# **Definición de arquitectura candidata**

## **Proyectos similares**

Wikipedia

Es una aplicación móvil de la enciclopedia del mismo nombre, desarrollada por la Fundación Wikimedia. Gracias a esta aplicación es posible consultar contenidos enciclopédicos, definiciones o simplemente leer por ocio, sin necesidad de acudir a la versión web [2].

La aplicación puede ser descargada desde la Apple App Store y la Google Play Store para los sistemas operativos iOS y Android respectivamente.

Como ventajas de que tiene esta aplicación se encuentra la posibilidad de consultar los contenidos de Wikipedia en cualquier momento y lugar gracias al uso del dispositivo móvil, utilizar la aplicación no tiene costo adicional alguno y se pueden guardar varios artículos para una posterior lectura sin conexión.

La principal desventaja de esta aplicación es que para obtener la información es necesario una conexión a los servidores de Wikipedia vía Internet.

Kiwix

Kiwix realmente es un proyecto de software libre que da la posibilidad de acceso a sitios web basados en MediaWiki sin conexión a Internet. Es más famoso por ser el lector offline de Wikipedia soportado oficialmente por la Fundación Wikimedia [3].

Gracias a que está destinado a un uso offline, esta aplicación trabaja con bases de datos con extensión ZIM, donde se encuentra toda la información. Estas bases de datos no se distribuyen junto a la aplicación por lo se suma la tarea de conseguir los archivos de bases de datos, al menos una vez.

EcuMóvil

EcuMóvil surge como la idea de crear una versión de la EcuRed Portátil para dispositivos móviles. Como antecedente, ya existía la propia EcuRed Portátil que se desplegaba para ordenador y que el proyecto Kiwix contaba con una versión para dispositivos móviles. Tras 3 años en que la aplicación se caracterizaba por un aspecto no acorde al estilo visual de Android, sistema operativo en el que se desplegaba, EcuMóvil implementa todo el contenido en una nueva versión basada en el proyecto Kiwix, lo que permitió implementar imágenes y mejorar la política de actualizaciones [4].

Guatiní (versión existente)

Como parte del proyecto extensionista Guatiní, se desarrollaron varias versiones de la aplicación móvil por parte del departamento del grupo de Investigación de Aplicaciones Móviles de la Facultad de Ingeniería Informática de la Universidad Tecnológica de La Habana, “José Antonio Echeverría” (CUJAE). Estas aplicaciones fueron muy diversas: unas con un aspecto muy apegado al ámbito de los juegos y otras con un estilo más parecido al resto de aplicaciones de Android, sistema operativo en donde se despliega [1].

La última versión logró un comportamiento de mutación en dependencia de los módulos que se le incorporaban: Proyecto Guatiní, Murciélagos cubanos, Artes Plásticas. De esta forma se lograba una “falsa unión” de los contenidos de estos módulos, pero no se alcanzaba la completa integración que se deseaba.

Animales

Animales es el nombre de un juego educativo para aprender el nombre y el aspecto de aproximadamente de 100 mamíferos, 90 aves, 20 reptiles, 4 anfibios, 45 peces y 45 artrópodos del mundo. Todo usuario de esta aplicación tendrá que adivinar el nombre de cada especie a partir de una imagen ilustrativa. La dificultad del juego se puede establecer en fácil y difícil y se puede utilizar en varios idiomas [5].

La aplicación está limitada a abordar sólo a los animales, dejando de lado al resto de especies. Cómo juego es un método de aprendizaje que funciona bien, sobre todo para los niños y jóvenes.

## **Plataformas objetivo**

Es evidente que la sociedad cubana se va informatizando cada vez más, por lo que es innegable que el uso de los dispositivos móviles y las computadoras crece a medida que pasa el tiempo [6]. Ambos son dispositivos con una capacidad de computo asombrosa y características como el factor de forma y el tamaño de los *smartphones* los sitúan sobre las computadoras a nivel de uso cotidiano para la mayoría de los usuarios. Las aplicaciones para *smartphones* tienen la ventaja de que pueden ser utilizadas prácticamente en cualquier lugar y momento. Está claro que los esfuerzos en desarrollar una aplicación tipo enciclopedia deben enfocarse en las plataformas móviles.

En la Figura 1 se muestra un gráfico extraído de StatCounter que muestra los sistemas operativos móviles más utilizados actualmente en el mundo.

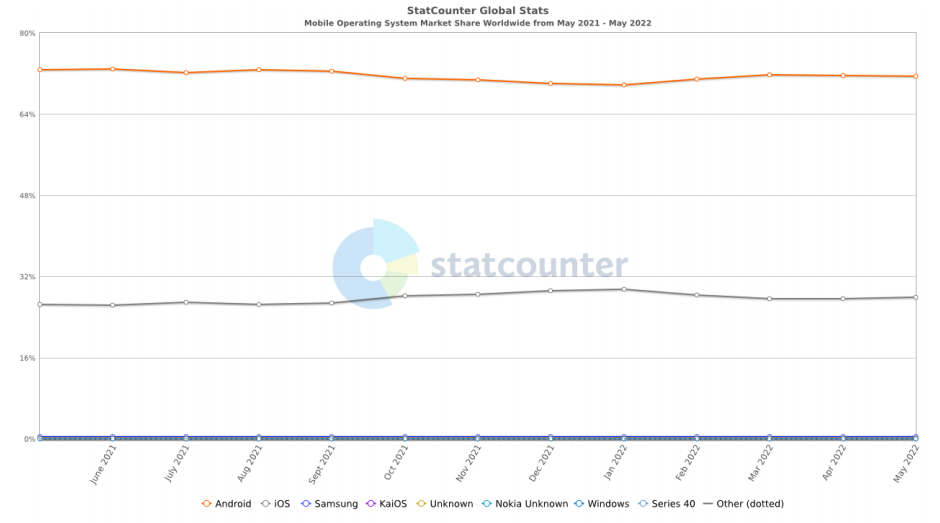


Figura 2: Cuota de mercado de los sistemas operativos móviles en el mundo en el período mayo 2021 - mayo 2022

Se puede evidenciar claramente que el sistema operativo más utilizado es Android, con alrededor del 71%, seguido de iOS con un 28% aproximadamente. El resto de sistemas operativos móviles no son notorios respecto a los anteriormente mencionados, por lo que adoptar iOS y Android como plataformas objetivo para el desarrollo de la aplicación garantiza una gran cuota de mercado en el proceso de despliegue.

## **Tecnologías para el desarrollo de aplicaciones móviles**

Para el desarrollo de aplicaciones móviles existen varios lenguajes de programación que se podrían adoptar. Para el sistema operativo Android se puede programar aplicaciones en Java donde se utilizarían librerías especiales para correr en Android Runtime (ART), sucesor de Dalvik; ambas son máquinas virtuales de Java modificadas, adaptadas para Android. También existe Kotlin, lenguaje desarrollado por JetBrains, que es interoperable con Java al utilizar la Máquina Virtual de Java (JVM) y más conciso al poder generar aplicaciones similares con un 40% menos de líneas de código que con Java [7]. Para iOS y iPadOS se puede desarrollar en Swift, lenguaje presentado en 2014 y desarrollados por Apple específicamente para el desarrollo en iOS, iPadOS y macOS [8]. Si bien programar de forma nativa las aplicaciones tiene el gran beneficio de poder implementar características específicas para cada sistema operativo y en ocasiones necesarias para la solución de *bugs*, el uso de frameworks es totalmente recomendable en el momento del desarrollo de dichas aplicaciones gracias a la velocidad que se alcanza en el proceso de escritura del código de las aplicaciones, además de proveer en muchas ocasiones la posibilidad de crear aplicaciones multiplataformas simultáneamente [9].

Ionic es un framework que permite desarrollar aplicaciones con tecnología web como HTML5, CSS y JS para desarrollar y ejecutar las aplicaciones móviles, lo que requiere el *wrapper* Cordova para acceder a los controladores de la plataforma nativa. No ofrece un rendimiento similar al de las aplicaciones nativas. En cuanto al desarrollo de aplicaciones, no utiliza componentes nativos pero ofrece una apariencia similar a la de las aplicaciones nativas. Esto se hace utilizando tecnologías web. La ventaja de usar Ionic es que la prueba de las aplicaciones es un proceso rápido. Esto agiliza todo el proceso de desarrollo de aplicaciones [10].

React Native es un *framework* que utiliza JavaScript, combina los beneficios de este lenguaje y ReactJS. La gran ventaja es que permite al desarrollador escribir las partes de código en Swift, Objetive-C o Java siempre que sea necesario. Si se desea manejar operaciones pesadas en la aplicación, existen acceso a módulos y bibliotecas nativas en aplicaciones basadas en React Native [11]. Proporciona a las aplicaciones un rendimiento similar al de las nativas. Esto se debe a que toma elementos del código que son específicos de las APIs (interfaz de programación de aplicaciones, del inglés: Application Programming Interface) de React Native. Estos módulos nativos se programan utilizando los lenguajes nativos, lo que facilita el acceso de los desarrolladores. Pero la única desventaja de esto es que no se puede reutilizar para otras plataformas. Estos módulos nativos sólo ofrecen un alto rendimiento. Los módulos de React Native trabajan con los controladores de UI (interfaz de usuario, del inglés: *User Interface*) nativos, lo que ofrece una mejor experiencia de usuario similar a la de las aplicaciones nativas. También hace uso de las librerías ReactJS y de los elementos de interfaz de usuario, lo que consolida el desarrollo de la interfaz de usuario.

Xamarin es un *framework* que combina lo mejor de C# y .Net para desarrollar aplicaciones para dispositivos móviles y Mac. Las aplicaciones desarrolladas con Xamarin funcionan de forma similar a las aplicaciones nativas. Se puede desarrollar la interfaz de usuario utilizando Xamarin.Android/iOS o Xamarin.Forms. El primer método consume mucho tiempo, pero ofrece un aspecto similar al de las aplicaciones nativas en lo que respecta a la UX (experiencia de usuario, del inglés: *User eXperience*). Cuando se utiliza Xamarin.Forms, el proceso de desarrollo es más fácil y consume menos tiempo, pero, este desarrollo no garantiza el aspecto y la sensación que es similar a las aplicaciones nativas [12].

Cuando se busca construir aplicaciones de alta calidad utilizando *frameworks*, muchos desarrolladores seleccionan Flutter, el cual utiliza Dart como lenguaje de programación. Dart es un lenguaje procedural que ofrece varios beneficios y está basado en C/C++ y Java. Gracias a Dart, los desarrolladores utilizan Flutter para desarrollar aplicaciones con cierta complejidad de una manera relativamente sencilla. Ofrece una velocidad increíble, se puede lograr una UI totalmente nativa y no se utiliza JavaScript para salvar la interacción con los componentes nativos del dispositivo [13].

## **Tecnologías a utilizar en el desarrollo de la aplicación móvil**

Se optó por utilizar el *framework* Flutter junto al lenguaje de programación Dart, dado que ambos son creados por Google, misma empresa que mantiene Android, el sistema operativo móvil más usado de la actualidad, lo que da la posibilidad de aprovechar la inmediatez de la implementación de las nuevas características y funcionalidades tanto en el sistema operativo como en el *framework* para su utilización en el proceso de desarrollo.

Herramientas de desarrollo:

* Flutter: Kit de desarrollo de código abierto desarrollado por Google y enfocado al desarrollo rápido de aplicaciones móviles multiplataformas.
* Gradle: Sistema de automatización de construcción de código abierto que se encarga de ensamblado de las aplicaciones para su posterior depuración o lanzamiento.
* Android SDK: Kit de desarrollo necesario para desarrollar aplicaciones en la plataforma Android. Contiene el depurador de código, las librerías, los simuladores de dispositivos, la documentación, ejemplos de código y tutoriales.
* Java Development Kit (JDK): Kit de desarrollo para programar en lenguaje Java. Necesario también para la programación para Android.
* Java Runtime Environment (JRE): Entorno de ejecución para aplicación escritas en Java. Necesario también para la programación para Android.

Lenguaje de programación:

* Dart: Lenguaje de programación creado por Google y utilizado para el framework Flutter.

Fundamentación de la selección

Flutter es un kit de desarrollo de código abierto, desarrollado por Google. Sus contribuciones se basan en la propia compañía, desarrolladora del sistema operativo Android, y la comunidad, por lo que tiende a ser un SDK (kit de desarrollo de software, del inglés: *Software Development Kit*) bastante actualizado acorde a las últimas características implementadas por el sistema Android o requeridas por los desarrolladores.

Este *framework* es capaz de crear aplicaciones de códigos diferentes (por ejemplo: Swift para iOS y Kotlin para Android) desde un mismo código fuente escrito en Dart, lenguaje de programación propio de Flutter.

Permite el desarrollo rápido con implementación de “*hot reload*” y “*hot restart*”, que son funciones de depuración que permiten ver al desarrollador los cambios realizados en el código en tiempo real y recargar la aplicación en el dispositivo virtual o físico sin necesidad de recompilar toda la aplicación nuevamente, proceso que normalmente demora. Está enfocado en la experiencia del usuario. Es capaz de crear aplicaciones que sean adaptativas o *responsives*, o sea, que se adapten a cualquier tamaño de pantalla. También posee un rendimiento prácticamente igual al nativo, brindando un comportamiento nativo según la plataforma y unas velocidades de carga rápidas.

Entorno de Desarrollo Integrado a utilizar (IDE, del inglés *Integrated*  
*Development Environment*)

Aunque es de obligación tener instalado Android Studio para el desarrollo de aplicaciones Android, dado que este es el software que gestiona nativamente todo lo relacionado con el desarrollo, se seleccionó Visual Studio Code.

Visual Studio Code es un IDE desarrollado por Microsoft. Soporta depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código y refactorización de código. Es muy personalizable por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Posee capacidad de adición de extensiones por lo que es posible añadirla características. Su desempeño es bastante bueno, consumiendo menos memoria RAM que otros IDE, como Android Studio o similares, basados en IntelliJ Platform. Es gratuito y de código abierto.

Base de datos: SQLite

Para almacenar información y administrarla es muy común utilizar una base de datos. Desde hace bastante tiempo, las bases de datos se manejan del lado del servidor y los dispositivos móviles se comunican con ellos mediante la red para obtener la información. Sin embargo, en ocasiones en necesario que la aplicación sea menos dependiente de la conectividad de la red. Para ello es necesario almacenar la base de datos en el almacenamiento del propio dispositivo, por lo que la base de datos debe ser ligera, rápida, segura y que consuma poca memoria y energía. La utilización de bases de datos SQLite es muy popular en el ámbito de los desarrolladores móviles ya cumple con la ligereza y la rapidez que se quiere, además de ser una base de datos relacional, con todos los beneficios que conlleva.

# **Estilos, patrones arquitectónicos y principios de diseño**

## **Estilos y patrones arquitectónicos**

Patrón Modelo-Vista-Controlador (Estilo Llamada y Retorno)

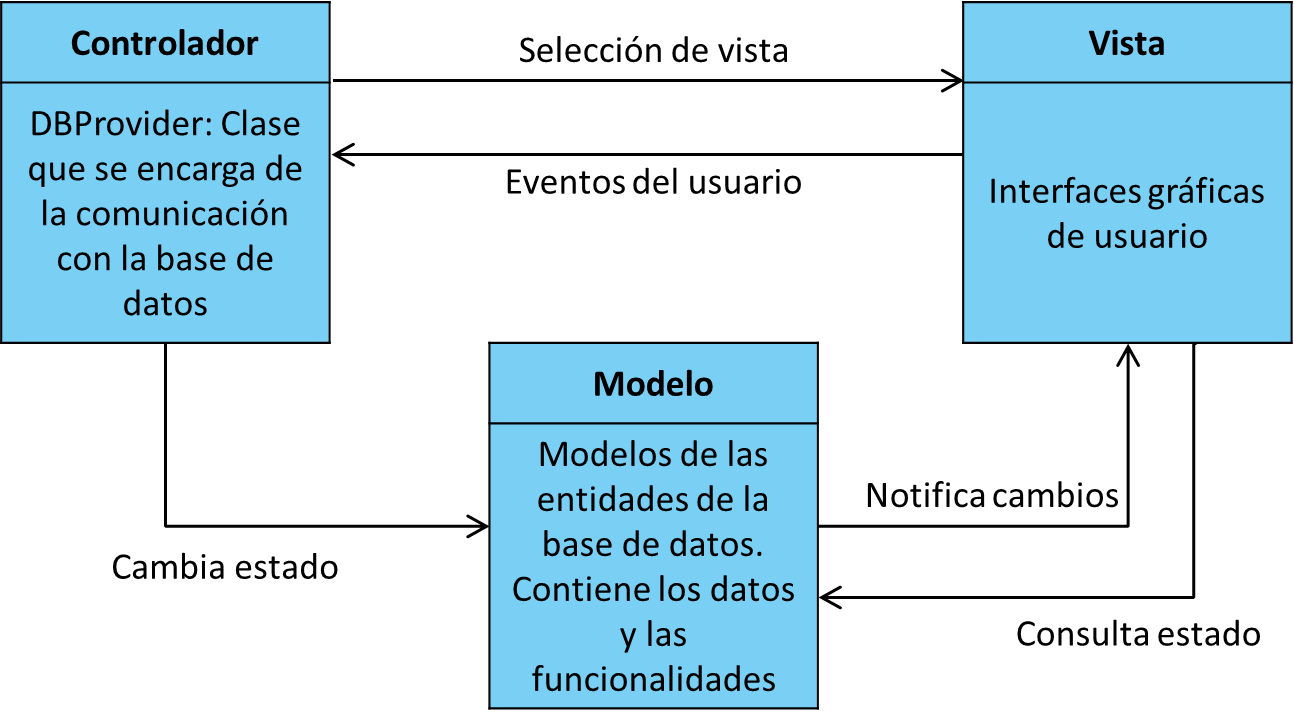


Figura 3: Representación del patrón Modelo-Vista-Controlador

Patrón N-capas (Estilo Llamada y Retorno)

Enfoque basado en Reutilización

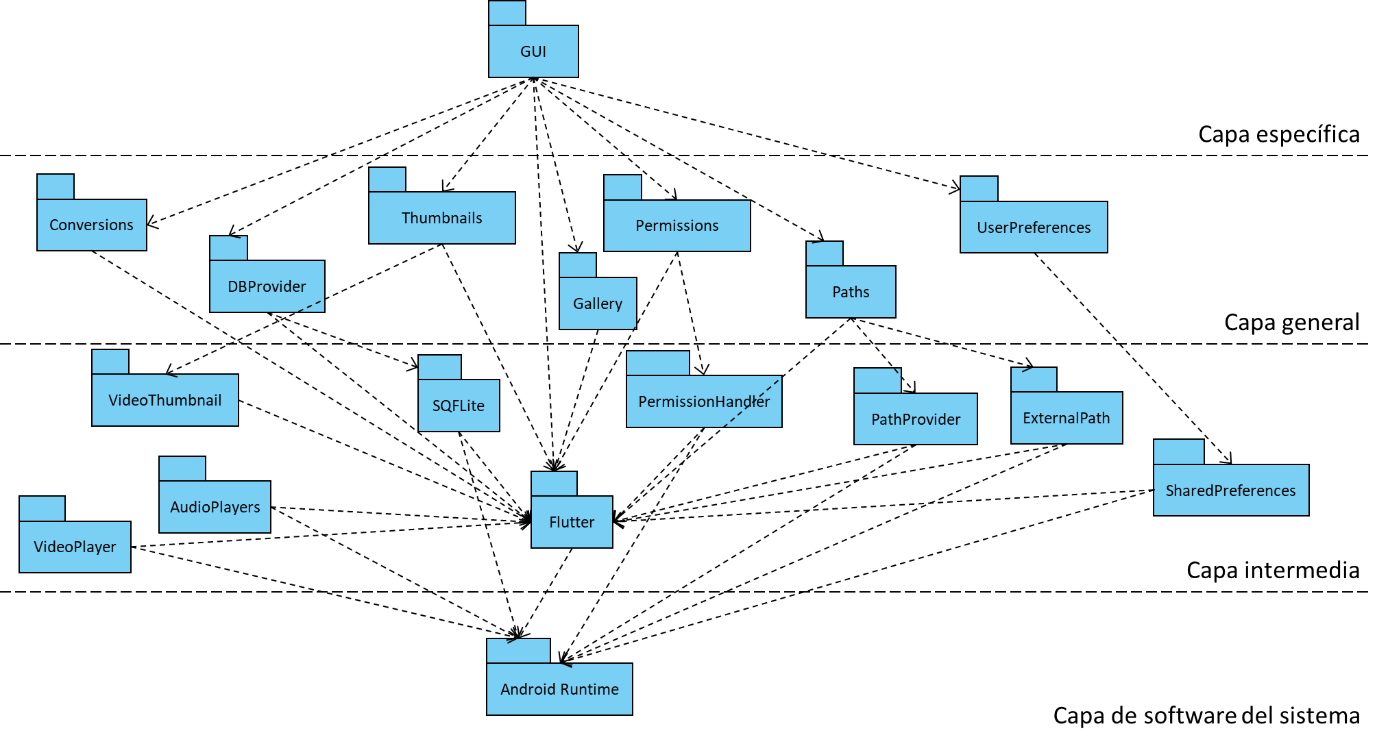


Figura 4: Representación del patrón N-capas con enfoque basado en Reutilización

Enfoque basado de Responsabilidades

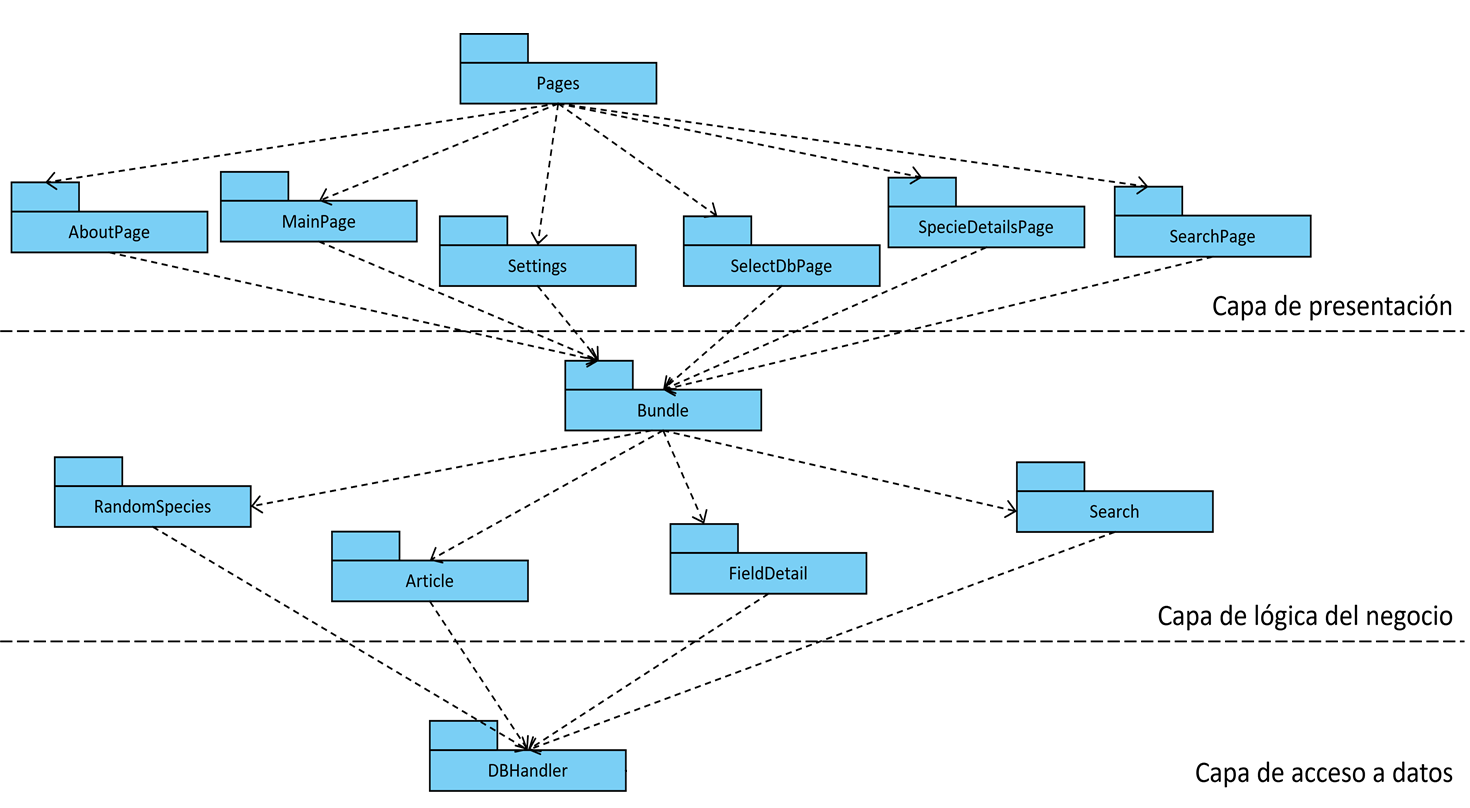


Figura 5: Representación del patrón N-capas con enfoque basado en Responsabilidades

Patrón Repositorio (Estilo Centrado en Datos)

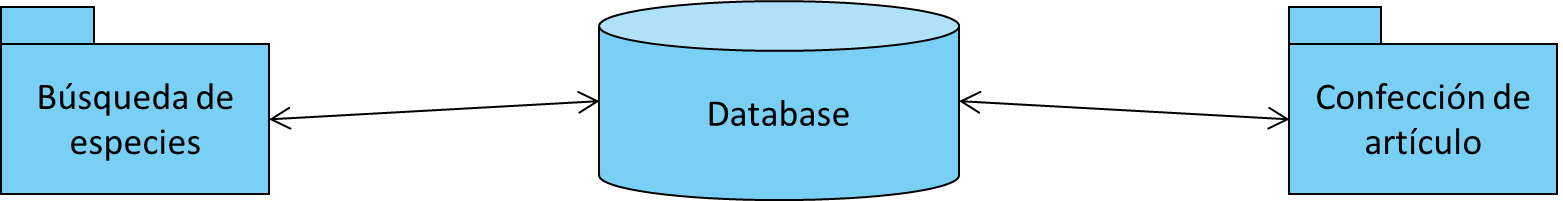


Figura 6: Representación del patrón Repositorio

Patrón Tuberías y Filtros (Estilo basado en Flujo de Datos)

Búsqueda de especies

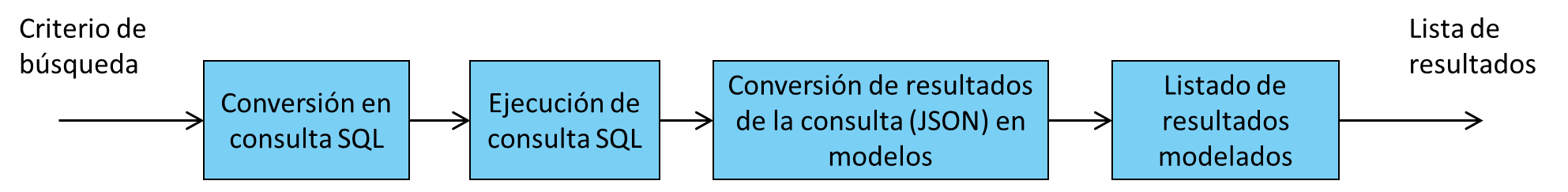


Figura 7: Representación del patrón Tuberías y filtros para la búsqueda de especies

Apertura de artículo

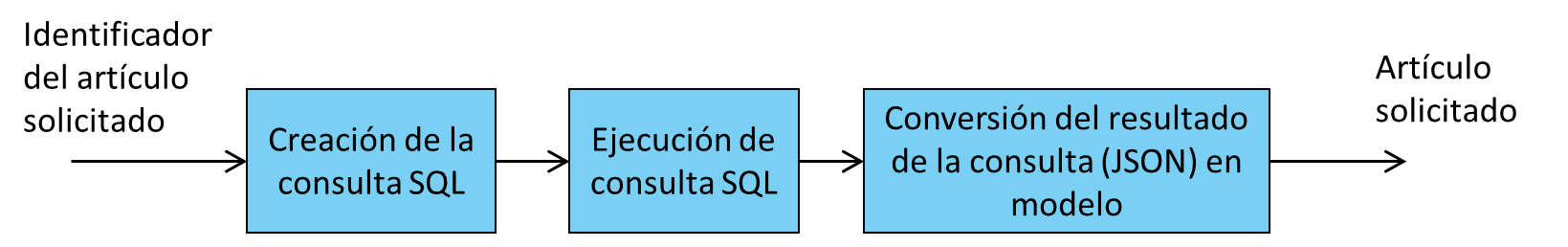


Figura 8: Representación del patrón Tuberías y filtros para la apertura de un artículo

Arquitectura basada en servicios (Estilo Peer to Peer)

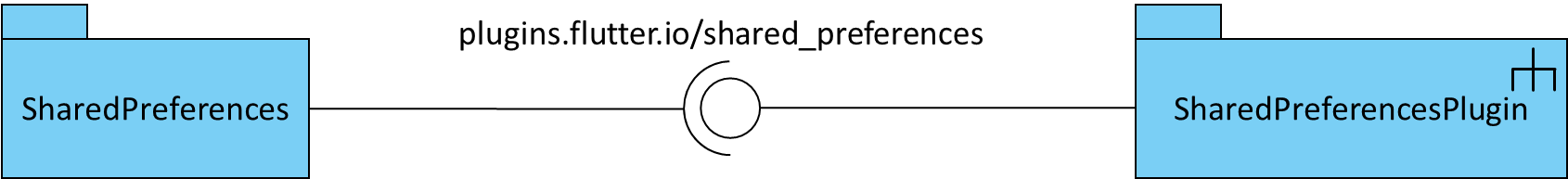


Figura 9: Representación del patrón SOA para el servicio de preferencias de usuario

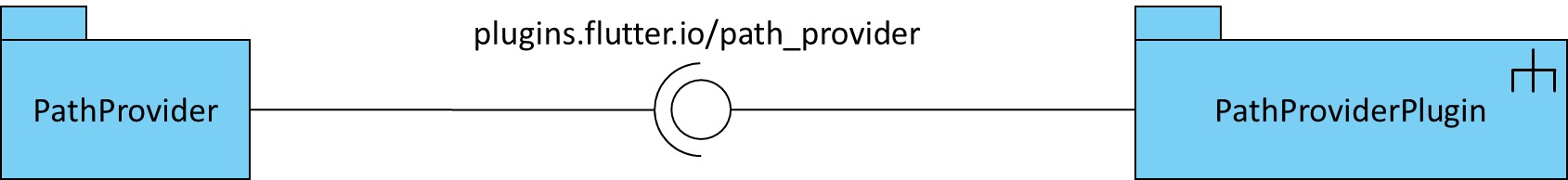


Figura 10: Representación del patrón SOA para el servicio de rutas de almacenamiento local

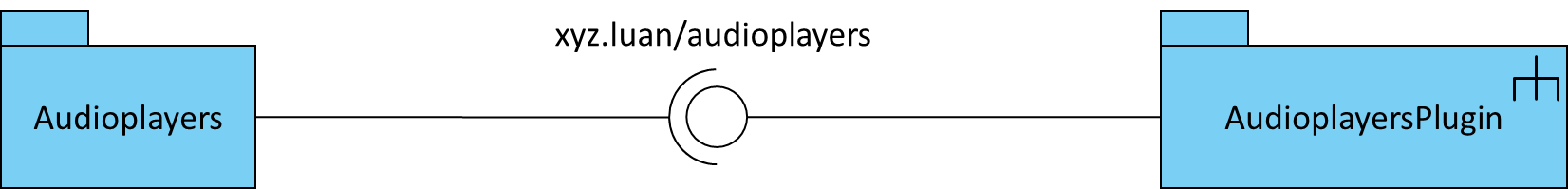


Figura 11: Representación del patrón SOA para el servicio reproducción de audio

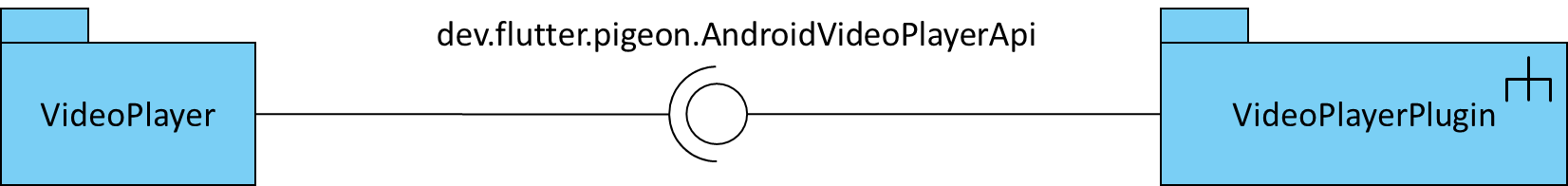


Figura 12: Representación del patrón SOA para el servicio de reproducción de video

## **Principios de diseño**

Single Responsibility Principle (SRP): Las clases sólo se encargan de una cosa en concreto. Ejemplo, hay clases específicas que se encargan de cada una de estas responsabilidades por separado: permisos de la aplicación, interacción con la base de datos, reproducción de audio, reproducción de video, almacenamiento en el dispositivo, búsqueda de artículos, visualización de un artículo.

Open-Close: Ejemplo, al visualizar un artículo se ven varios campos que tienen información de la especie. Estos campos no se programan dentro del artículo sino como *widgets* aparte, no permitiendo la modificación de la clase sino que se le añaden estos *widgets* en forma de extensión.

Ley de Demeter (“No hables con extraños”): Ejemplo, las clases que se encargan de la interfaz gráfica de usuario, sólo se comunican con otras, como la encargada de la interacción con la base de datos y la manejadora de preferencias de usuario, si es estrictamente necesario; solo efectúan llamadas a objetos creados por ellas, como los *widgets*.

Principio de Hollywood (“No nos llames, nosotros te llamaremos”): Las partes de más alto nivel de la aplicación llaman a las partes de un nivel inferior y se evitan las llamadas a la inversa. Ejemplo, las clases encargadas de la interfaz gráfica de usuario realiza llamadas de métodos a la clase que interactúa con la base de datos y esta a su vez realiza los cambios pertinentes en dicha base de datos, evitando a toda costa que las llamadas de métodos se realicen en sentido contrario.

*Once and Only Once Rule*: Cada parte de la aplicación tiene una sola responsabilidad y esta no se repite en otra parte. Ejemplo, clases que se encargan de la base de datos, las de las preferencias usuario, las de audio y las de video.

Modularidad-Cohesión-Acoplamiento: La aplicación está programada de tal forma que sea lo más modular posible, como se evidencia en la representación del patrón N-capas con enfoque basado en reutilización. La cohesión alta garantiza que cada parte de la aplicación se centre en una sola cosa, como con el principio *Once and Only Once Rule*, y el bajo acoplamiento va de la mano con la Ley de Demeter.

# **Patrones de diseño**

## **Singleton**

Este patrón se utiliza al momento de utilizar la base de datos, en los casos que se necesita hacer alguna consulta, ya sea para las sugerencias de búsqueda o directamente para mostrar el contenido de los artículos.

Este patrón de utiliza en este caso para asegurar de que una clase que inicializa y maneja la base de datos tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia. Cada vez que se necesite realizar alguna acción con la base de datos basta con referenciar dicha instancia [14].

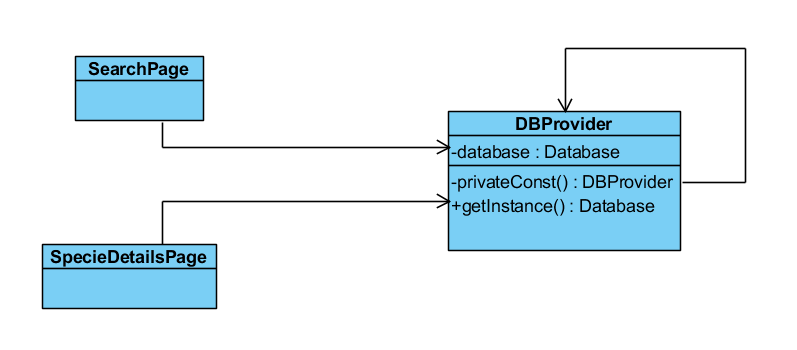


Figura 13: Representación del patrón Singleton

Principios de diseño empleados:

Dado que Singleton es considerado por muchos como anti-patrón, más que emplear principios de diseño viola una muy importante, que el Principio de Responsabilidad Única, puesto que resuelve dos problemas a la vez: tener un punto global de una instancia y tener la certeza de que una clase tiene una única instancia [14].

## **Proxy**

Este patrón se utiliza en los momentos que se tiene que cargar imágenes, videos y miniaturas de videos, mostrando en su lugar un gif con una animación de “cargando”.

Este patrón se utiliza en este caso dado que permite proporcionar un sustituto visual antes de que se muestre la imagen o video en cuestión, ya que en dependencia del dispositivo donde se ejecute la aplicación, el proceso de carga de la multimedia puede durar más o menos, mientras tanto el usuario tiene alguna información visual de que algo se está procesando [14].

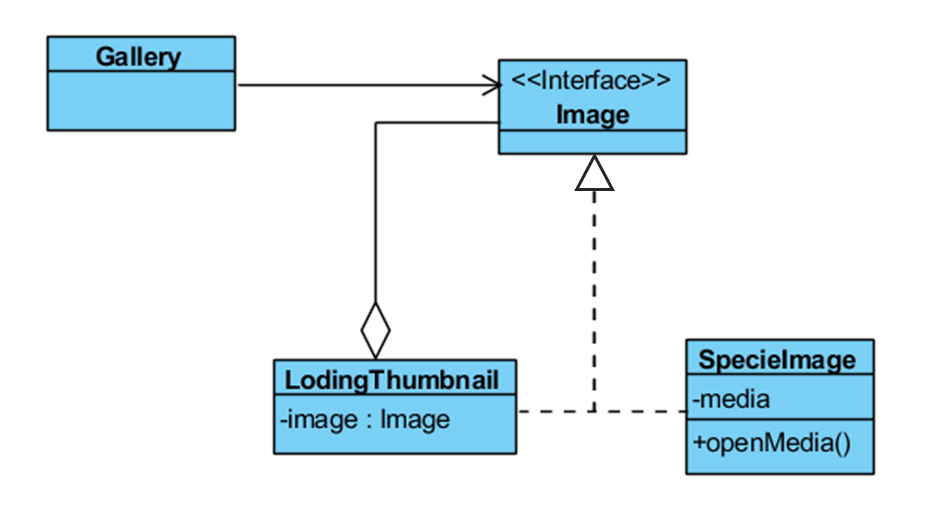


Figura 14: Representación del patrón Proxy

Principios de diseño empleados:

* Open-Close: Las operaciones están abiertas para que sean transformadas por la imagen real (la imagen de la especie) y por la temporal (imagen de carga).
* Encapsular la variabilidad: Cada imagen tiene su operación, la imagen de carga se encarga de informar sobre el proceso mientras que la real muestra toda la lógica de una imagen.
* Diseñar hacia las interfaces, no hacia las implementaciones: Existe una interfaz abstracta Imagen donde se pueden incorporar más proxys si es necesario.
* Bajo acoplamiento: Minimiza el acoplamiento de la clase Gallery con todas las posibles clases reales (Imágenes) con las que hay que comunicarse porque lo hace a través del proxy.