Universidad de las Ciencias Informáticas
Facultad de Ciencias y Tecnologías Computacionales (CITEC)

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

Aplicación móvil para la votación en sala.

Autor:

Christián Raúl Vazquez del Valle

Tutores:

Ing. Lester Collado Rolo

Ing. Rene Orta Martínez

La Habana, noviembre de 2022

Declaración de autoría

Declaro ser el único autor del trabajo de diploma *“Aplicación móvil para la votación en sala”*, concedo a la Universidad de las Ciencias Informáticas y en especial al centro CREAD la autorización a hacer uso del mismo en su beneficio.

Para que conste firmo el presente documento a los ___ días del mes de _____ del año 2022.

Christián Raúl Vazquez del Valle

_____

Firma del autor

Ing. Lester Collado Rolo

_____

Firma del tutor

Ing. Rene Orta Martínez

_____

Firma del tutor

Datos de contacto

Ing. Lester Collado Rolo

Correo: lcollado@uci.cu

Ing. Rene Orta Martínez

Correo: rorta@uci.cu

Dedicatoria

Una vez llegado a este punto en el que me encuentro hoy quiero dedicar todos mis esfuerzos y estudios a las personas más importantes de mi vida que siempre han estado conmigo en las buenas y en las malas, esas que siempre han estado para ayudarme a levantarme y me han dado las fuerzas para seguir estudiando día a día; esta tesis va dedicada en especial a:

Mis padres que durante toda mi vida se esforzaron por hacerme un hombre de bien y con valores. Se los dedico a ellos que, durante toda mi vida, me han brindado todos los recursos y el apoyo necesarios para que ahora pueda estar cumpliendo mi mayor sueño.

A mi esposa. Que, aunque no nos conocemos desde hace mucho, ha estado a mi lado durante mi carrera universitaria y me ha animado a seguir cuando yo creía que no podría más. Se la dedico porque tuvo mucha paciencia conmigo en las noches en que me quedaba despierto estudiando, porque me entendía cuando estaba estresado y cambiaba de humor, y porque

siempre estaba dispuesta a ayudarme con todo lo que necesitara.

Esta tesis va dedicada a todas esas personas que de una manera u otra contribuyeron a que hoy me encuentre en esta etapa de mi vida, a esas personas que llegaron y a las que ya no están en mi vida, a mis amigos por su apoyo, en especial Alvaro Ernesto y también a todos mis profesores que a lo largo de la carrera me aportaron todos los conocimientos que hoy poseo.

Resumen

El presente trabajo de diploma se desarrolla debido a la necesidad de la existencia de un software donde se informatice el proceso de votación en sala, de modo que permita optimizar recursos materiales y humanos, así como posibles errores. Por tal motivo, se determinó implementar una aplicación móvil que contribuya a mejorar las actividades relacionadas con los votos en las asambleas nacionales, como además en otros entornos, ya que estas se realizan actualmente de forma manual, ya sea mediante boletas o mediante el método de mano alzada. Para asegurar el buen funcionamiento del sistema desde el punto de vista jurídico y se abordan diferentes sistemas de voto electrónico. Para llevar a cabo el desarrollo del sistema, se estudiaron y documentaron las metodologías, herramientas, tecnologías de desarrollo de software y lenguajes a emplear durante la implementación, como la definición de los elementos necesarios para el exitoso desarrollo de la misma, así como los artefactos requeridos por la metodología de desarrollo. Esta aplicación fue sometida a pruebas de software para verificar su calidad y correcto funcionamiento, los cuales arrojaron valores satisfactorios. Tras finalizar la investigación se obtuvo un sistema que cumple los objetivos establecidos, permitiendo al usuario emitir un voto desde cualquier dispositivo móvil.

Palabras clave: aplicaciones móviles, dispositivos móviles, voto en sala

Abstract

This diploma work is developed due to the need for a software to computerize the voting process in room, in order to optimize material and human resources, as well as possible errors. For this reason, it was decided to implement a mobile application to help improve the activities related to voting in national assemblies, as well as in other environments, since these are currently performed manually, either by ballots or by the show of hands method. To ensure the proper functioning of the system from the legal point of view and different electronic voting systems are addressed. To carry out the development of the system, the methodologies, tools, software development technologies and languages to be used during the implementation were studied and documented, as well as the definition of the necessary elements for the successful development of the same, as well as the artifacts required by the development methodology. This application was subjected to software tests to verify its quality and correct operation, which yielded satisfactory values. At the end of the research, a system was obtained that meets the objectives set, allowing the user to vote from any mobile device.

Keywords: mobile applications, mobile devices, floor voting

Tabla de contenido

Índice de ilustraciones

Índice de tablas

Introducción

Las TIC (Tecnologías de la Información y la Comunicación) están moldeando el futuro de las sociedades modernas. La acción de estas tecnologías no solo está afectando a la propia economía, destruyendo puestos de trabajo, alterando los trabajos que siguen siendo desarrollados por mano de obra humana o creando nuevos trabajos que hasta hace poco no existían (Kumano 2017).

Al igual que los ciudadanos y las empresas, los gobiernos han comenzado su particular transición a la esfera digital, trasladando parte de sus servicios al mundo virtual e interactuando en la red con los distintos actores sociales (Kumano 2017).

En el sector gubernamental también se ha llevado a cabo un nivel de informatización, desplegando aplicaciones para perder ejercer votos electrónicos.

El principal objetivo de la votación es el de permitir que los electores ejerzan el derecho a expresar su opinión respecto a ciertas cuestiones, fragmentos de la legislación, iniciativas ciudadanas, enmiendas constitucionales, recordatorios y para que elijan a sus gobernantes y representantes políticos. Cada vez con mayor frecuencia, la tecnología se usa como una herramienta para ayudar a los electores para que emitan su voto. Para permitir que se ejerza este derecho, casi todos los sistemas electorales alrededor del mundo incluyen los siguientes pasos (Aguilar, Gutiérrez, Howlet 2017):

- Identificación y autenticación de los electores.
- Votación y registro de los votos emitidos.
- Escrutinio de votos.
- Publicación de los resultados electorales.

En Cuba, los procesos eleccionarios siempre han contado con el máximo apoyo de nuestro pueblo para garantizar con mayor rapidez y eficacia el flujo de información desde las estructuras bases hasta la Comisión Electoral Nacional. Estos procesos incluyen a todos los ciudadanos cubanos mayores de 16 años y que no presenten incapacidad mental con previa declaración judicial de esta, además de los incapacitados por causas penales.

De igual manera pasa en las votaciones en sala realizadas en las asambleas. Pero estas presentan algunos obstáculos que la ralentizan, y esto es que el proceso de votación se realiza mediante el voto a mano o mediante boletas. El voto de mano alzada es un proceso el cual toma su tiempo debido a que se debe de realizar un voto donde todos los votantes que estén de acuerdo levanten la mano y se realice el conteo de los mismos, un nuevo voto a mano alzada para los que no están a favor y se realiza el mismo proceso nuevamente y otro para los que se abstiene, para de esta forma determinar una decisión. Aunque esta variante es más ágil que el voto mediante boletas, no impide que se puedan cometer errores humanos a la hora del conteo cuando se realiza a una escala superior a diez votantes, esta puede presentar fallas de precisión. Punto clave del voto en esta variante es que el voto no permanece en el anonimato.

El voto mediante boleta es el más tradicional, ya que el votante permanece en el anonimato. Sin embargo, presenta diversos problemas, como el caso de las boletas nulas o el error en el conteo de las mismas debido a grandes cantidades. Aunque es un trabajo que es realizado por expertos, no puede quitarse el riesgo de errores humanos. Este proceso sin duda alguna es un proceso que requiere de mucho tiempo. Pero llevar a cabo el desarrollo de nuevas tecnologías de votación no es una tarea fácil, viene marcada por una serie de ventajas e inconvenientes. Aunque el voto electrónico puede suponer un avance para el desarrollo de los procesos democráticos, los inconvenientes que puede provocar una mala implementación, un mal funcionamiento o incluso la disrupción completa del proceso electoral suponen una gran desventaja a la hora de decidir si se implementa o no.

Entre las ventajas que ofrece el voto electrónico encontramos (Kumano 2017):

- La rapidez del sistema a la hora de contar los votos emitidos y, gracias a ello, la obtención más acelerada de los resultados de los comicios.
- La eliminación de posibles errores humanos a la hora del conteo de los votos, debido a la mayor precisión de las máquinas.
- Una mayor comodidad para los ciudadanos que podría conllevar una mayor participación en las elecciones, dando mayor legitimidad a los procesos electorales.
- Evita las conductas fraudulentas en las mesas electorales y en el conteo de los votos.

- Da una mayor accesibilidad a las personas con alguna discapacidad, con problemas de movilidad o con dificultades para ejercer su derecho al voto.
- Ahorros en la gestión de los procesos electorales al eliminar papeletas, reducir el número de personas necesarias en las mesas o resolver problemas logísticos asociados al voto por correo.

A partir de la problemática antes mencionada se obtiene como **problema de investigación**: ¿Cómo contribuir al desarrollo de servicios de votación en sala?

El **objeto de estudio** de la presente investigación es el desarrollo de aplicaciones para sistemas operativos móviles, enfocando el **campo de estudio** en aplicaciones móviles que utilizan servicios web para votaciones en línea, enmarcando al **campo de acción**: Desarrollar aplicación móvil para las votaciones en sala del proyecto decidimOS.

Teniendo en cuenta el problema a resolver se define como **objetivo general**: Desarrollar una aplicación móvil para realizar votaciones en sala, que contribuya a mejorar y agilizar los procesos de votaciones.

Para dar cumplimiento al objetivo general se han derivado los siguientes **objetivos específicos**:

- Definir el marco teórico de la investigación mediante el estudio y análisis de los principales estándares de desarrollo de aplicaciones para móviles.
- Describir la propuesta de la aplicación y el proceso realizado para su desarrollo
- Definir la metodología, técnicas y herramientas para el desarrollo de la propuesta de solución.
- Desarrollar el análisis y diseño de la aplicación móvil para la votación en sala.
- Implementar las funcionalidades de la propuesta de solución.
- Realizar las pruebas de software al funcionamiento de la aplicación.

Para garantizar el cumplimiento de los objetivos específicos de la investigación se identifican las siguientes tareas de investigación:

- Estudio del estado del arte sobre el desarrollo de los Sistemas Electorales.

- Selección de las herramientas, tecnologías y metodologías para el desarrollo de la aplicación móvil.
- Definición de requisitos con el cliente.
- Definición de la arquitectura a utilizar para el desarrollo de la aplicación móvil.
- Realización del análisis y diseño de la aplicación.
- Realización de la implementación de la aplicación.

Para el desarrollo de la investigación se utilizaron los métodos de la investigación científica teóricos que a continuación se plantean:

Métodos Teóricos

- Histórico – Lógico: permite hacer un estudio del estado del arte relacionada con la situación problemática de la investigación a través del cual se profundizó en los antecedentes y tendencias actuales de las votaciones mediante aplicaciones móviles, así como los argumentos que antecedieron al problema. A su vez, permite analizar las ventajas y desventajas de las herramientas, tecnologías y metodología utilizadas actualmente en la informatización del proceso de reprocesamiento.
- Analítico – Sintético: permite procesar la información y llegar a conclusiones a partir de la revisión de conceptos, teorías, técnicas y herramientas. De aquí se extraen las ideas fundamentales y al mismo tiempo se detalla la información necesaria para que de esta manera identificar las tecnologías y herramientas que se utilizarán en el desarrollo de la aplicación.

Método empírico

- **Entrevista**: Se utilizará la entrevista como una conversación planificada con los clientes para obtener información acerca del problema que se investiga. Su uso constituye un medio para el conocimiento cualitativo de las características particulares de un proceso y puede influir en el posterior análisis y diseño del producto de software.

- **Observación**: se utilizó para obtener información e identificar algunas características mediante la percepción de objetos y recopilación de conocimientos para analizar la situación actual de sistemas similares a la solución propuesta. Se usa además para diagnosticar el problema e identificar posibles funcionalidades de la propuesta de solución.

Esta investigación estará estructurada en tres capítulos, donde se describe el trabajo realizado y los resultados obtenidos del mismo.

Capítulo 1: Fundamentación teórica de la investigación.

Se reflejan los conceptos asociados al tema, necesarios para la comprensión de la solución del problema planteado. Además, se exponen los elementos teóricos utilizados en la investigación, se realiza una descripción de las diferentes tecnologías, metodología, herramientas y lenguajes de programación a utilizar en el desarrollo de la solución, donde se arriba a conclusiones que permiten dar respuesta a la problemática de la investigación.

Capítulo 2: Propuesta de solución.

En este capítulo se exponen los elementos que permiten describir la propuesta de solución tales como: modelado de dominio y los principales requisitos funcionales y no funcionales a implementar. Además, se hace una descripción textual de cada uno de ellos, incluyendo los prototipos de interfaz de cada una de las historias de usuario para lograr un entendimiento claro del sistema a desarrollar.

Capítulo 3: Validación de la propuesta de solución.

Contiene la descripción de los elementos establecidos para el proceso de desarrollo empleado, se define la estructura y ubicación de los componentes que se van a desplegar, se describen los casos de prueba que se le realizaron al sistema, así como los estándares de codificación empleados, con el objetivo de validar el correcto funcionamiento de la aplicación.

Capítulo 1. Fundamentación teórica de la investigación

1.1 Introducción

En este capítulo se abordan los conceptos relacionados con la investigación, los cuales constituyen el conocimiento sobre los procesos y tecnologías asociadas al desarrollo de la solución requerida. Se exponen y explican los lenguajes de programación, metodologías, herramientas y tecnologías a utilizar en el desarrollo de la aplicación, justificando el motivo de su selección.

1.2 Conceptos asociados a la investigación

1.2.1 *Elección*

La elección constituye un método de designación del titular o titulares de un órgano, caracterizado por la pluralidad de los llamados a tomar parte en aquella, integrantes del «colegio electoral». El método electivo consiste en un proceso por el que la pluralidad de las declaraciones de voluntad de los componentes del colegio se reconduce a una voluntad de los componentes del colegio se reconduce a una voluntad única, normalmente identificada por el sistema de la mayoría, pero que vale como voluntad del colegio en su conjunto. La elección se caracteriza por la libertad formal con que se forman las declaraciones de voluntad que la integran, diferenciándose de los métodos de designación que tienen por objeto verificar si concurren o están ausentes ciertos requisitos necesarios para ser titular de un órgano (*Elecciones* 2014).

1.2.2 *El sistema electoral*

Es el conjunto de reglas y procedimientos destinados a regular las diversas etapas de los procesos de votación por los cuales la voluntad de la ciudadanía se transforma en órganos de gobierno de representación política. A través del sistema electoral se definen funciones básicas como quiénes pueden votar, quiénes ser votados, de cuántos votos dispone cada elector, cómo pueden y deben desarrollarse las campañas de propaganda y difusión electoral, cuántos representantes se eligen en cada demarcación electoral, cómo se determinan y delimitan los distritos y secciones electorales, quiénes y cómo deben encargarse de organizar los comicios, cómo deben emitirse y contarse los sufragios, cuántas vueltas electorales pueden y/o deben realizarse para determinar al triunfador, cómo se resuelven los conflictos postelectorales, entre otras (Díaz 2022).

1.2.3 *Votación*

Operación de ejercicio del derecho de voto por los electores durante un período de tiempo legalmente establecido. Sistema ordinario de adopción de acuerdos parlamentarios cuando el Reglamento no dispone otra clase de votación. Puede ser electrónica, por levantamiento de los parlamentarios o mediante mano alzada, manifestándose primero quienes aprueban, después quienes rechazan y por último quienes se abstienen (RAE 2020a).

1.2.4 *Voto Electrónico*

El voto electrónico forma parte de lo que hoy en día se conoce como nuevas tecnologías de votación (New Voting Technologies). Además del voto electrónico, dentro de este grupo se encuentran el escaneo de papeletas o el voto por Internet. Según la OSCE, las nuevas tecnologías de votación se pueden definir como el uso o aplicación de las Tecnologías de la Información y la Comunicación para la emisión del voto, su recuento y posterior tabulación. El Instituto IDEA, por su parte, define el voto electrónico como aquellos «sistemas en que el registro, la emisión o el conteo de los votos en elecciones para cargos políticos y referendos involucra el uso de tecnologías de la información y las comunicaciones (TIC) (Kumano 2017).

1.2.5 *Dispositivo Móvil*

Se puede definir como un aparato de tamaño pequeño, el cual tiene ciertas capacidades de procesamiento y que pueden estar conectados a una red telefónica o de Internet, tienen capacidad de almacenamiento y memoria limitada en comparación con las computadoras convencionales. Por lo general los dispositivos móviles son de pequeñas magnitudes y con un peso ligero. Estos cuentan con un procesador, cuyas características son de acuerdo a las funciones para las que han sido diseñados, en su mayoría tienen acceso a internet (Guevara Soriano 2010).

1.2.6 *Aplicación Móvil*

Es aquel software que utiliza en un dispositivo móvil como herramienta de comunicación, gestión, venta de servicios-productos orientados a proporcionar al usuario las necesidades que demande de forma automática e interactiva (Artica Navarro 2014).

Programa informático destinado a ser ejecutado en teléfonos inteligentes, tabletas u otros dispositivos móviles (RAE 2020b).

Una aplicación móvil, o app es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles y que permite al usuario efectuar una tarea concreta de cualquier tipo profesional, de ocio, educativas, de acceso a servicios, etc., facilitando las gestiones o actividades a desarrollar.

1.3 Estado del arte

En este acápite se hace un análisis de las principales aproximaciones existentes para la gestión de las votaciones en sala mediante el uso de tecnologías móviles. La investigación está centrada en dos aspectos fundamentales: la integración de los procesos electorales y las votaciones a aplicaciones móviles y el análisis de las aplicaciones móviles desarrolladas específicamente para elecciones y votaciones en sala.

1.3.1 *Aplicaciones para votaciones*

En el presente sub acápite se abordan las características de las principales soluciones de aplicaciones para ejercer el voto. El análisis de estas soluciones constituye el punto de partida para determinar si alguna de ellas puede ser utilizada para dar solución a la problemática de la presente investigación.

APPSAMBLEA

Es un sistema de votación online, cómodo y seguro. Ahorra tiempo y recursos votando desde cualquier lugar del mundo; solo necesitarás conexión a Internet. Lo pueden utilizar organizaciones de cualquier tipo: Empresas, Asociaciones, Entidades Públicas, Clubes Deportivos, o colectivos de cualquier tamaño, para facilitar sus votaciones, porque estas serán verificadas, secretas e inalterables (*Appsamblea - Votaciones online seguras, rápidas y fáciles* 2018).

La aplicación ayuda a los coordinadores a verificar la identidad de los participantes a través de las cargas de ID y fotos y utiliza tecnología de curva elíptica para garantizar un proceso de votación encriptado. Permite a los administradores de eventos ajustar la configuración de sondeo, cargar datos del censo del votante y controlar las sesiones. También puede enviar notificaciones a los votantes sobre la autenticación y proporcionar acceso a las personas para permitirles participar en la votación. Ofrece una funcionalidad de cadena de bloques, que permite a los organizadores proteger los votos enviados, eliminando la posibilidad de cualquier modificación o manipulación de resultados. También

proporciona un modo quiosco que permite a los miembros del equipo instalar estaciones o dispositivos de sondeo y coordinar el proceso en múltiples ubicaciones (*Appsamblea - Votaciones online seguras, rápidas y fáciles* 2018).

AssociationVoting

Es una herramienta de administración de elecciones basada en la web. La plataforma está diseñada para ayudar a maximizar la participación de votantes con características que incluyen nominaciones, elecciones híbridas, servicios de voto por poder y voto en línea con capacidad de respuesta móvil. Permite la personalización de las boletas en función de la membresía de la región o subgrupo (*Association Voting – Online Voting Made Simple* 2016).

ElectionBuddy

Es un software de votación en línea diseñado para administrar de manera precisa y segura las elecciones. Ofrece muchas opciones de votación, incluidos los teléfonos celulares, tabletas y computadoras de los votantes, o boletas impresas enviadas por correo. El software es compatible con una variedad de sistemas de votación que incluyen primero después de la publicación (FPTP), votación acumulativa, voto único transferible (STV), boleta preferencial, votación puntuada, referéndums, escala de calificación o votación de aprobación. Las papeletas se pueden personalizar para las elecciones de candidatos, puestos en la junta, aprobaciones presupuestarias, mociones, enmiendas a los estatutos, ratificaciones de contratos, encuestas de miembros y más (*ElectionBuddy* 2018).

Conclusiones del estudio de las soluciones existentes:

Tras analizar las aplicaciones similares a la solución a desarrollar, en las cuales se observaron los diseños de sus interfaces, las funcionalidades que ofrecen, el grado de dificultada, la hora de interactuar con la aplicación, entre otros rasgos importantes que contribuyeran a obtener un producto con la mejor calidad posible, se determinó que estas aplicaciones no son una propuesta de solución factible debido a que:

- Dado que los servidores de estas aplicaciones no se encuentran en el país, estaríamos exponiendo toda la información, lo que permitiría que se cometan intentos de fraude y manipulación de los votos emitidos.

- Las aplicaciones pertenecen a compañías privadas que cobran intereses por su uso; estos costos pueden llegar a ser elevados en determinados momentos debido a la licencia, el soporte y la capacitación.

A pesar de ello, el análisis de estas propuestas brindó una mayor comprensión del sistema a implementar. Las soluciones existentes permitieron una comprensión más precisa del flujo de datos; así como también, a la hora de la obtención de los requisitos, se pudo tener una visión más clara de las principales funcionalidades que deberían de estar presente en la propuesta de solución.

1.3.2 *Estudio de los sistemas operativos móviles*

En el presente sub acápite se abordan las características de las principales de los sistemas operativos más usados en la actualidad. El análisis de estos sistemas constituye el punto de partida para determinar una plataforma a seguir en la que se pueda abarcar la mayor cantidad de usuarios posibles.

Sistema Operativo

Los sistemas operativos son muy importantes en la actualidad, estos ha tenido un gran avance en la era moderna, basándose en el desarrollo de nuevos lenguajes de programación que permiten que esto tenga nuevas tendencias actualizadas para su buen funcionamiento en el uso de las máquinas informáticas, cabe mencionar que son importante las actualizaciones con el fin de presentar nuevos servicios que el usuario necesite e incluso la forma de obtener nuevas innovaciones que favorezcan los servicios que la sociedad requiera, es por ello que debemos determinar y llegar a la conclusión del uso de estas nuevas herramientas que destacan en el comercio la venta de sistemas operativos también conocidos como comerciales, y determinar los sistemas operativos libres. La importancia de los aparatos que posteriormente son electrónicos y que utilizan microprocesadores para su buen funcionamiento lleva incorporados en ellos un sistema operativo que facilita su buen funcionamiento (Monterrubio-Hernandez 2019)

Sistemas operativos más utilizados

Según datos de StatCounter, en el cierre del mes de mayo del 2022, los dispositivos Android representarán el 71.43% de la cuota de mercado de sistemas operativos móviles en todo el mundo, iOS representa el 27.85%, Samsung el 0.42%. La figura 1 muestra una representación de estos datos

Figura 1 Cuota de mercado de sistemas operativos móviles en todo el mundo.

Fuente: https://gs.statcounter.com/os-market-share/mobile/worldwide

Sistema operativo Android

Android es un sistema operativo para móviles diseñado por la compañía estadounidense Google. Basado en el sistema operativo Linux, su objetivo inicial fue fomentar el uso de un sistema de tipo abierto, gratuito, multiplataforma y muy seguro, adaptado a los dispositivos móviles como smartphones y tablets. Desde su creación, el sistema ha realizado una fuerte apuesta para atraer a desarrolladores, por ello cuenta con una variación de Java denominada Dalvik que permite desarrollar aplicaciones que exploten las utilidades de los dispositivos de manera muy sencilla (Polanco, Taibo 2011).

Este sistema operativo ofrece una gran variedad de ventajas respecto a otros, que presentamos a continuación y que deben ser tenidas en cuenta, para poder comparar y ejercer un juicio acertado y con datos, sobre cuál es mejor para nuestros intereses (YEICY JULIANA MOLINA RIVERA, JONATHAN SANDOVAL CARDONA, SANTIAGO ALBERTO TOLEDO FRANCO 2012).

Código abierto

La primera ventaja de Android, y la más notable, es que funciona con un código abierto, lo que se traduce en que cualquiera puede crear aplicaciones y contribuir a aumentar la oferta de esta en el universo Android, siendo la gran mayoría de ellas, gratuitas.

El hecho de que su código sea abierto, también posibilita una gran variedad de ventajas más, como por ejemplo el hecho de que los errores puedan ser revisados y reparados con mayor rapidez, y otras ventajas que veremos a continuación.

Mayor libertad

Otra característica positiva que define este sistema operativo, que se desprende de la primera, es la libertad total con la que trabajan los desarrolladores de Android para realizar todo tipo de aplicaciones sin límites y sin pedir ningún tipo de permiso.

Además de eso, también es destacable el hecho de que Android no depende de ningún fabricante u operadora para implementar las mejoras del sistema, lo que redunda en su nivel de libertad.

Diversidad y versatilidad

Por todo lo comentado anteriormente, Android es también el sistema operativo que cuenta con un mayor número de fabricantes y está presente en la mayoría de marcas comerciales, así como de usuarios, lo que lo convierte en el sistema más utilizado.

A ese hecho también debemos añadir la versatilidad de dispositivos en los que podemos encontrar este sistema operativo, entre los cuales podemos encontrar, además de teléfonos móviles, tabletas, relojes inteligentes, ordenadores, dispositivos GPS y toda clase de electrodomésticos de última generación que dispongan de la tecnología necesaria para conectarse.

Sistema multitarea

El sistema operativo Android cuenta también con un sistema multitarea que permite abrir distintas aplicaciones a la vez y hacerlas funcionar simultáneamente, así como ponerlas en modo suspensión, si no las estamos utilizando.

Todo ello nos servirá para trabajar con más agilidad y también para ahorrar en el consumo de memoria y de batería, ya que el sistema también permite cerrar las aplicaciones que no nos sirvan de utilidad, mientras trabajamos con otras.

Mayor interacción

Gracias a la libertad de la que hace gala este sistema, otra de sus ventajas características es su gran capacidad de personalización, es decir, el hecho de que los usuarios y los fabricantes de Android cuentan con una mayor capacidad de interacción y puedan personalizar a su gusto instalando tanto fondos de pantalla personales, como animaciones o temas de todo tipo.

Así pues, gracias a Android podremos elegir aquellos temas, estilos o interfaces que más se adapten a nuestro gusto, lo que sin duda dará un toque especial a nuestros dispositivos.

Comunidad

Otro hecho destacable del sistema operativo Android, es que sus propias características basadas en la libertad de creación y de comunicación entre sus desarrolladores, fomentan y potencian el feedback entre creadores de todo el mundo.

Es por eso que el sistema cuenta con la mayor comunidad del mundo, siempre en constante movimiento y generando permanentemente todo tipo de foros, eventos colectivos y reuniones de toda clase.

A pesar de poseer una gran variedad de ventajas, el sistema Android también cuenta con una serie de desventajas que deben ser tenidas en cuenta si queremos conocer en profundidad y en su globalidad las características de este sistema operativo, pero es de vital importancia recalcar que: El hecho de tener un código abierto, también posibilita que este sistema sea más susceptible a ataques y a que algunos hackers aprovechen errores del propio sistema para atacarlo. Esto proporciona a que el sistema sea vulnerable con respecto a otros sistemas operativos que no son de código abierto.

Sistema operativo iOS

IOS es un sistema operativo móvil desarrollado por Apple Inc. para iPhones, iPads y otros dispositivos móviles de Apple. Su se basa en capas. Su comunicación no ocurre directamente. La capa entre la capa de aplicación y la capa de hardware ayudará a la comunicación. El nivel inferior proporciona servicios básicos de los que dependen todas las aplicaciones y las capas de nivel superior proporcionan gráficos y servicios relacionados con la interfaz. La mayoría de las interfaces del sistema vienen con un paquete especial llamado marco. Un marco es un directorio que contiene bibliotecas compartidas dinámicas como archivos, archivos de encabezado, imágenes y aplicaciones auxiliares que admiten la biblioteca. Cada capa tiene un conjunto de marcos que son útiles para los desarrolladores (*Arquitectura del sistema operativo IOS – Acervo Lima* 2020).

1.3.3 *Tipos de aplicaciones Móviles*

En la actualidad se han desarrollado tres tipos de aplicaciones las cuales son: app nativa, aplicaciones web y aplicaciones híbridas, esta última es el resultado de la combinación entre las aplicaciones nativas y las webs apps que se desarrollan usando tecnologías web tales como: HTML, CSS, JavaScript que se compila y empaqueta de tal forma que el resultado final es una app para dispositivos móviles (Thomas et al. 2018) (Ver Figura 2).

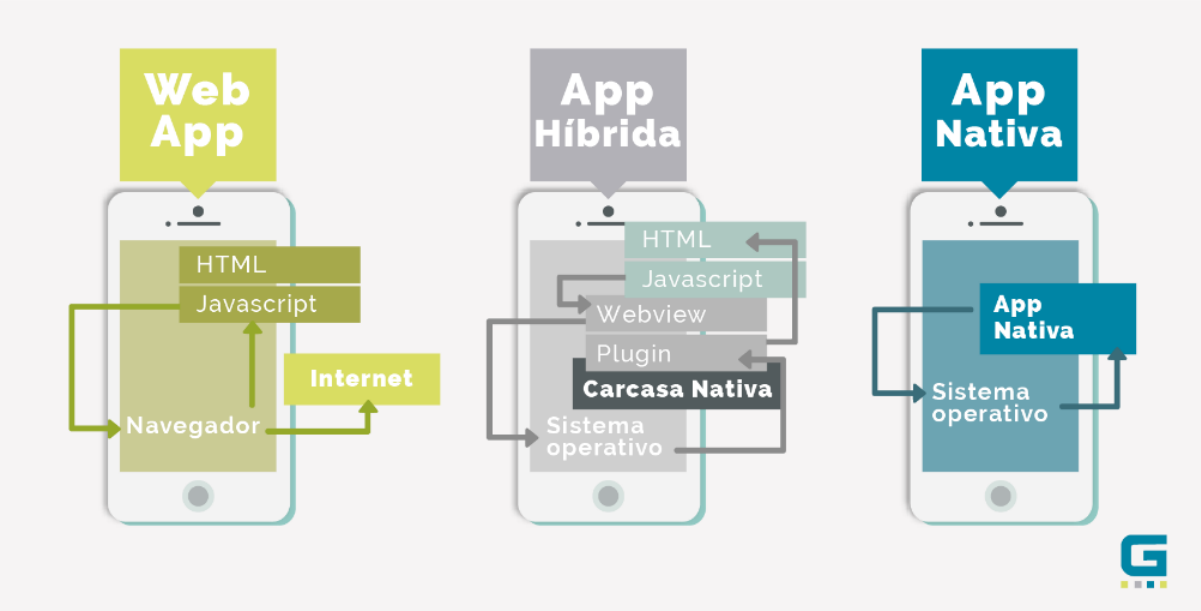


Figura 2 Tipos de aplicaciones móviles

Fuente: Gsoft Informática

Aplicaciones nativas

Las aplicaciones nativas así desarrolladas poseen un conjunto de características ventajosas, entre las que sobresalen el acceso a todas las capacidades del dispositivo (cámara, GPS, acelerómetro y agenda, entre otras), el alto rendimiento, la posibilidad de trabajar sin acceso a Internet y de correr en segundo plano notificando al usuario únicamente en caso de requerir su atención. Estas aplicaciones pueden distribuirse a través de las tiendas en línea correspondientes. Sin embargo, el precio de todas estas ventajas es alto: no es posible reusar el código fuente entre plataformas diferentes, el esfuerzo se multiplica y se elevan los costos de desarrollo, actualización y distribución de nuevas versiones (Thomas et al. 2018).

Aplicaciones Web

Una aplicación Web (Web based application) es una aplicación cliente/servidor, donde tanto el cliente (el navegador, explorador o visualizador) como el servidor (el servidor Web) y el protocolo mediante el que se comunican (HTTP) están estandarizados y no han de ser creados por el programador de aplicaciones (Ríos et al. 2018).

La construcción de aplicaciones Web Móviles constituye un ejemplo representativo de este enfoque. Estas aplicaciones se diseñan para correr dentro de un navegador, se desarrollan con tecnología web bien conocida (HTML, CSS y JavaScript), no necesitan adecuarse a ningún entorno operativo; su puesta en marcha es rápida y sencilla. Las desventajas de las aplicaciones Web Móviles recaen sobre su rendimiento. Los tiempos de respuesta se dilatan afectados por la interacción cliente-servidor y las restricciones de seguridad impuestas a la ejecución de código por medio del navegador limitan el acceso a todas las capacidades del dispositivo. Además, al no poseer el look and feel de las aplicaciones nativas, resultan menos atractivas para el usuario final (Thomas et al. 2018).

Aplicaciones híbridas

Las aplicaciones híbridas combinan lo mejor de los dos tipos de aplicaciones anteriores. Se utilizan tecnologías multiplataformas como HTML, Javascript y CSS, pero se puede acceder a buena parte de las capacidades específicas de los dispositivos. En resumen, son desarrolladas utilizando tecnología web y son ejecutadas dentro de un contenedor web sobre el dispositivo móvil. Entre las principales ventajas de esta metodología se pueden mencionar la posibilidad de distribución de la aplicación a través de las tiendas de aplicaciones, la reutilización de código para múltiples plataformas y la posibilidad de utilizar las características de hardware del dispositivo. Una de las desventajas es que, al utilizar la misma interfaz para todas las plataformas, la apariencia de la aplicación no será como la de una aplicación nativa. Finalmente, la ejecución será más lenta que la ejecución en una aplicación nativa (Delía et al. 2013).

Las aplicaciones híbridas constituyen otro tipo de desarrollo multiplataforma basado en tecnologías web (HTML, JavaScript y CSS) pero que, a diferencia de las anteriores, no son ejecutadas por un navegador. En su lugar, corren en un contenedor web especial con mayor acceso a las capacidades del dispositivo a través de una API específica. Las aplicaciones híbridas permiten la reutilización de código en las distintas plataformas, el acceso al hardware del dispositivo, y la distribución a través de las tiendas de aplicaciones. Sin embargo, conservan algunas de las desventajas de las aplicaciones Web Móviles: la utilización de componentes no nativos en la interfaz perjudica la experiencia de usuario, y la ejecución se ve ralentizada por la carga asociada al contenedor web (Thomas et al. 2018).

Selección del tipo de aplicación a implementar

Se ha decidido implementar una aplicación móvil híbrida debido a las numerosas ventajas que esta aporta, teniendo en cuenta que con el mismo código podemos desplegar en diferentes dispositivos y sistemas operativos: iOS, Android, web, etc. esta característica las convierte en una opción más económica frente a una app nativa. Además, se desea implementar un sistema que sea capaz de desplegar en cualquier plataforma para que alcance a la mayor cantidad de usuarios finales posibles.

1.4 Metodologías de desarrollo de software

El desarrollo de un software es un proceso que abarca distintos puntos de análisis antes de su partida inicial, estos puntos permiten su construcción con calidad. Para la implementación de una aplicación, es necesario tener en cuenta aspectos tales como: la metodología, el lenguaje de programación, las herramientas de desarrollo y modelado a emplear, ya que de estos aspectos dependerá el éxito y la calidad de desarrollo de la misma.

La metodología se define como la disciplina que indicará qué métodos y técnicas hay que usar en cada fase del ciclo de vida del desarrollo del proyecto. Los elementos que componen la metodología son: Fases, Documentación, Técnicas y Herramientas, Métodos, Control y Evaluación (Ríos et al. 2017).

1.4.1 *Metodologías tradicionales*

Según (Pressman, 2013), las metodologías de desarrollo tradicionales o clásicas son también llamados modelos de proceso prescriptivo, y fueron planteadas originalmente para poner orden en el caos del desarrollo de software que existía cuando se empezó a generar masivamente. La historia revela que estos modelos tradicionales, que fueron presentados en la década de los 60, dieron cierta estructura útil al trabajo de La Ingeniería de software y constituyen un mapa razonablemente eficaz para los equipos de desarrollo. En las metodologías tradicionales se concibe al proyecto como uno solo de grandes dimensiones y estructura definida; el proceso es de manera secuencial, en una sola dirección y sin marcha atrás; el proceso es rígido y no cambia; los requerimientos son acordados de una vez y para todo el proyecto, demandando grandes plazos de planeación previa y poca comunicación con el cliente una vez ha terminado esta (Vite Cevallos, Montero, Cuesta 2018).

A continuación, se presenta gráficamente las fases del modelo en cascada:

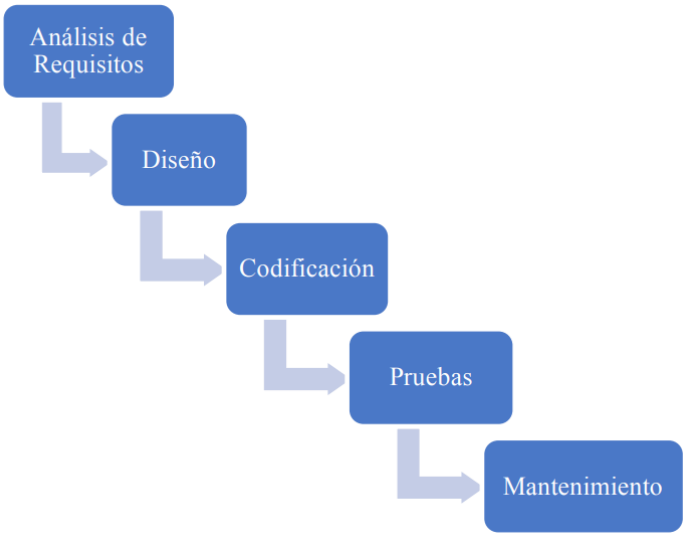


Figura 3 Fases del modelo en cascada.

El modelo en cascada se define como una secuencia de fases, que al final de cada etapa reúne toda la documentación para garantizar que cumple con los requerimientos y especificaciones. El modelo en cascada para la época se convirtió en un pilar fundamental de ejemplo de proceso dirigido, donde se planificaría todas las actividades antes de comenzar a trabajar en ellas. Al pasar el tiempo se empieza a detectar los principales problemas tales como la dificultad de responder a los requerimientos cambiantes del cliente (Vite Cevallos, Montero, Cuesta 2018).

1.4.2 *Metodologías Ágiles*

Las metodologías ágiles presentan como principal particularidad la flexibilidad, los proyectos en desarrollo son subdivididos en proyectos más pequeños, incluye una comunicación constante con el usuario, son altamente colaborativos y es mucho más adaptable a los cambios. De hecho, el cambio de requerimientos por parte del cliente es una característica especial, así como también las entregas, revisión y retroalimentación constante (Vite Cevallos, Montero, Cuesta 2018).

Entre las más notables metodologías de desarrollo ágil, se encuentran: Scrum, XP.

Metodología Scrum

Scrum es un método ágil de entrega de productos iterativos e incrementales que utiliza la retroalimentación frecuente y la toma de decisiones en colaboración (*Agile project management with Scrum* 2011).

Scrum es una variación de la metodología Ágil, la más clásica, capaz de soportar la variación de los requerimientos y reaccionar favorablemente a ellos; siempre y cuando sea bien aplicada, esta permite la flexibilidad en el desarrollo de software y su proceso no se basa en rigurosas reglas de administración. Scrum define ciclos cortos de ejecución del proceso llamados “sprints”, el cual debe durar entre 1 y 2 semanas y cada una de ellas debe cumplir sus metas y obtener una retroalimentación que sirva para el siguiente sprint (Merizalde Medina 2018).

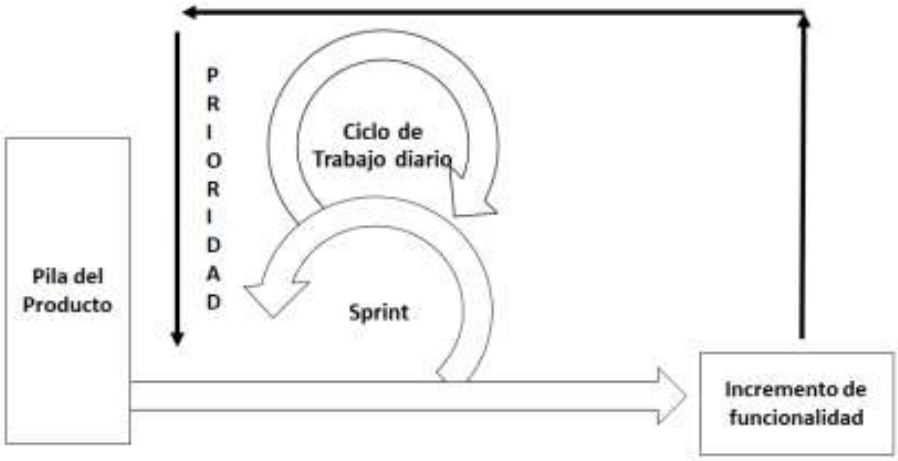


Figura 4 Modelo Scrum Iterativo.

Metodología XP

La metodología extreme programming o XP, es la metodología ágil más conocida. Fue desarrollada por Kent Beck en la búsqueda por guiar equipos de trabajo pequeños o medianos, entre dos y diez programadores, en ambientes de requerimientos imprecisos o cambiantes. La principal particularidad de esta metodología son las historias de usuario, las cuales corresponden a una técnica de especificación de requisitos; se trata de formatos en los cuales el cliente describe las características y funcionalidades que el sistema debe poseer. En esta metodología se realiza el proceso denominado Planning game, que define la fecha de cumplimiento y el alcance de una entrega funcional, el cliente define las historias de usuario y el desarrollador con base en ellas establece las características de la entrega, costos de implementación y número de interacciones para terminarla. Para cada iteración, el cliente estipula cuáles son las historias de usuario que componen una entrega funcional. Se realizan entregas pequeñas que son el uso de ciclos cortos de desarrollo, llamado iteraciones, que muestra al cliente una funcionalidad del software terminado y se obtiene una retroalimentación de él. Algo muy característico de esta metodología es la programación en parejas, indica que cada funcionalidad debe de ser desarrollada por dos programadores, las parejas deben cambiar con cierta frecuencia, para que

el conocimiento no sea solo de una persona, sino de todo el equipo (Vite Cevallos, Montero, Cuesta 2018).

Metodología AUP-UCI

La Universidad de las Ciencias Informáticas (UCI) desarrolló una versión de la metodología de desarrollo de software AUP (Proceso Ágil Unificado), con el fin de crear una metodología que se adapte al ciclo de vida definido por la actividad productiva de la universidad. Esta versión decide mantener para el ciclo de vida de los proyectos la fase de Inicio, pero modificando el objetivo de la misma y se unifican las restantes fases de la metodología de desarrollo de software AUP en una sola, nombrada Ejecución y agregándose también una nueva fase llamada Cierre. A continuación, se muestra una tabla con las fases de la metodología AUP-UCI (Morales, Borrell, Armas 2019):

Fases AUP	Fases Variación AUP-UCI	Objetivos de las fases (Variación AUP-UCI)
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización fundamental acerca del alcance del proyecto, realiza estimaciones de tiempo, esfuerzo y costo y decide si se ejecuta o no el proyecto.
Elaboración	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto.
Construcción		
Transición		
	Cierre	En esta fase se analizan los resultados del proyecto como la ejecución y se realizan las actividades formales de cierre del proyecto.

Tabla 1 Fases de la variación de AUP para la UCI.

Selección de la metodología a usar:

A partir del análisis e investigación realizada, la metodología AUP-UCI es apropiada para proyectos pequeños, permitiendo disminuir las probabilidades de fracaso, por ser un producto de fácil uso utilizando cualquier herramienta. Con la adaptación de AUP que se propone para la actividad productiva de la UCI se logra estandarizar el proceso de desarrollo de software. Se logra hablar un lenguaje común en cuanto a fases, disciplinas, roles y productos de trabajos. Se redujo a 1 la cantidad

de metodologías que se usaban y de más de 20 roles en total que se definían se redujeron a 11. De igual manera, también es la metodología que más utilidad tiene en la UCI por ser una versión desarrollada en dicha institución (SÁNCHEZ RODRÍGUEZ 2015).

1.5 Lenguajes y herramientas a utilizar

1.5.1 *Lenguajes*

Lenguaje de Modelado UML (v.8.0)

Un Lenguaje Unificado de Modelado (UML: Unified Modeling Language) es una herramienta que permite modelar software orientado a objetos a través de un amplio vocabulario gráfico enfocado a la representación conceptual y física de los sistemas de software (Bueno 2002).

UML es un lenguaje de modelado para visualizar, especificar, construir y documentar partes de un sistema software desde distintos puntos de vista (García-Holgado et al. 2020):

- Puede usarse con cualquier proceso de desarrollo, a lo largo de todo el ciclo de vida y puede aplicarse a todos los dominios de aplicación y plataformas de implementación
- También puede usarse en otras áreas, como la ingeniería de negocio y modelado de procesos gracias a los mecanismos de adaptación/extensión mediante perfiles

El lenguaje de modelado UML es uno de los estándares más utilizado para especificar y documentar sistemas. Y ha sido utilizado para especificar patrones de diseño. Pero ocurre que a veces no es suficientemente formal para lograr una especificación precisa. Los Perfiles UML constituyen el mecanismo que proporciona el propio UML para extender su sintaxis y su semántica de manera de expresar los conceptos específicos de un determinado dominio de aplicación. Los perfiles no solo pueden ser utilizados para dominios específicos, sino también para resolver problemas particulares en diferentes dominios (Cortez, Garis, Riesco 2011).

El objetivo de UML es la unificación de los métodos de modelado de objetos (Booch, OMT y OOSE) por medio de la identificación y definición de la semántica de los conceptos fundamentales y elección de una representación gráfica con una sintaxis simple, expresiva e intuitiva. La especificación UML se define usando el enfoque de un metamodelo (García-Holgado et al. 2020).

Ventajas que proporciona el lenguaje de modelado UML (*La guía sencilla para la diagramación de UML y el modelado de la base de datos* 2019)

- Simplifica las complejidades
- Mantiene abiertas las líneas de comunicación
- Automatiza la producción de software y los procesos
- Ayuda a resolver los problemas arquitectónicos constantes
- Aumenta la calidad del trabajo
- Reduce los costos y el tiempo de comercialización

Framework Flutter (v.3.3.2)

Flutter es un framework multiplataforma que tiene como objetivo desarrollar aplicaciones móviles de alto rendimiento. Además de funcionar en Android y iOS, las aplicaciones de Flutter también se ejecutan en Fuschia. Este es elegido como el marco principal en el ámbito de aplicación de Google para su sistema operativo de próxima generación. Flutter es excepcional porque depende de los widgets OEM del dispositivo en lugar de consumir vistas web. Utiliza un motor de renderizado de alto rendimiento para renderizar cada componente de la vista utilizando el suyo propio. Esto ofrece la posibilidad de crear aplicaciones de tan alto rendimiento como las aplicaciones nativas. En cuanto a la arquitectura, el código C o C++ del motor implica la compilación con NDK de Android y LLVM para iOS respectivamente, y durante el proceso de compilación, el código Dart es compilado en código nativo. La función de recarga en caliente de Flutter se denomina Stateful hot reload y es un factor importante factor para impulsar el ciclo de desarrollo. La recarga en caliente Stateful se implementa enviando el código fuente actualizado a la máquina virtual de Dart (Dart VM) sin cambiar la estructura interna de la aplicación, por lo que las transiciones y acciones de la de la aplicación se conservarán bien después de la recarga en caliente (Tashildar et al. 2020).

Lenguaje de programación Dart (v.2.18.1)

Flutter utiliza Dart como lenguaje de programación, también desarrollado por Google. Dart es un lenguaje de programación de propósito general, orientado a objetos y de código abierto desarrollado por Google. Es usado para construir aplicaciones web, móviles y dispositivos IoT (Internet of Things). Dart ofrece herramientas para realizar análisis personalizado sobre el árbol sintáctico de un código fuente Dart. Estas herramientas pueden ser integradas a los ambientes de desarrollo (IDE) mediante extensiones, lo que permite al usuario analizar sus programas de forma interactiva (*Desclasificación basada en tipos en DART: Implementación y elaboración de herramientas de inferencia* 2018).

El lenguaje de programación Dart en un lenguaje de carácter open source (de código abierto o con acceso al público mediante este tipo de software), establecido con Google para lograr en conjunto con los desarrolladores de programación un lenguaje que permita estar centralizado en los objetos, así

como también logrando un análisis estático en estos mismos. Básicamente, el lenguaje de programación Dart consiste y se sostiene a la vez en la medida de su facilidad y sencillez al momento de establecerse de manera más rápida y cómoda frente a los desarrolladores. A partir de esta particularidad, Dart ofrece un conjunto de herramientas que logran en el usuario la utilización de estas, que son de tipo de compiladores, analizadores y de formatear. Asimismo, una gran ventaja que ofrece Dart es la ejecución del código en el momento en el que se va desarrollando. También es importante mencionar las características o conocimientos similares que comparte con JavaScript o C ++ (Higuera Fombuena, González Antona 2017).

1.5.2 *Herramientas*

Herramienta para el modelado Visual Paradigm (v.5.0)

Visual Paradigm for UML (VP-UML) es una herramienta de diseño UML y herramienta CASE del inglés (*Computer Aided Software Engineering*) diseñada para ayudar al desarrollo de software. Soporta los principales estándares de la industria tales como el Lenguaje de Modelado Unificado (UML), y la notación de Modelado de Procesos de Negocio (BPMN). Ofrece un completo conjunto de herramientas de equipos de desarrollo de software necesario para la captura de requisitos, planificación de software, planificación de controles, modelado de clases y modelado de datos. Además, permite modelar todos los tipos de diagramas de clases, código inverso y generar código desde los diagramas (*Herramienta CASE Visual Paradigm - MARCO TEÓRICO* 2021).

IDE para el desarrollo de aplicaciones móviles *Android Studio* (v.2021.3)

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android y está basado en IntelliJ IDEA. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece incluso más funciones que aumentan tu productividad cuando desarrollas apps para Android, como las siguientes (CASTILLO 2019):

- Un sistema de compilación flexible basado en Gradle
- Un emulador rápido y cargado de funciones
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android
- Aplicación de cambios para insertar cambios de código y recursos a la app en ejecución sin reiniciarla
- Integración con GitHub y plantillas de código para ayudarte a compilar funciones de apps comunes y también importar código de muestra
- Variedad de marcos de trabajo y herramientas de prueba

- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones, entre otros
- Compatibilidad con C++ y NDK
- Compatibilidad integrada con Google Cloud Platform, que facilita la integración con Google Cloud Messaging y App Engine

1.6 Conclusiones del capítulo

En el presente capítulo se estudiaron los principales conceptos asociados al objeto y campo de la investigación para respaldar la labor a desarrollar, puntualizando los conceptos necesarios referentes al campo de acción de manera detallada, se realiza un análisis donde se seleccionan las metodologías, herramientas y lenguajes de modelado que permitieron una selección idónea para desarrollar la propuesta de solución. Se logra crear un marco de trabajo adecuado a las necesidades y características de una aplicación para dispositivos móviles. También, se realizó un estudio sobre los software similares a la propuesta planteada para el apoyo a los procesos, elecciones y votaciones, donde se resaltaron las características que poseen las aplicaciones similares; donde vale destacar los grandes costos que tendrían para el país la adquisición de cualquiera de los sistemas mencionados anteriormente, así como la seguridad, confidencialidad de los datos y el cumplimiento de los términos legales y constitucionales que protegen el principio electoral. Esto llevó a la conclusión de la necesidad de crear un sistema para gestionar la información de las votaciones en la sala. Se desarrolla un análisis donde se seleccionan la metodología AUP-UCI como metodología de desarrollo más idónea para el desarrollo de la propuesta de solución planteada.

Capítulo 2. Propuesta de solución

El presente capítulo contiene la descripción de la propuesta de solución al problema de investigación a través de una imagen que representa los flujos principales de la misma, así como el modelo conceptual como una representación visual de las clases conceptuales del dominio de interés, y también la descripción de estas. Además, se especifican los requisitos funcionales y no funcionales como características imprescindibles que debe cumplir el software; así como la descripción de los requisitos funcionales utilizando las Historias de Usuario y se define la arquitectura de software y los patrones de diseño.

2.1 Descripción del sistema

La propuesta de solución tiene como objetivo acceder a los datos del Sistema de votación en sala para la realización de votos en línea a través de la utilización de una capa de servicios web que garantiza la comunicación entre ambos sistemas. Esta permitirá el acceso a la principal funcionalidad del sistema desde dispositivos móviles con sistema operativo Android 4.4.2 o superior y cuenten con una conexión a internet, ya sea Wifi, Ethernet o conexión de datos móviles.

La propuesta de solución permite la autenticación en el sistema mediante el consumo de servicios web. Una vez que se haya realizado la autenticación y el sistema haya verificado que las credenciales son válidas, enviará a la aplicación móvil una respuesta de aceptación que, además de las credenciales, contendrá un token que identificará al usuario. Las credenciales se cifran mediante el cifrado *Advanced Encryption Standard* (AES), la clave secreta AES se cifran con RSA por sus siglas en inglés (*Rivest, Shamir y Adleman*) y la clave RSA se almacena en *KeyStore* del dispositivo móvil. Una vez que el usuario haya iniciado sección en el sistema, la aplicación móvil solicitara la boleta activa para poder emitir el voto. Para llevar a cabo esta solicitud que realiza la aplicación móvil, esta debe enviar el token al sistema de votación en sala para comprobar que ese usuario tiene autorización para visualizar dicha boleta. Una vez visualizada la boleta, se procederá a emitir el voto, el cual se mantendrá en anonimato todo el tiempo. En la ejecución del voto solo se envía al servidor la dirección del Protocolo de Internet (IP) del dispositivo móvil para registrar que esa persona ya votó y no pueda repetir el voto, también se enviara enviar el ID de la boleta, el ID de la opción seleccionada, y la opinión en caso de haberla redactado. Los votos son almacenados en Sistema de votación en sala.

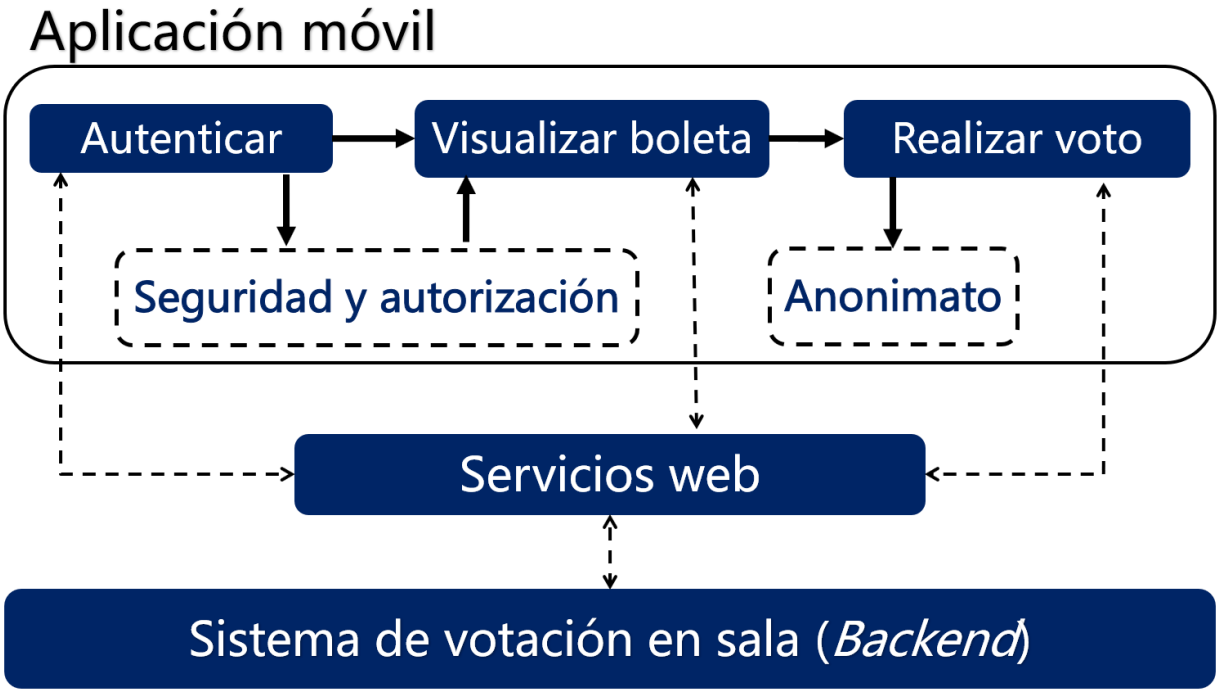


Figura 5 Propuesta de solución.

2.2 Modelo conceptual

Según lo planteado por (Jacobson, Booch, Rumbaugh 2000), el modelo del dominio, captura, comprende y describe los objetos más importantes dentro del contexto de un sistema. Los objetos del dominio o clases pueden obtenerse a partir de una especificación de requisitos o mediante la entrevista con los expertos del tema. Las clases que lo componen suelen aparecer en tres formas típicas:

- Objetos del negocio que representan los elementos que se manipulan en el negocio.
- Objetos del mundo real y conceptos de los que el sistema debe hacer un seguimiento.
- Sucesos que ocurrirán o que han ocurrido.

Un modelo conceptual tiene como objetivo identificar y explicar los conceptos significativos en un dominio de problema, identificando los atributos y las asociaciones existentes entre ellos. Puede ser visto, también, como una representación de las cosas, entidades, ideas, conceptos u objetos del "mundo real" o dominio de interés. También es considerado como un diccionario visual de abstracciones, conceptos, vocabulario e información del dominio de problema (Marciszack et al. 2019).

El Modelo de Dominio puede ser tomado como punto inicial para el diseño del sistema; este puede utilizarse para capturar y expresar los aspectos más importantes del contexto del sistema. Además, es una representación visual de los conceptos u objetos del mundo real, significativos para un problema o área de interés. A continuación, en la figura 5 se presenta del diagrama del modelo conceptual, el cual constituye una representación gráfica de los conceptos fundamentales a tener en cuenta para su desarrollo

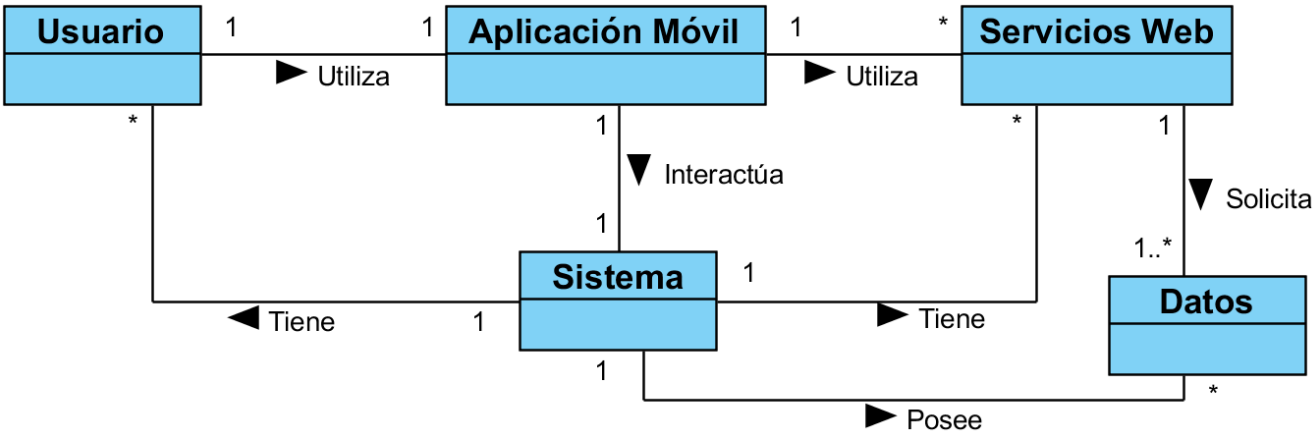


Figura 6 Diagrama de clases del modelo conceptual

2.2.1 *Descripción del diagrama de clase del modelo conceptual*

Concepto	Descripción
Usuario	Todo aquel que tendrían acceso al sistema con el objetivo de interactuar con el mismo mediante la aplicación móvil
Aplicación móvil	Identifica a la aplicación para el sistema operativo Android que se encarga de solicitar los datos del sistema mediante el uso de los servicios web.
Servicios Web	Es la herramienta que facilita la interoperabilidad entre el sistema y aplicaciones externas al mismo.
Sistema	Se refiere a la aplicación web del sistema de votaciones en línea.
Datos	Es la información que posee el sistema.

Tabla 2 Descripción del diagrama de clase del modelo conceptual

2.3 Levantamiento de requisitos

El levantamiento de requisitos es la primera etapa en el ciclo de desarrollo de software. En esta etapa es importante aplicar técnicas de levantamiento de requerimientos para culminar exitosamente los proyectos de desarrollo software en el tiempo estimado y cumpliendo con calidad con los requisitos del cliente (Rondon Suarez 2019).

El levantamiento de requerimientos es una de las etapas iniciales y primordiales para el desarrollo de un sistema de información. Un buen levantamiento conlleva a desarrollar un sistema lo más apegado posible a los requerimientos del usuario final. Para poder llevar a cabo esta etapa, en el desarrollo se deben seleccionar y aplicar algunas de las técnicas existentes para el levantamiento de requerimientos. En este artículo se presentan algunas técnicas para el levantamiento de requerimientos de tal forma que puedan ser utilizadas en otros sistemas de información (Sánchez, Venegas, Romero 2018).

2.3.1 *Requisitos funcionales*

RF 1. Autenticar.
RF 2. Visualizar boleta.
RF 3. Introducir opinión.
RF 4. Votar.
RF 5 Guardar voto en la base de datos
RF 6 Cargar voto de la base de datos
RF 7 Eliminar voto de la base de datos

2.3.2 *Requisitos No Funcionales*

Disponibilidad:
RNF 1. El sistema debe estar disponible solo el día en que se realizan las votaciones.

Confidencialidad:
RNF 2. El sistema solo puede ser manipulado por el personal autorizado.

Apariencia o interfaz externa:
RNF 3. El sistema debe tener una interfaz agradable para los usuarios, con colores que no distraigan su atención.

Ambiente:
RNF 4. Las funcionalidades del sistema deben estar agrupadas de conformidad con sus características y los datos con los que opera.

Seguridad:
RNF 5. La contraseña del usuario autenticado debe almacenarse utilizando algún método de cifrado.

RNF 6. Enviar las credenciales al servidor cifradas en consecuencia al esquema de autenticación que se utilice.

Hardware:

RNF 7. El dispositivo móvil debe de utilizar una memoria RAM de 512 megabytes como mínimo.

RNF 8. Almacenamiento disponible 5 megabytes

Software:

RNF 9. El dispositivo móvil que se disponga debe tener el sistema operativo Android versión 4.4.2 o superior.

2.4 Historia de Usuario

Las historias de usuario se usan, en el contexto de la ingeniería de requisitos ágil, como una herramienta de comunicación que combina las fortalezas de ambos medios: escrito y verbal. Describen, en una o dos frases, una funcionalidad de software desde el punto de vista del usuario, con el lenguaje que este emplearía (MENZINSKY 2018).

Historia de usuario	
Número: HU_1	**Nombre Historia de Usuario:** Autenticar
Prioridad en negocio: Alta	
Descripción: Comienza cuando el usuario introduce la contraseña, el sistema comprueba que sean válidos y permite el acceso de acuerdo al rol que este ocupe.	
Observaciones: Si las credenciales son incorrectas, se muestra un diálogo de error.	
Prototipo:	

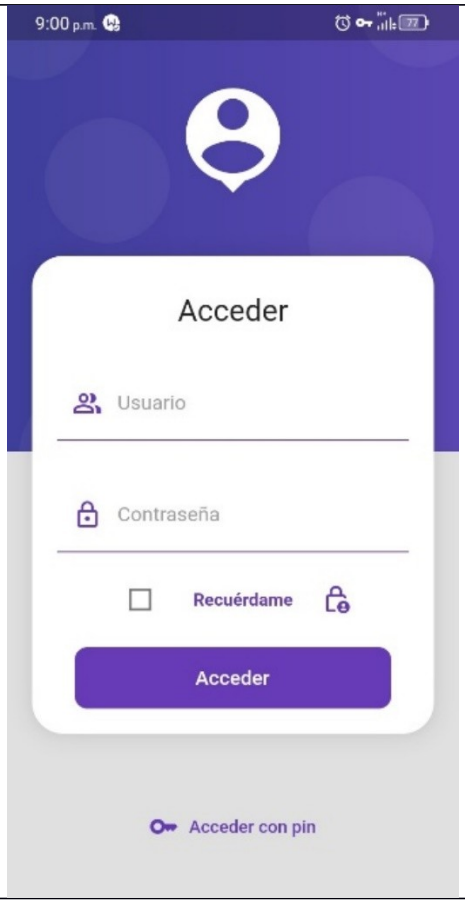


Tabla 3 Historia de usuario 1 RF Autenticar

Historia de usuario	
Número: HU_2	**Nombre Historia de Usuario:** Visualizar boleta
Prioridad en negocio: Media	
Descripción: Comienza cuando el usuario sé auténtica, de forma automática se cargará la boleta.	
Observaciones: Si no se encuentran boletas activas, la aplicación mostrará la pantalla con un texto informando que no se encuentran boletas activas.	
Prototipo:	

Tabla 4 Historia de usuario 2 RF Visualizar boleta

Historia de usuario	
Número: HU_3	**Nombre Historia de Usuario:** Introducir descripción
Prioridad en negocio: Media	
Descripción: Comienza cuando el usuario introduce en el campo de texto una opinión	
Observaciones: El campo de texto tiene un límite de 500 caracteres	
Prototipo: 	

Tabla 5 Historia de usuario 3 RF Introducir opinión

Historia de usuario	
Número: HU_4	**Nombre Historia de Usuario:** Votar
Prioridad en negocio: Alta	

Descripción: Comienza cuando el usuario da clic en el botón votar.
Observaciones: El sistema debe cambiar la vista una vez que se realiza la votación.
Prototipo: 

Tabla 6 Historia de usuario 4 RF Votar

2.5 Estilo arquitectónico

Un patrón arquitectónico define la estructura básica de una aplicación, provee un subconjunto de subsistemas predefinidos, incluyendo reglas, lineamientos para conectarlos y pautas para su organización, además constituye una plantilla de construcción (Mejía, Agudelo, Gil 2021).
Entre las ventajas del uso de patrones, se pueden encontrar:

- Permiten la reutilización de soluciones arquitectónicas de calidad.
- Son de gran ayuda para controlar la complejidad de un diseño.
- Facilitan la documentación de diseños arquitectónicos.
- Proporcionan un vocabulario común que mejora la comunicación entre diseñadores.

Los patrones arquitectónicos expresan una organización estructural para un sistema, permiten estructurar los componentes y subsistemas de un sistema. La abstracción más alta en cuanto a soluciones a través de patrones se obtiene a través del uso de patrones de arquitectura.

2.5.1 *Patrón arquitectónico N-Capas*

Se propone el uso del patrón arquitectónico N-Capas para el desarrollo de la aplicación, ya que los entornos de ejecución de dicha aplicación estarán distribuidos en diferentes elementos o nodos, tanto lógicos como físicos. Aquí aparecen dos conceptos importantes: las capas que se encargan de la distribución lógica de los componentes, y los niveles que se refieren a la colocación física de los recursos. La característica principal de este patrón es que cada capa oculta las capas inferiores de las siguientes superiores a esta.

El estilo arquitectural en capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades, la funcionalidad que implementan. Algunos de los veneficios que posee la arquitectura son (Chininin 2012):

- Abstracción ya que los cambios se realizan a alto nivel y se puede incrementar o reducir el nivel de abstracción que se usa en cada capa del modelo.
- Aislamiento ya que se pueden realizar actualizaciones en el interior de las capas sin que esto afecte al resto del sistema.
- Rendimiento ya que distribuyendo las capas en distintos niveles físicos se puede mejorar la escalabilidad, la tolerancia a fallos y el rendimiento.
- Testeabilidad ya que cada capa tiene una interfaz bien definida sobre la que realizar las pruebas y la habilidad de cambiar entre diferentes implementaciones de una capa.
- Independencia ya que elimina la necesidad de considerar el hardware y el despliegue, así como las dependencias con interfaces externas.
- Mejoras en las posibilidades de mantenimiento, debido a que cada capa es independiente de la otra, los cambios o actualizaciones pueden ser realizados sin afectar la aplicación como un todo.
- Escalabilidad, ya que las capas están basadas en diferentes máquinas, el escalamiento de la aplicación hacia afuera es razonablemente sencillo.
- Flexibilidad, como cada capa puede ser manejada y escalada de forma independiente, la flexibilidad se incrementa.
- Disponibilidad, las aplicaciones pueden aprovechar la arquitectura modular de los sistemas habilitados usando componentes que escalan fácilmente lo que incrementa la disponibilidad.

La descomposición en capas que se propone es:

Capa de presentación: es la única que interactúa directamente con el usuario presentándole el sistema, permitiendo el intercambio de información entre ambos. Contiene una interfaz gráfica que posibilita mostrar a los usuarios los resultados de sus peticiones y estados de la aplicación. Esta capa solo interactúa con la capa de negocio, que es su inferior inmediata.

Lógica de programación: es la que recibe las peticiones de la capa de presentación y le envía las respuestas tras el proceso. Aquí se realiza la mayor parte del procesamiento de la información del dominio de la aplicación, donde se ejecutan los algoritmos específicos del programa. Esta capa interactúa hacia arriba, con la capa de presentación, para recibir sus solicitudes y presentarle los

resultados al usuario e interactúa hacia abajo haciendo solicitudes a la capa de red y a la capa de acceso de datos.

Capa de red: permite acceder a fuentes de datos externas a la aplicación y el dispositivo a través del uso de las redes de cómputo. Se realizan llamadas a servicios web en servidores de internet para obtener datos e imágenes necesarios.

Capa de acceso a datos: el código que permite acceder a las fuentes de datos. Esencialmente, trata sobre 4 operaciones básicas, llamadas CRUD (por Create, Retrieve, Update y Delete), que se realizan sobre cualquier fuente de datos (normalmente alguna base de datos relacional).

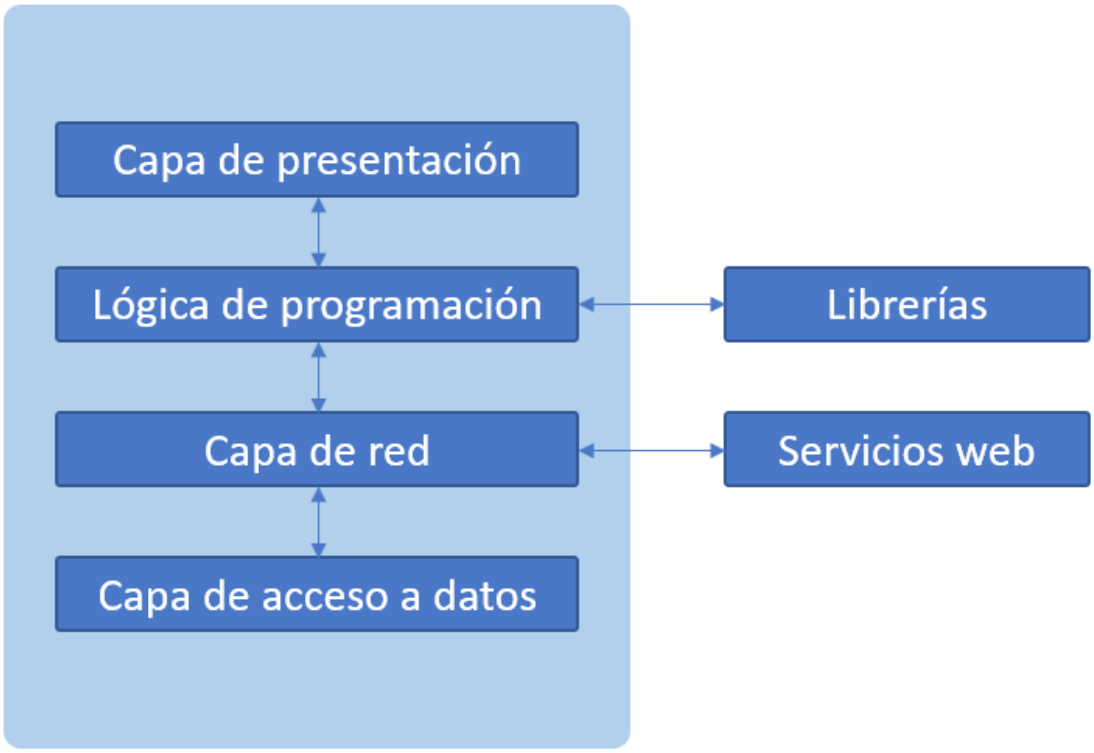


Figura 7 Diagrama de la arquitectura del proyecto

Descripción de los elementos de cada capa:

- Capa de presentación: En esta capa se encuentra la actividad Android que contiene los componentes visuales que se muestran al usuario.
- Lógica de programación: En esta capa se representan todos los componentes lógicos del flujo de trabajo, o sea las clases que se encargan de ejecutar las tareas propias de la aplicación.

- Capa de red: En esta capa se realizan todas las consultas necesarias al servidor mediante la utilización de servicios web.
- Capa de acceso de datos: En esta capa se crean las tablas necesarias para el almacenamiento local de los datos, así como también se realizan consultas a las mismas en caso de que la capa de red no brinde una respuesta satisfactoria.

2.6 Patrones de diseño

Los patrones de diseño se han manifestado como una importante técnica para construir software orientado a objetos. Un patrón de diseño ofrece una solución para un problema de diseño común, describiendo cómo una sociedad de clases (y objetos) que trabajan juntos para resolver ese problema en un contexto particular. El valor principal de los patrones de diseño se encuentra en el aprovechamiento de la experiencia de los expertos que los han identificado y documentado (Martínez, Molina, García 1999)

Los patrones de diseño son soluciones a problemas de diseño recurrentes en software científico. Estos patrones se usan para mitigar la ausencia de algunos aspectos de calidad inherentes al software (Calle Gallego 2016).

Los patrones de diseño, son estrategias que permiten la documentación del software, la transmisión de conocimientos y el punto de partida para una terminología compartida (Cortez et al. 2020).

A continuación, se presentan los patrones de diseño identificados durante el desarrollo de la propuesta de solución clasificada en dos grandes grupos: patrones GRASP y patrones GoF.

2.6.1 *Patrones GRASP*

Los patrones GRASP, más que patrones propiamente dichos, son una serie de “buenas prácticas” de aplicación recomendable en el diseño de software. Entre ellos, los patrones Experto, Creador, Bajo acoplamiento y Alta cohesión pueden ser tratados en un curso de lógica de programación, porque guardan directa relación con la creación y asignación de responsabilidades a los objetos. Sin embargo, el patrón Controlador, que también conforma los patrones GRASP, se puede estudiar en un curso superior de ingeniería de software o en un laboratorio de programación, dado que sirve como intermediario entre la interfaz de usuario y las clases que encapsulan los datos (Tabares 2010).

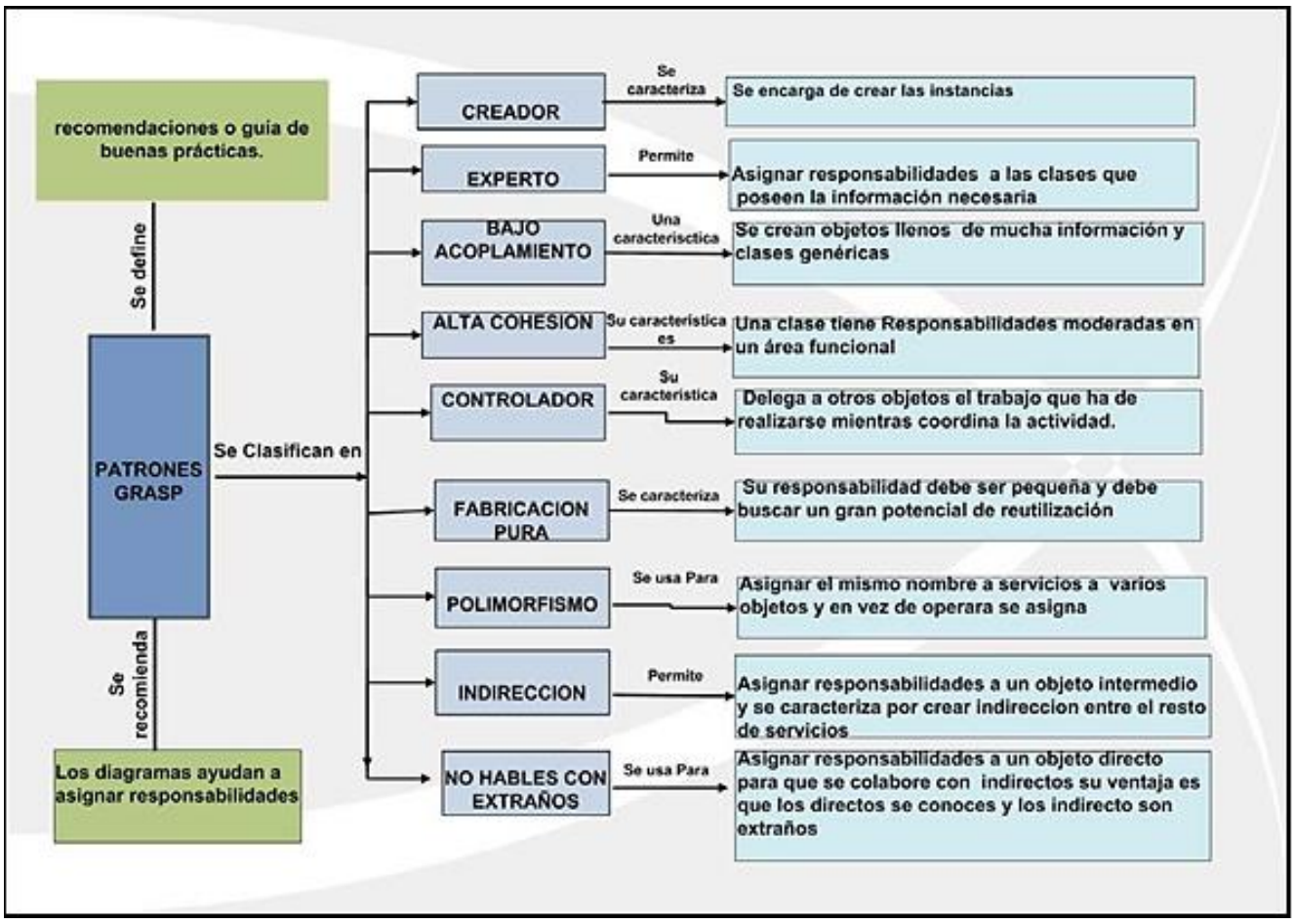


Figura 8 Mapa conceptual diseños de GRASP

Fuente: http://revistas.uniguajira.edu.co/rev/index.php/cei/article/view/252

A continuación, se abordarán los patrones GRASP identificados en el desarrollo de la propuesta de solución.

Controlador (Ortega 2021):

- Problema: Dentro de las operaciones del sistema se generan eventos que son producto de acciones derivadas de otras operaciones, y estos están encargados de realizar el enlace con cada uno de los procesos, por lo tanto, se debe definir quien se encargara, y en qué momento, atender dichos eventos.
- Solución: Con este patrón se intenta asignar responsabilidades, en el manejo de eventos, a las clases que tienen la capacidad de resolverlas manteniendo una alta cohesión con estas.

Experto (Ortega 2021):

- Problema: Cuáles son las clases y qué nivel de responsabilidad se les puede llegar a asignar dentro del sistema.
- Solución: Asignar responsabilidades a las clases cuyos objetos poseen la información necesaria para cumplirlas. Para poder cumplir con esto es posible que requiera información distribuida en otros objetos, es decir, otros expertos. Para asignar las responsabilidades es indispensable apoyarse en los diagramas de colaboración (UML), quienes son la base para la toma de decisiones en la elaboración del diseño orientado a objetos bajo los Patrones GRASP.

Este patrón se ve evidenciado en la clase: _CheckLogic

Creador (Ortega 2021):

- Problema: A cuáles clases asignar la responsabilidad de crear las instancias u objetos de otra clase.
- Solución: Utilizando los diagramas como el de colaboración se puede tener una relación de las actividades asignadas a las clases para que en conjunto con las siguientes condiciones se pueda definir claramente cuál o cuáles son los patrones creadores, permitiendo así tener una clara secuencia de las actividades que realiza la aplicación y el control en la generación de código excesivo que solo creara problemas en el momento del mantenimiento.

Este patrón se ve evidenciado en la clase: BallotsView

Alta Cohesión (Ortega 2021):

- Problema: Como crear clases que contengan la suficiente información, sin que se llegue al exceso de contenido, para que puedan funcionar de una manera óptima y así evitar la alta dependencia con otras que le suministren datos necesarios para su existencia.
- Solución: Una clase con alta cohesión posee una importante funcionalidad y poco trabajo, colabora con otros objetos para compartir tareas muy grandes. Su alto grado de funcionalidad permite simplificar el mantenimiento y los mejoramientos.

Este patrón se ve evidenciado en la clase: _Provider

Bajo Acoplamiento (Ortega 2021):

- Problema: Como evitar la gran dependencia generada entre las clases que forman parte del sistema sin que esto conlleve a traumatismo y la generación de código excesivo para compensar en algún momento la ruptura en la relación de clases muy dependientes.
- Solución: Se debe tener poca dependencia entre las clases, el tener una alta dependencia entre clases creará una gran cantidad de software, el objetivo de este patrón es invitar a la creación de clases con la suficiente información para que su relación con otras se reduzca y puedan en algún momento subsistir si una de la clase a las cuales se relaciona deja de existir.

Este patrón se ve evidenciado en las clases: CardContainer y AuthBackground.

2.6.2 *Patrones GoF*

Una especificación formal que permita definir en forma precisa la semántica de patrones de diseño, en particular patrones como GoF, puede ser la base para definir una herramienta que ayude a desarrollar software más seguro. En ciertos dominios, certificar el uso de métodos y técnicas es indispensable para asegurar la calidad del software producido (Cechich, Moore 2001).

La propuesta de solución hace uso de algunos de estos patrones, los cuales serán mostrados a continuación.

***Singleton*:** Es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase solamente tenga una instancia y proporcionar un punto de acceso global a ella, por ejemplo al consumir una API con una instancia única de un objeto que contenga la misma se hacen menos llamadas al servidor donde se aloja, por lo que esto ayuda a ahorrar datos de transferencia y el lag (tiempo que tarda el servidor en responder a la petición) que esto produce. Este patrón se implementa creando en nuestra clase un método que crea una instancia del objeto solo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con modificadores de acceso como protegido o privado) (Astorga et al. 2019).

Singleton restringe el número de instancias de una clase a únicamente una en todo el sistema (Calle Gallego 2016).

Este patrón tiene como objetivo asegurar que una clase determinada solo pueda poseer una instancia, proporcionando un método de clase único que la devuelva.

Adaptador (*Adapter*): Este patrón permite que una clase haga uso de una interfaz a la cual no tiene acceso, adaptando la primera especialmente para que sea convertida en otra parecida (Astorga et al. 2019).

Adapter le permite a un cliente trabajar con clases cuya interfaz no posee una interfaz permitida en un sistema específico. Este patrón adapta la interfaz del objeto externo al de las clases del sistema al que se integra (Calle Gallego 2016).

Este patrón de diseño adapta dos componentes que trabajan de manera diferente, pero que necesitan un traductor para que ambas puedan funcionar en conjunto. Se enfoca en crear una interfaz que traduzca o mapee un componente con otro, por medio de la encapsulación de una clase para que la otra se comunique por medio de la interfaz, ya que ambas clases no se pueden comunicar directamente.

2.7 Diagrama de clase del diseño

En la ingeniería de software, un diagrama de clases en el Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que describe la estructura de un sistema, mostrando clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. En un diagrama de clases se pueden distinguir principalmente dos elementos: clases y sus relaciones (Fajardo 2019).

A continuación, en la figura 8, se muestra el diagrama de clases del diseño.

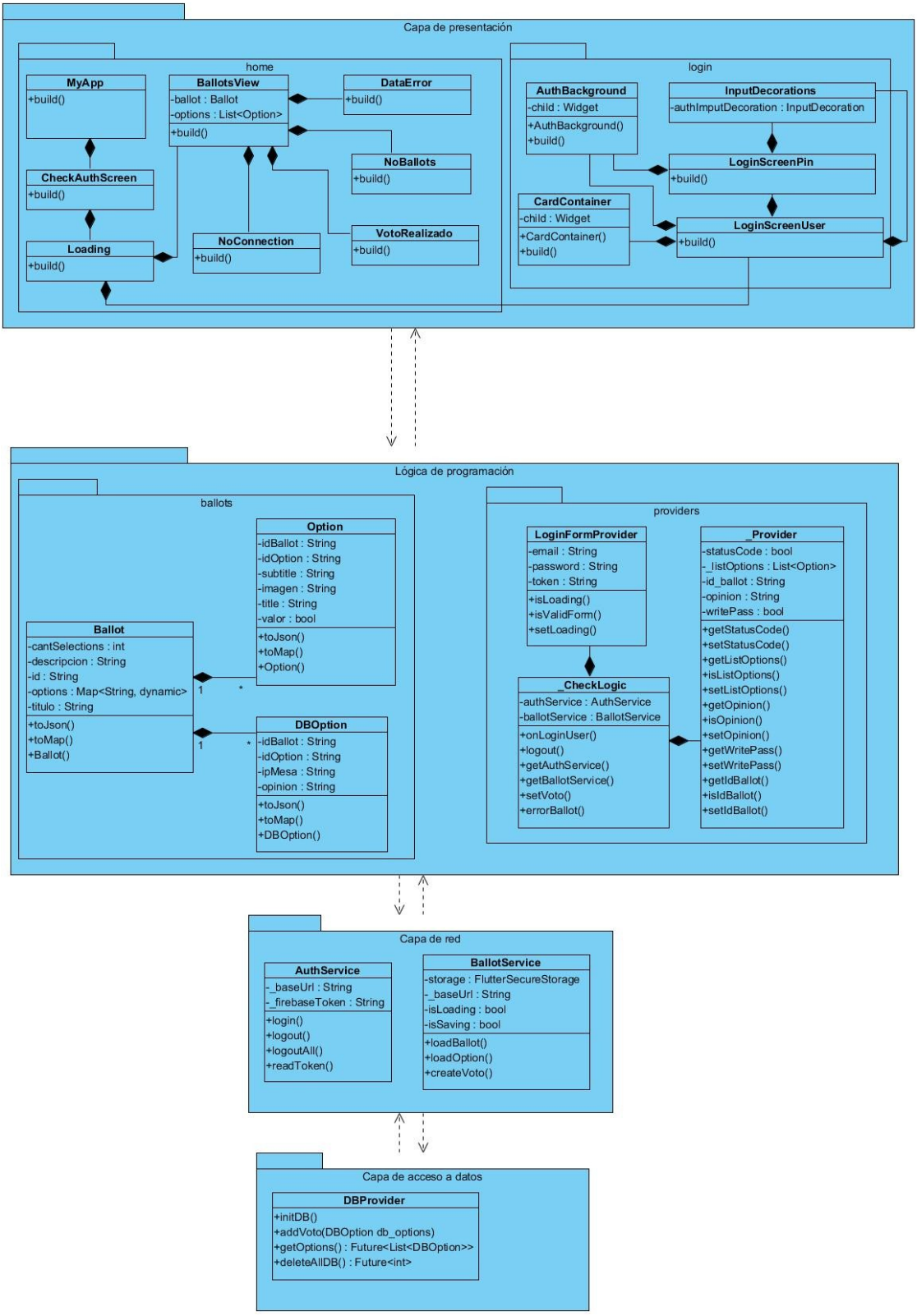


Figura 9 Diagrama de clases del sistema

2.8 Modelo entidad-relación de los datos

Lo que hace al modelaje entidad-relación tan universal es que no está enfocado al diseño de un modelo de bases de datos particular y que dispone de técnicas de transformación para muchos de ellos. El problema central del diseño lógico, como se le concibe hoy día, es el de la correcta y completa representación de propiedades de un dominio de aplicación y el de la transformación de dicha representación en especificaciones de almacenamiento, recuperación y desarrollo de programas de aplicación. El modelaje entidad-relación se centra en la representación de propiedades del dominio de forma independiente de la implementación y en proveer técnicas de transformación a distintos modelos de bases de datos (Hilario 2007).

El modelo entidad-relación de los datos es un modelo conceptual que representa la vista estructural de datos de un sistema a través de entidades y relaciones entre las entidades pertinentes de un sistema y se utiliza en el proceso de diseño de bases de datos. (Rafael, Wilson 2017)

A continuación, en la figura 9 se muestra el modelo entidad-relación de los datos.

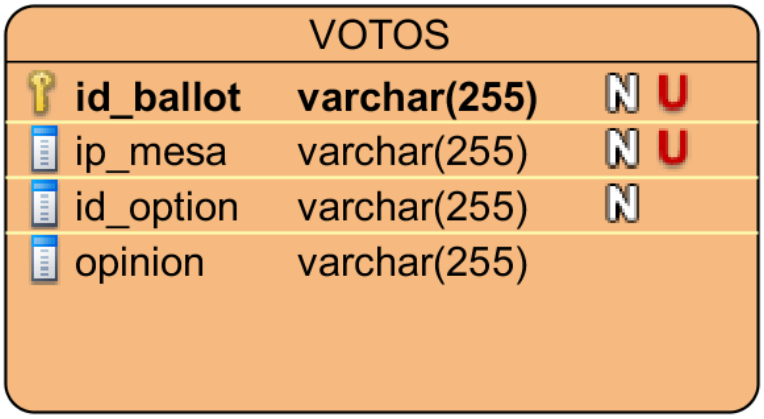


Figura 10 Modelo entidad-relación de los datos

2.9 Conclusiones del capítulo

A partir del desarrollo del presente capítulo es posible afirmar que la captura de los requisitos y la elaboración de las historias de usuario correspondientes permitió comprender mejor las funcionalidades de la aplicación que se desea desarrollar y tener una visión más clara sobre las características y restricciones que debe cumplir el mismo. Igualmente, la confección de los artefactos definidos en la disciplina Análisis y Diseño de la metodología AUP en su variante UCI (AUP-UCI), permitió definir las bases necesarias y posibilitó que se obtuvieran de una forma concreta y detallada

las relaciones que se establecen entre las entidades del sistema. Además, la investigación y definición de la arquitectura N-Capas y los patrones de diseño GRASP y GoF contribuyeron al diseño de la propuesta de solución, proporcionando una estructura para la misma y posibilitando lograr una implementación centrada en el diseño realizado.

Capítulo 3. Implementación y validación de la propuesta de solución

En el presente capítulo se tratan los temas referentes a la implementación y validación del sistema, donde figura el diagrama de despliegue que contiene los componentes del sistema, el diagrama de componentes, el cual muestra las relaciones entre los elementos que dan sustento a las funcionalidades del software, se describen los estándares de codificación definidos, se realizarán las pruebas a la solución para la validación del sistema y se describirán las funcionalidades obtenidas en este.

3.1 Diagrama de despliegue

Es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes (Giraldo Gómez, Acevedo O., Moreno N. 2011).

La realización del modelo de despliegue del Módulo de Registro de Electores, permite la obtención de una visión más realista de los recursos que se necesitarán para la implantación del sistema en el negocio real, permitiendo definir de forma ilustrativa qué actividades se desarrollarán específicamente en cada puesto de trabajo.

A continuación, en la figura 10, se muestra el diagrama de despliegue para el sistema.

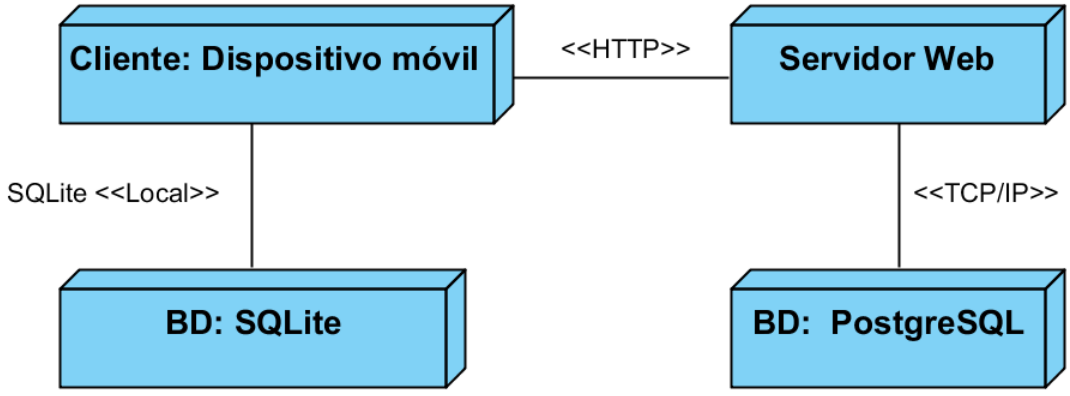


Figura 11 Diagrama de despliegue

3.1.1 *Descripción del diagrama de despliegue*

Cliente: Dispositivo Móvil: representa el hardware y software donde se desplegará la solución propuesta, en este estará instalada la aplicación móvil para que el usuario acceda al servidor.

BD: SQLite: se refiere a nodo determinado donde serán guardados los datos. . En el mismo estarán almacenados todos los datos de los votos realizados por el usuario en el momento en que falla la conexión con el servidor.

Servidor Web: Es la herramienta principal para ejecutar la lógica de negocio en el lado del servidor. Es el responsable de ejecutar el código de las páginas servidor. la conexión del sistema con el dispositivo móvil que se utiliza HTTPS, del inglés (*Hypertext Transfer Protocol Secure*)

BD: PostgreSQL: se refiere a nodo determinado donde serán guardados los datos. En el mismo estarán almacenados todos los datos recopilados por todos los nodos.

3.2 Diagrama de componentes

Los diagramas de componentes constituyen en esta disciplina el modelo de implementación, al describir los componentes a construir, su organización y dependencias. Un componente es una parte física y reemplazable de un sistema que se conforma por un conjunto de interfaces y proporciona la realización de dicho conjunto. Estos diagramas se usan para modelar los elementos físicos que pueden hallarse en un nodo, por lo que empaquetan elementos como clases, colaboraciones e interfaces.

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes de software. Muestra la organización y las dependencias entre un conjunto de componentes y sus relaciones de manera gráfica a través del uso de nodos y arcos entre estos (Burgués 2018).

A continuación, en la figura 11, se muestra el diagrama de Componente.

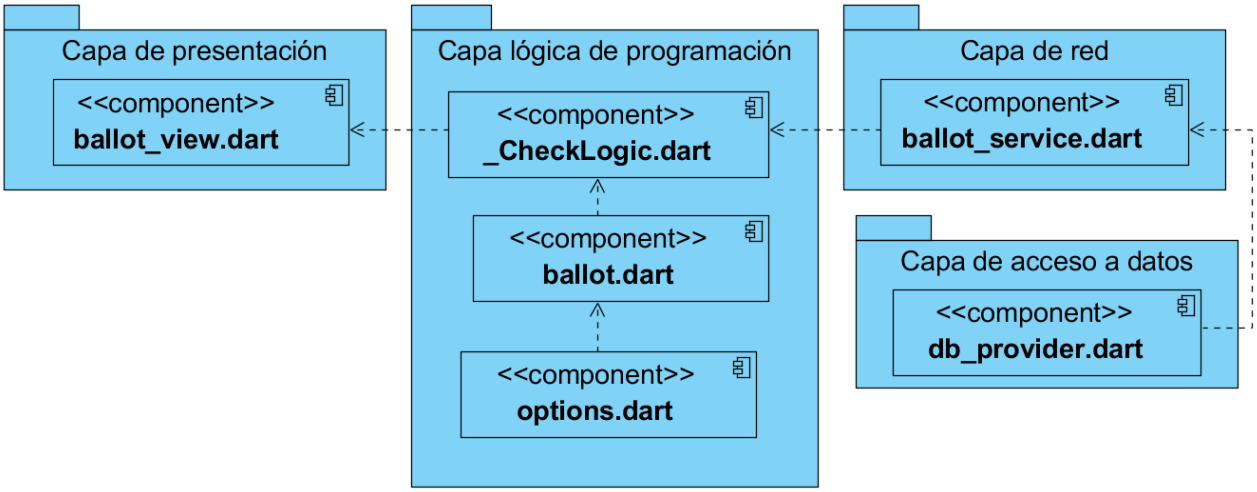


Figura 12 Diagrama de Componente

3.3 Estándar de codificación

En la propuesta de solución se utilizan los estándares de codificación, los cuales son muy importantes para cualquier proyecto de desarrollo, pues establecen la estructura del lenguaje, el orden y normas de estilos. Contribuyen a asegurar que el código contenga menos errores, pueda ser mantenido fácilmente y posea una alta calidad. Un estándar de codificación son reglas que se siguen para la escritura del código fuente. A continuación, se muestran los estándares de codificación utilizados en la solución:

- No se usarán nombres de variables que coincidan con palabras reservadas. Los nombres de las variables y los métodos responderán al estilo de capitalización *CamelCase,* con el cual se capitaliza la primera letra de las palabras excepto la primera palabra. Ejemplo: myApp. Los nombres de las variables son cortos, pero con significados lógicos, capaces de permitir a un observador identificar su función (Ver Figura 12).
- El código será tabulado y espaciado a través del formato que aplica la combinación de teclas *Ctrl + S* definida en la configuración del IDE Visual Studio Code.
- Los comentarios deben ser añadidos de forma que resulten prácticos, para explicar el flujo del código y el propósito de las funciones y variables. Los comentarios de una solo línea deben emplear los caracteres “//” al inicio. Los comentarios que ocupen múltiples líneas deben comenzar con los caracteres “/*” y concluir con “*/” (Ver Figura 12).
- Se utilizará el estilo de capitalización PascalCase donde los nombres de las clases presentan la primera letra en mayúscula, en caso de ser un nombre compuesto, la inicial de cada palabra se representa en mayúscula y cada salto de palabras debe iniciar con mayúsculas (Ver Figura 12).
- Se debe colocar un espacio antes y después de los operadores binarios (+, -, +=, !=, ==, etc.)

```
5   class Ballot {
6     //declaración de variables
7     int cantSelections;
8     String description;
9     String id;
10    Map<String, dynamic> options;
11    String title;
12
13    //Constructor
14    Ballot({
15      required this.cantSelections,
16      required this.description,
17      required this.id,
18      required this.options,
19      required this.title,
20    });
21  }
```

Figura 13 Estándar de codificación

3.4 Pruebas realizadas a la solución

El concepto de pruebas de calidad de software permite brindar productos con altos estándares de calidad y con una disminución de fallos en estos. Con la claridad de la importancia del proceso de pruebas de calidad en los productos software se eleva la calidad y fiabilidad dentro del ciclo de vida de un proyecto. Asimismo, al implementar un proceso de pruebas de calidad de software se genera un control y seguimiento a los defectos o faltas y a los fallos, de manera que las soluciones que se generen tengan un mayor nivel de calidad (Paz 2016).

3.4.1 *Pruebas unitarias*

El primer nivel de las pruebas es el de la prueba unitaria o de componente que consiste en la verificación de unidades del software de forma aislada, es decir, probar el correcto funcionamiento de una unidad de código. Se indica que una unidad de código es considerada una unidad de programa, como una función o método de una clase que es invocada desde fuera de la unidad y que puede invocar otras unidades. Es por ello que hay que probar que cada unidad funcione separada de las demás unidades de código (Sanchez Peño 2015).

Las pruebas de unidad se realizan mediante el método de caja blanca o estructural, denominada a veces prueba de caja de cristal, tienen el objetivo de garantizar que las operaciones internas se

realizan de acuerdo a las especificaciones y que se han probado todos los componentes internos de manera adecuada, además de tener por objetivo el aislamiento de partes del código y la demostración de que estas partes no contienen errores.

Las pruebas unitarias se realizaron sobre los servicios del sistema para validar que las salidas son correctas y asegurar al desarrollador que su solución no presenta errores en la lógica de programación, pues las respuestas son acertadas ante una entrada de datos determinada por el probador. A continuación, se muestran las pruebas unitarias aplicadas a los servicios (ver figuras 13 y 14)

```
void main() {
    test('Prueba cargar boleta', () async {
      final String _baseUrl = 'votos-en-sala-default-rtdb.firebaseio.com';
      final url = Uri.https(_baseUrl, 'active_ballot.json', {'auth': await storage.read(key: 'token') ?? ''});
      final client = MockClient();
      // Usa Mockito para devolver una respuesta exitosa cuando llama al
      // http.Client proporcionado.
      when(client.get(url))
          .thenAnswer((_) async => http.Response(resp, 200));
      expect(await fetchPost(client), respBallot);
    });

}

Future<Ballot> fetchPost(http.Client client) async {
  final url = Uri.https(_baseUrl, 'active_ballot.json', {'auth': await storage.read(key: 'token') ?? ''});
  final response = await client.get(url);
  if (response.statusCode == 200) {
    // Si la llamada al servidor fue exitosa, analice el JSON
    return Ballot.fromJson(json.decode(response.body));
  } else {
    // Si esa llamada no fue exitosa, lance un error.
    throw Exception('Error al cargar post');
  }
}
```

Figura 14 Prueba unitaria a la funcionalidad cargar boleta.

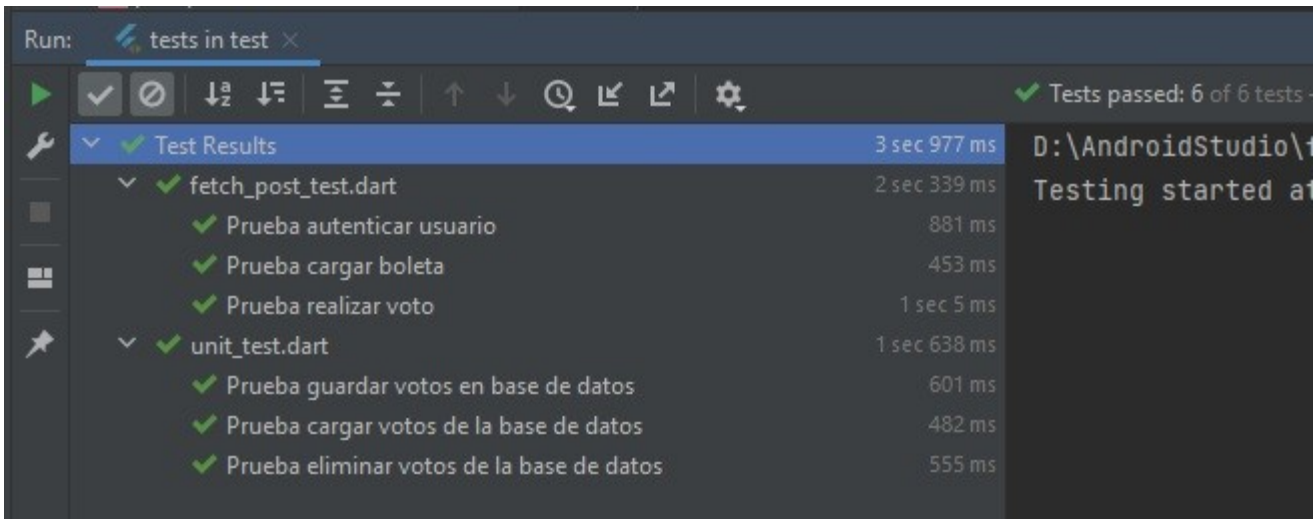


Figura 15 Pruebas unitarias realizadas

Las pruebas unitarias son prácticas para verificar el comportamiento de una sola función, método o clase. El paquete **test** proporciona el marco principal para escribir pruebas unitarias

Para la realización de las pruebas unitarias se utilizaron las librerías: **test** en su versión 1.22.0 y **Mockito** en su versión 5.3.2**,** debido a que, en ciertos casos, las pruebas unitarias pueden depender de las clases que obtienen datos de servicios web en vivo o bases de datos. Esto es inconveniente por algunas razones:

- Llamar a servicios en vivo o bases de datos ralentizará la ejecución de la prueba.
- Una prueba de aprobación puede comenzar a fallar si un servicio web o una base de datos arroja resultados inesperados. Esto se conoce como una “prueba escamosa”.
- Es difícil probar todos los posibles escenarios de éxito y falla utilizando un servicio web en vivo o una base de datos.

Por lo tanto, en lugar de confiar en un servicio web o base de datos en línea, puedes “simular” esas dependencias. Los Mocks nos permiten emular un servicio web en vivo o una base de datos y devolver resultados específicos según la situación.

En la figura 13, se prueba de forma unitaria la función fetchPost de la receta Obtener datos desde internet, en este caso cargar boleta. Para probar esta función, necesitas hacer dos cambios:

Proporciona un http.Client a la función. Esto te permitirá proporcionar el http.Client correcto según la situación. Para proyectos de Flutter y del lado del servidor, puedes proporcionar un http.IOClient. Para

las aplicaciones del navegador, puedes proporcionar un http.BrowserClient. Para las pruebas, proporciona un http.Client simulado.

Se utiliza el client proporcionado para buscar datos de Internet, en lugar del método estático http.get, que es difícil de simular. Si piensas en la función fetchPost hará una de estas dos cosas:

- Devolver un Post si la llamada http tiene éxito
- Lanzar una Excepción si falla la llamada http

Por lo tanto, quieres probar estas dos condiciones. Podemos usar la clase MockClient para devolver una respuesta “Ok” para la prueba de éxito y una respuesta de error para la prueba fallida. Para lograr esto, usa la función when proporcionada por Mockito.

3.4.2 *Pruebas funcionales*

Este tipo de prueba se basa en las funcionalidades de un sistema que se describen en la especificación de requisitos, es decir, lo que hace el sistema. También pueden estar no documentadas, pero se requiere un nivel de experiencia elevado para interpretar estas pruebas (Sanchez Peño 2015).

El método de prueba de caja negra, también denominada prueba de comportamiento, se centra en los requisitos funcionales del software. O sean, permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del programa (Benitez, Calderón, Manresa 2019).

Para el diseño de estos casos de pruebas se tuvo en cuenta la técnica de Partición equivalente. Esta técnica divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Es por esto que por cada requisito se obtuvo la Descripción de las variables (DV) para luego dividir el campo de entrada en clases de datos válidos e inválidos. Una vez definida la DV, se procede a diseñar los escenarios de pruebas para cada caso de prueba

Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato.

A continuación, la tabla 7 y 8 se presentan las descripciones de las variables de los casos de pruebas realizados:

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	Usuario	Campo de texto	No	Contiene el usuario que desea acceder al sistema

2	contraseña	Campo de texto	No	Contiene la contraseña que desea acceder al sistema

Tabla 7 Descripción de variable caso de prueba "Autenticar usuario"

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	Opciones	Lista de Radio Button	No	Contiene la elección del usuario
2	Opinión	Campo de texto	Si	Contiene la opinión del usuario con respecto al voto realizado

Tabla 8 Descripción de variable caso de prueba "Realizar votación"

A continuación, la tabla 9 presenta el Diseño de Casos de Prueba perteneciente al requisito autenticarse:

Escenario	Descripción	Usuario	Contraseña	Respuesta del sistema	Flujo central
EC 1.1 Datos Válidos	El usuario introduce los valores de usuario y contraseña válidos	**V** christianrvdv	**V** 123456*	Se cambia de pantalla y se muestra una boleta en caso de esta estar activa, en caso de no estar activa muestra una pantalla con un texto informando que no se encuentran boletas activas.	1 Seleccionar capo usuario. 2 Introducir usuario válido. 3 Seleccionar campo contraseña. 4 Introducir contraseña válida. 5 Seleccionar botón acceder.
EC 1.2 Datos inválidos	El usuario introduce el usuario correctamente , pero la contraseña no es correcta	**V** christianrvdv	**I** 123	Muestra un diálogo con la información de credenciales inválidas.	1 Seleccionar capo usuario. 2 Introducir usuario válido. 3 Seleccionar campo contraseña. 4 Introducir contraseña inválida. 5 Seleccionar botón acceder.
EC 1.3 Campos en blanco	El usuario introduce el usuario correctamente , pero deja el campo contraseña en blanco	**V** christianrvdv	**I**	Muestra debajo del campo vacío letras en rojo indicando que el campo no puede estar vacío.	1 Seleccionar capo usuario. 2 Introducir usuario válido. 3 Dejar campo contraseña en blanco. 4 Seleccionar botón acceder.

Tabla 9 Diseño de caso de prueba "Autenticar Usuario"

Escenario	Descripción	Lista Radio Button	Opinión	Respuesta del sistema	Flujo central
EC 1.1 Datos correctos	El usuario selecciona uno o varios de las opciones brindadas en la boleta	Selecciona al menos un Radio Button		El sistema realiza el voto satisfactoriamente .	1 Seleccionar el número de opciones requeridos en la boleta. 2 Seleccionar el botón Votar.
		V	N/A		
EC 1.2 Datos incorrectos	El usuario no selecciona uno ninguna de las opciones brindadas en la boleta			Muestra un diálogo indicando que debe marcar un número determinado de opciones en la boleta.	1 No seleccionar ninguna de las opciones de la boleta. 2 Seleccionar el botón Votar
		I	N/A		

Tabla 10 Diseño de caso de prueba "Realizar votación"

3.4.3 *Resultados de las pruebas funcionales*

Fueron probadas todas las funcionalidades del sistema que se corresponden con los casos de prueba de la aplicación. Para lograr la validación de los requisitos funcionales se efectuaron dos iteraciones donde se encontraron en general 9 no conformidades, siendo en su mayor parte errores de faltas de ortografía y funcionalidades. En general fueron identificadas 8 no conformidades en la primera iteración realizada y 1 en la segunda iteración; al realizar una tercera iteración no fue encontrada ninguna no conformidad, una vez terminadas todas las iteraciones, fueron solucionadas las no conformidades en su totalidad. En la figura 15 se observan los resultados de cada una de las iteraciones de pruebas que se efectuaron a la aplicación móvil voto en sala.

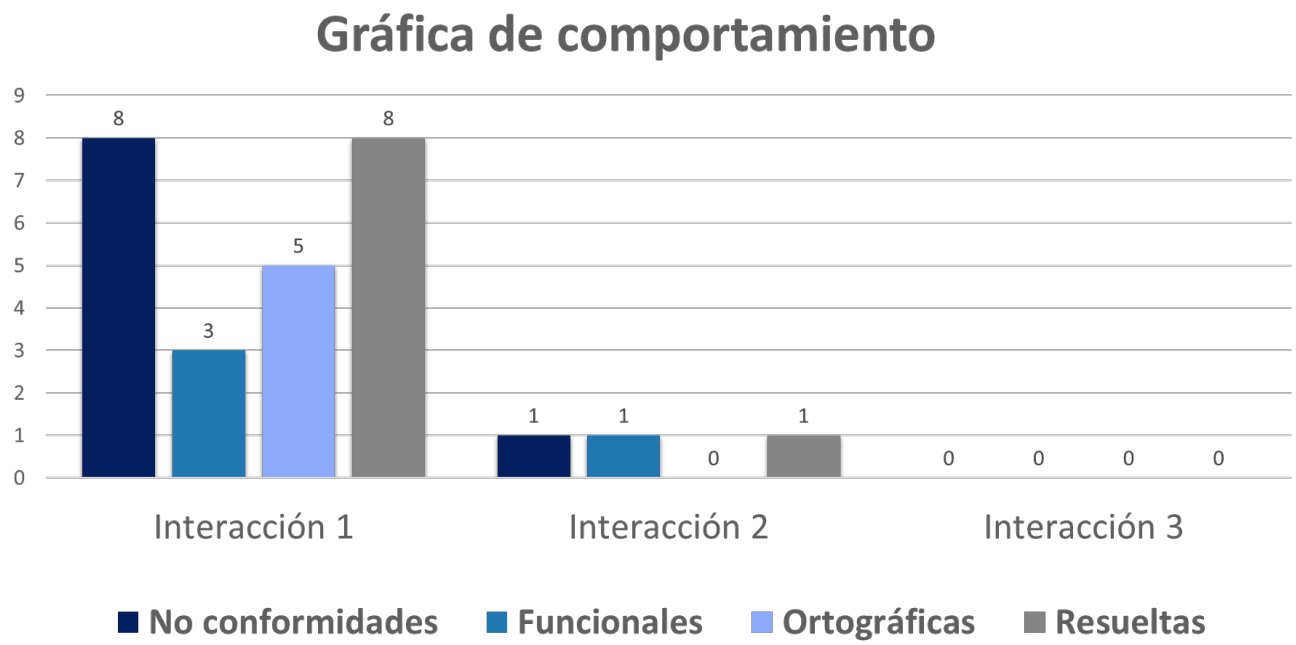


Figura 16 Gráfica de comportamiento de las no conformidades según las interacciones

3.5 Conclusiones del capítulo

El uso de estándares de codificación permitió guiar el proceso de implementación, obteniendo un código más estructurado y entendible. Además, los artefactos generados en la fase de implementación y prueba constituyen elementos fundamentales en el desarrollo de la solución propuesta. Con el diagrama de componentes se representa la vista lógica de la implementación, posibilitando la comprensión más a fondo del proceso. De igual forma con el diagrama de despliegue se obtiene una visión de la distribución de los nodos que serán necesarios para su expansión. La ejecución de las pruebas unitarias al código de la aplicación, así como de las pruebas funcionales, permitieron presentar los resultados arrojados en cada iteración logrando finalmente una aplicación que responde al conjunto de funcionalidades identificadas.

Conclusiones

Con la realización del presente trabajo de diploma se ha cumplido con el objetivo general propuesto, así como con las tareas de la investigación definidas, llegando a las siguientes conclusiones:

- Luego de un análisis de las aplicaciones que tienen como objetivo la votación en sala, se demostró que no se puede utilizar ninguna de estas como propuesta de solución, aunque sirvieron de guía para el desarrollo de la ampliación móvil.
- Las herramientas, metodología y tecnologías puestas en práctica, para el desarrollo de la solución propuesta, permitieron la obtención de aplicaciones que resuelven los problemas planteados.
- La fase de implementación permitió establecer estándares de codificación con el objetivo de identificar los elementos de código que estarán presentes en dicho proceso, además de obtener una versión funcional de este.
- La implementación de las funcionalidades de la herramienta, acorde a las pautas de diseño, garantizó el cumplimiento de lo establecido en los objetivos específicos de la investigación.
- La realización de las pruebas definidas, garantizan la calidad del producto, además de lograr identificar un conjunto de no conformidades, las cuales se fueron erradicando en cada una de las iteraciones pautadas.
- Se obtuvo una aplicación móvil desarrollada con tecnologías libres.

Recomendaciones

Teniendo en cuenta lo anteriormente planteado se recomienda:

- Que se perfeccione el sistema de manera que pueda ser utilizado en cualquier institución.
- Agregar la funcionalidad de que te muestre en pantalla los votos que se van realizando en tiempo real.

Referencias bibliográficas

Agile project management with Scrum, 2011. en línea. [Accedido 10 junio 2022]. Recuperado a partir de: https://www.pmi.org/learning/library/agile-project-management-scrum-6269

AGUILAR, Alma Lilia Sapién, GUTIÉRREZ, María y HOWLET, Laura Cristina Piñón, 2017. Voto electrónico: confiabilidad y utilización de tecnología. *Investigación y Ciencia: de la Universidad Autónoma de Aguascalientes*. 2017. No. 70, pp. 77-83.

Appsamblea - Votaciones online seguras, rápidas y fáciles, 2018. *Appsamblea - Votaciones online seguras, rápidas y fáciles*. en línea. [Accedido 11 junio 2022]. Recuperado a partir de: https://appsamblea.com/

Arquitectura del sistema operativo IOS – Acervo Lima, 2020. en línea. [Accedido 8 junio 2022]. Recuperado a partir de: https://es.acervolima.com/arquitectura-del-sistema-operativo-ios/

ARTICA NAVARRO, Robertho Lutty, 2014. Desarrollo de aplicaciones móviles. . 2014.

Association Voting – Online Voting Made Simple, 2016. en línea. [Accedido 11 junio 2022]. Recuperado a partir de: https://www.associationvoting.com/

ASTORGA, José Alejandro Ibarra, TIRADO, Jorge Luis Osuna, RAMÍREZ, Ricardo Ulises Reyes, MENDOZA, José Carlos Lepe y GARZÓN, Alvaro Peraza, 2019. CLASIFICACIÓN DE LOS PATRONES DE DISEÑO IDÓNEOS EN PROGRAMACIÓN ANDROID. *Revista Digital de Tecnologías Informáticas y Sistemas*. en línea. 17 diciembre 2019. Vol. 3, no. 3. [Accedido 24 septiembre 2022]. Recuperado a partir de: https://redtis.org/index.php/Redtis/article/view/33

BENITEZ, Karla García, CALDERÓN, Leyanys Acosta y MANRESA, Yosbel Lázaro Guirola, 2019. Implementación de pruebas de software para un sistema para la gestión de datos de urocultivo. *Serie Científica de la Universidad de las Ciencias Informáticas*. 21 mayo 2019. Vol. 12, no. 1, pp. 54-68.

BUENO, Luis Ignacio Lizcano, 2002. UML: Un lenguaje de modelo de objetos. *Respuestas*. 2002. Vol. 7, no. 1, pp. 25-29.

BURGUÉS, Julián Esteban Gracia, 2018. *Aprende a Modelar Aplicaciones con UML- Tercera Edición*. IT Campus Academy. ISBN 978-1-985133-43-3. Google-Books-ID: 2cJKDwAAQBAJ

CALLE GALLEGO, Johnathan Mauricio, 2016. Identificación de patrones de diseño para software científico a partir de esquemas preconceptuales. en línea. 18 noviembre 2016. [Accedido 24 septiembre 2022]. Recuperado a partir de: https://repositorio.unal.edu.co/handle/unal/59127Accepted: 2019-07-02T15:25:37Z

CASTILLO, José Dimas Luján, 2019. *Desarrollo de aplicaciones Android con Android Studio: Conoce android studio*.

CECHICH, Alejandra y MOORE, Richard, 2001. Una especificación precisa para patrones GoF. En: *III Workshop de Investigadores en Ciencias de la Computación*. en línea. 2001. [Accedido 24 septiembre 2022]. Recuperado a partir de: http://sedici.unlp.edu.ar/handle/10915/21723

CHININ, Zorem, 2012. GUIA DE ARQUITECTURA EN N-CAPAS: Estilos Arquitecturales. *GUIA DE ARQUITECTURA EN N-CAPAS*. en línea. 14 octubre 2012. [Accedido 23 septiembre 2022]. Recuperado a partir de: http://arquitecturancapas.blogspot.com/2012/10/estilos-arquitecturales.html

CORTEZ, Alberto, GARIS, Ana Gabriela y RIESCO, Daniel Eduardo, 2011. Formalización de patrones de diseño de comportamiento. En: *XIII Workshop de Investigadores en Ciencias de la Computación*. en línea. 2011. [Accedido 23 septiembre 2022]. ISBN 978-950-673-892-1. Recuperado a partir de: http://sedici.unlp.edu.ar/handle/10915/20072

CORTEZ, Alberto, NAVEDA, Claudia, GARIS, Ana y RIESCO, D., 2020. Especificación del patrón de diseño Memento a través de un Perfil UML. . 24 junio 2020.

DELÍA, Lisandro Nahuel, GALDAMEZ, Nicolás, THOMAS, Pablo Javier y PESADO, Patricia Mabel, 2013. Un análisis experimental de tipo de aplicaciones para dispositivos móviles. En: *XVIII Congreso Argentino de Ciencias de la Computación*. 2013.

Desclasificación basada en tipos en DART: Implementación y elaboración de herramientas de inferencia, 2018. en línea. [Accedido 10 junio 2022]. Recuperado a partir de: https://repositorio.uchile.cl/handle/2250/168368

DÍAZ, Emilio Antonio Duharte, 2022. EL SISTEMA ELECTORAL Y LA REFORMA POLÍTICA INTEGRAL EN CUBA. *International Journal of Cuban Studies*. 2022. Vol. 14, no. 1, pp. 36-62.

Elecciones, 2014. en línea. [Accedido 4 junio 2022]. Recuperado a partir de: http://www.enciclopedia-juridica.com/d/elecciones/elecciones.htm

ElectionBuddy, 2018. *ElectionBuddy*. en línea. [Accedido 11 junio 2022]. Recuperado a partir de: https://electionbuddy.com/

FAJARDO, Adolfo Vega, 2019. Método para complementar la generación de códigos de aplicaciones web desde el diagrama de clases UML. *Ciencia y Tecnología*. 2019. pp. 35-64. DOI 10.18682/cyt.v19i19.1864.

GARCÍA-HOLGADO, A., MORENO GARCÍA, M. N., VÁZQUEZ-INGELMO, A. y GARCÍA-PEÑALVO, F. J., 2020. UML. Unified Modeling Language. en línea. 26 febrero 2020. [Accedido 23 septiembre 2022]. DOI 10.5281/zenodo.3688621. Accepted: 2020-02-26T16:29:03Z

GIRALDO GÓMEZ, Gloria Lucia, ACEVEDO O., Juan F. y MORENO N., David A., 2011. Una ontología para la representación de conceptos de diseño de software. en línea. 2011. [Accedido 11 octubre 2022]. Recuperado a partir de: https://repositorio.unal.edu.co/handle/unal/34966Accepted: 2019-06-27T23:48:44Z

GUEVARA SORIANO, Anaid, 2010. Dispositivos Móviles. en línea. 6 agosto 2010. [Accedido 23 septiembre 2022]. Recuperado a partir de: https://www.ru.tic.unam.mx/xmlui/handle/123456789/1731Accepted: 2017-08-03T16:01:28Z

Herramienta CASE Visual Paradigm - MARCO TEÓRICO, 2021. en línea. [Accedido 10 junio 2022]. Recuperado a partir de: https://1library.co/article/herramienta-case-visual-paradigm-marco-te%C3%B3rico.qvlerrgy

HIGUERA FOMBUENA, Jorge y GONZÁLEZ ANTONA, Daniel, 2017. *Plataforma de aplicaciones con Dart*. ETSI_Sistemas_Infor.

HILARIO, Ana Belén Ríos, 2007. Análisis de la funcionalidad de la parte descriptiva de las reglas de catalogación españolas mediante la aplicación del modelo entidad-relación (ER). En: *Anales de documentación*. Facultad de Comunicación y Documentación y Servicio de Publicaciones de la …. 2007. pp. 345-359.

JACOBSON, Ivar, BOOCH, Grady y RUMBAUGH, James, 2000. *El proceso unificado de desarrollo de software*. Addison Wesley Madrid.

KUMANO, Mayumi Yasunaga, 2017. Las nuevas tecnologías de votación: ¿una puerta abierta a la injerencia externa? *bie3: Boletín IEEE*. 2017. No. 5, pp. 703-716.

La guía sencilla para la diagramación de UML y el modelado de la base de datos, 2019. en línea. [Accedido 10 junio 2022]. Recuperado a partir de: https://www.microsoft.com/es-ww/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling

MARCISZACK, Marcelo Martín, MORENO, Juan Carlos, SÁNCHEZ, Claudia Evangelina, MEDINA, Oscar Carlos, DELGADO, Andrea Fabiana y CASTRO, Claudia Susana, 2019. *Patrones en la construcción del modelo conceptual para sistemas de información*. en línea. edUTecNe. [Accedido 23 septiembre 2022]. ISBN 978-987-1896-96-7. Recuperado a partir de: http://ria.utn.edu.ar/xmlui/handle/20.500.12272/5062

MARTÍNEZ, José, MOLINA, Jesús y GARCÍA, Pedro, 1999. *Ana Arquitectura para una Herramienta de Patrones de Diseño.*

MEJÍA, Juan Camilo Giraldo, AGUDELO, Fabio Alberto Vargas y GIL, Kelly Garzón, 2021. Marco de Trabajo para Seleccionar un Patrón Arquitectónico en el Desarrollo de Software. . julio 2021. No. E43, pp. 568-581.

MENZINSKY, Alexander, 2018. *Historias de usuario. Ingeniería de requisitos ágil*.

MERIZALDE MEDINA, Mildred Veronica, 2018. ARTÍCULO CIENTÍFICO: Aplicación de la metodología Scrum en la gestión y desarrollo de proyectos. Caso de estudio: Empresas Consultoras de Software de Guayaquil. en línea. 2018. [Accedido 10 junio 2022]. Recuperado a partir de: http://biblioteca.uteg.edu.ec/xmlui/handle/123456789/239Accepted: 2019-10-29T05:23:20Z

MONTERRUBIO-HERNANDEZ, Elias, 2019. Sistema Operativo. *Con-Ciencia Serrana Boletín Científico de la Escuela Preparatoria Ixtlahuaco*. en línea. 5 enero 2019. Vol. 1, no. 1. [Accedido 23 septiembre 2022]. Recuperado a partir de: https://repository.uaeh.edu.mx/revistas/index.php/ixtlahuaco/article/view/3672

MORALES, Daynelis Brito, BORRELL, Johan Bravo y ARMAS, Lijandy Jiménez, 2019. Aplicación móvil para el análisis de la información captada en SIGEv3.0. *Serie Científica de la Universidad de las Ciencias Informáticas*. 9 julio 2019. Vol. 12, no. 6, pp. 55-71.

ORTEGA, Gilberto Andrés Vargas, 2021. Lineamientos para el diseño de aplicaciones web soportados en patrones GRASP. *Ciencia e Ingenieria*. 20 noviembre 2021. Vol. 8, no. 2, pp. e5716304-e5716304.

PAZ, Julián Andrés Mera, 2016. Análisis del proceso de pruebas de calidad de software. *Ingeniería Solidaria*. 1 octubre 2016. Vol. 12, no. 20, pp. 163-176. DOI 10.16925/in.v12i20.1482.

POLANCO, Kristel Malave y TAIBO, José Luis Beauperthuy, 2011. “Android” el sistema operativo de Google para dispositivos móviles. *Negotium: revista de ciencias gerenciales*. 2011. Vol. 7, no. 19, pp. 79-96.

RAE, 2020a. Definición de votación ordinaria - Diccionario panhispánico del español jurídico - RAE. *Diccionario panhispánico del español jurídico - Real Academia Española*. en línea. 2020. [Accedido 4 junio 2022]. Recuperado a partir de: https://dpej.rae.es/lema/votaci%C3%B3n-ordinaria

RAE, 2020b. Definición de aplicación móvil - Diccionario panhispánico del español jurídico - RAE. *Diccionario panhispánico del español jurídico - Real Academia Española*. en línea. 2020. [Accedido 5 junio 2022]. Recuperado a partir de: https://dpej.rae.es/lema/aplicaci%C3%B3n-m%C3%B3vil

RAFAEL, Capacho, José y WILSON, Nieto Bernal, 2017. *Diseño de bases de datos*. Universidad del Norte. ISBN 978-958-741-825-5. Google-Books-ID: TLBJDwAAQBAJ

RÍOS, Jimmy Rolando Molina, ORDÓÑEZ, Mariuxi Paola Zea, SEGARRA, María José Contento y ZERDA, Fabricio Gustavo García, 2017. Estado del arte: Metodologías de desarrollo en aplicaciones web. *3c Tecnología: glosas de innovación aplicadas a la pyme*. 2017. Vol. 6, no. 3, pp. 54-71.

RÍOS, Jimmy Rolando Molina, ORDÓÑEZ, Mariuxi Paola Zea, SEGARRA, María José Contento y ZERDA, Fabricio Gustavo García, 2018. Comparación de metodologías en aplicaciones web. *3C Tecnología: glosas de innovación aplicadas a la pyme*. 2018. Vol. 7, no. 1, pp. 1-19.

RONDON SUAREZ, Lidis Mayerly, 2019. Calidad en el levantamiento de requerimientos en proyectos de software. en línea. 13 junio 2019. [Accedido 23 septiembre 2022]. Recuperado a partir de: http://repository.unimilitar.edu.co/handle/10654/31989Accepted: 2019-08-29T17:16:47Z

SÁNCHEZ, F. Barrón, VENEGAS, HA Montes y ROMERO, JR Marcial, 2018. Técnicas para el levantamiento de requerimientos en el desarrollo de un sistema de información. *Pistas Educativas*. 2018. Vol. 36, no. 114.

SANCHEZ PEÑO, José Manuel, 2015. Pruebas de software. fundamentos y técnicas. . 2015.

SÁNCHEZ RODRÍGUEZ, Tamara, 2015. Metodología de desarrollo para la Actividad productiva de la UCI. . 2015. Vol. v1.2.

TABARES, Ricardo Botero, 2010. Patrones Grasp y Anti-Patrones: un Enfoque Orientado a Objetos desde Lógica de Programación. *Entre Ciencia e Ingeniería*. 2010. Vol. 4, no. 8, pp. 161-173.

TASHILDAR, Aakanksha, SHAH, Nisha, GALA, Rushabh, GIRI, Trishul y CHAVHAN, Pranali, 2020. APPLICATION DEVELOPMENT USING FLUTTER. . 2020. Vol. 02, no. 08, pp. 5.

THOMAS, Pablo, DELÍA, Lisandro Nahuel, CORBALÁN, Leonardo, CÁSERES, Germán, FERNÁNDEZ SOSA, Juan, TESONE, Fernando, CUITIÑO, Alfonso y PESADO, Patricia Mabel, 2018. Tendencias en el desarrollo de aplicaciones para dispositivos móviles. en línea. abril 2018. Vol. XX WICC 2018

(UNNE, 26 y 27 de abril de 2018). [Accedido 7 noviembre 2022]. Recuperado a partir de: https://digital.cic.gba.gob.ar/handle/11746/8316

VITE CEVALLOS, Harry, MONTERO, Kelvin y CUESTA, Jefferson, 2018. Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software. *Espirales: Revista Multidisciplinaria de Investigación*. 8 junio 2018. Vol. 2. DOI 10.31876/re.v2i17.269.

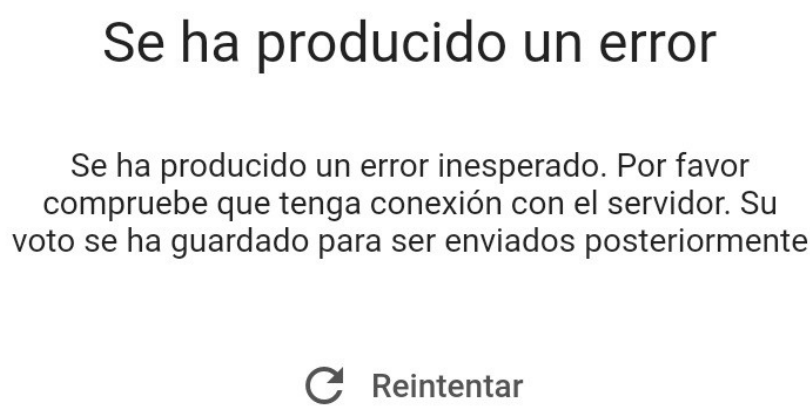
YEICY JULIANA MOLINA RIVERA, JONATHAN SANDOVAL CARDONA, y SANTIAGO ALBERTO TOLEDO FRANCO, 2012. *SISTEMA OPERATIVO ANDROID: CARACTERÍSTICAS Y FUNCIONALIDAD PARA DISPOSITIVOS MÓVILES*. en línea. UNIVERSIDAD TECNOLÓGICA DE PEREIRA. Recuperado a partir de: https://repositorio.utp.edu.co/server/api/core/bitstreams/2108e109-d69f-41cd-aa89-964053b44ac3/content

Anexos

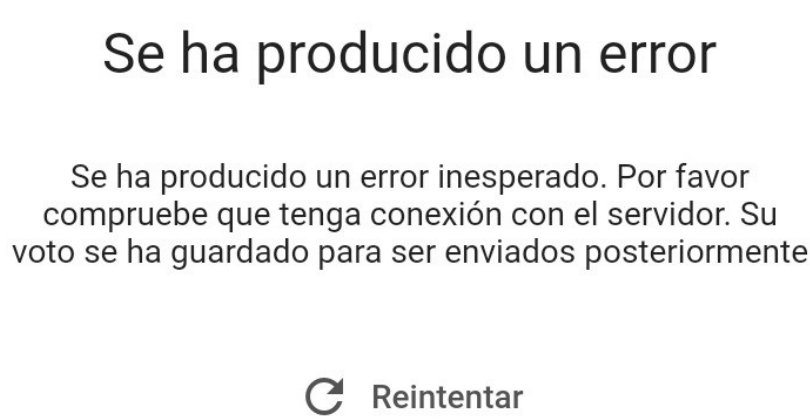
Historia de usuario	
Número: HU_5	**Nombre Historia de Usuario:** Guardar voto en la base de dato
Prioridad en negocio: Alta	
Descripción: Comienza cuando el usuario realiza un voto y este por cualquier motivo ya sé por falla en la conexión u otro, el sistema guarda automáticamente los datos en una base de datos local para luego ser enviado ese voto al servidor.	
Observaciones: Se muestra una pantalla como que se encontró un error inesperado.	
Prototipo: 	

Anexo 1 Tabla 11 Historia de Usuario 5 RF Guardar voto en la base de dato

Historia de usuario	
Número: HU_6	**Nombre Historia de Usuario:** Cargar voto de la base de datos
Prioridad en negocio: Alta	
Descripción: Comienza cuando el usuario selecciona el botón recargar, este hace la consulta a la	

base de datos local obteniendo el voto guardado para enviarlo al servidor.
Observaciones: Si se realiza correctamente el voto, se muestra una pantalla informando que el mismo se realizó correctamente.
Prototipo: 

Anexo 2 Tabla 12 Historia de Usuario 6 RF Cargar voto de la base de datos

Historia de usuario	
Número: HU_7	**Nombre Historia de Usuario:** Eliminar voto de la base de datos
Prioridad en negocio: Media	
Descripción: Comienza una vez que realizada la consulta a la base de datos y se envía el voto al servidor, automáticamente ese boto es eliminado de la base de dato local.	
Observaciones: Si se realiza correctamente la eliminación, se muestra una pantalla informando que el mismo se realizó correctamente.	
Prototipo: 	

Anexo 3 Tabla 13 Historia de Usuario 7 Eliminar voto de la base de datos