque.
- Bajo: El atacante está autenticado con privilegios con capacidades básicas de usuario.
- Alto: El atacante está autorizado con privilegios administrativos y posee el control sobre el componente vulnerable.

Interacción de usuario

La interacción de usuario es una métrica que captura el requerimiento para un usuario, en lugar de un atacante, para participar en la satisfactoria explotación de una vulnerabilidad. Indica si un usuario debe participar en el ataque o un atacante no necesita de la interacción de un usuario. Esta métrica es mayor si no se requiere de la interacción de un usuario.

- Ninguno: El sistema vulnerable puede ser explotado sin la interacción del usuario.
- Requerido: Requiere de la interacción de usuario para la satisfactoria explotación de la vulnerabilidad.

Alcance

El alcance es una métrica que representa la capacidad de una vulnerabilidad para impactar en los recursos de una vulnerabilidad más allá de sus medios o privilegios.

- Incambiable: Una vulnerabilidad explotada solo puede afectar los recursos gestionados por la misma autoridad. En este caso, el componente vulnerable y el componente impactado son el mismo.

- Cambiable: Una vulnerabilidad explotada puede afectar los recursos más allá de la autorización de privilegios previstos por el componente vulnerable. En este caso, el componente vulnerable y el componente impactado son diferentes.

Impacto de integridad

El impacto de integridad es una métrica que mide el impacto a la integridad de una vulnerabilidad explotada satisfactoriamente. La integridad se refiere a la confiabilidad y garantía de veracidad de la información. El incremento en el impacto a la integridad incrementa la puntuación de la vulnerabilidad.

Valor de la métrica:

- Ninguno: No existe impacto a la integridad del sistema.
- Parcial: Es posible la modificación de algunos archivos del sistema o información, pero el atacante no tiene control sobre lo modificable, o el alcance sobre lo afectado es limitado.
- Completo: Existe un compromiso total de la integridad del sistema. Existe una completa pérdida de protección del sistema, resultando en el compromiso total del sistema. El atacante es capaz de modificar cualquier archivo sobre el sistema objetivo.

Impacto de disponibilidad

El impacto de disponibilidad es una métrica que mide el impacto a la disponibilidad de una vulnerabilidad explotada satisfactoriamente. La disponibilidad se refiere a la accesibilidad de los recursos de información. Los ataques consumiendo ancho de banda, ciclos de procesador, o espacio en disco impactan en la disponibilidad de un sistema. El incremento en el impacto a la disponibilidad incrementa la puntuación de la vulnerabilidad.

Valor de la métrica:

- Ninguno: No existe impacto en la disponibilidad del sistema.
- Parcial: Existen interrupciones o desempeño reducido en la disponibilidad del recurso.
- Completo: Existe un reinicio total del recurso afectado. El atacante puede hacer el recurso completamente no disponible.

Luego de ser calculadas las métricas se genera una puntuación final o *CVSS Score.* Las puntuaciones varían de 0 a 10, siendo esta última la más severa. Según la versión CVSS v3.0 el impacto de una vulnerabilidad se considera nulo o informativo si es de cero, bajo si la puntuación se encuentra entre 0.1 y 3.9, medio entre 4 y 6.9, alto será si varía de 7 a 8.9 y crítico de 9 a 10 (García, 2016).

Los estándares CVE y CVSS funcionando en conjunto permiten conocer la fiabilidad de un sistema y la seguridad de la información que ellos contienen. Estos son estándares ampliamente utilizados y constantemente se publican actualizaciones de vulnerabilidades.

1.2 Sistemas de gestión de información de vulnerabilidades

En la actualidad existen empresas, proyectos, institutos y organizaciones gubernamentales que realizan búsquedas específicas de vulnerabilidades de software y hardware desde hace muchos años. La información sobre las vulnerabilidades es almacenada en repositorios y verificada para luego, mediante sistemas, ser mostrada a los usuarios.

En el estado del arte de esta investigación se analizan varios sistemas de información. Los criterios que se tuvieron en cuenta para el análisis son:

- Método de proveer los datos.
- Estándares que utilizan para almacenar la información.
- Alcance de portabilidad de la información.
- Dependencia de la red para la obtención de la información.
- Envío de notificaciones sobre descubrimientos de vulnerabilidades.
- Visibilidad de la información en Android.

Los sistemas de información para el análisis son los siguientes:

National Vulnerabilities Database

National Vulnerabilities Database (NVD, según sus siglas en inglés) es el repositorio gubernamental de los Estados Unidos perteneciente al Instituto Nacional de Estándares y Tecnologías (NIST) de gestión de vulnerabilidades basados en estándares. La información que proporciona este repositorio permite la automatización de la gestión de vulnerabilidades, la medición y el cumplimiento de seguridad. El NVD incluye bases de datos de listas de verificación de seguridad, fallas de software relacionadas con la seguridad, configuraciones incorrectas, nombres de productos y métricas de impacto. Esta base de datos de vulnerabilidades está desarrollada y completamente sincronizada con la lista CVE (NVD, 2018).

Actualmente el repositorio provee un enlace de descarga de prueba de ficheros en formato JSON y XML para obtener la información de las vulnerabilidades a través de su sitio oficial disponible en nvd.nist.gov. Este sistema provee, mediante un servicio RSS, la información de las vulnerabilidades en un rango de 8 días en formato XML.

CVE – Corporación MITRE

Es el sitio web oficial de la lista CVE, es una lista de entradas, cada una con un número de identificación, una descripción y al menos una referencia pública, para las vulnerabilidades de ciberseguridad públicamente conocidas. Se encuentra disponible a través de su sitio web cve.mitre.org.

Aunque están separados, tanto la lista CVE como NVD están patrocinados por US-CERT en la oficina de Seguridad Cibernética y Comunicaciones en el Departamento de Seguridad Nacional de Estados Unidos. Ambos están disponibles para el público y son de uso gratuito (CVE, 2017).

SIGEV

Este sistema es desarrollado por la DSI de la UCI. Almacena en su base de datos centralizada toda la información recopilada de los repositorios de vulnerabilidades anteriormente mencionados y es actualizada periódicamente. Provee un servicio web capaz de brindar esta información a otros sistemas y de acceder al listado de todas las vulnerabilidades, acceder a ellas de manera individual o realizar una búsqueda por la descripción de la vulnerabilidad.

Este servicio web utiliza la arquitectura REST, por tanto, se considera un servicio web RESTful. Según (Chanchi, 2011) la principal ventaja de REST es que utiliza bajo consumo de recursos y posee una alta velocidad para la consulta de los datos. Además, el servicio provisto por SIGEV utiliza como lenguaje de intercambio de datos a JSON que, según (Chanchi, 2011) es el menos pesado de los lenguajes de intercambios existentes. Por tanto, se agiliza la consulta de los datos por parte de los dispositivos móviles inteligentes que utilicen este servicio web.

Análisis sobre los sistemas descritos

Todos estos sistemas brindan la información de las vulnerabilidades utilizando los estándares CVE y CVSS y, aunque posean una amplia base de datos, dependen de la conectividad a la red para el acceso a la información. La portabilidad de estos datos está limitada a un ordenador debido a que estos sitios no utilizan un diseño web adaptativo por lo que el acceso a la información desde un dispositivo móvil inteligente es trabajoso, debido a que estos sitios no se adaptan correctamente a la dimensión de la pantalla del dispositivo móvil inteligente. NVD utiliza un servicio RSS para la notificación de la información de las vulnerabilidades mientras que los demás sistemas carecen de un método de notificación de registros de vulnerabilidades.

La forma de acceder a la información varía de uno a otro; de esta forma NVD y la lista CVE establecen un enlace de descarga para la obtención de las vulnerabilidades mientras que SIGEV la tramita a través de su servicio web resultando en una consulta de datos más agilizada y un uso más óptimo de recursos para los dispositivos móviles.

Se llega a la conclusión de que no existen soluciones que sean ejecutables en Android, que permitan la obtención de la lista de vulnerabilidades sobre una tecnología, el almacenamiento de esta información en un dispositivo móvil y notifiquen sobre el descubrimiento de nuevas vulnerabilidades.

1.3 Dispositivos móviles inteligentes

Los dispositivos móviles inteligentes utilizan los servicios web para obtener y enviar datos para el funcionamiento de aplicaciones, como Facebook, Twitter y GoogleMaps, por citar algunas, obteniendo las mismas funcionalidades que desde sus sitios web con una mayor portabilidad.

Existen varios tipos de dispositivos móviles inteligentes entre los que destacan teléfono inteligente o *smartphone* y la tableta o *Tablet*. El teléfono celular con pantalla táctil permite al usuario conectarse a internet, gestionar cuentas de correo electrónico e instalar otras aplicaciones y recursos a modo de pequeño computador. La tableta es un dispositivo electrónico que tiene un tamaño intermedio entre el ordenador y el móvil, ligero, con manejo intuitivo utilizando las manos, elevada autonomía de uso e independiente de otros accesorios complementarios (Alonso, 2011).

Estos dispositivos inteligentes son potenciados por los sistemas operativos que poseen que, en correspondencia con el fabricante que desarrolle el dispositivo, se les incorpora. Ejemplo: en el caso de la marca iPhone de la compañía Apple Inc. se les incorpora el sistema operativo iOS creado por la misma empresa.

Los sistemas operativos móviles más abundantes son Symbian OS, BlackBerry OS, Firefox OS, Windows Phone, iOS y Android, siendo este último, el sistema operativo predominante del planeta (ver Fig. 2).

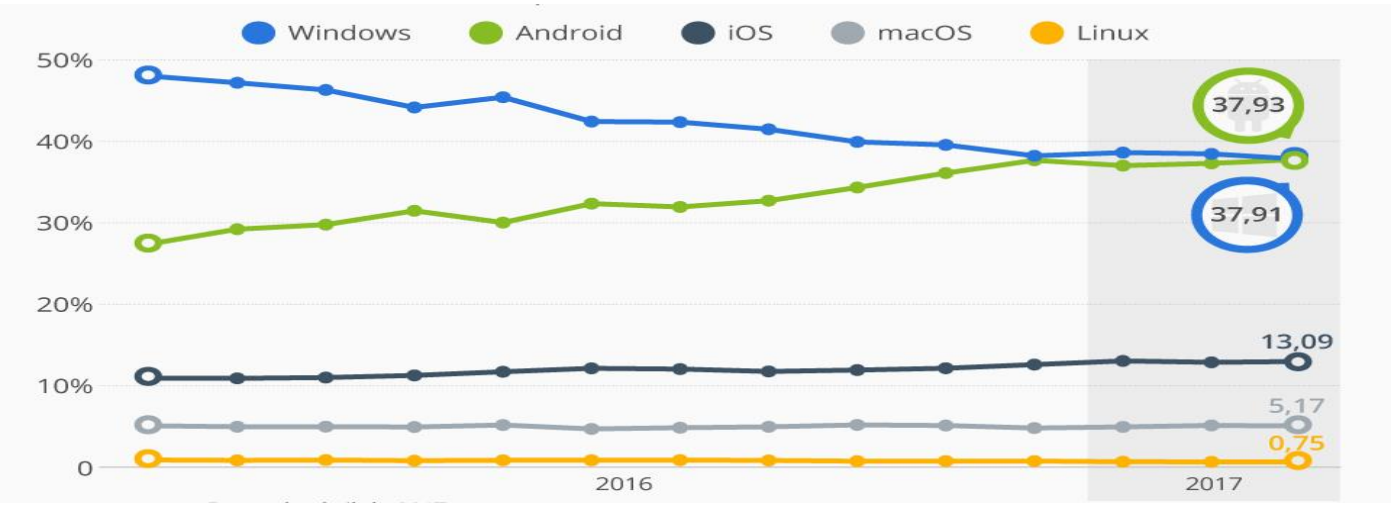


Fig. 2 Distribución de uso de sistemas operativos en el mundo en %. Fuente: StatCounter. Abril 2017

Por otra parte, Google es la principal compañía desarrolladora de aplicaciones, patentes, sistemas operativos y actualmente es patrocinado por el consorcio de empresas Open Handset Alliance (OHA).

Sistema operativo Android

Es orientado a dispositivos móviles, por lo general con pantalla táctil. De este modo, es posible encontrar tabletas, teléfonos inteligentes y relojes equipados con Android, aunque el software también se utiliza en automóviles, televisores y otras máquinas.

Fue creado por Android Inc., una compañía adquirida por Google en el 2005. El objetivo inicial de Android fue promover los estándares abiertos en teléfonos y ordenadores móviles (Android, 2017).

Consta de una plataforma para dispositivos móviles que contiene una pila de software donde se incluye el sistema operativo, el software que permite la conexión entre redes y las aplicaciones básicas para el usuario. Las capas de la arquitectura de Android son las que aparecen en la siguiente figura:

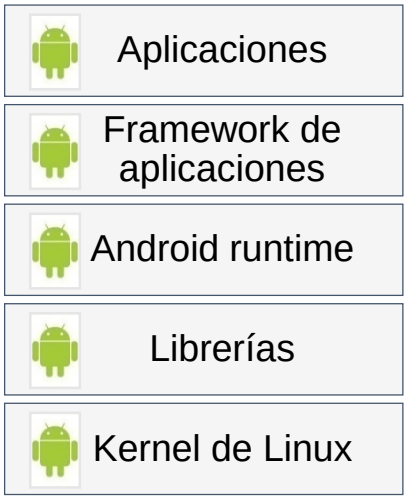


Fig. 3 Capas de la arquitectura del sistema operativo Android

Cada una de las capas de la arquitectura utiliza servicios ofrecidos por las anteriores y ofrece, a su vez, los suyos propios a las capas de niveles superiores (Brahler, 2010).

El logo es de color verde y es el dibujo de un robot con forma redondeada, que fue diseñado con la fuente Droid y que la tipografía se llama Norad (Android, 2017).

Desarrollo de aplicaciones Android

APK es la abreviatura para aplicación empaquetada de Android y puede ser entendida como un software instalado en una terminal Android. (…) El archivo APK está, de hecho, en formato zip con un nombre de extensión modificado para apk, y descomprimido por UnZip para obtener un archivo Dex. Dex representa la ejecución de la máquina virtual de Dalvik para luego ejecutarla directamente (Wang, 2017).

Android hace uso de aplicaciones propias del sistema operativo, añadidas por el fabricante del dispositivo o creadas por un desarrollador para ser instaladas por el usuario para su uso. Todas estas aplicaciones se

pueden tipificar de acuerdo a su lenguaje de desarrollo en: nativas, webApp o híbridas (Charland, 2011). Las características de cada una de ellas son:

Las **aplicaciones web para móviles o webApp** son diseñadas para ser ejecutadas en el navegador del dispositivo móvil. Estas aplicaciones son desarrolladas utilizando HTML, CSS y JavaScript. Una de las ventajas de este enfoque es que los dispositivos no necesitan la instalación de ningún componente en particular, ni la aprobación de algún fabricante para que las aplicaciones sean publicadas y utilizadas. Se requiere solo acceso a internet. Además, las actualizaciones de la aplicación son visualizadas directamente en el dispositivo ya que los cambios son aplicados sobre el servidor y están disponibles de inmediato. La principal ventaja de este tipo de aplicación es su independencia de la plataforma. No necesita adecuarse a ningún entorno operativo. Solo es necesario un navegador. Por contrapartida, esto disminuye la velocidad de ejecución y requieren de mayor almacenamiento en memoria por el uso del navegador. Además, podrían tener bajo rendimiento por problemas de conectividad y una dependencia del servidor. Finalmente este tipo de aplicaciones no pueden utilizar todos los elementos de hardware del dispositivo, como por ejemplo, cámara, GPS, entre otros.

Las aplicaciones **nativas** son aquellas que se conciben para ejecutarse en una plataforma específica, es decir, se debe considerar el tipo de dispositivo, el sistema operativo a utilizar y su versión. Tiene como desventaja que se debe desarrollar una aplicación para cada plataforma. La principal ventaja de este tipo de aplicaciones es la posibilidad de interactuar con todas las capacidades del dispositivo, como cámara, GPS, acelerómetro, agenda, entre otros. Además no es estrictamente necesario poseer acceso a internet. Su ejecución es rápida, puede ejecutarse en segundo plano y permite notificar al usuario cuando ocurra un evento que necesite su atención.

Una aplicación **híbrida** es una combinación de las dos anteriores. Se desarrollan con lenguajes propios de las webApp: HTML, Javascript y CSS; por lo que permite su uso en diferentes plataformas. También dan la posibilidad de acceder a parte de las características del hardware del dispositivo, aunque no brindan un acceso completo por lo que sus funciones son limitadas. Generalmente, requieren de una conexión a internet para su funcionamiento y su rendimiento es menor comparado con las aplicaciones nativas.

Debido a la rapidez de respuesta, el control y uso de los recursos de hardware del dispositivo móvil y la no dependencia de la conexión a la red, se decide que la aplicación que se desarrolla en este trabajo sea nativa. Esta decisión está avalada por la necesidad de almacenar los datos de las vulnerabilidades en el dispositivo móvil inteligente, visualizarlas para su consulta utilizando los recursos del dispositivo móvil inteligente y establecer la conexión con el servicio web del SIGEV para la transferencia de información.

1.4 Herramientas y tecnologías empleadas

Las herramientas y tecnologías que se describen en este epígrafe son las establecidas por Google para el desarrollo de aplicaciones nativas en el sistema operativo Android.

Lenguaje de programación Java 7

En el desarrollo de aplicaciones en Android se utilizan comúnmente varios lenguajes de programación, sin embargo, se establece como lenguaje oficial Java para el desarrollo de aplicaciones nativas en Android (Android Open Source Project 2017), por lo que fue seleccionado como lenguaje de desarrollo para la realización de la propuesta solución.

Java es un lenguaje de programación creado por Sun Microsystems para poder funcionar en distintos tipos de procesadores. Java es relativamente simple, orientado a objeto, seguro y lenguaje portable, es. El código Java, una vez compilado, puede llevarse sin modificación alguna sobre cualquier máquina, y ejecutarlo. Esto se debe a que el código se ejecuta sobre una máquina hipotética o virtual, la *Java Virtual Machine* (JVM), que se encarga de interpretar el código (ficheros compilados .class) y convertirlo a código particular de la CPU que se esté utilizando (siempre que se soporte dicha máquina virtual) (Meenakshi, 2015).

Java Development Kit (JDK) es el kit de desarrollo oficial del lenguaje de programación Java. Además de la máquina virtual de Java, indispensable para ejecutar las clases de los programas, cuenta con diversas herramientas, como javac, el compilador de *bytecode* de Java, javap, el desensamblador de clases, y jdb, el depurador de *bugs*. Estas herramientas se encuentran en el subdirectorio bin del JDK. Además de estas herramientas, el JDK contiene ejemplos y demostraciones. A continuación se presentan las nuevas características de esta versión:

- Cambios en el lenguaje, para incrementar la productividad del desarrollador y simplificar las tareas comunes de programación disminuyendo la cantidad de código necesario, aclarando la sintaxis y haciendo que el código pueda leerse más fácilmente.
- Soporte mejorado, para lenguajes dinámicos (entre ellos: Ruby, Python y JavaScript), lo que da como resultado un aumento considerable del desempeño en JVM.
- Una completa interfaz de entrada y salida, para trabajar con sistemas de archivo que pueden acceder a una variedad más amplia de atributos de archivo y ofrecer más información cuando ocurren errores.
- Nuevas características de redes y seguridad.
- Mayor soporte de la internacionalización, incluido soporte para Unicode 6.0.
- Versiones actualizadas de numerosas bibliotecas (Friesen, 2012).

Bibliotecas de desarrollo

Una biblioteca de desarrollo le ofrece al desarrollador una funcionalidad bien definida, son más sencillas de implementar en un proyecto y es más organizado trabajar debido a que cada una posee su propia funcionalidad. Para el desarrollo de aplicaciones en Android se han seleccionado algunas de estas bibliotecas para realizar la propuesta de solución.

GSON

Es una biblioteca de Java que se puede utilizar para convertir objetos de Java en su representación de JSON. También se puede usar para convertir una cadena JSON a un objeto Java equivalente. Gson puede trabajar con objetos arbitrarios de Java, incluidos objetos preexistentes de los que no tiene código fuente (Google, 2008).

Volley

Es una biblioteca HTTP que hace que las consultas en redes para las aplicaciones de Android sean más fáciles y, lo que es más importante, más rápidas. Volley está disponible en GitHub.

Volley ofrece beneficios como programación automática de solicitudes de red, múltiples conexiones de red simultáneas, almacenamiento en memoria caché, soporte para la priorización de solicitudes, solicitud de cancelación de petición, facilidad de personalización, una ordenación sólida que facilita el llenado correcto de la interfaz con datos obtenidos de forma asíncrona de la red.

Se integra fácilmente con cualquier protocolo y posee soporte para cadenas sin procesar, imágenes y JSON (Android Developer, 2017).

MaterialSearchView

Es una biblioteca que permite implementar una barra de búsqueda con diseño material directamente en la *Action Bar*, un menú auxiliar de las aplicaciones Android, que se ubica en la parte superior de cada actividad, es un elemento que por lo general es persistente. Le permite al usuario realizar búsquedas desde la actividad de inicio o desde cualquier otra actividad (Catalan, 2018).

Panter-Dialog

Esta es una biblioteca que permite mostrar diálogos, una ventana pequeña que le indica al usuario que debe tomar una decisión o ingresar información adicional, y que estos sean personalizados por el desarrollador con iconos, colores, líneas, logos, etcétera. Ayuda al usuario a esclarecer la acción que se está realizando o debe realizar (Ayar, 2016).

RefreshActionItem

Es una biblioteca que incorpora un elemento a la barra de acciones que implementa un botón de actualización que, al ser pulsado, muestra una barra de progreso para indicar que existe una actividad ocurriendo en segundo plano, le permite conocer al usuario que se está produciendo la actualización de sus datos o aplicación (Peinado, 2013).

SmartScheduler

Una biblioteca que permite realizar tareas únicas o periódicas dado un tiempo, exista la aplicación en ejecución o no. Permite realizar actualizaciones y notificaciones programadas (Hypertrack, 2016).

CarousalNotification

Esta biblioteca permite realizar una notificación donde el usuario puede navegar dentro de la notificación. Permite crear notificaciones con una visualización agradable y textos de diferentes tamaños siendo personalizable por el desarrollador (Mamgainn, 2017).

BadgeView

Esta es una biblioteca de código abierto que permite asignarle un símbolo a un elemento de la interfaz. Esto permite dar a conocer al usuario la cantidad de elementos de los que se dispone o darle un significado a una acción (Sheng, 2016).

TinyDB

Esta biblioteca simplifica las llamadas a las preferencias compartidas, una forma de lograr la persistencia de datos en Android, permite guardar de manera privada los conjuntos clave-valor, a menudo es utilizada para almacenar las configuraciones de la aplicación (Kcochibili, 2018).

InfiniteScrollListener

Esta biblioteca le permite al usuario dada una colección de elementos, incorporar nuevos elementos según sea necesario o si existen más elementos disponibles al realizar una acción deslizable hacia arriba una vez se haya alcanzado el límite de elementos fijado. Esto le permite al usuario tener control de los elementos que quiere visualizar (Wittchen, 2016).

Mapeo de Objeto Relacional

El Mapeo de Objeto Relacional (ORM, por sus siglas en inglés). Es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las

tareas básicas de acceso a los datos para el programador. El mapeo objeto-relacional es una técnica de programación para convertir datos del sistema de tipos utilizado en un lenguaje de programación orientado a objetos al utilizado en una base de datos relacional. En la práctica esto crea una base de datos virtual orientada a objetos sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (Del Busto, 2012).

ORMLite 4.41

ORMLite proporciona algunas funcionalidades simples y livianas para objetos persistentes de Java en bases de datos SQL, al tiempo que se evita la complejidad y la sobrecarga de paquetes de ORM más estándar. Permite realizar operaciones con tiempo de ejecución de O(1) ya que este implementa una tabla hash que le permite conocer el elemento que desea gestionar (OrmLite, 2017).

Base de datos Android SQLite

SQLite es un motor de bases de datos que ofrece características como: pequeño tamaño, no necesita servidor, precisa poca configuración, es transaccional y de código libre. La plataforma Android proporciona herramientas para el almacenamiento y consulta de datos estructurados (Park, 2018). Esta herramienta es utilizada como base de datos de las aplicaciones a desarrollar, se utiliza porque está integrada a la plataforma Android.

Android Studio v2.1.3

Android Studio fue seleccionado para el desarrollo de la solución por ser el IDE oficial para el desarrollo de aplicaciones para Android, este se basa en IntelliJ IDEA, un IDE desarrollado por la compañía JetBrains, el cual brinda un potente editor de códigos y herramientas para desarrolladores.

Android Studio ofrece funciones que aumentan la productividad durante la compilación de aplicaciones para Android, un sistema de compilación basado en Gradle, compilador de Android, un emulador rápido con múltiples funciones, un entorno unificado en el que se pueden realizar desarrollos para todos los dispositivos Android, *Instant Run* función que permite aplicar cambios mientras la aplicación se ejecuta sin la necesidad de compilar una nueva apk, integración de plantillas de código y GitHub, para ayudar a compilar funciones comunes de las aplicaciones e importar ejemplos de código, gran cantidad de herramientas y bibliotecas de prueba, herramienta Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versión, etcétera (Android Developer, 2017).

Android Studio además provee de un emulador, la integración con un kit de desarrollo y una terminal para la interacción con el emulador, para su configuración o ejecución de acciones.

Kit de Desarrollo de Software de Android 24.0.0

El Kit de Desarrollo de Software de Android (Android SDK) le proporciona al desarrollador las bibliotecas y las herramientas necesarias para crear, probar y depurar aplicaciones para Android. Para la realización de la solución se seleccionaron las bibliotecas de *CardView*, un elemento visual en forma de tarjeta que nos permite mostrar con un diseño agradable la información, *RecyclerView*, un contenedor de elementos que funciona en forma de lista y permite reciclar los elementos que ya no son visibles por el usuario al hacer el deslizamiento, y las bibliotecas de design, *Support* y *AppCompat*, que brindan los componentes visuales para el diseño, construcción de la aplicación y compatibilidad entre las diferentes versiones del sistema operativo (Android Developer, 2017).

Puente de Depuración de Android

El puente de depuración de Android (ADB por sus siglas en inglés). Es una herramienta de línea de comandos la cual ha sido desarrollada para permitir la comunicación con una instancia de un emulador o un dispositivo Android conectado al equipo. Con la herramienta se pueden realizar diferentes acciones en los dispositivos Android tales como la instalación y la depuración de aplicaciones, proporcionar acceso a un terminal desde el cual puede ser usado para ejecutar varios comandos en un emulador o un dispositivo conectado (Android Developer, 2017).

ADB es un programa cliente-servidor que incluye tres componentes básicos, cliente, daemon y servidor. El cliente es el encargado de enviar comandos. Este cliente se ejecuta en la máquina de desarrollo. El daemon se puede invocar a un cliente desde un terminal de línea de comandos emitiendo un comando de ADB, es el encargado de ejecutar comandos en un dispositivo. El daemon se ejecuta como un proceso en segundo plano en cada instancia del emulador o dispositivo Android conectado. Por último, el servidor**,** encargado de administrar la comunicación entre el cliente y el daemon (Android Developer, 2017).

Lenguaje de Modelado Unificado

El Lenguaje de Modelado Unificado (UML por sus siglas en inglés). Es un lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas de software, así como para el modelado del negocio y otros sistemas no software. Ayuda a capturar la idea de un sistema para comunicarla posteriormente a quien esté involucrado en su proceso de desarrollo; esto se lleva a cabo mediante un conjunto de símbolos y diagramas. Cada diagrama tiene fines distintos dentro del proceso de desarrollo (Larman, 2003).

Visual Paradigm v8.0

Es una suite de desarrollo de software y gestión empresarial, que ofrece características para la arquitectura empresarial, la gestión de proyectos, el desarrollo de software y la colaboración en equipo en una solución integral. Es una herramienta CASE profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar

diagramas de clases, código inverso, generar código desde diagramas y generar documentación (Visual Paradigm, 2014).

Metodologías de desarrollo

El software es el producto que construyen los programadores y al que después le dan mantenimiento, incluye programas que se ejecutan en una computadora de cualquier tamaño y arquitectura. La ingeniería de software está formada por un proceso, un conjunto de métodos y un arreglo de herramientas que permite a los profesionales elaborar software de computo de alta calidad (Pressman, 2014).

Una metodología de desarrollo es parte de ese conjunto de métodos y puede definirse como "un proceso mediante el cual un proyecto de software es completado o desarrollado a través de procesos o etapas bien definidas" (Chandra, 2015).

Entre las diferentes metodologías de desarrollo que existen, la metodología ágil es más recomendada para ser utilizadas en proyectos de aplicaciones móviles debido a que no son proyectos grandes y requieren de constantes cambios (Khalid, 2014).

En la UCI, se realizó una variante de una metodología ágil, Agile Unified Process (AUP), para ser utilizada por sus proyectos productivos. De esta manera, se adapta a sus procesos personalizados. Se decidió utilizar esta metodología por ser ágil y adaptar la propuesta solución a los procesos de la universidad. Aplica buenas prácticas apoyándose en el Modelo CMMI-DEV v1.3, el cual constituye una guía para aplicar las mejores prácticas que se centran en el desarrollo de productos y servicios de calidad (UCI, 2015).

AUP variación UCI

La metodología define 3 fases para el ciclo de vida de un proyecto, inicio, ejecución y cierre. En la fase inicio se llevan a cabo las actividades relacionadas con la planeación del proyecto, en ejecución se ejecutan actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura, y por último la fase de cierre, donde se analizan tanto los resultados del proyecto como su ejecución y se realizan actividades formales de cierre del proyecto.

Propone 7 disciplinas que son modelo del negocio, requisitos, análisis y diseño, implementación, pruebas internas, pruebas de liberación y pruebas de aceptación.

En el modelado del negocio para modelar el sistema en los proyectos existen 4 escenarios diferentes, de acuerdo a las necesidades de la propuesta solución se seleccionó el escenario 2 por sus características que son:

Escenario No. 2: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no es necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio. Se recomienda este escenario para proyectos donde el objetivo primario es la gestión y presentación de información (UCI, 2015).

1.5 Conclusiones del capítulo

Luego de analizar los sistemas existentes para la gestión de la información de vulnerabilidades y concluir que estas no resuelven el problema planteado se decidió la implementación de una nueva propuesta de solución mediante el consumo del servicio web brindado por el SIGEV.

Se fundamenta que la solución a desarrollar debe ser una aplicación Android nativa y ejecutarse en dispositivos móviles inteligentes.

Las herramientas y tecnologías con que se desarrolla la solución se corresponden con las establecidas por Google, patrocinadores de Android. El entorno de desarrollo para Android a utilizar es el IDE oficial Android Studio 2.1.3, así como el lenguaje de programación oficial Java en su versión 7 y la herramienta de modelado seleccionada fue Visual Paradigm for UML 8.0 capaz de representar el ciclo de vida completo de software.

El desarrollo de la aplicación objetivo de este trabajo está guiado por la metodología AUP en su variante UCI, escenario 2, respetando las políticas del proceso de desarrollo de software empleadas en la universidad.

Capítulo 2. Características de la aplicación de gestión de información de vulnerabilidades

La solución que se propone como resultado de esta investigación es una aplicación que se ejecuta sobre el sistema operativo Android, que maneja las vulnerabilidades recopiladas por el SIGEV de la DSI y que mantiene comunicado al usuario sobre las vulnerabilidades que se descubren.

A continuación se describen, siguiendo las especificaciones del escenario 2 de AUP variación UCI las disciplinas negocio, requisitos y análisis de la aplicación de gestión de información de vulnerabilidades.

2.1 Modelado del negocio

Para el escenario 2 se especifica que el modelo de negocio se represente a través del modelo de dominio puesto que el negocio que se trabaja, no tiene procesos claros si no conceptos relacionados. Un modelo de dominio es una representación de las clases conceptuales del mundo real. Ayuda a comprender mejor la estructura y dinámica de la organización, los problemas actuales dentro de esta e identificar las mejoras potenciales (Pressman, 2014).

El modelo de dominio tiene el objetivo de simbolizar los conceptos más significativos relacionados con el negocio en cuestión. El modelo específico de la solución propuesta se muestra en la figura 4.

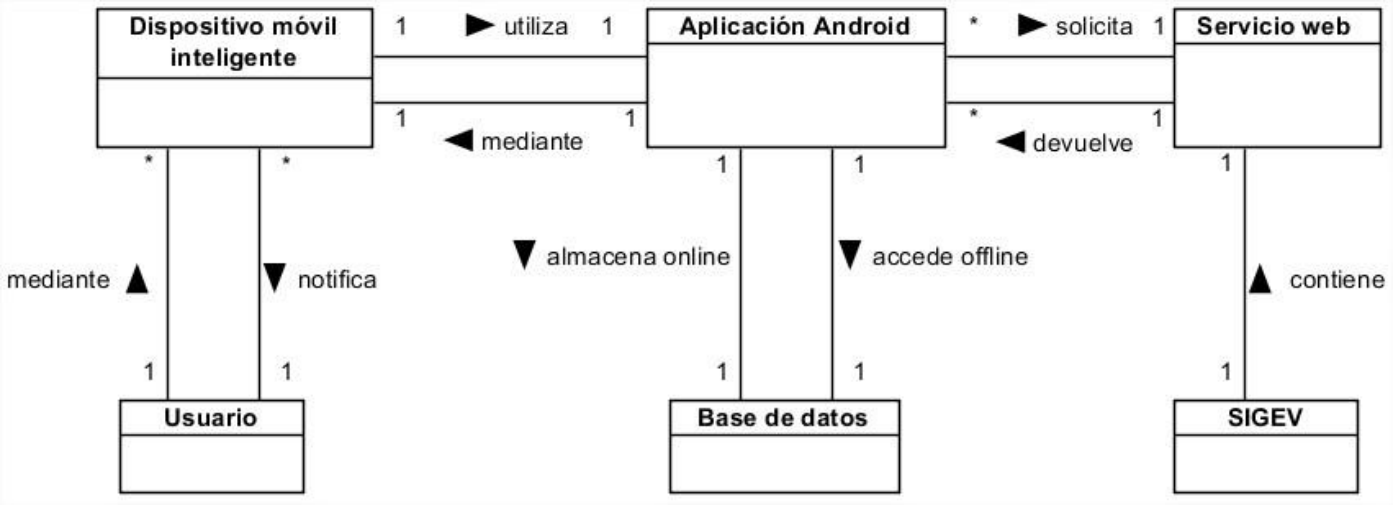


Fig. 4 Diagrama de modelo de dominio de la aplicación

Descripción del modelo de dominio

El usuario, mediante el dispositivo móvil, utiliza la aplicación Android y solicita la tecnología al servicio web del SIGEV. El servicio accede a SIGEV, obtiene las vulnerabilidades almacenadas en el sistema y se las entrega a la aplicación Android. Esta las almacena en su base de datos.

El usuario consulta la información proporcionada por SIGEV sin necesidad de estar conectado a la red, utilizando la aplicación Android y es notificado cuando se registran nuevas vulnerabilidades.

Descripción de las clases del modelo de dominio

SIGEV: Es el sistema que provee los datos de las vulnerabilidades que luego son tratados por la aplicación Android. Contiene el servicio web.

Servicio web: Establece la comunicación entre la aplicación Android y el SIGEV.

Dispositivo móvil: Este contiene el sistema operativo Android en el cual se ejecutará la aplicación para la gestión de las vulnerabilidades.

Aplicación Android: Aplicación que gestiona y organiza las vulnerabilidades que obtiene a través del servicio web. Posibilita la presentación de las vulnerabilidades al usuario sin que se necesite conexión activa de red.

Base de datos: Contiene los datos organizados de la aplicación Android.

Usuario: Utiliza el dispositivo móvil que tiene instalado la aplicación Android.

2.2 Descripción de los requisitos del sistema de la propuesta solución

El análisis de los requisitos da como resultado la especificación de las características del software, estos establecen las restricciones que lo limitan. Además, brindan la información que se traduce en diseños de arquitectura, interfaz y componentes, y otorgan al desarrollador y cliente los medios para evaluar la calidad una vez construido el software (Pressman, 2014).

A continuación, se muestran los requisitos funcionales y no funcionales de la aplicación para la gestión de información de vulnerabilidades, que fueron recopilados utilizando como técnica de recopilación de información, la tormenta de ideas, que se utiliza para identificar las características con las que debe cumplir teniendo en cuenta los criterios del cliente, tutores y el autor.

Requisitos funcionales:

Tabla.1 Requisitos funcionales

Número	Nombre	Descripción	Prioridad para el cliente	Complejidad
RF 1	Adicionar tecnología	Adiciona una tecnología con sus vulnerabilidades asociadas.	Alta	Alta

RF 2	Actualizar tecnología	Actualiza la tecnología incorporando nuevas vulnerabilidades y actualizando las incorporadas.	Alta	Alta
RF 3	Eliminar tecnología	Elimina una tecnología, así como de todas sus vulnerabilidades asociadas y la cancelación de su actualización.	Media	Media
RF 4	Seleccionar tecnologías a visualizar	Selecciona la tecnología que se desea visualizar.	Alta	Media
RF 5	Mostrar listado de tecnologías	Muestra un listado de todas las tecnologías existentes en la aplicación.	Baja	Baja
RF 6	Mostrar vulnerabilidad(es) por tecnología	Muestra un listado de todas las vulnerabilidades asociadas a una tecnología.	Alta	Alta
RF 7	Mostrar descripción de la vulnerabilidad	Muestra la información detallada de una vulnerabilidad.	Alta	Media
RF 8	Buscar vulnerabilidades sin conexión	Busca dentro de las tecnologías existentes y sus vulnerabilidades, por su fecha, descripción o identificador sin necesitar conexión a la red.	Alta	Alta
RF 9	Notificar información sobre actualizaciones.	Envía notificaciones, para mostrar una descarga, error de conexión o resultados descargados.	Alta	Media
RF 10	Configurar la aplicación	Permite la configuración de la aplicación para su personalización y adaptación, como establecimiento de tiempo de actualización, activar notificaciones, activar vibración, definir dirección del servicio web y establecer orden de prioridad para la visualización de las vulnerabilidades.	Media	Media

Requisitos no funcionales:

Tabla 2 Requisitos no funcionales de la aplicación

RNF de software	**RNF_1:** Es necesario un dispositivo móvil inteligente con sistema operativo Android con versión 4.1 o superior.
RNF de usabilidad	**RNF_2:** La aplicación debe poseer una interfaz intuitiva y amigable para asegurar la fácil navegación del usuario por la misma.
	RNF_3: Debe seguir los patrones del diseño material, creado y recomendado por Google, para el diseño de colores, íconos y componentes de la interfaz.
	RNF_4: La aplicación debe estar disponible en inglés y español.
RNF de hardware	**RNF_5:** La aplicación necesita al menos 4 megabytes de almacenamiento para ser instalada y 512 megabytes de memoria RAM.
RNF de seguridad	**RNF_6:** La comunicación externa con el servidor se realiza a través de la Capa Segura de Comunicaciones (SSL).
RNF de soporte	**RNF_7:** La aplicación se construye utilizando los principales estándares de codificación de Java, el diseño utilizará Patrones Generales de Software para Asignar Responsabilidades (GRASP) y los patrones de diseño Gang of Four (GoF) para garantizar la escalabilidad del sistema.

Descripción del sistema

Luego de recopilados los requisitos funcionales estos se agrupan en casos de uso. (Pressman, 2014) define los casos de uso como un conjunto de escenarios que identifican la naturaleza de los usos para el sistema que se va a construir, estos proporcionan la descripción en la que se utilizará el sistema. En la matriz de trazabilidad siguiente se identifica cómo se agruparon los requisitos identificados y los nombres de los casos de uso que los relaciona.

Tabla.3 Matriz de trazabilidad requisitos-casos de uso del sistema

Referencia a requisitos	Nombre del caso de uso
RF: 1,2,3,4,5 y 6	Gestionar tecnología
RF: 7	Mostrar vulnerabilidad

RF: 8	Buscar vulnerabilidad
RF: 9	Notificar información sobre actualizaciones
RF: 10	Configurar aplicación

Un actor es un papel que desempeñan las personas (usuarios) o los dispositivos cuando interactúan con el software (Pressman, 2014). En el análisis de la aplicación se identifica como actor al usuario que interactúa con el sistema y que puede estar desempeñado por un administrador, un especialista de seguridad informática o cualquier otra persona que requiera la información sobre vulnerabilidades.

Modelación de los casos de uso del sistema

Los diagramas de casos de uso del sistema permiten representar como el actor interactuará con los casos de uso del sistema anteriormente modelados (Larman 2003). A continuación se muestra la relación del actor con los casos de uso del sistema identificados.

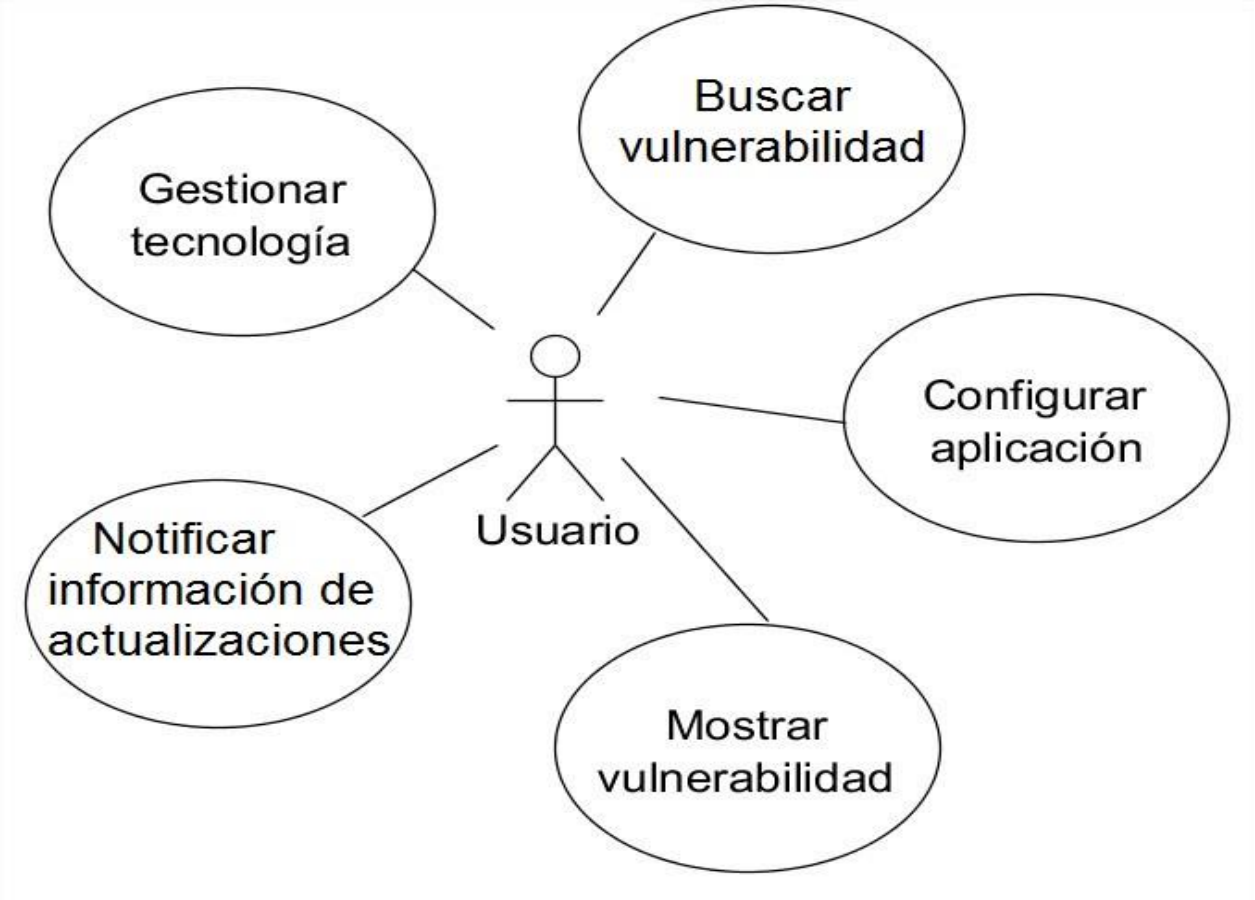


Fig. 5 Diagrama de caso de uso del sistema

Para completar la modelación se describen los casos de uso representados. Para determinar la complejidad del caso de uso se analizan la de los requisitos funcionales de la siguiente forma:

Tabla.4 Descripción del caso de uso del sistema: Gestionar tecnología

Objetivo	Gestionar las tecnologías	
Actor	Usuario	
Descripción	El caso de uso comienza cuando el usuario desea adicionar, actualizar o eliminar la tecnología.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	El usuario debe completar el tutorial.	
Poscondiciones	Se actualiza la base de datos de la aplicación	
Referencias	RF 1, RF 2, RF 3, RF 4, RF 5, RF 6	
Flujo de eventos		
Flujo básico Adicionar tecnología		
	Actor	**Sistema**
1	El usuario pulsa el botón de adicionar tecnología.	El sistema muestra un diálogo con un campo para introducir la tecnología.
2	El usuario escribe la tecnología y pulsa el botón Aceptar, en caso de pulsar el botón Cancelar ver **Flujo alterno 1: Cancelar**.	El sistema comprueba si los datos no están vacíos, si están vacíos ver **Flujo alterno 1: Cancelar**, si la tecnología está escrita incorrectamente ver **Flujo alterno 2: Adicionar tecnología incorrectamente**, si la tecnología está escrita correctamente, pero existe actualmente, entonces ver **Flujo alterno 3: Adicionar tecnología existente**. En caso contrario la tecnología es adicionada a la base de datos.
Flujos alternos		
No 1: Cancelar		
	Actor	**Sistema**
1		El sistema retira el diálogo y muestra la interfaz detrás del diálogo.
Flujos alternos		
No. 2 Adicionar tecnología incorrectamente		
	Actor	**Sistema**
1		El sistema envía un mensaje notificando que el nombre de la tecnología no se introdujo correctamente.
2	El usuario pulsa el botón de Aceptar del mensaje.	
Flujos alternos		
No. 3 Adicionar tecnología existente		

	Actor	**Sistema**
1		El sistema envía un mensaje notificando que la tecnología ya se encuentra añadida.
2	El usuario pulsa el botón de Aceptar del mensaje.	
Flujo de eventos		
Flujo básico Mostrar vulnerabilidad(es) por tecnología		
	Actor	**Sistema**
1	El usuario presiona del listado de tecnologías la que desea visualizar.	El sistema busca en la base de datos todas las vulnerabilidades asociadas a la tecnología, sino encuentra ninguna vulnerabilidad asociada a esa tecnología muestra un texto notificando que no se encontraron elementos, en caso de que si encuentre vulnerabilidades se muestra un listado de vulnerabilidades asociadas a la tecnología seleccionada.
Flujo de eventos		
Flujo básico Actualizar tecnologías		
	Actor	**Sistema**
1	El usuario presiona el botón Actualizar	El sistema realiza una sincronización con el servicio web, si en el momento de la sincronización existe conexión entonces ver **Flujo alterno 1: Actualizar con conexión**, sino presenta conexión entonces ver **Flujo alterno 2: Actualizar sin conexión**, sino existen tecnologías que actualizar ver **Flujo alterno 3: Actualizar sin tecnología.**
Flujos alternos		
No. 1 Actualizar con conexión		
	Actor	**Sistema**
1		Realiza una actualización de todas las tecnologías, muestra un contador con la cantidad de vulnerabilidades nuevas y actualiza la fecha de última actualización.
Flujos alternos		
No. 2 Actualizar sin conexión		
	Actor	**Sistema**
1		Muestra un símbolo de que no existe conexión con el servicio web.
Flujos alternos		
No. 3 Actualizar sin tecnologías		
	Actor	Sistema
		El sistema muestra un diálogo al usuario notificándole de que debe adicionar antes una tecnología.

Flujo de eventos		
Flujo básico Eliminar tecnología		
	Actor	**Sistema**
1	El usuario presiona el botón Eliminar la tecnología	El sistema muestra un diálogo de confirmación.
	El usuario pulsa el botón Aceptar, en caso de pulsar el botón Cancelar ver Flujo 1: Cancelar.	El sistema elimina de la base de datos la tecnología y todas las vulnerabilidades asociadas, así como cualquier actualización de la tecnología y muestra la pantalla “Inicio”.
Flujos alternos		
No. 1: Cancelar		
1		El sistema retira el diálogo y muestra la interfaz detrás del diálogo.

Prototipo elemental de interfaz gráfica de usuario

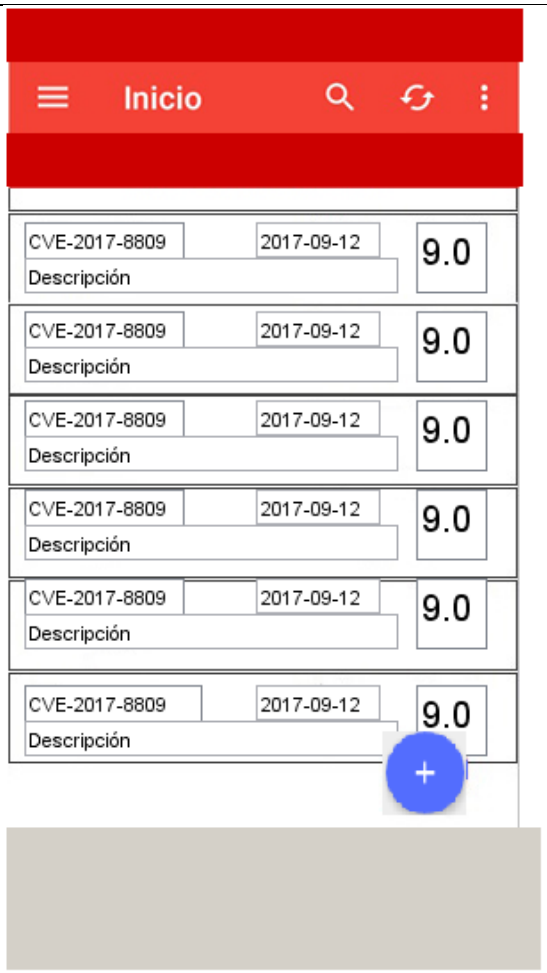


Tabla.5 Descripción del caso de uso del sistema: Buscar vulnerabilidad

Objetivo	Buscar vulnerabilidad
Actor	Usuario
Descripción	El caso de uso comienza cuando el usuario realiza una búsqueda, luego puede visualizar un listado con los resultados. El caso de uso terminará una vez mostrado los resultados de la búsqueda o el mensaje de que no se encontraron resultados que coincidan con la búsqueda.
Complejidad	Alta
Prioridad	Alta
Precondiciones	El usuario debe adicionar al menos una tecnología que presente al menos una vulnerabilidad.
Poscondiciones	Se realizó la búsqueda con resultados y serán mostrados en un listado.
Referencias	RF 8
Flujo de eventos	
Flujo básico Buscar	

	Actor	**Sistema**
1		El usuario decide realizar una búsqueda, en caso de no existir ningún resultado ver Flujo alterno 1: No existen resultados.
2	El usuario realiza la búsqueda	El sistema realiza una búsqueda en su base de datos y crea un listado con las coincidencias para mostrárselos al usuario.
	El usuario visualiza los resultados	Termina el caso de uso del sistema.

Flujos alternos

No. 1 No existen resultados

	Actor	**Sistema**
1		El sistema envía un mensaje notificando que no existen resultados
2	El usuario pulsa el botón de aceptar del mensaje.	Termina el caso de uso del sistema

Prototipo elemental de la interfaz gráfica de usuario

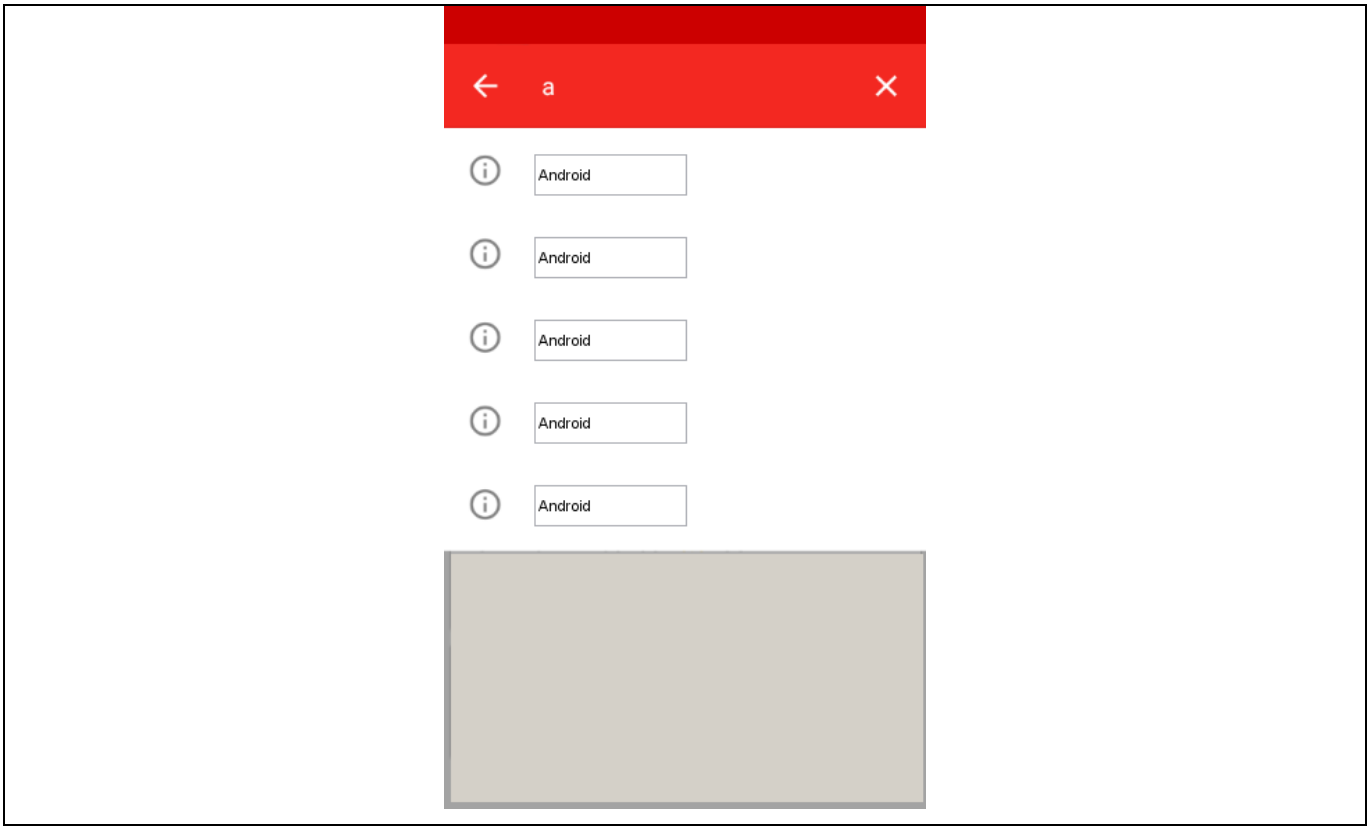


Tabla 6 Descripción del caso de uso del sistema: Mostrar vulnerabilidad

Objetivo	Mostrar vulnerabilidad
Actor	Usuario
Descripción	El caso de uso comienza cuando el usuario selecciona una tecnología a visualizar y se muestra un listado de vulnerabilidades y desea ver la descripción específica de una vulnerabilidad. El caso de uso termina cuando se muestra la vista de detalles de una vulnerabilidad.
Complejidad	Media
Prioridad	Alta
Precondiciones	El usuario debe adicionar al menos una tecnología que presente al menos una vulnerabilidad y seleccionar una tecnología a visualizar.
Poscondiciones	Se muestra la descripción detallada de una vulnerabilidad.
Referencias	RF 7
Flujo de eventos	
Flujo básico Mostrar vulnerabilidad	

	Actor	Sistema
1		El usuario decide visualizar la descripción detallada de una vulnerabilidad.

2	El usuario selecciona una vulnerabilidad	El sistema obtiene el identificador que el usuario desea visualizar y realiza una búsqueda en la base de datos de la vulnerabilidad y obtiene todos los datos sobre esta y muestra en una vista de detalle la descripción específica de la vulnerabilidad.
	El usuario visualiza la vista de detalles de la vulnerabilidad.	Termina el caso de uso del sistema.
Prototipo elemental de la interfaz gráfica de usuario		

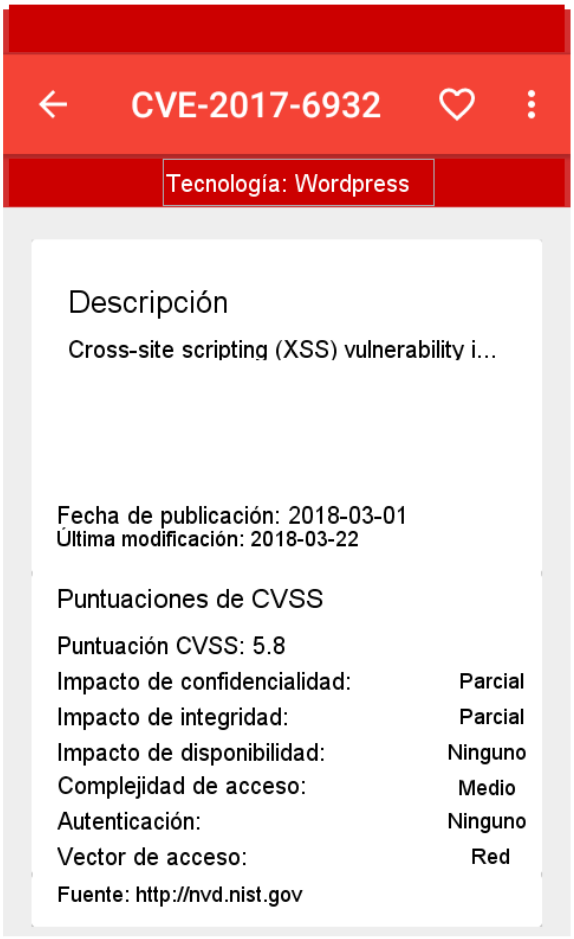


Tabla 7 Descripción del caso de uso del sistema: Notificar información de actualizaciones

Objetivo	Notificar información sobre actualizaciones
Actor	Usuario

Descripción	El caso de uso comienza cuando se realiza una sincronización con el servicio web y se le notifica al usuario la información obtenida sobre el registro de nuevas vulnerabilidades.
Complejidad	Media
Prioridad	Alta
Precondiciones	El usuario debe adicionar al menos una tecnología.
Poscondiciones	Se muestra la notificación de las nuevas vulnerabilidades registradas por tecnología
Referencias	RF 9

Flujo de eventos

Flujo básico Notificar

	Actor	**Sistema**
1		El sistema mediante una alarma, se sincroniza con el servicio web, en caso de no existir conexión ver Flujo alterno 1: Sin conexión, en caso contrario, obtiene los datos de las vulnerabilidades según cada tecnología correspondiente, luego las compara con las que posee actualmente, adiciona las que no tenga registradas y actualiza las que posean modificaciones en el servicio. El sistema envía una notificación al dispositivo con esta información.
2	El usuario visualiza esta información.	Termina el caso de uso.

Flujos alternos

No. 1: Sin conexión

	Actor	**Sistema**
1		El sistema envía una notificación al dispositivo informando que no existe conexión con el servicio web.
2	El usuario visualiza esta información.	Termina el caso de uso.

Prototipo elemental de la interfaz gráfica de usuario

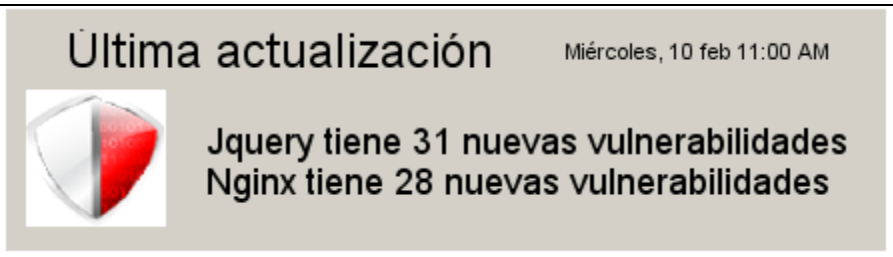


Tabla 8 Descripción del caso de uso del sistema: Configurar aplicación

Objetivo	Configurar aplicación	
Actor	Usuario	
Descripción	El caso de uso comienza cuando el usuario selecciona la opción de configuración, luego puede modificar los valores establecidos anteriormente y configurar la aplicación.	
Complejidad	Media	
Prioridad	Media	
Precondiciones		
Poscondiciones	Se configura la aplicación.	
Referencias	RF 10	
Flujo de eventos		
Flujo básico Configurar aplicación		
	Actor	**Sistema**
1		El usuario decide configurar la aplicación
2	El usuario modifica una preferencia en la configuración.	El sistema obtiene la preferencia y el valor modificado para persistir los datos a través de las preferencias compartidas. Actualiza la aplicación y la vista de configuración.
3	El usuario visualiza los cambios.	Termina el caso de uso.
Prototipo elemental de la interfaz gráfica de usuario		

2.3 Análisis de la propuesta de solución

El análisis pone énfasis en una solución conceptual que satisface a los requisitos (Larman, 2003). El resultado de esta disciplina se representa mediante modelos de análisis a través de los distintos casos de usos.

Diagrama de clases de análisis

Mediante los diagramas de clases de análisis se estructuran los requisitos funcionales definidos con anterioridad utilizando los casos de uso realizados, permitiendo una mayor comprensión del problema para modelar la solución propuesta (Larman, 2003). A continuación, se presentan los diagramas de análisis de la solución propuesta:

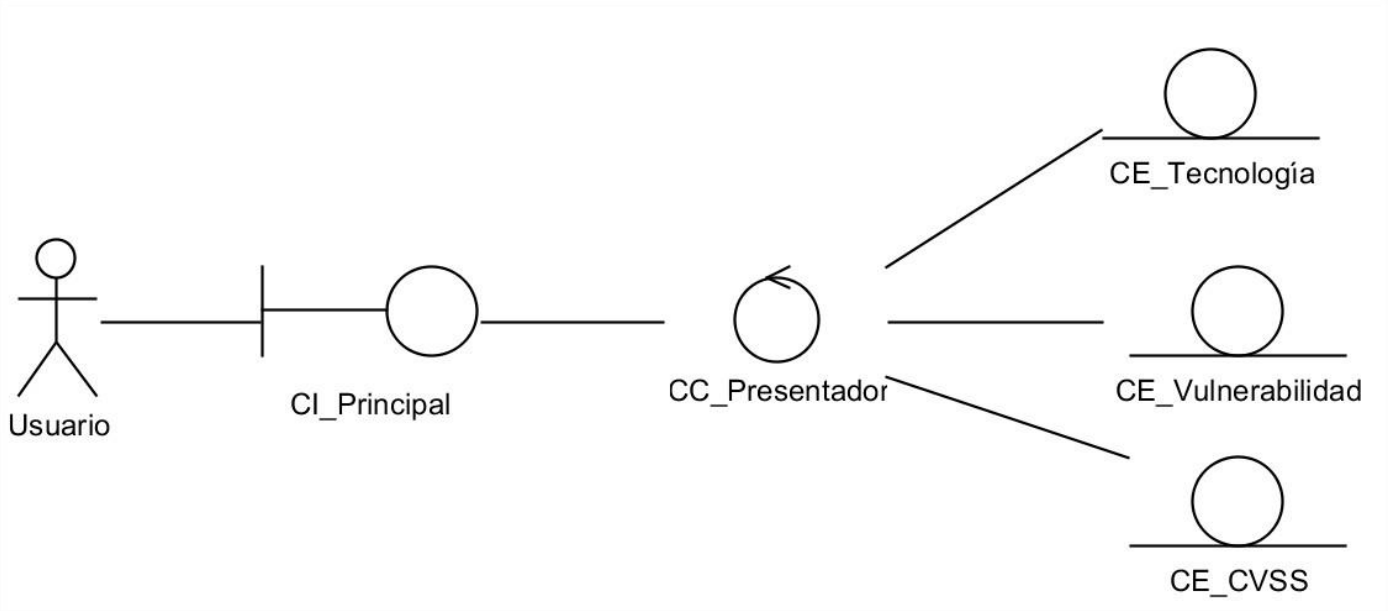


Fig. 6 Diagrama de clases del análisis: Gestionar tecnología

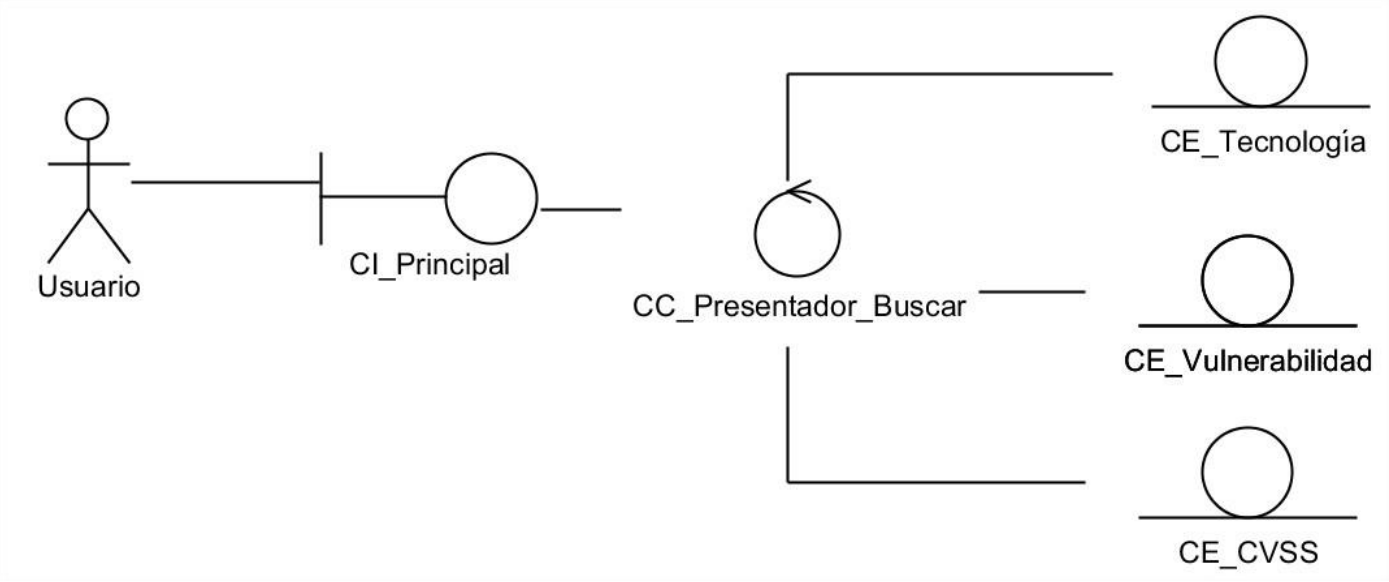


Fig. 7 Diagrama de clases del análisis: Buscar vulnerabilidad

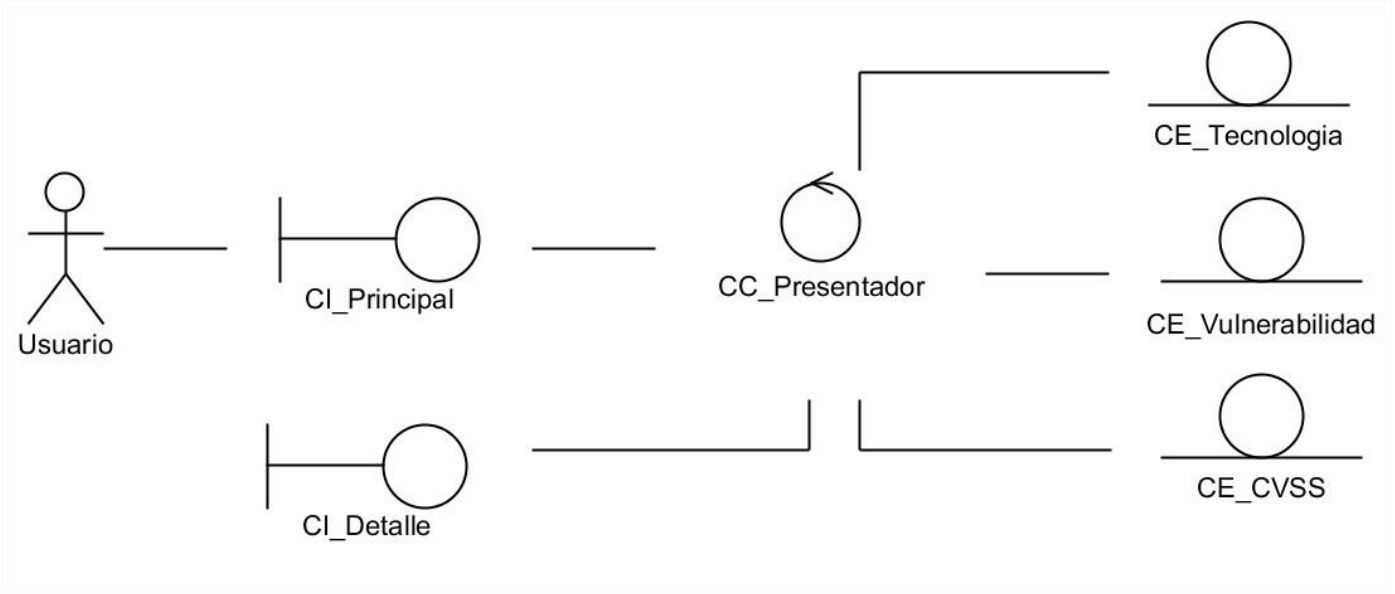


Fig. 8 Diagrama de clases del análisis: Mostrar vulnerabilidad

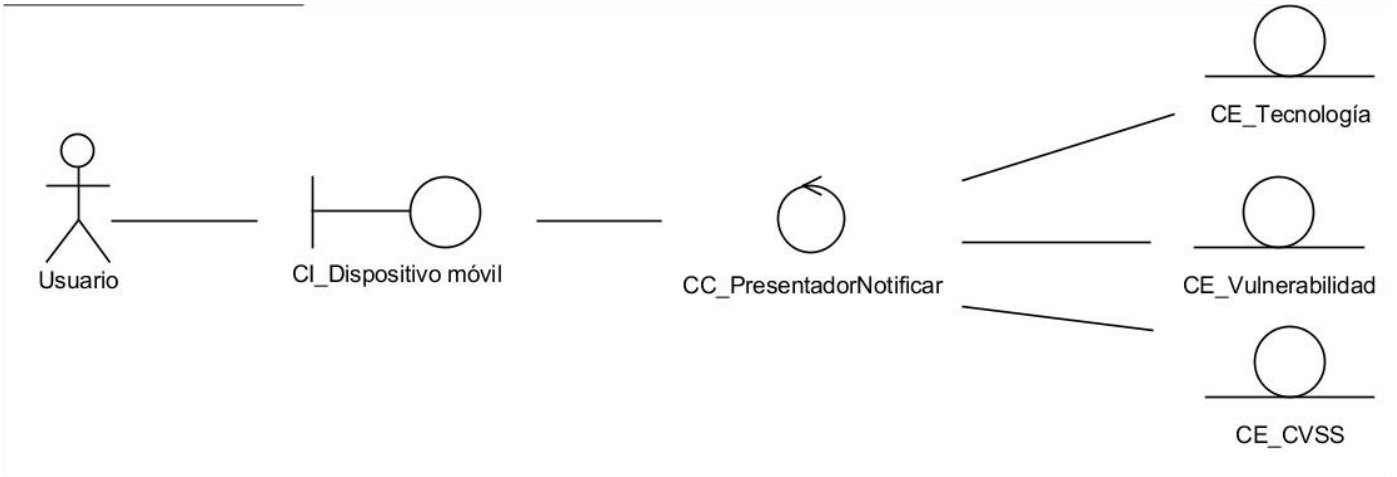


Fig. 9 Diagrama de clases del análisis: Notificar información de actualizaciones

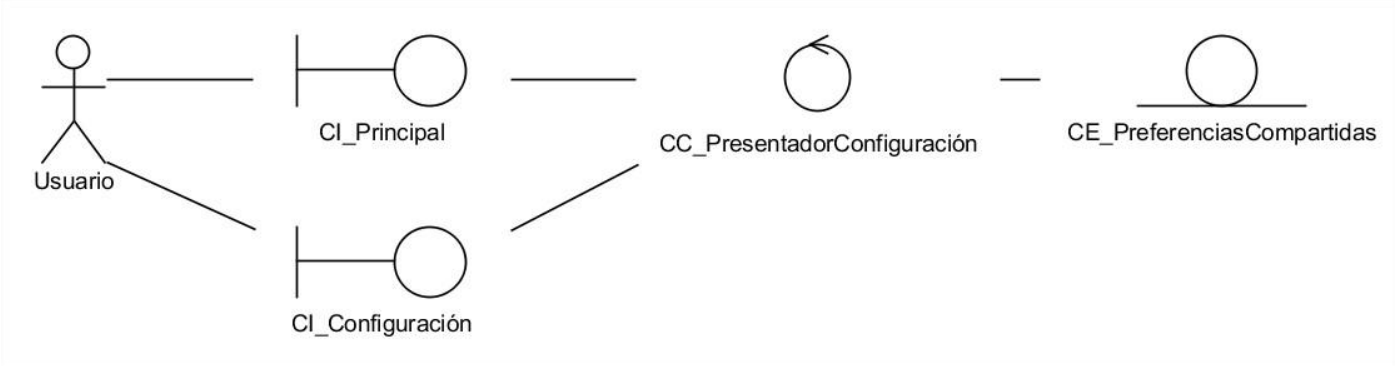


Fig. 10 Diagrama de clases del análisis: Configurar aplicación

Como se puede apreciar en los diagramas, el usuario interactúa con clases interfaces que se comunican con las entidades a través de las clases controladoras. Las clases interfaces se definen teniendo en cuenta la información con la que debe interactuar el usuario, de ahí que varíe de un caso de uso a otro, pero siempre partiendo de la clase principal excepto en el de notificación que el usuario solo recibe información a través de mensajes de la aplicación.

Diagramas de secuencia

Los diagramas de secuencia son según (Jacobson, 2000) una "sociedad de clases y otros elementos que colaborarán para realizar el comportamiento expresado en un caso de uso". Estos muestran mediante las interacciones entre objetos, creando enlaces entre ellos y añadiendo mensajes. A continuación se representan los diagramas de secuencias de los casos de uso incorporados a la propuesta solución:

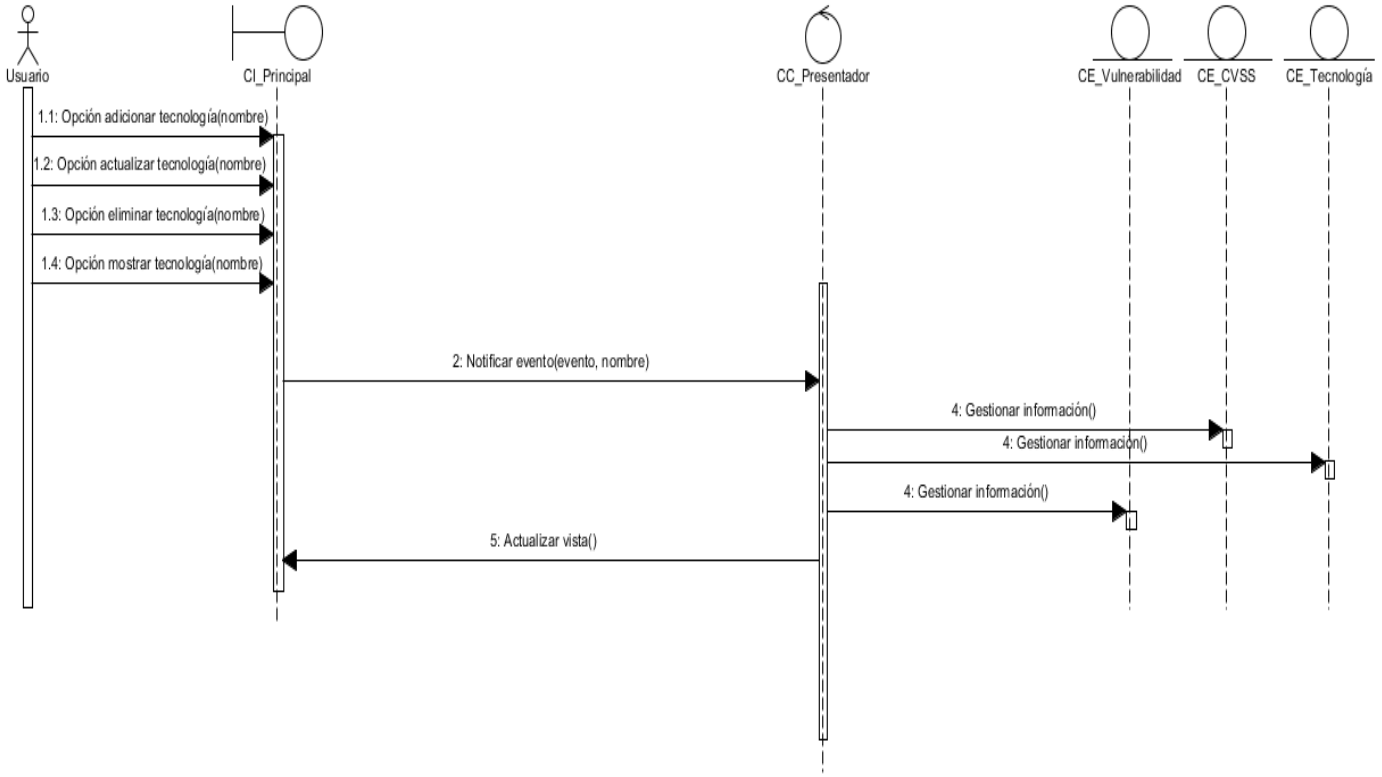


Fig. 11 Diagrama de secuencia: Gestionar tecnología

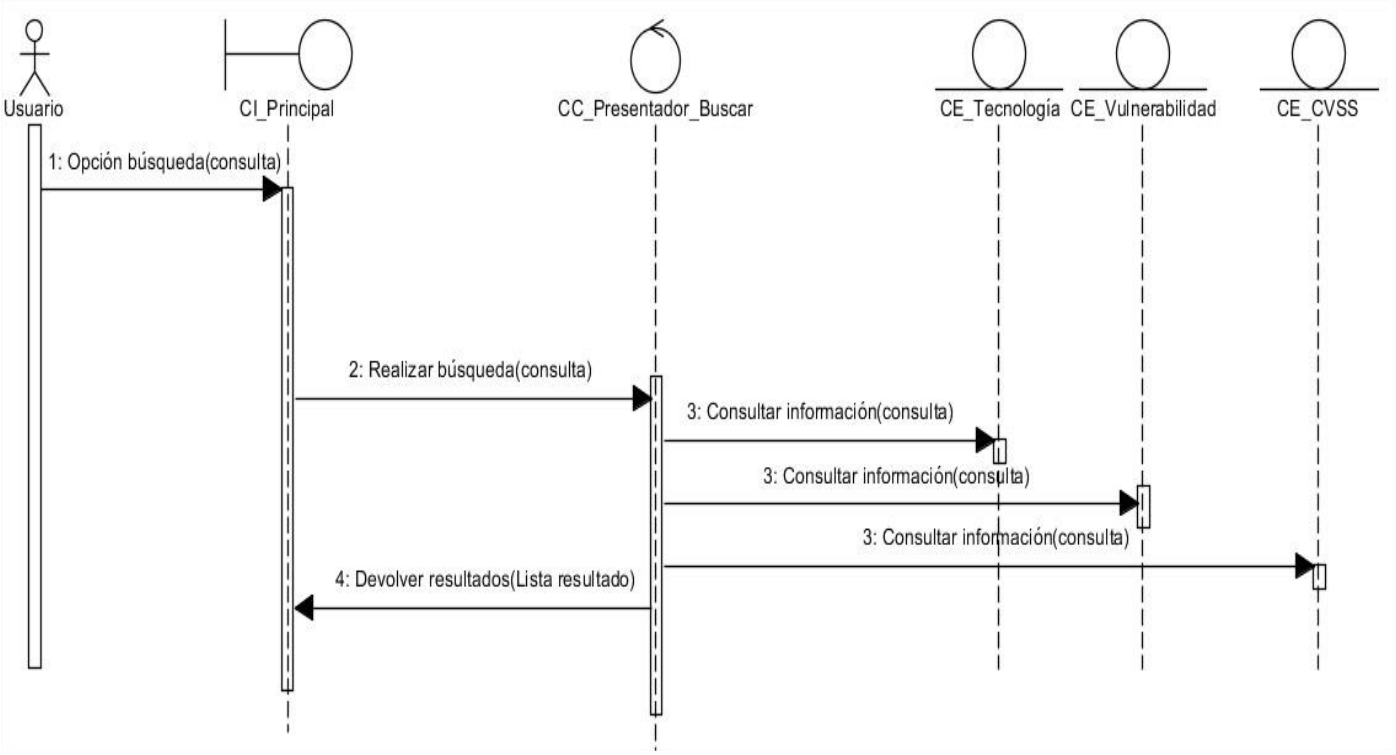


Fig. 12 Diagrama de secuencia: Buscar vulnerabilidad

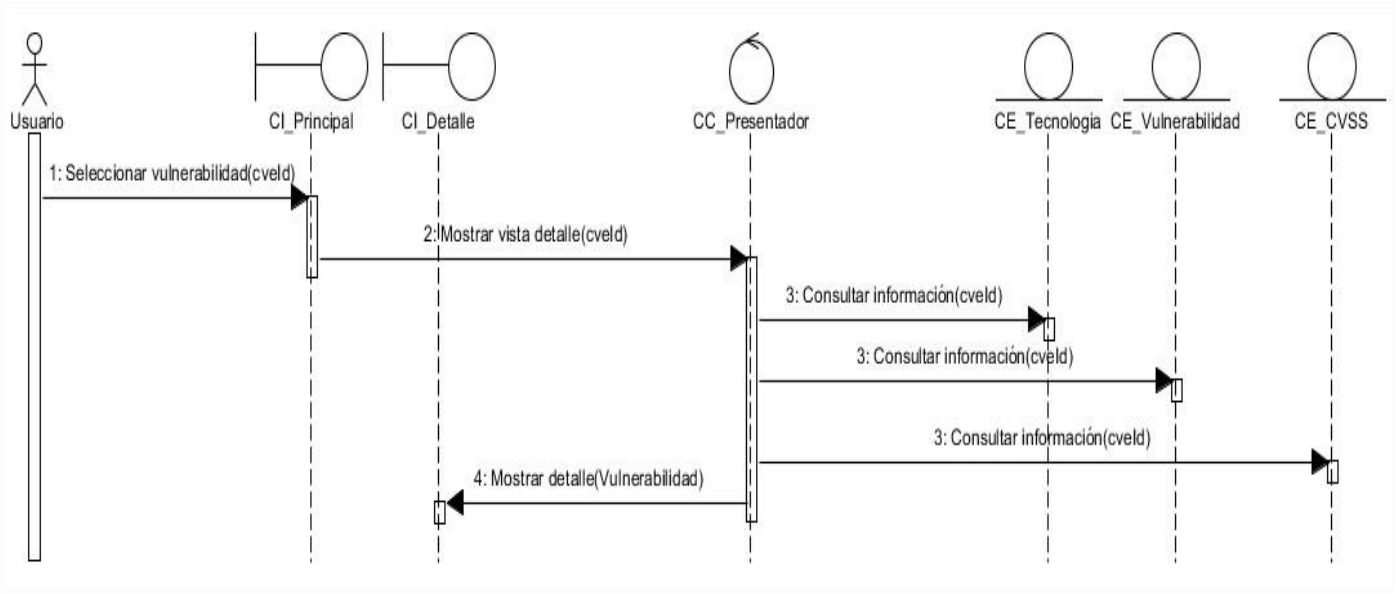


Fig. 13 Diagrama de secuencia: Mostrar vulnerabilidad

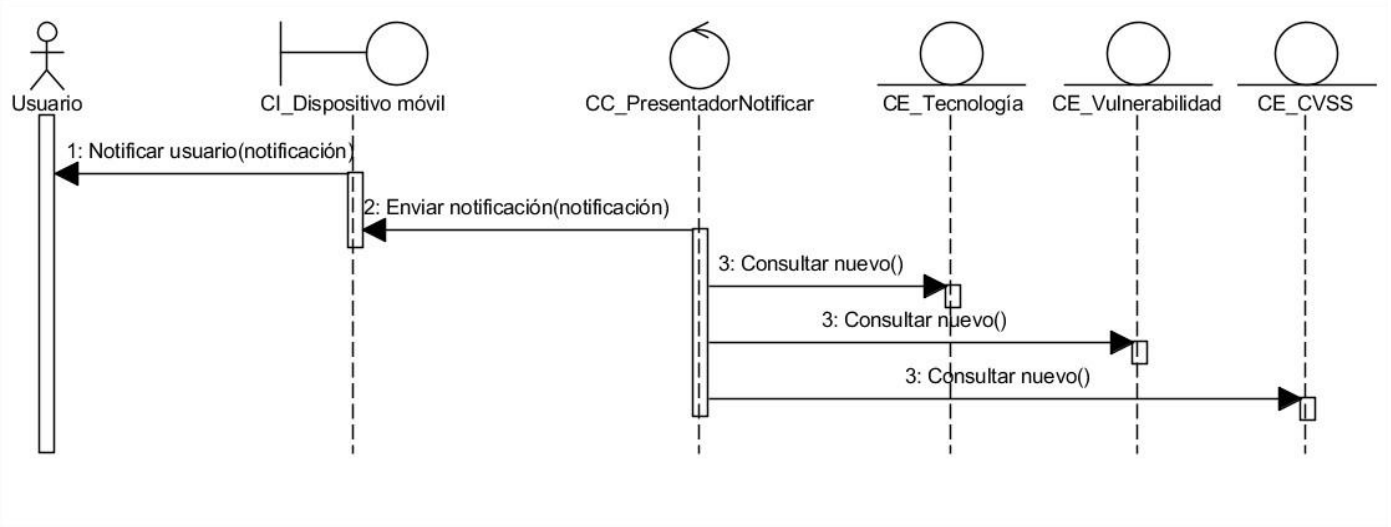


Fig. 14 Diagrama de secuencia: Notificar información de actualizaciones

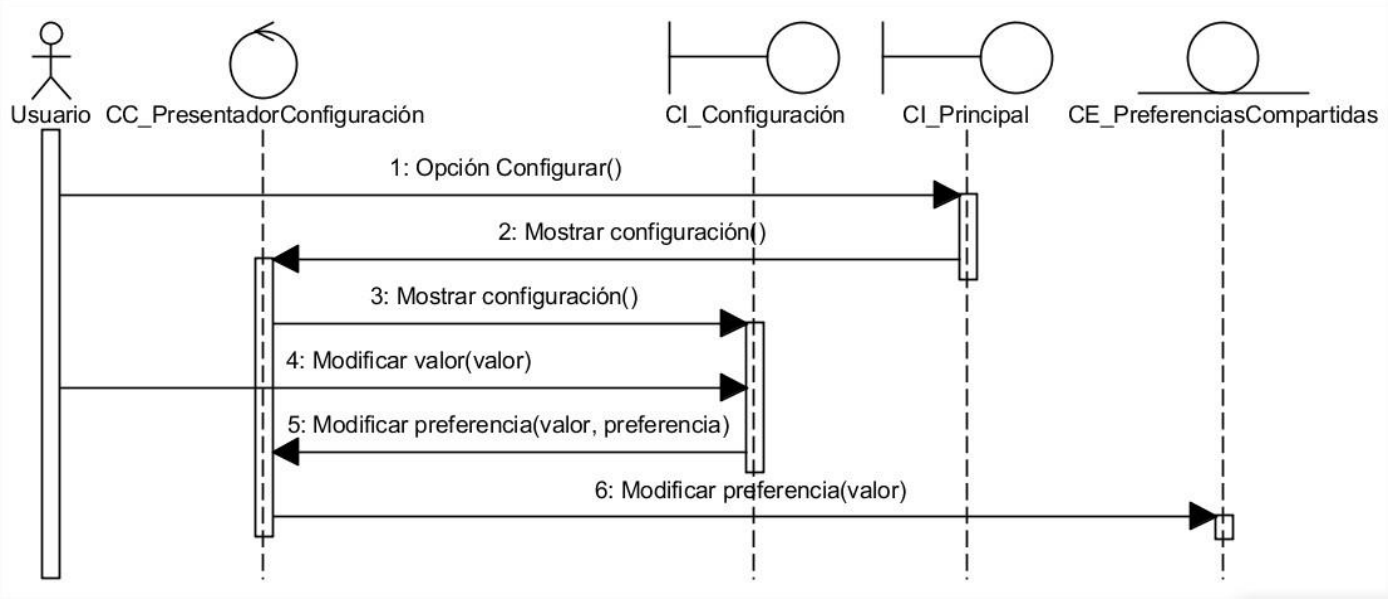


Fig. 15 Diagrama de secuencia: Configurar aplicación

2.4 Conclusiones del capítulo

Se definieron un total de 10 requisitos funcionales que se agruparon en 5 casos de uso y 7 no funcionales que representan las características que debe cumplir el sistema identificando solo un actor usuario que representa a toda aquella persona que puede interactuar con el sistema.

Fueron elaborados los diagramas de análisis y secuencia siguiendo la organización de los casos de uso definida en la disciplina requisitos.

Capítulo 3. Diseño, implementación y validación de la aplicación de gestión de información de vulnerabilidades

La disciplina diseño tiene como entrada los modelos generados en la disciplina de análisis y el estilo arquitectónico con el que se estructura la aplicación. Para el modelado de los productos de trabajo del diseño se analiza a continuación la arquitectura y los patrones de diseño que se van a emplear.

Arquitectura de desarrollo

El diseño arquitectónico representa la estructura de los datos y de los componentes del programa que se requieren para construir un sistema. La arquitectura del software de un programa o sistema de cómputo es la estructura o estructuras del sistema, lo que comprende a los componentes del software, sus propiedades externas visibles y las relaciones entre ellos (Guano, 2014).

Existen diferentes estilos de arquitecturas para el diseño de software. Para la aplicación se utiliza el estilo llamar y regresar. Este estilo permite obtener una estructura de programa relativamente fácil de modificar y escalar. Dentro de este se seleccionó el patrón de arquitectura de software Modelo-Vista-Presentador (MVP).

Este patrón es una variante del Modelo Vista Controlador (MVC). MVP tiene la finalidad de separar la lógica de la presentación, de tal forma que todo lo relacionado con cómo funciona la interfaz queda separado del cómo representarlo en pantalla, debido a que en Android las actividades, fragmentos y demás elementos que componen a la vista están íntimamente acopladas tanto con la interfaz como con las mecánicas de acceso de datos, hasta tal punto que elementos de la interfaz acceden a la base de datos o realizan peticiones a un servicio web. Esto a nivel estructural no es una buena práctica, ya que, al no separarse estas responsabilidades; el acoplamiento, la legibilidad y otros principios de calidad del software no se ven respetados. El MVP independiza la vista de la forma de conseguir esos datos y divide la aplicación en tres capas distintas, facilitando la realización de las pruebas unitarias. Estas capas son:

Presentador: es el encargado de actuar de intermediario entre la vista y el modelo. Recupera los datos del modelo y se los devuelve formateados a la vista. Pero a diferencia del MVC, también decide qué ocurre cuando se interactúa con la vista y el presentador no tiene conciencia de ella.

Vista: está compuesta por actividades, fragmentos, adaptadores y demás elementos que heredan de la clase View. La vista contendrá una referencia al presentador y es la encargada de crear el objeto presentador. Lo único que hará la vista será llamar a un método del presentador cada vez que se realice una acción sobre la interfaz, normalmente el pulsado de un botón, un elemento de una lista, etc. Esta no se relaciona con el modelo.

Modelo: es la representación de los objetos del negocio que le provee los datos que necesita el presentador y los métodos para poder modificarlos. El modelo nunca interactúa con la vista (Guano, 2014).

Fig. 16 Arquitectura de software Modelo Vista Presentador

Descripción de los patrones de Principios Generales para Asignar Responsabilidades

El software ha ido aumentando su complejidad de forma pareja a la potencia de las computadoras, el aumento del conocimiento de los usuarios, incremento de las posibilidades de la informática, de las nuevas posibilidades que ofrecen las redes, los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo. Estos mejoran la flexibilidad, modularidad y extensibilidad, reduce los esfuerzos de desarrollo y mantenimiento, y mejora la seguridad, eficiencia y consistencia de nuestros diseños. Los patrones de Principios Generales para Asignar Responsabilidades (GRASP) se describen a continuación (Larman, 2003):

- **Experto**

Este patrón consiste en asignar la responsabilidad a la clase que tiene información necesaria para realizarla. Se conserva el encapsulamiento y el comportamiento se distribuye entre todas las clases implicadas en la ejecución de la función. Por ejemplo, la clase Presentador.java es la clase encargada de realizar la transformación de los datos provistos por el servicio web y enviárselos al modelo.

- **Creador**

La creación de instancias bien asignadas permite que el diseño pueda soportar un bajo acoplamiento, una mayor claridad y reutilización. La clase DatabaseHelper.java es la clase que tiene la instancia de los objetos que serán almacenados en la base de datos y tiene la función de enviarle los objetos DAO a la clase Presentador.java cuando esta los solicite.

- **Controlador**

El patrón controlador es un intermediario entre determinada interfaz y un algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a otras clases según el llamado. Una evidencia del uso de este patrón es la clase Presentador.java debido a que es la clase con la responsabilidad de controlar el flujo de eventos de la aplicación.

- **Bajo acoplamiento**

El acoplamiento es una medida de la fuerza con que un elemento está conectado a otro. En la aplicación se aplica minimizando la dependencia entre las clases para garantizar que al modificar una clase, otras no se vean afectadas. Un ejemplo de ello es la clase Singleton.java ya que esta clase no depende de otras para realizar la petición y obtener los datos del servicio web.

- **Alta cohesión**

La cohesión es la medida de la fuerza que une a las responsabilidades de una clase. Mientras menos relación tenga la clase para lograr sus responsabilidades, menor será su cohesión.

Este patrón es el encargado de asignar responsabilidades, de manera que la información que se almacena en una clase, sea la necesaria y esté bien delimitada. Se evidencia su uso en PlaceholderFragment.java. La responsabilidad de esta clase es listar las vulnerabilidades y simplifica el trabajo de otras clases que, para hacerlo, solo deben utilizarla.

Descripción de los patrones de diseño Gang of Four

Los patrones de diseño proporcionan un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe estructuras repetitivas de comunicar componentes que resuelven un problema de diseño en un contexto particular.

Los patrones de diseño Gang of Four (GoF) son un conjunto de 23 patrones de diseño que están divididos en 3 categorías denominadas creacionales, estructurales y de comportamiento. Estos reducen la complejidad del sistema nombrando y definiendo abstracciones utilizando clases e instancias, constituyen

una base reusable de experiencia para la construcción de diseño y proporcionan una reorganización de clases mediante jerarquías. En esta investigación se utilizan los siguientes (Gamma, 1994):

- ***Singleton* (Creacional)**

El objetivo de este patrón es asegurarse que una clase solo tiene una instancia y ofrecer un punto de acceso a ella. En la clase Singleton.java se utiliza este patrón, debido a que esta clase utiliza la función *synchronized* de java, que establece una sección crítica para esta clase, limitando la instancia de esta clase a ser única.

- ***Builder* (Creacional)**

Este patrón consiste en hacer una separación de la construcción de objetos complejos o compuestos de su representación de modo que el mismo proceso de construcción pueda crear diferentes representaciones Este patrón se evidencia al crear un *RecyclerView*, debido a que este objeto complejo se crea a través de elementos más simples como el *LinearLayoutManager*, *VulnerabilidadAdapter* y *List<Vulnerabilidad>*.

- ***Observer* (Comportamiento)**

Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él. Este patrón se pone de manifiesto en la clase Singleton.java al realizar peticiones al servicio web, este verifica el estado de la conexión y lo notifica al Presentador para la actualización de la vista.

- ***Adapter* (Estructural)**

Permite que objetos que se encuentran implementados puedan ser adaptados para ser reutilizados sin que se necesiten cambios en ellos, simplemente envolviéndolos con una capa alrededor que nos permitirá adaptarlos al nuevo interfaz deseado. Esto se evidencia en la clase VulnerabilidadAdapter.java que a través de un adaptador simplifica la creación de objetos de la lista.

3.1 Diseño de la aplicación de gestión de información de vulnerabilidades

El diseño de software agrupa al conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema o producto de alta calidad. Los principios de diseño establecen una filosofía general que guía el trabajo de diseño que debe ejecutarse. Deben entenderse los conceptos de diseño antes de aplicar la mecánica de éste, y la práctica del diseño en si lleva a la creación de distintas representaciones del software que sirve como guía para la construcción que siga (Pressman, 2014).

Diagramas de clases de diseño

Los diagramas de clases de diseño representan las especificaciones de clases e interfaces de software en una aplicación. Estas muestran las definiciones de las clases de software en lugar de los conceptos del mundo real (Pressman, 2014). A continuación son representados algunos de los diagramas de clases de diseño de la propuesta solución:

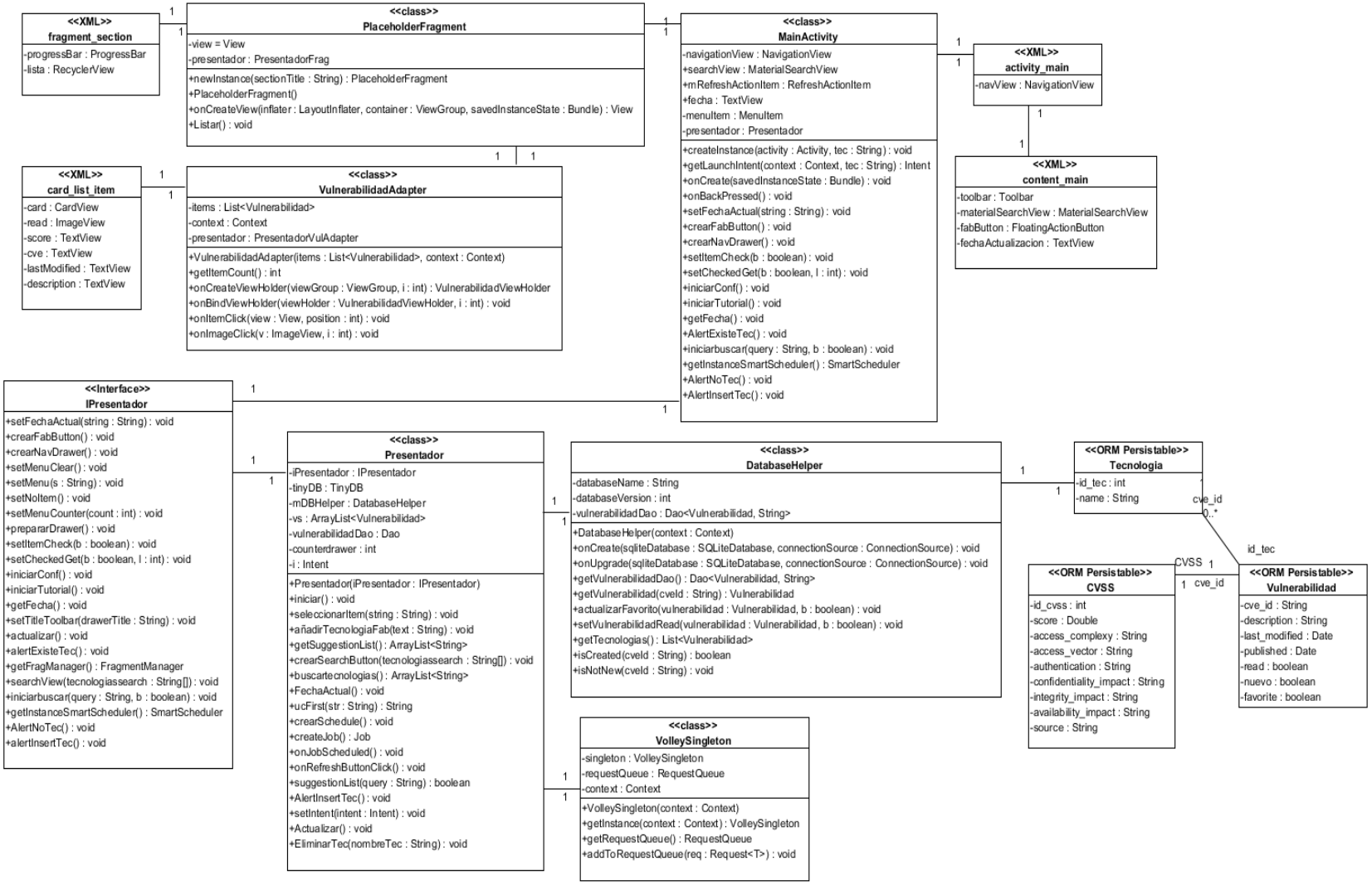


Fig. 17 Diagrama de clases de diseño: Gestionar tecnología

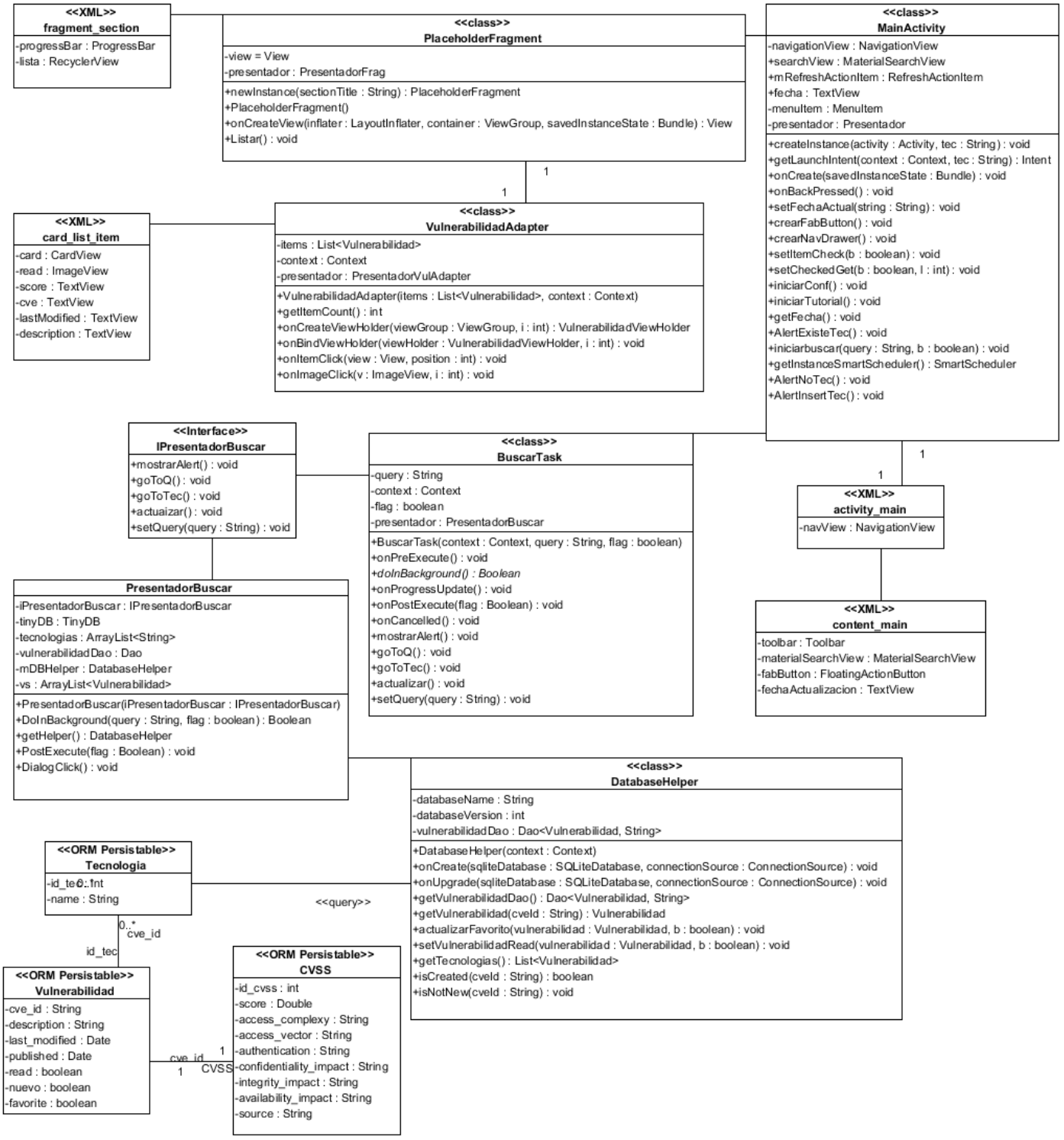


Fig. 18 Diagrama de clases de diseño: Buscar vulnerabilidad

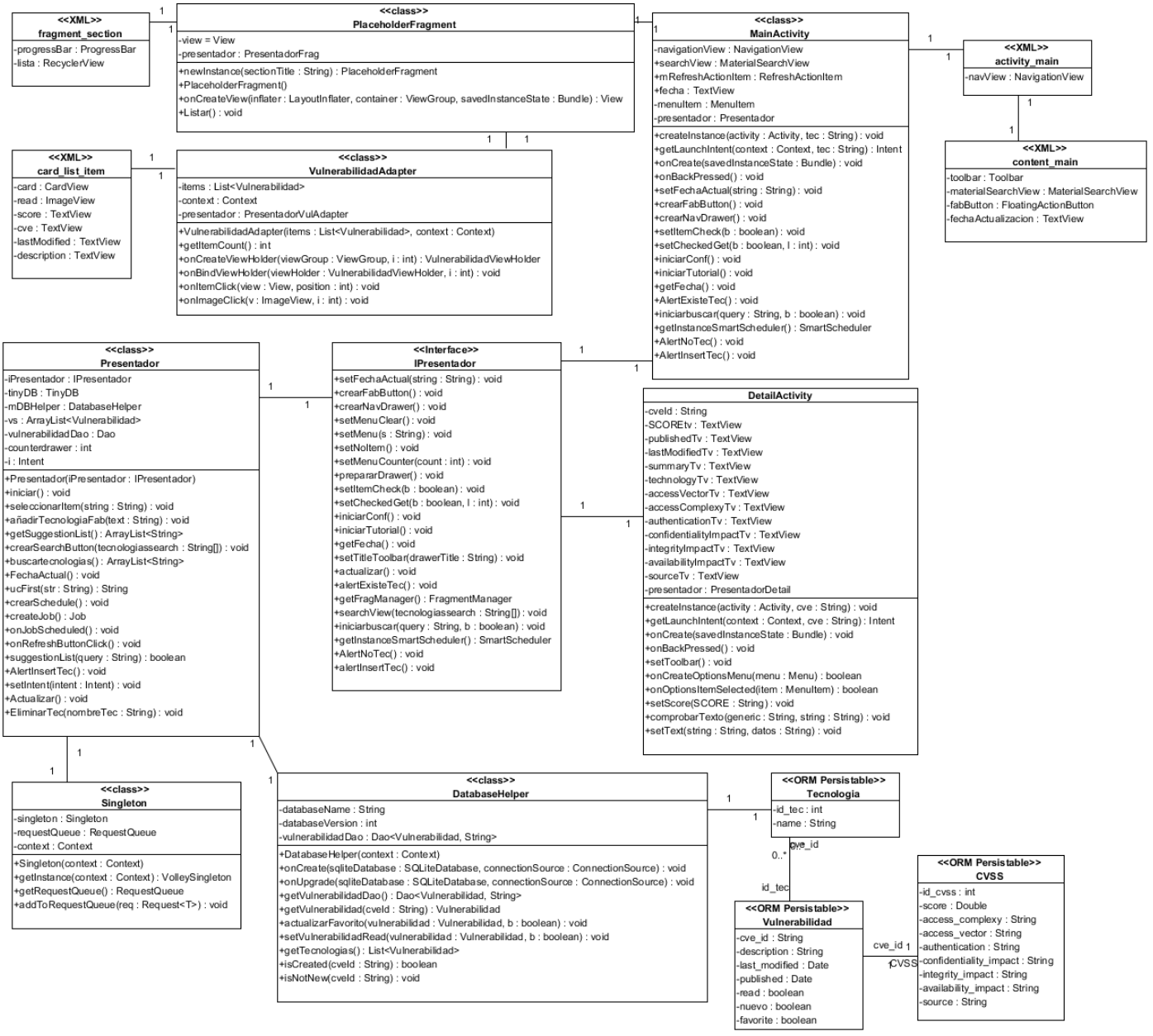


Fig. 19 Diagrama de clases de diseño: Mostrar vulnerabilidad

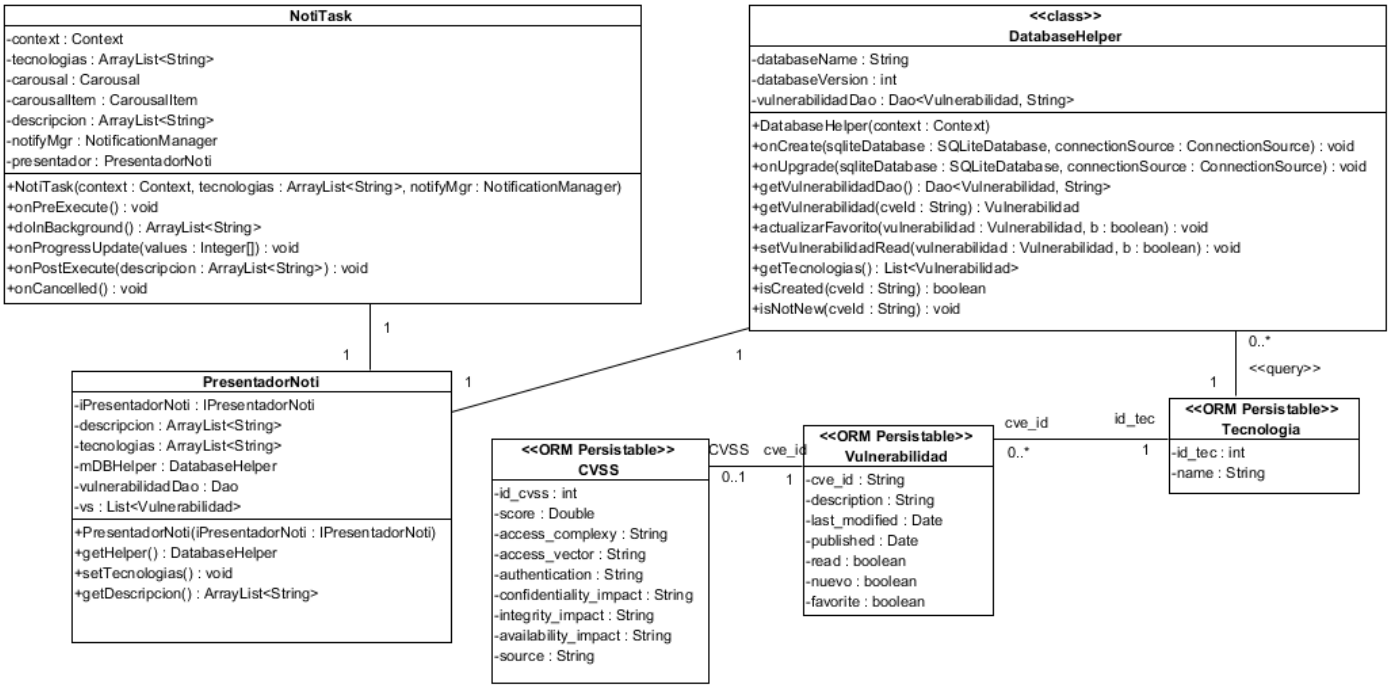


Fig. 20 Diagrama de clases de diseño: Notificar información de notificaciones

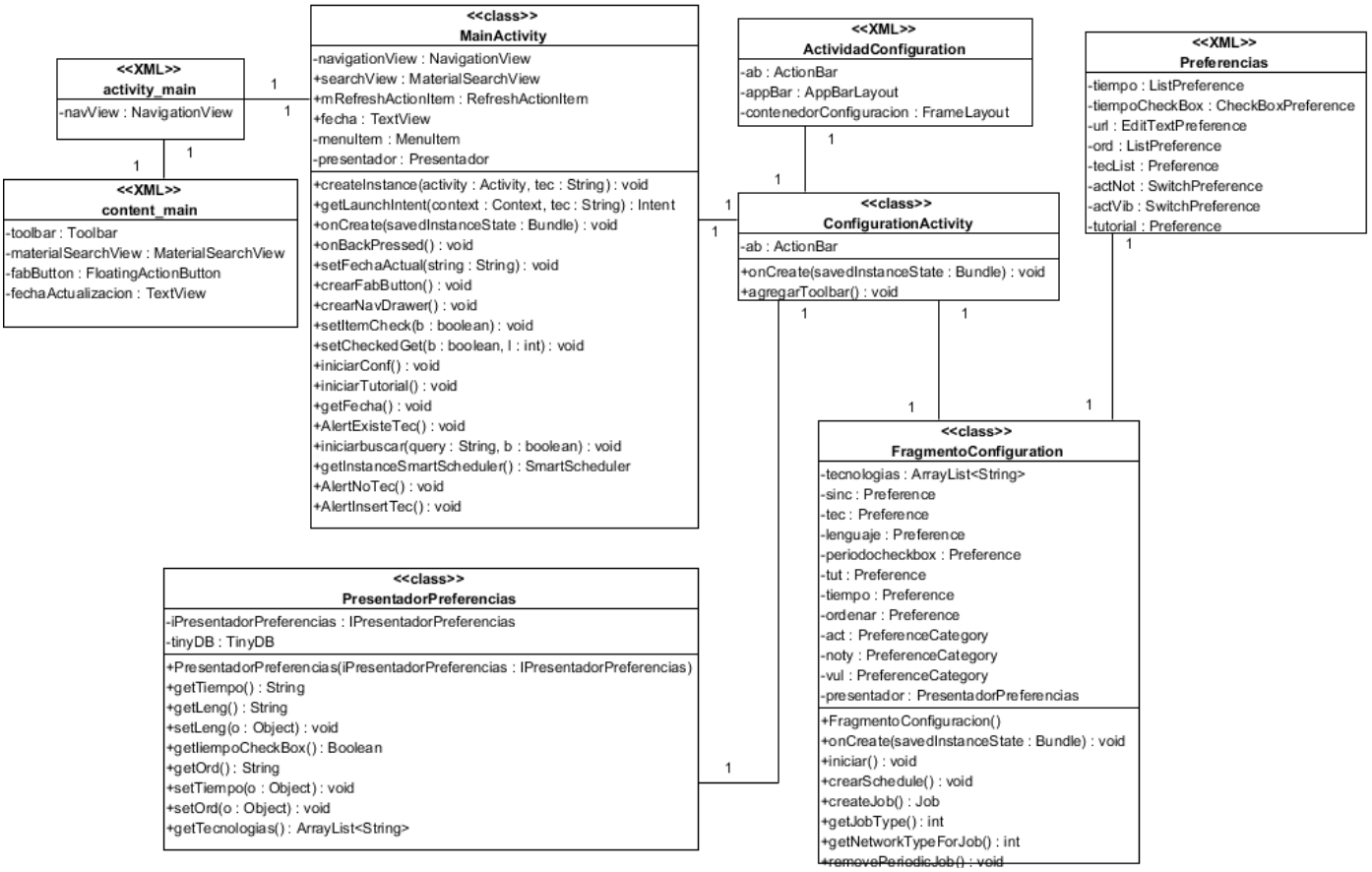


Fig. 21 Diagrama de clase de diseño: Configurar aplicación

Diagrama de la base de datos

Durante el diseño se identifican las clases persistentes y se representan en el diagrama de clases de diseño. Este muestra la estructura lógica de la base de datos mediante clases y cómo se relaciona con el resto de las clases de diseño. Para la elaboración de este diagrama se tuvo en cuenta los datos obtenidos por el servicio web:

1. **Nombre:** Listado de vulnerabilidades

Descripción: Proporciona los datos de todas las vulnerabilidades almacenadas en la base de datos.

URI: /api/cve/

Datos de entrada: Ninguno.

Datos devueltos por cada vulnerabilidad:

- **cve_id:** Identificador de la vulnerabilidad.
- **published:** Fecha de publicación de la vulnerabilidad.
- **last_modified:** Fecha de la última modificación de la vulnerabilidad.
- **sumary:** Descripción de la vulnerabilidad.

2. **Nombre:** Vulnerabilidades por palabras claves

Descripción: Proporciona los datos de las vulnerabilidades, a partir del nombre de su tecnología.

URI: /api/cve/summary/{nombre_tecnologia}

Datos de entrada: nombre_tecnologia: Nombre de la tecnología.

Datos devueltos por cada vulnerabilidad:

- **cve_id:** Identificador de la vulnerabilidad.
- **published:** Fecha de publicación de la vulnerabilidad.
- **last_modified:** Fecha de la última modificación de la vulnerabilidad.
- **sumary:** Descripción de la vulnerabilidad.

3. **Nombre:** Vulnerabilidad por identificador

Descripción: Brinda los datos de una vulnerabilidad, a partir de su identificador CVE-ID.

URI: /api/cve/{cve_id}

Datos de entrada: cve_id: Identificador de la vulnerabilidad.

Datos devueltos de la vulnerabilidad:

- **cve_id:** Identificador de la vulnerabilidad.
- **published:** Fecha de publicación de la vulnerabilidad.
- **last_modified:** Fecha de la última modificación de la vulnerabilidad.
- **sumary:** Descripción de la vulnerabilidad.
- **cvss:** Valor CVSS otorgado por la fuente a esta vulnerabilidad.

CVSS es un objeto compuesto por otros valores que se describen a continuación:

- **score:** Puntuación de la vulnerabilidad.
- **access_vector:** Vector de acceso para explotar la vulnerabilidad.
- **access_complexy:** Complejidad de acceso para explotar la vulnerabilidad.
- **authentication:** Autenticación requerida para explotar la vulnerabilidad.
- **confidentiality_impact:** Impacto de confidencialidad si se explota la vulnerabilidad.
- **integrity_impact:** Impacto de integridad si se explota la vulnerabilidad.
- **availability_impact:** Impacto de disponibilidad si se explota la vulnerabilidad.
- **generated_on_datetime:** Momento en el que se almacenó la vulnerabilidad en el SIGEV.
- **source:** Fuente que registró la vulnerabilidad.

Un modelo de datos, llamado modelo de entidad relación, incorpora algunas de las semánticas más importantes sobre el mundo real. Este puede ser utilizado como base para la unificación de diferentes vistas de datos (Chen, 1988). A continuación se muestra el diagrama entidad relación de la base de datos generado a partir del diagrama de clases de diseño referente a la aplicación a desarrollar:

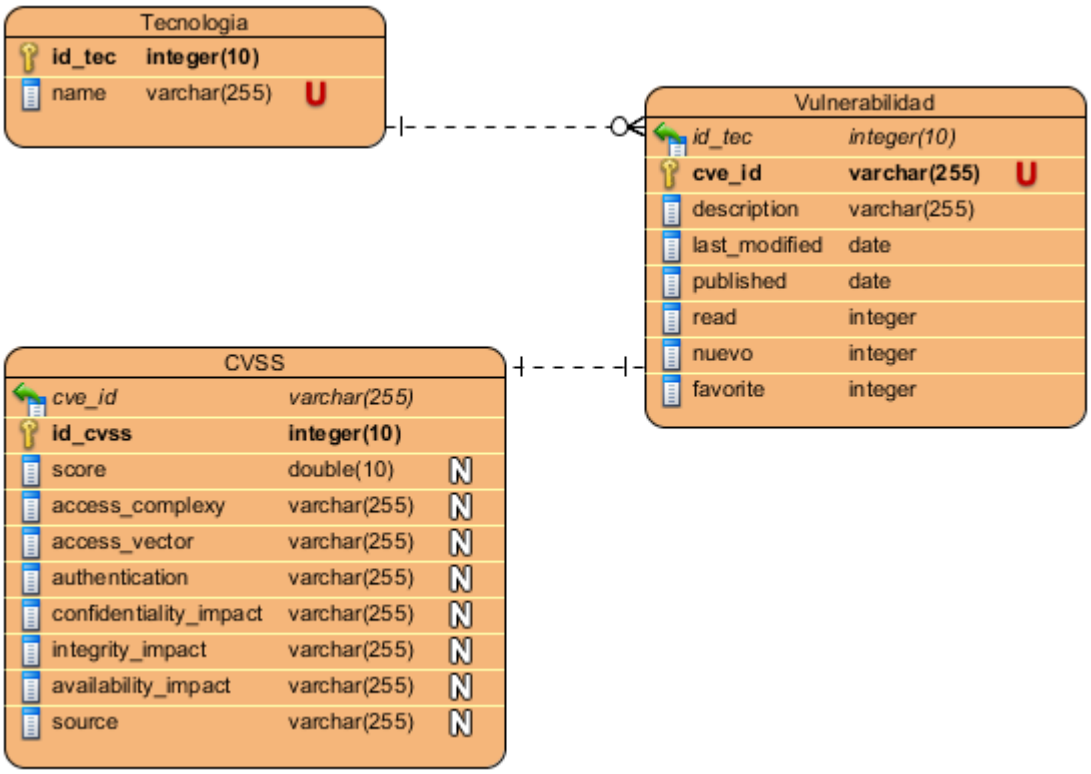


Fig. 22 Diagrama de entidad-relación

Diagrama de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la topología del sistema, las comunicaciones y la correspondencia entre los elementos ejecutables y los nodos (Larman, 2003).

El despliegue se realiza sobre un dispositivo móvil donde se instala aplicación Android y que debe tener conexión WIFI para la obtención de los datos del servicio web a través del protocolo HTTPS. Se representa este diagrama en la figura 23:

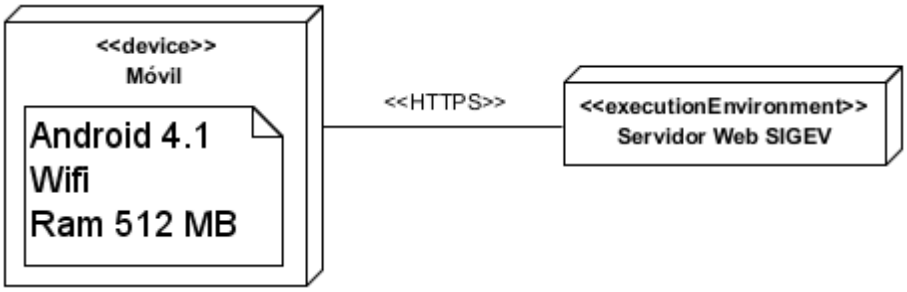


Fig. 23 Diagrama de despliegue de la propuesta solución.

Móvil: Representa el hardware y el software necesario donde se desplegará la aplicación Android. Este dispositivo requiere de una capacidad de 512 megabytes (MB) de RAM, sistema operativo 4.1 como mínimo y conexión Wifi para la obtención de los datos del servicio web.

Servidor web SIGEV: Sistema que provee los datos mediante el servicio web a través del protocolo HTTPS.

3.2 Implementación de la aplicación de gestión de información de vulnerabilidades

Después de completar el diseño de la solución se dispone de suficiente detalle para codificar la aplicación. Los productos de trabajo creados en el diseño se utilizan como entradas en el proceso de implementación junto a un conjunto de buenas prácticas de Android que se describen en el epígrafe 3.2.1 y otras referentes a la codificación y nomenclatura en Java.

3.2.1 Buenas prácticas para la implementación

Las buenas prácticas aseguran la calidad de la aplicación, visualización de la información y la interacción con el usuario. Además, se mejora el rendimiento y la estabilidad para administrar los recursos del dispositivo móvil inteligente.

Este conjunto de buenas prácticas es propuesto por Google definidas en (Android Developer 2017), y aunque no es un estándar constituye una guía en la que los desarrolladores se pueden apoyar para la implementación de aplicaciones Android. En la solución se aplican las siguientes:

Diseño estándar

- La aplicación no redefine la función prevista de un ícono del sistema.
- La aplicación no reemplaza un ícono del sistema con un ícono completamente diferente.
- La aplicación no redefine ni utiliza de forma inadecuada patrones de la interfaz de usuario de Android, de modo que los íconos o los comportamientos pudieran desorientar o confundir a los usuarios.
- Uso de diálogos para la confirmación de descargas y de eventos.
- Mostrar ayuda o tutorial en caso de que sea solicitado por el usuario o cuando la aplicación es ejecutada por primera vez.

Navegación

- La aplicación admite la navegación estándar del sistema con el botón Atrás y no utiliza avisos personalizados en pantalla para el botón Atrás.

Notificaciones

- Se agrupan múltiples notificaciones en un solo objeto de notificación.

- Las notificaciones solo son recurrentes si están relacionadas con eventos actuales.
- Las notificaciones no contienen publicidad ni contenido que no esté relacionado con la función central de la aplicación.

Permisos

- La aplicación solicita solo la cantidad mínima de permisos necesarios para respaldar la funcionalidad central.
- La aplicación no solicita permisos para acceder a datos confidenciales ni a servicios que puedan costarle dinero al usuario.

Interfaz gráfica

- La aplicación no utiliza rotación de pantalla para conservar una correcta visualización del contenido.

Estado del usuario o la aplicación

- La aplicación solo deja en ejecución el servicio de actualización en segundo plano para la obtención de datos y el envío de notificaciones.

Estabilidad

- La aplicación no falla, no impone el cierre, no se inmoviliza ni funciona de ningún otro modo anormal en ninguno de los dispositivos donde esté instalada.

Rendimiento

- La aplicación se carga en 5 segundos o le proporciona al usuario información en pantalla en caso de que demore más tiempo en cargarse.

Calidad visual

- La aplicación ofrece relación de imágenes para todos los tamaños de pantalla y formatos admitidos, incluidos aquellos para dispositivos con pantallas más grandes como las tabletas.
- No se observa distorsión en los bordes de los menús, los botones ni otros elementos de la interfaz de usuario.
- Se acepta la composición en todos los formatos de textos admitidos, incluidos aquellos para dispositivos con pantallas más grandes como las tabletas.
- No se visualizan ajustes automáticos de línea incorrectos en botones ni íconos.
- Existe espacio entre el texto y los elementos que lo rodean para su visualización.

3.2.2 Estándares de nomenclatura y codificación utilizados

Los estándares de codificación son un conjunto de directrices, normas y reglamentos enfocados a la especificación de cómo debe escribirse el código fuente de la aplicación. Estos incluyen pautas sobre la nomenclatura de las variables, clases y paquetes; la correcta indentación del código, cómo escribir estructuras de control, entre otros aspectos. La correcta elaboración y utilización de los estándares de codificación permiten que todos los desarrolladores implementen siguiendo las mismas pautas y así puedan entender el código del resto como si estuvieran mirando el de ellos, así la aplicación será más legible, uniforme y fácil de mantener (Vermeulen, 2001).

Nomenclatura general

- El nombre de todas las variables y métodos comenzarán con letra minúscula. Si está compuesto por varias palabras se utilizará el estilo de escritura lowerCamelCase, que la palabra inicial comienza con minúscula, pero todas las palabras internas se escriben con inicial mayúscula. Ejemplo: navigationView

Paquetes

- Por defecto todos los paquetes se escriben en minúsculas y sin utilizar caracteres especiales. El paquete base queda definido como uci.cu y en él no se definirá ninguna clase.
- El nivel extra se identifica dentro del paquete anterior con el nombre del proyecto o de la capa. Ejemplo: vulnerapp.uci.cu

Indentación

- En el contenido siempre se indenta con tabulaciones, nunca utilizando espacios en blanco.

Clases

- El nombre de las clases comienza con mayúsculas y cada salto de palabras debe iniciar con mayúsculas. Ejemplo: MainActivity.java

Estilos de codificación

Comentarios

- Los comentarios deben añadir claridad al código. Deben contar el por qué y no el cómo.
- Deben ser concisos.
- Se debe escribir la documentación antes de desarrollar el código.

Sentencias

- Una sentencia por línea de código.
- Todo bloque de sentencias entre llaves, aunque sea una sola sentencia después de un if.

3.3 Pruebas

Una etapa fundamental en el ciclo de desarrollo del software es la etapa de las pruebas, proceso que permite verificar y revelar la calidad del producto. La metodología AUP variación UCI define tres pruebas, estas son, pruebas internas, pruebas de liberación y pruebas de aceptación (UCI, 2015).

A continuación se muestran los resultados de las pruebas internas y las pruebas de aceptación. No se realizarán las pruebas de liberación debido a que estas son emitidas por entidades de certificación de calidad.

3.3.1 Pruebas internas

Las pruebas internas de software se realizarán mediante pruebas unitarias. Las pruebas unitarias son la verificación de una unidad de código del software.

En esta investigación se realizan las pruebas internas a través del entorno de trabajo JUnit4 que está integrado con el SDK de Android. Este entorno permite realizar la ejecución de clases de Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como es esperado. Todo el código debe tener pruebas unitarias y retornar resultados satisfactorios antes de realizar las pruebas de aceptación con el cliente (Jing, 2012). A continuación se muestra el resultado de una comprobación de las pruebas unitarias aplicadas a todo el código de la aplicación:

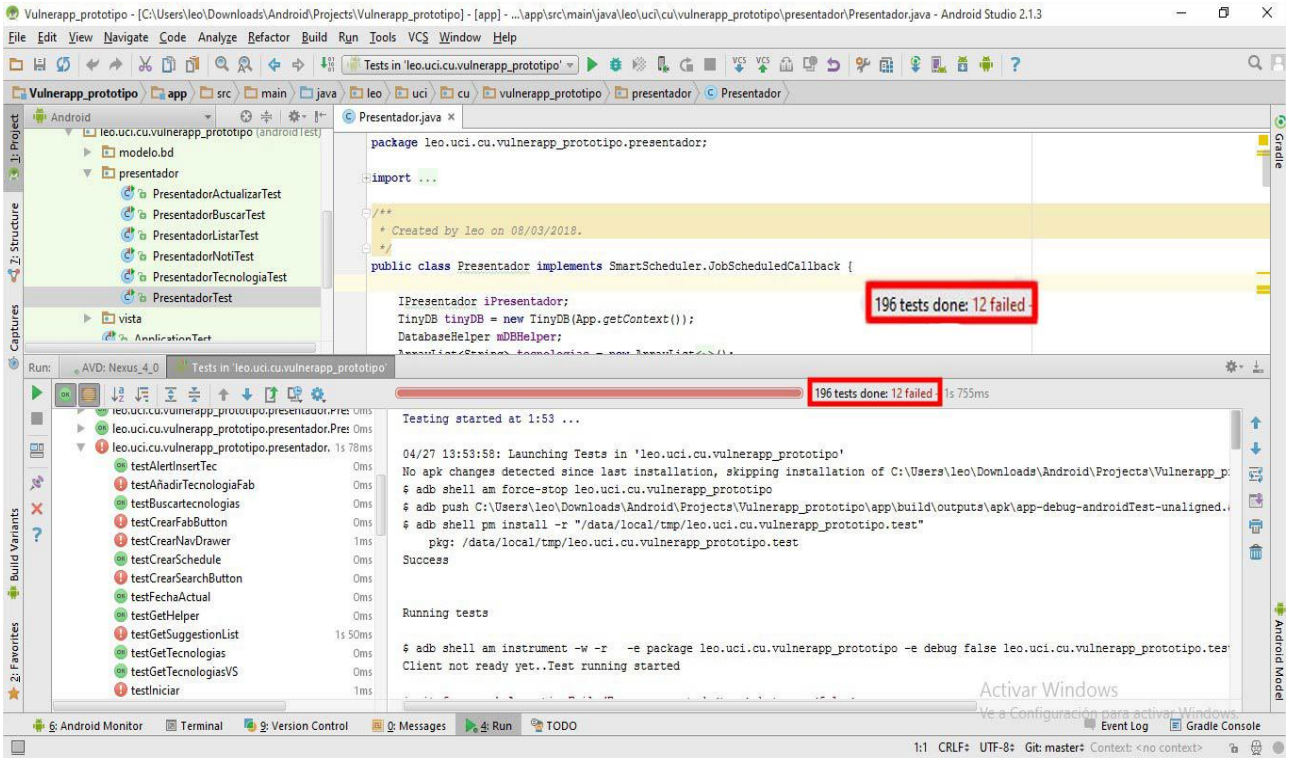


Fig. 24 Primera comprobación de pruebas unitarias.

En una primera comprobación se arrojaron los resultados de 12 pruebas fallidas relacionadas con:

- Validación de entrada de datos.
- Actualizar datos en la base de datos con elementos vacíos.
- No existía una validación al no existir conexión Wifi.
- No se actualizaba la lista al eliminar las tecnologías.

Se realizan actualizaciones al código para reparar estas deficiencias y se procedió a realizar otra comprobación:

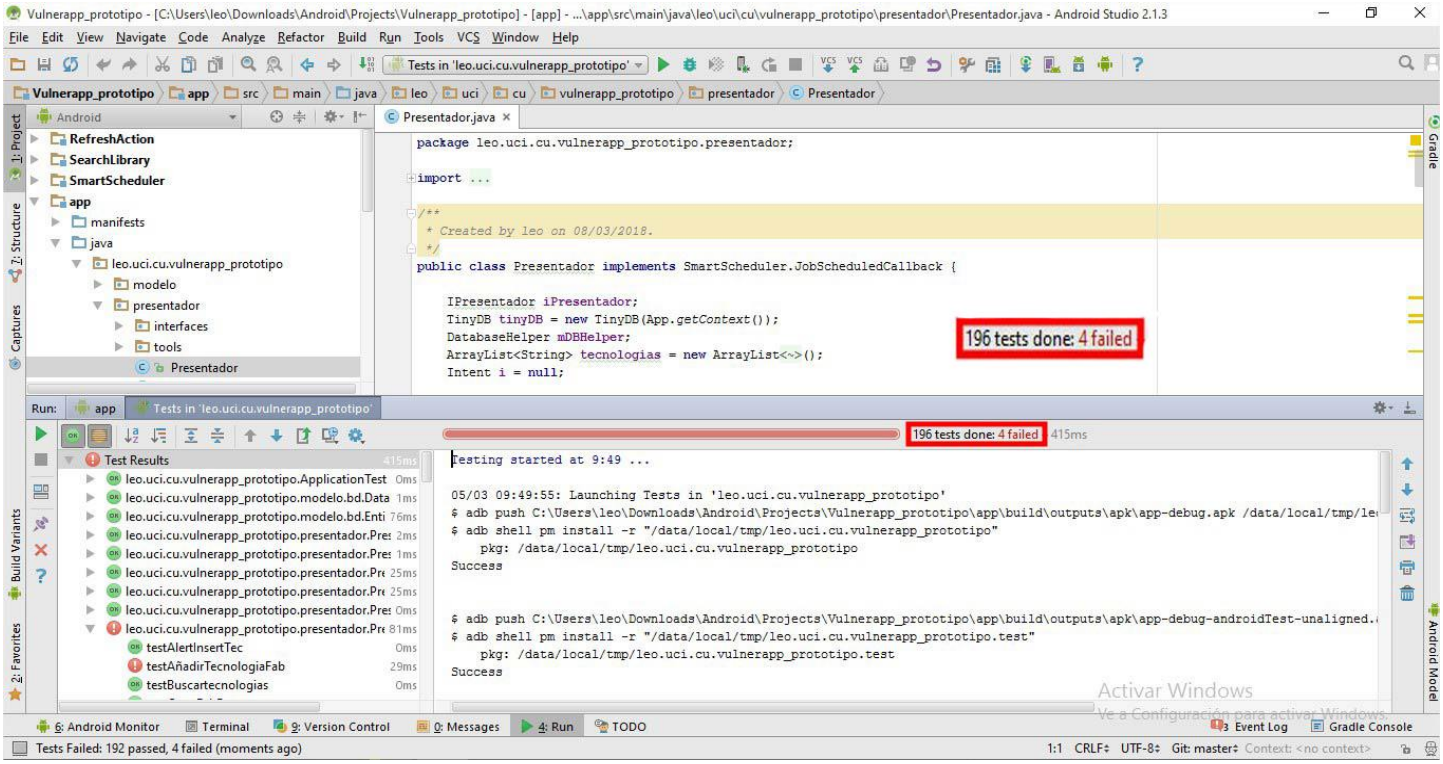


Fig. 25 Segunda comprobación de pruebas unitarias

Se detectaron en la segunda comprobación 4 pruebas fallidas, detectándose errores como los siguientes:

- No se pueden realizar una búsqueda de vulnerabilidades mientras se producía una actualización.
- No se notifican las actualizaciones de las vulnerabilidades.

Luego de reparar las inconformidades anteriores se procedió a realizar una nueva comprobación:

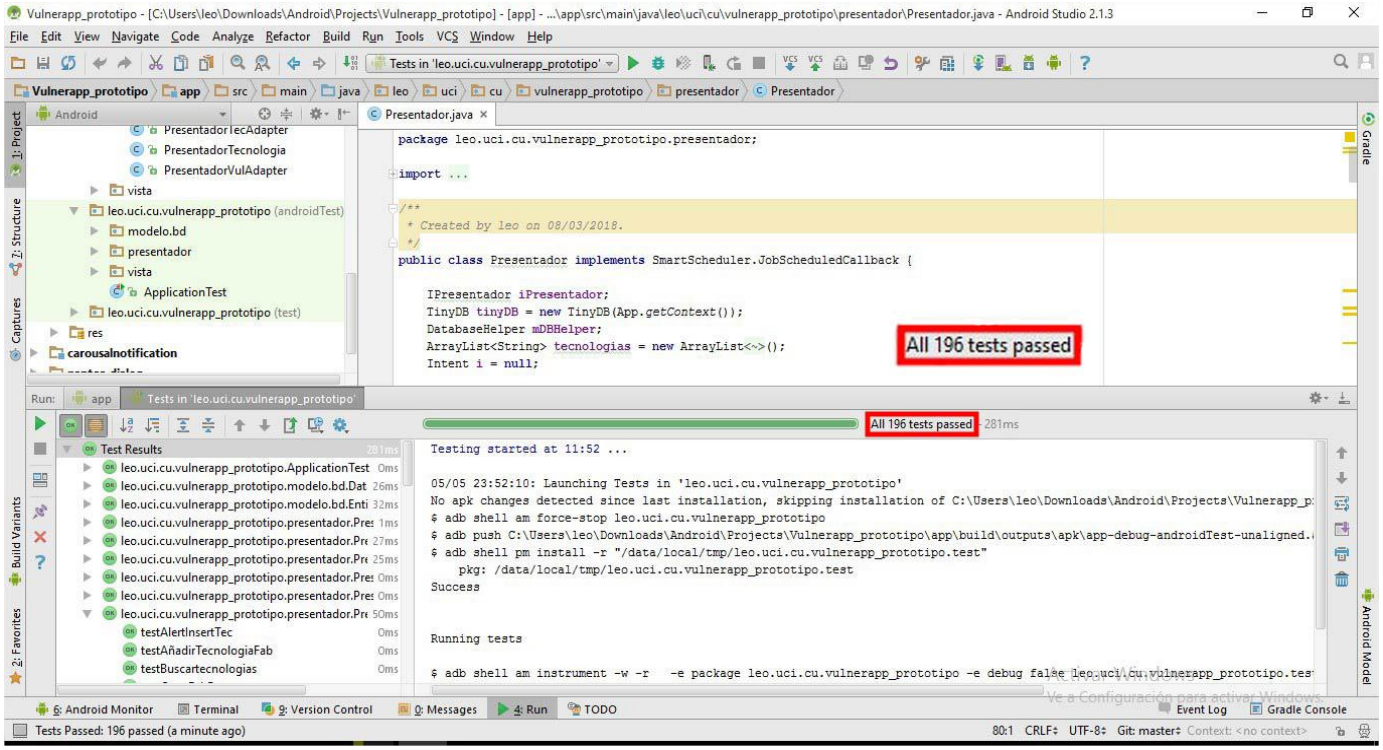


Fig. 26 Tercera comprobación de las pruebas unitarias.

En esta última comprobación la aplicación supera todas las pruebas satisfactoriamente, determinando que todos sus componentes individualmente funcionan correctamente. Se procede a realizar las pruebas de aceptación con el cliente.

3.3.2 Pruebas de aceptación

Las pruebas de aceptación son determinadas por el cliente y se centran en los requisitos funcionales. Son del tipo de pruebas de caja negra y se realiza mediante el diseño de casos de pruebas a partir de los casos de uso del sistema elaborados anteriormente.

El objetivo de estas pruebas es verificar los requisitos, por este motivo, los propios requisitos del sistema son la principal fuente de información para construir las pruebas de aceptación (Pressman, 2014).

A continuación se muestran los casos de prueba de aceptación correspondientes a los casos de usos:

Tabla.9 Diseño de caso de prueba del caso de uso Gestionar tecnología

Nombre de la sección	Escenarios de la sección	Descripción de la sección	Flujo central	Resultados esperados	Resultados obtenidos
SC 1: Adicionar una tecnología	**EC 1.1: Adicionar tecnología**	El usuario presiona el botón Adicionar tecnología. El sistema muestra un diálogo, el usuario entra el nombre da la tecnología y pulsa aceptar	Gestionar tecnología/ Adicionar tecnología	El sistema debe mostrar un símbolo con la cantidad de vulnerabilidades descargadas.	El sistema muestra un símbolo con la cantidad de vulnerabilidades descargadas.
	EC 1.2: Adicionar incorrectamente	El usuario presiona el botón Adicionar tecnología. El sistema muestra un diálogo, el usuario introduce el nombre de la tecnología conteniendo un signo de pregunta.	Gestionar tecnología/ Adicionar tecnología/ Adicionar incorrectamente	El sistema debe mostrar un mensaje notificando que el nombre de la tecnología no se introdujo correctamente.	El sistema muestra un mensaje notificando que el nombre de la tecnología no se introdujo correctamente.
	EC 1.3: Adicionar tecnología existente	El usuario presiona el botón Adicionar tecnología. El sistema muestra un diálogo, el usuario introduce el nombre de una tecnología existente.	Gestionar tecnología/ Adicionar tecnología/ Adicionar tecnología existente	El sistema debe mostrar un mensaje notificando que la tecnología se encuentra adicionada.	El sistema muestra un mensaje notificando que la tecnología se encuentra adicionada.
SC 2 : Actualizar tecnologías	**EC 2.1: Actualizar tecnologías**	El usuario presiona el botón Actualizar tecnologías.	Gestionar tecnología/ Actualizar tecnologías	El sistema debe mostrar un símbolo con el número de vulnerabilidades descargadas.	El sistema muestra un símbolo con el número de vulnerabilidades descargadas

	EC 2.2: Actualizar sin conexión	El usuario presiona el botón Actualizar tecnologías sin existir conexión con el servicio web.	Gestionar tecnología/ Actualizar tecnologías/ Actualizar sin conexión	El sistema debe mostrar un símbolo de error.	El sistema muestra un símbolo de error.
	EC 2.3: Actualizar sin tecnologías	El usuario presiona el botón Actualizar tecnologías sin existir tecnologías adicionadas.	Gestionar tecnología/ Actualizar tecnologías/ Actualizar sin tecnologías	El sistema debe mostrar un mensaje notificando que se debe adicionar una tecnología para actualizar.	El sistema muestra un mensaje notificando que se debe adicionar una tecnología para actualizar.
SC 3: Eliminar tecnología	**EC 3.1: Eliminar tecnología**	El usuario presiona el botón Eliminar tecnología, el sistema despliega un diálogo de confirmación y el usuario presiona el botón Aceptar.	Gestionar tecnología/ Eliminar tecnología	El sistema debe eliminar de la base de datos la tecnología y todas las vulnerabilidades asociadas, así como cualquier actualización de la tecnología y mostrar la pantalla “Inicio”.	El sistema elimina de la base de datos la tecnología y todas las vulnerabilidades asociadas, así como cualquier actualización de la tecnología y muestra la pantalla “Inicio”.

Tabla.10 Diseño de caso de prueba del caso de uso Buscar vulnerabilidad

Nombre de la sección	Escenarios de la sección	Descripción de la sección	Flujo central	Resultados esperados	Resultados obtenidos
SC 1: Buscar vulnerabilidad	**EC 1.1: Buscar vulnerabilidades**	El usuario inserta una búsqueda.	Buscar/Buscar vulnerabilidades	El sistema muestra un listado de vulnerabilidades como resultado de la búsqueda.	El sistema muestra un listado de vulnerabilidades como resultado de la búsqueda.
	Buscar vulnerabilidades y no existen resultados	El usuario inserta una búsqueda.	Buscar/Buscar vulnerabilidades y no existen resultados	El sistema envía un mensaje notificando que no existen resultados.	El sistema envía un mensaje notificando que no existen resultados.

Tabla.11 Diseño de caso de prueba del caso de uso Mostrar vulnerabilidad

Nombre de la sección	Escenarios de la sección	Descripción de la sección	Flujo central	Resultados esperados	Resultados obtenidos
SC 1: Mostrar vulnerabilidad	**EC 1.1: Mostrar vulnerabilidad**	El usuario selecciona la vulnerabilidad que desea visualizar.	Mostrar vulnerabilidad	El sistema muestra la vista detallada de la vulnerabilidad.	El sistema muestra la vista detallada de la vulnerabilidad.

Tabla 12 Diseño de caso de prueba del caso de uso: Notificar información de actualizaciones

Nombre de la sección	**Escenarios de la sección**	**Descripción de la sección**	**Flujo central**	**Resultados esperados**	**Resultados obtenidos**
SC 1: Notificar información de actualizaciones	**EC 1.1: Notificar con conexión**	El sistema se sincroniza con el servicio web.	Notificar/Notificar con conexión	El sistema muestra una notificación sobre el registro de nuevas vulnerabilidades.	El sistema muestra una notificación sobre el registro de nuevas vulnerabilidades.
	EC 1.1: Notificar sin conexión	El sistema no obtiene conexión con el servicio web	Notificar/Notificar sin conexión	El sistema muestra una notificación informando que no existe conexión con el servicio web.	El sistema muestra una notificación informando que no existe conexión con el servicio web.

Tabla 13 Diseño del caso de prueba: Configurar aplicación

Nombre de la sección	**Escenarios de la sección**	**Descripción de la sección**	**Flujo central**	**Resultados esperados**	**Resultados obtenidos**
SC 1: Configurar aplicación	**EC 1.1: Configurar aplicación**	El usuario selecciona la opción de configurar la aplicación. El usuario modifica las preferencias en la configuración.	Configurar aplicación	El sistema muestra en la vista de configuración los cambios realizados en las preferencias.	El sistema muestra en la vista de configuración los cambios realizados en las preferencias.

Como resultado de las pruebas de aceptación se obtuvo los resultados esperados determinando que la aplicación funciona correctamente.

3.4 Conclusiones del capítulo

Se seleccionó la arquitectura de desarrollo MVP, que, al permitir separar la lógica de la interfaz, facilita las modificaciones que se necesiten en la aplicación.

El empleo de los patrones de diseño seleccionados: GRASP y GoF, fortalece la estructura, el comportamiento y la creación de instancias al proporcionarle cohesión, independencia, reutilización y facilidad de mantenimiento.

Con la determinación de buenas prácticas, tanto para la implementación en Android como de nomenclatura y codificación se logra un código más robusto y que responda a los requisitos no funcionales de la aplicación.

El despliegue de la aplicación se realiza sobre un dispositivo móvil inteligente que satisfaga los requisitos de hardware y software definidos, sin necesitar otras demandas.

Para la validación de la aplicación se necesitan 3 comprobaciones de pruebas internas y la realización de pruebas de aceptación. Ambas arrojaron resultados favorables que concluyen que el sistema cumple con los requisitos esperados.

Conclusiones

Al término del trabajo de diploma se afirma que se cumplió el objetivo general de la investigación y se concluye que:

- El estudio sobre la gestión de vulnerabilidades, los estándares que se utilizan internacionalmente y fundamentan la aplicación, permiten esclarecer un conjunto de características que debe tener el software como que: sea desarrollada en Android como una aplicación nativa, que se emplee el estándar CVE, que se empleen herramientas y tecnologías propuestas por Google y la metodología de desarrollo empleada en el proceso de desarrollo de software de la UCI.
- El análisis del negocio da paso a la definición de los requisitos funcionales y no funcionales de la aplicación que sirven de base para el diseño de la estructura del software y para la validación, luego de implementado.
- Las interfaces de la aplicación se diseñan respondiendo a los requisitos no funcionales y son aceptadas por el usuario durante la etapa de pruebas.
- Los resultados de la validación conducen a que la aplicación es funcional y que soluciona la problemática planteada en el diseño de la investigación.
- Se obtiene una aplicación apk, que se conecta al servicio web del SIGEV para gestionar la información de las vulnerabilidades y proveer a los usuarios de portabilidad para su consulta.

Recomendaciones

Teniendo en cuenta el estudio realizado durante el proceso de desarrollo de software de la presente investigación y con el objetivo de enriquecer la solución el autor recomienda las siguientes acciones:

- Permitir la búsqueda de vulnerabilidades en el servicio web mientras se mantiene la conexión de red para que la aplicación pueda proveer la funcionalidad de navegación general entre vulnerabilidades, aspecto que no fue previsto en el alcance del proyecto actual.
- Incorporar la exportación e importación de la base de datos para su transmisión entre dispositivos, de manera tal que puedan compartir la información sobre las vulnerabilidades sin requerir la conexión al servicio del Sistema de Gestión de Vulnerabilidades.

Referencias bibliográficas

AKAMAI, 2018. Akamai *vulnerability management*. [en línea]. [Consulta: 25 noviembre 2017]. Disponible en: https://www.akamai.com/es/es/resources/vulnerability-management.jsp

ALONSO, 2011 Arturo Baz, et al. Dispositivos móviles. EPSIG Ing. Telecomunicación Universidad de Oviedo, 2011.

ANDROID, 2017. Android - Historia. [en línea]. [Consulta: 29 noviembre 2017]. Disponible en: https://www.android.com/intl/es_es/history/#/donut.

ANDROID DEVELOPER, 2017. Conoce Android Studio | Android Studio. [en línea]. [Consulta: 5 enero 2018]. Disponible en: https://developer.android.com/studio/intro/index.html.

ANDROID OPEN SOURCE PROJECT, 2017. Android Open Source Project. [en línea]. [Consulta: 23 enero 2018]. Disponible en: https://source.android.com/.

AYAR, Muhammed Emin, 2016. *Review of Panter Dialog* GitHub. [en línea]. [Consulta: 15 febrero 2018]. Disponible en: https://github.com/kngfrhzs/panter-dialog

BRAHLER, 2010 Stefan. Analysis of the android architecture. *Karlsruhe institute for technology*, 2010, vol. 7, no 8.

CATALAN, Miguel 2015. *Review of Material SearchView* GitHub. [en línea]. [Consulta: 16 febrero 2018]. Disponible en: https://github.com/MiguelCatalan/MaterialSearchView

CASTILLO-PÉREZ, D. Sergio, 2011. 5. VULNERABILIDADES Y SOFTWARE MALICIOSO. En II Congreso sobre las Nuevas Tecnologías y sus repercusiones en el seguro: Internet, Biotecnología y Nanotecnología. Barcelona, España. 2011. p. 5.

CHANCHÍ, Gabriel E., 2011, et al. Esquema de servicios para Televisión Digital Interactiva, basados en el protocolo REST-JSON. *Cuadernos de Informática*, 2011, vol. 6, no 1, p. 233-240.

CHANDRA, Vishal, 2015. Comparison between Various Software Development Methodologies. *International Journal of Computer Applications*, 2015, vol. 131, no 9, p. 7-10.

CHARLAND, Andre, 2011; LEROUX, Brian. *Mobile application development: web vs. native. Communications of the ACM*, 2011, vol. 54, no 5, p. 49-53.

CHEN, Peter Pin-Shan, 1988. The entity-relationship model—toward a unified view of data. En *Readings in artificial intelligence and databases*. 1988. p. 98-111.

CVE, 2017 Common Vulnerabilities. Exposures [en línea]. [Consulta: 25 noviembre 2017]. Disponible en: http://cve.mitre.org/

DEL BUSTO, Hansel Gracia; ENRIQUEZ, Osmel Yanes, 2012. Mapeo Objeto/Relacional (ORM). *Revista Telem@ tica*, 2012, vol. 10, no 3, p. 1-7.

FIRST, 2017. *CVSS V3.0 Specification Document* [en línea]. [Consulta: 8 diciembre 2017]. Disponible en: https://www.first.org/cvss/cvss-v30-specification-v1.8.pdf

FLORATOU, Avrilia; OZCAN, Fatma, 2018. *Transforming an ontology query to an sql query*. U.S. Patent Application No 15/213,227, 18 Ene. 2018

FRIED, I., 2010. Steve Jobs at D8: *Post-PC era is nigh* - CNET. [en línea]. [Consulta: 26 noviembre 2017]. Disponible en: https://www.cnet.com/news/steve-jobs-at-d8-post-pc-era-is-nigh/.

FRIESEN, Jeff, 2012. *Beginning Java 7*. Apress, 2012.

GAMMA, Helm, Johnson, Vlissides, 2003. Patrones de diseño. Elementos de software orientado a objetos reutilizable. Madrid - Pearson Educación, 2003.

GARCÍA, Cristina, 2016. *Reputation management of an Open Source* Software *system based on the trustworthiness of its contributions*. 2016.

GOOGLE, Inc. 2008. Github Gson. [en línea]. [Consulta: 8 enero 2018]. Disponible en: https://github.com/google/gson

GUANOLUISA, Carrera; GERMANIA, Jenny, 2014. *Análisis comparativo de la productividad entre los patrones de dieño Modelo Vista Controlador (MVC) y Modelo Vista Presentador (MVP) aplicado al desarrollo del Sistema Nómina de Empleados y Rol de Pagos de la Distribuidora Soria CA*. 2014. Tesis de Licenciatura.

HUANG, Huai-Ying, 2015, et al. *Stable SRAM cell*. U.S. Patent No 8,947,900, 3 Feb. 2015.

JACOBSON, Ivar; BYLUND, Stefan, 2000. *The road to the unified* software *development process*. Cambridge University Press, 2000.

JING, Yiming; AHN, Gail-Joon; HU, Hongxin, 2012. Model-based conformance testing for android. En *International Workshop on Security*. Springer, Berlin, Heidelberg, 2012. p. 1-18.

KHALID, Asra; ZAHRA, Sobia; KHAN, Muhammad Fahad, 2014. Suitability and contribution of agile methods in mobile software development. *International Journal of Modern Education and Computer Science*, 2014, vol. 6, no 2, p. 56.

KCOCHIBILI, 2018. *Review of TinyDB GitHub*. [en línea]. [Consulta: 6 febrero 2018]. Disponible en: https://github.com/kcochibili/TinyDB--Android-Shared-Preferences-Turbo.

LABRADA MARTÍNEZ, Esther; SALGADO CEBALLOS, Cristina, 2013. *Diseño web adaptativo o responsivo*. *Tema del mes*, 2013.

LARMAN, Craig, 2003. *UML y Patrones*. ^ eMadrid Madrid: Pearson Educación, 2003.

LÓPEZ FERNÁNDEZ José Luis, 2018. *Análisis y gestión de vulnerabilidades de sistemas informáticos con* software *libre,* 2018*.*

MAMGAINN, Shailesh, 2017 *Review of Carousal Notification GitHub*. [en línea]. [Consulta: 6 febrero 2018]. Disponible en: https://github.com/shaileshmamgain5/Carousel-Notification

MEENAKSHI, Sanjiv Kumar Singh, 2015, et al. Implementing Java Distributed Objects. *International Journal Of Engineering And Computer Science*, 2015, vol. 4, no 06.

MELL, Peter; SCARFONE, Karen; ROMANOSKY, Sasha, 2006. *Common vulnerability scoring system*. *IEEE Security & Privacy*, 2006, vol. 4, no 6.

NVD, 2018. *General information* - NVD. [en línea]. [Consulta: 23 noviembre 2017]. Disponible en: https://nvd.nist.gov/general

ORMLITE, 2017. OrmLite - Lightweight Object Relational Mapping (ORM) Java Package. [en línea]. [Consulta: 8 febrero 2018]. Disponible en: http://ormlite.com/.

PARK, Jin Hyung, 2018, et al. Security Architecture for a Secure Database on Android. *IEEE Access*, 2018, vol. 6, p. 11482-11501.

PEINADO, Manuel, 2013. *Review of Refresh Action Item* GitHub. [en línea]. [Consulta: 23 enero 2018]. Disponible en: https://github.com/ManuelPeinado/RefreshActionItem

PRESSMAN, Roger S, 2014. Ingeniería de Software. Un enfoque práctico. 2014.

ROUSE, M., 2016. *What is integrated development environment (IDE)? - Definition from* WhatIs.com. [en línea]. [Consulta: 6 diciembre 2017]. Disponible en: http://search*software*quality.techtarget.com/definition/integrated-development-environment.

SHENG, Alex Liu, 2016. *Review of BadgeView* GitHub. [en línea]. [Consulta: 4 febrero 2018]. Disponible en: https://github.com/AlexLiuSheng/BadgeView

SUÁREZ, R., 2015. *Sobre los avances y retos de la informatización de la sociedad - Cubadebate*. [en línea].

[Consulta: 25 noviembre 2017]. Disponible en: http://www.cubadebate.cu/noticias/2015/09/18/avances-retos-informatizacion-sociedad/.

TICWEB, 2017. *Los dispositivos móviles y su incidencia en nuestra vida cotidianaTIC's en la Web*. [en línea]. [Consulta: 25 noviembre 2017]. Disponible en: https://www.ticweb.es/los-dispositivos-moviles-y-su-incidencia-en-nuestra-vida-cotidiana/.

UCI, 2015. *Metodología de desarrollo para la actividad productiva de la UCI*. Universidad de las Ciencias Informáticas. La Habana : s.n.,2015 pág 16.

VERMEULEN, Allan, et al, 2001 *The Elements of Java (TM) Style*. Cambridge University Press, 2001. ISBN:0521777682.

VISUAL PARADIGM, 2014. *Visual Paradigm*. [en línea]. [Consulta: 26 noviembre 2017]. Disponible en: https://www.visual-paradigm.com/.

WANG, Xun; ZHANG, Xu, 2017. *Method and device for extracting characteristic code of APK virus*. U.S. Patent No 9,600,668, 21 Mar. 2017.

WITTCHEN, Piotr, 2016. *Review of Infinite Scroll Listener GitHub*. [en línea]. [Consulta: 12 febrero 2018]. Disponible en: https://github.com/pwittchen/InfiniteScroll.

Bibliografías

ARAYA SOLÍS, Gloriana; MÉNDEZ MARÍN, Geovanny; JIMÉNEZ SEGURA, Ronald, 2014. Pruebas de software para dispositivos móviles android. 2014.

BALAGUERA, Yohn Daniel Amaya, 2013. Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual. *Revista de Tecnología*, 2013, vol. 12, no 2, p. 111-123.

BRAY Tim, 1997, et al. Extensible markup language (XML). *World Wide Web Journal*, 1997, vol. 2, no 4, p. 27-66.

CROCKFORD, Douglas, 2006. The application/json media type for javascript object notation (json). 2006.

CUEVAS LÓPEZ, Sandra, 2017. *Herramienta de gestión de vulnerabilidades en sistemas*. 2017. Tesis de Licenciatura.

DABBISH, Laura, 2012, et al. *Social Coding in* GitHub: *Transparency and Collaboration in an Open* Software *Repository*. 2012.

ESPADA, Jordán Pascual, 2011, et al. Virtual objects on the internet of things. *IJIMAI*, 2011, vol. 1, no 4, p. 23-29.

FIELDING, Roy, 1999, et al. *Hypertext transfer protocol--HTTP/1.1*. 1999.

FORTES ALAYÓN, José, 2016. API de servicios web REST extensible dinámicamente basada en una arquitectura Plug-in para el procesado de imágenes recibidas desde dispositivos móviles. 2016.

FREIER, Alan; KARLTON, Philip; KOCHER, Paul, 2011. The secure sockets layer (SSL) protocol version 3.0. 2011.

GIRONÉS, Jesús Tomás, 2012. *El gran libro de Android*. Marcombo, 2012.

HERNÁNDEZ SAMPIERI, Roberto; FERNÁNDEZ COLLADO, Carlos; BAPTISTA LUCIO, Pilar, 2014. *Metodología de la investigación*. Sexta Edición. Editorial Mc Graw Hill. México. 2014• Hernández, R. Metodología de la Investigación. 6a Edición, Mc Graw Hill, México, 2014.

IBM, 2017 *Definition of* Software *Stack*. [en línea]. [Consulta: 4 diciembre 2017]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SS2GNX_7.2.2/com.ibm.tivoli.tpm.scenario.doc/software/csfm_sftstack.html

JACKSON, Wallace, 2012. Android Framework Overview. En *Android Apps for Absolute Beginners*. *Apress, Berkeley, CA, 2012. p. 99-123.*

LABRADA MARTÍNEZ, Esther; SALGADO CEBALLOS, Cristina, 2013. *Diseño web adaptativo o responsivo*. *Tema del mes*, 2013.

LÓPEZ FERNÁNDEZ, José Luis, 2018. Análisis y gestión de vulnerabilidades de sistemas informáticos con software libre, 2018.

MORA, Diego Leonardo Arias; CARVAJAL, Jean Paul Barquero; ÁLVAREZ, María José Fonseca, 2017. Metodologías de desarrollo de software.

MULLER, Hausi A.; NORMAN, Ronald J.; SLONIM, Jacob (ed.), 2012. *Computer Aided* Software *Engineering*. Springer Science & Business Media, 2012.

PALACIOS AGUILAR, Pedro, 2015, et al. Propuesta de implementación de un marco de trabajo para el desarrollo de aplicaciones Android. 2015.

POLANCO, Kristel Malave; TAIBO, José Luis Beauperthuy, 2011. "ANDROID" EL SISTEMA OPERATIVO DE GOOGLE PARA DISPOSITIVOS MÓVILES. *Revista Negotium*, 2011, no 19.

SOLARTE, Francisco Nicolás Solarte; ROSERO, Edgar Rodrigo Enriquez; DEL CARMEN BENAVIDES, Mirian, 2015. Metodología de análisis y evaluación de riesgos aplicados a la seguridad informática y de información bajo la norma ISO/IEC 27001. *Revista Tecnológica-ESPOL*, 2015, vol. 28, no 5.

SORIANO, José Enrique Amaro, 2012. *El gran libro de programación avanzada con Android*. Marcombo, 2012.

VELOZ VIDAL, Carlos Alberto, 2016. Modelo para el desarrollo de aplicaciones nativas en android basado en mejores prácticas, metodologías ágiles y elementos del área interacción humano-computadora. 2016.

Glosario de términos

Apk: es la abreviatura para aplicación empaquetada de Android y puede ser entendida como un software instalado en una terminal Android.

CVE: siglas de *Common Vulnerabilities and Exposures*, lista de vulnerabilidades registradas, estándar que permite identificar las vulnerabilidades.

CVSS: siglas de *Common Vulnerabilities Score System*, estándar que puntúa a las vulnerabilidades registradas.

Diseño web adaptativo: o *web responsive design*, es una técnica de diseño y desarrollo web que, mediante el uso de estructuras e imágenes fluidas consigue adaptar el sitio web al entorno del usuario.

Falso positivo: Son aquellas vulnerabilidades que fueron detectadas por la herramienta automatizada y que el sistema no presenta realmente.
Falso negativo: Son las vulnerabilidades que la herramienta automatizada no fue capaz de detectar y se detectaron manualmente.

GitHub: una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se la utiliza principalmente para la creación de código fuente de programas de computadora, posee foros de debate y tutoriales de uso de librerías, etc.

HTTP: es un protocolo de nivel de aplicación para distribuir y colaborar información hipermedia de sistemas.

HTTPS: es un protocolo que utiliza SSL para encriptar los datos y crear un canal de cifrado más apropiado para el tráfico de información sensible que el protocolo HTTP.

JSON: siglas de *JavaScript Object Notation* es un formato de intercambio de datos.

Pila de software: o software *stack*: es un tipo especial de definición de software que puede utilizarse para identificar grupos de software que desea instalar al mismo tiempo en una secuencia determinada en los sistemas de destino.

Sistema operativo: es considerado el programa principal y encargado de administrar todos los recursos para ser utilizados de manera eficiente, cómoda y sin interrupciones, de tal manera que el usuario pueda mantener una comunicación sin problema haciendo uso de los recursos que el hardware le suministra.

SSL: es un protocolo en capas. En cada capa, los mensajes pueden incluir campos de duración, descripción y contenido. SSL lleva mensajes a ser transmitido, fragmenta los datos en bloques manejables,

opcionalmente comprime los datos, aplica un MAC, encripta y transmite resultado. Los datos recibidos se descifran, verifican, descomprimen y re ensamblado, luego entregado a clientes de mayor nivel.

URI: *Unified Resource Identifier* o identificador de recurso unificado, es una cadena de caracteres que identifica los recursos de una red.

XML: Lenguaje de marcado extensible abreviado en siglas en inglés XML, describe una clase de objetos de datos llamados Documentos XML, y describe parcialmente el comportamiento de los programas informáticos que los procesan.