# Construcción de la aplicación móvil

**Breve descripción:**

Mediante este componente el aprendiz se afianzará en el desarrollo de aplicaciones móviles híbridas, sumando elementos clave para el correcto desempeño y funcionalidad, flujo de navegación e información, componentes gráficos e informativos de la app, entre otros. Identificará recursos externos e internos de uso, definiendo las API necesarias e integrables, todo ello, en concordancia con su modelo de negocio.

**Septiembre 2023**

# Tabla de contenido

# Introducción

Para comenzar, lo invitamos a ver el siguiente video:

**Video 1.** Construcción de la aplicación móvil


[Enlace de reproducción del video]

| **Síntesis del video: Construcción de la aplicación móvil** |
| --- |
| Para lograr el correcto funcionamiento de las aplicaciones móviles es importante manejar una estructura en el “Front-end” y en el “Back-end”, capaz de responder a los flujos de navegación requeridos en cada uno de los sistemas operativos donde se va a desplegar la aplicación. |

Por esta razón es importante conocer cada uno de ellos, de manera detallada y no cometer errores en el diseño de la estructura de la aplicación.

Otro tema relevante para el desarrollo de aplicaciones móviles es la capacidad de generar, utilizar y consumir las API (Interfaz de Programación de Aplicaciones), que permitan la integración con elementos necesarios para cumplir con los requerimientos del cliente.

# 1. Flujos de navegación en app multiplataforma

El flujo de navegación de una aplicación hace referencia a la sucesión de acciones requeridas para la navegación del usuario en una aplicación móvil.

El objetivo del diseño del flujo de navegación para una app es buscar que esta sea eficiente y fácil de usar.

## 1.1. Flujos de navegación por usuario

El flujo de navegación define la interacción que se tendrá con la aplicación, precisando cada uno de los roles o grupos de usuarios, personalizando la interfaz basados en el proceso de negocio.

Se pueden definir varias categorías de flujos de navegación:

- **Global.** Todo usuario tiene acceso a este flujo de navegación sin restricciones.
- **Rol.** Este flujo de navegación, está definido por el perfil del usuario, el cual permitirá o restringirá el acceso, según los permisos o privilegios que este posea. Por ejemplo, el flujo de navegación de un usuario administrador.
- **Grupo.** Este flujo de navegación no se da por el rol, sino por el grupo al cual pertenece, y permite realizar actividades diferentes o tener restricciones, según las características del grupo.

## 1.2. Flujos de navegación según sistema operativo

Los flujos de navegación deben contar con una estructura sencilla e intuitiva que se familiarice con los antecedentes de las aplicaciones que se utilizan sobre ese sistema operativo a trabajar y, si es el caso trabajar sobre varios S.O., se debe adaptar y facilitar la navegación y, experiencia al usuario.

Todo el recorrido y navegación del usuario debe tener características previsibles, coherentes y sencillas, independientemente de la navegación que se utilice; gran parte del éxito de la aplicación móvil recae sobre estos tipos de detalles. Si la aplicación no cuenta con un flujo de desplazamiento sencillo, los usuarios simplemente no la utilizarán por complicada o inapropiada.

### Flujo de navegación en Android

Desde sus primeras versiones, la navegación en Android se realizó por medio de un patrón o comportamiento nombrado como “hacia adentro, hacia atrás y hacia arriba”. Esta navegación se caracteriza porque cuenta con dos botones “up” y “back”: arriba y atrás, como se muestra a continuación.

**Figura 1.** Flujo de navegación en Android

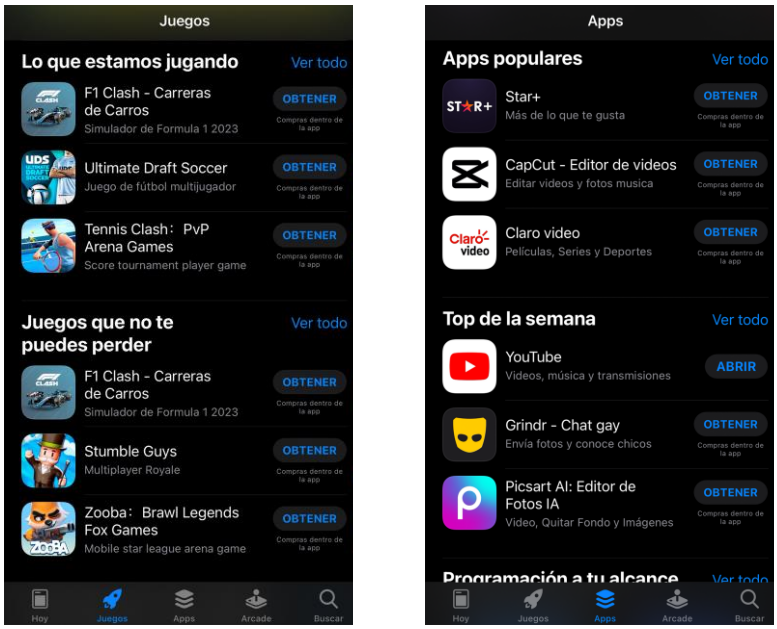
La imagen presenta el flujo de navegación en Android, desde el inicio, luego la lista de conversación y finaliza con el componer.

- **“Up”.** Por lo general, se encuentra en la parte superior o inferior izquierda y permite retroceder en la navegación, en un orden jerárquico establecido. La navegación permite replegarse de manera escalonada hacia el inicio o hacia la pantalla principal de la aplicación móvil.
- **“Back”.** Este botón deshace ordenadamente la navegación que se ha ejecutado, restableciendo y ubicando al usuario, de nuevo en la pantalla de navegación de la que proviene.

## Flujo de navegación en sistema operativo iOS

En este sistema operativo se encuentran tres estilos de navegación bien definidos que se adaptan y ajustan a la estructura de las aplicaciones, como se muestra a continuación.

**Figura 2.** Flujo de navegación en iOS



Nota. Tomado de ejercicio práctico de descargas en app “store”.

La imagen presenta el flujo de navegación en iOS.

- **Navegación jerárquica.** Se organiza de manera descendente o ascendente para mostrar la información y los elementos que componen la aplicación, por ejemplo, Facebook. La navegación con estructura jerárquica permite a los usuarios elegir la pantalla de navegación y realizar un recorrido hasta llegar a su destino o realizar la actividad. Si desea realizar otra actividad, debe retroceder o iniciar desde cero, regresando hasta la pantalla de inicio de la aplicación. Un ejemplo de estas aplicaciones que manejan esta estructura es el correo y la opción de configuración del sistema operativo.

- **Navegación plana.** La información está organizada para visualizarse y tener acceso a ella por medio de desplazamientos laterales de izquierda a derecha. La navegación con estructura plana permite la navegación desplazándose de una categoría a otra directamente y se cuenta con una pantalla principal que presenta todas las categorías de manera centralizada, como ejemplo se puede ver el funcionamiento de la app “store”.

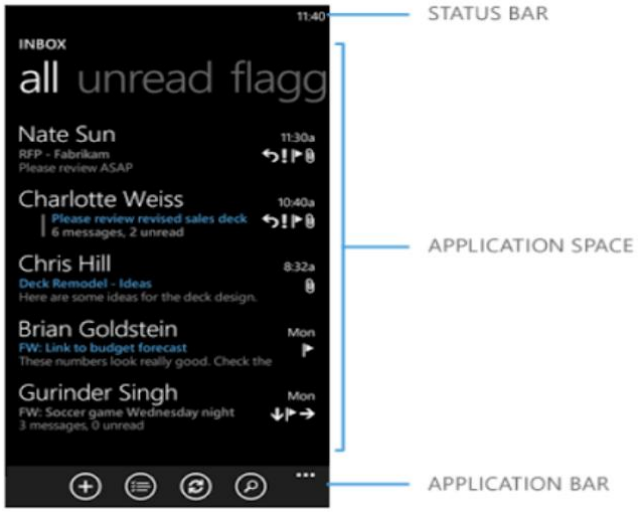
- **Navegación contenida.** Esta navegación combina las dos anteriores, la jerárquica y la plana.

## Navegación en Windows phone

La navegación en este sistema operativo busca asemejar el funcionamiento de los sistemas operativos implementados en los computadores, ofreciendo una barra de estado o barra de navegación, ubicada en la parte inferior. Permite la interacción

sencilla del usuario y recibe actualizaciones de información constantemente, mostrando diferentes notificaciones, como se muestra a continuación.

**Figura 3.** Navegación en Windows phone




http://catedraocampo.com.ar/diferencias-de-flujos-de-navegacion-en-apps-multiplatforma/

Imagen que presenta la navegación en Windows phone, donde se presenta el "status bar", el "application space" y el "application bar".

- **La barra de estados.** Se encuentra ubicada en la parte superior y ofrece la posibilidad de cambiar sus propiedades; puede estar visible o replegarse, ocultándose mientras no es utilizada o solicitada; por lo general, se manipula y visualiza para indicar el proceso de algunas actividades que están en progreso.

- **Barra de aplicaciones.** Esta barra se asemeja a la barra del sistema operativo de Windows, permite ubicar y visualizar de manera sencilla los elementos habituales. También permite generar un acceso rápido a las

aplicaciones más utilizadas, incluyendo un menú con los ejecutables más usados o relevantes; su ubicación se puede orientar de manera horizontal o vertical, por esa razón, las aplicaciones desarrolladas deben estar estructuradas, pensando en esta posibilidad de cambio de orientación.

## 2. Arquitectura de la aplicación móvil híbrida

Para realizar una aplicación móvil híbrida se debe contar con una arquitectura que permita comprender e integrar cada uno de los elementos necesarios para realizar los procesos; por lo general se usa el modelo de tres capas para la arquitectura de "software" para aplicaciones móviles. Estas capas se pueden dividir y trabajar con otras subcapas, según la complejidad y robustez del aplicativo, estas son: la capa de presentación, la capa de aplicación o capa de negocio y la capa de datos.

- **Componentes de la capa de datos.** El componente de la lógica de datos permite visualizar de manera más clara los contenedores de información y sus relaciones, concentrando la funcionalidad de este proceso que permite el acceso a los datos, generando una estructura que hará que la aplicación sea más fácil de diseñar, configurar y mantener a largo plazo.

- **Componentes de ayuda o servicios.** Las tareas para acceder a los datos demandan mantener una lógica general y estandarizada que debe ser implementada en los componentes que, por lo general, se encuentran separados, pero mantienen una identidad, que permite simplificar la complejidad de los componentes y su integración, minimizando la cantidad de código a mantener. Estos componentes son las librerías especializadas y las rutinas planificadas para mejorar e incrementar el rendimiento en el acceso a los datos, reducir el tiempo de desarrollo y la cantidad de requerimientos de componentes lógicos.

- **Parámetros de diseño de capa de acceso a datos.** Un correcto diseño de la capa de datos reduce el tiempo de desarrollo y permite un fácil

mantenimiento de la misma después de que la aplicación se encuentre desarrollada. Por lo general, es fundamental trabajar con el diagrama de clases y el diagrama de objetos UML, para lograr la claridad en lo que se busca realizar (Korth y Sudarshan, 2006).

- **Diseño global de la capa de acceso a datos.** Identificar y determinar las fuentes de datos y el acceso a ellos, definiendo cómo se realizará la conexión para la recolección y presentación de los datos, generando estrategias que permitan evitar los errores o contar con un plan de contingencia, si es necesario.

- **Diseño de componentes de acceso a datos.** Definir los patrones de diseño y los métodos de acceso a utilizar para cada fuente de datos, definiendo, enumerando y clasificando los componentes necesarios para simplificar el acceso a los datos.

- **Diseño de componentes de ayuda o servicios.** Centralizar la información con el fin de reutilizarla, identificar librerías y elementos que permitan la reutilización de los datos de manera eficiente y segura, implementar procedimientos para generar un buen control de acceso a los datos y realizar pruebas en cada uno de los elementos utilizados, identificando y creando mecanismos de ayuda para suplir problemas más frecuentes, como por ejemplo los "`strings`" de conexión, la comprobación y autenticación de las fuentes de datos, el monitoreo de los servicios y excepciones particulares en los diferentes procesos.

- **Agentes de servicio.** Se emplean cuando un componente de negocio lo provee un servicio o un servidor externo; estos servicios externos son, por lo general, ofrecidos por un tercero, por ejemplo, la conexión a correos electrónicos y el almacenamiento de la información que contienen los mensajes; se almacenan de manera remota y por esa razón se necesitará crear un código que administre el protocolo de comunicación con ese servicio. Los protocolos de servicios controlan la petición y la integración, logrando mejorar el rendimiento y la fiabilidad en los procesos.

  El agente de servicio determina cómo este será utilizado en la aplicación; se debe determinar cuál es la herramienta más apropiada para utilizar y referenciar el servicio. Por lo general, esto crea unos canales de comunicación que forman los contratos o la negociación entre los servicios.

## 3. Instrucciones de diseño

Son aquellas características a tener en cuenta para el diseño de la capa de acceso a datos, siguiendo estos parámetros se asegura el acceso a la capa de datos, cumpliendo con los requerimientos de manera eficiente y segura; de esta manera se facilita el mantenimiento y los cambios. Las modificaciones se podrán atender de manera más sencilla.

a. **Tecnología adecuada.** Es de suma importancia elegir la tecnología adecuada para el acceso a los datos. Para definir qué tecnología utilizar, lo primero que se debe plantear es qué tipo de datos se van a manipular, identificando los posibles tipos de datos y su formato. Así se puede generar una interfaz que utilice una capa de abstracción intermedia que organice el acoplamiento para la capa de datos.

b. **Componentes de interfaz.** Este proceso se puede lograr por medio de la definición de componentes de interfaz, como por ejemplo:

   - Teniendo claro la puerta de enlace con entradas y salidas conocidas, que interpreten todas las peticiones a un formato comprensible para los componentes de las capas.
   - Encapsulando las funciones que permiten el acceso a los datos para que no sean asequibles fácilmente. Esta capa de acceso a datos, debe ocultar y no permitir el acceso a los detalles de las fuentes de los datos.
   - Negociando y gestionando todas las conexiones y las funciones que generan las consultas.

c. **Interfaces abstractas.** Por lo general, los procesos en los cuales se adquieren y consumen datos interactúan por medio de las interfaces abstractas que, utilizando objetos y funciones en las aplicaciones, adquieren la información en formatos personalizados y XML. La capa principal para el acceso a los datos debe crear y gestionar todas las conexiones a todas las fuentes de datos requeridas por la aplicación.

d. **Protección de la información.** Se debe seleccionar el método apropiado para el almacenamiento y protección de la información, en estas conexiones y sus transacciones, la cuales deben estar acordes con los requisitos de seguridad de las aplicaciones. La capa que permite el acceso a los datos debe capturar, manejar y administrar todas las excepciones asociadas con las fuentes de datos y las operaciones CRUD (“create”, “read”, “update”, “delete”), determinando cómo se van a manejar las excepciones de datos, evitando anomalías en las consultas de información y teniendo el control de todas las transacciones y operaciones realizadas.

## 3.1. Riesgos de seguridad

La capa de acceso a los datos debe estar protegida de posibles incursiones y ataques que buscan robar o dañar la información. Proteger todos los elementos que permiten las transacciones y por donde fluye la información evitando que se tenga un fácil acceso a las fuentes de datos es una necesidad. Se debe manejar, en todos los niveles y capas, unos mínimos de seguridad restringiendo el acceso a las fuentes de datos.

También reduce los procesos de ida y vuelta reiterativos, considerando la utilización de instrucciones “batching” o también conocido como manejo por lotes, dentro de una única operación de base de datos.

Los objetivos de desempeño para el acceso a la capa de datos deben de ser tomados en cuenta durante el diseño del modelo de negocio y la interacción con los elementos que hacen parte de los procesos que se identifican en el diagrama de objetos.

## 3.2. Capa de negocio

Sobre la capa de negocio se debe realizar una serie de actividades y/o consideraciones, que son relevantes para el correcto funcionamiento en la lógica de la aplicación. Estas actividades se deben realizar de manera paralela con las diseñadas y desarrolladas para el diseño de la capa de datos:

- Diseño global para la capa de negocio.
- Diseño de componentes del negocio.
- Componentes de flujo de trabajo.
- Consideraciones de diseño.
- Componentes de negocio.
- Concurrencia y transacciones.

## 3.3. Diseño global para la capa de negocio

Es fundamental identificar las características relevantes de los usuarios sobre los cuales se enfoca la aplicación y cómo ellos van a utilizar la capa de negocio, si ha de ser con niños, personas adultas mayores, público en general, etc.

Definir cómo se va a exponer la capa de negocio, cuáles son los requerimientos de seguridad, identificando cuáles son las necesidades y generar las estrategias de validación para la capa de negocio.

## 3.4. Diseño de componentes del negocio

Identificar y definir las reglas del negocio, generando patrones que se ajusten a los requisitos y necesidades.

- **Flujo de información.** Se trata entonces de establecer la manera en que se utilizará para realizar el envío de información entre los componentes de la aplicación móvil.
- **Ubicación, acoplamiento e interacciones apropiadas.** Se trata, además, de definir qué tipo de soporte es el apropiado para utilizar entre las transacciones, asegurando que son las adecuadas para lograr la mejor ubicación, el mejor acoplamiento, y lograr las interacciones de los componentes de negocio.
- **Formatos de datos.** Elegir e identificar los formatos de los datos que serán comunes para las entidades de negocio que se interrelacionan.

## 3.5. Componentes de flujo de trabajo

Es clave identificar y definir el flujo de trabajo que debe ser utilizado en cada escenario. Definir, entonces, un modelo de edición estándar que permita realizar cambios sin afectar otros componentes cuando se realicen actualizaciones y mejoras. También es importante el diseño de los componentes de negocio, conociendo el flujo de trabajo para generar la sinergia necesaria en estas actividades.

- **Consideraciones de diseño.** El diseño de la capa de negocio busca minimizar la complejidad de interconexión y flujo de datos por medio de la división de las actividades en diferentes áreas, por ejemplo, los objetos de procesamiento de información que definen el negocio, el flujo de trabajo y las relaciones entre las entidades que definen la lógica del negocio.

- **Responsabilidades de la capa de negocio.** Es importante comprender e identificar las responsabilidades de la capa de negocio en cada una de las transacciones o en cada operación, es una buena idea y muy sana usar una capa de negocio separada para el procesamiento de las reglas de negocio más complejas.

- **Separación adecuada de componentes.** Desde luego, y después de la acción anterior, se debe evitar mezclar los diferentes tipos de componentes, impidiendo un estrecho acoplamiento y dependencia entre las capas, que podrían generar graves fallas en el correcto funcionamiento de la aplicación.

## 3.6. Componentes de negocio

Los componentes que hacen parte fundamental del negocio se deben trabajar bajo el enfoque de módulos que permitan la reutilización y el acoplamiento, evitando la dependencia y, por consiguiente, las fallas generales.

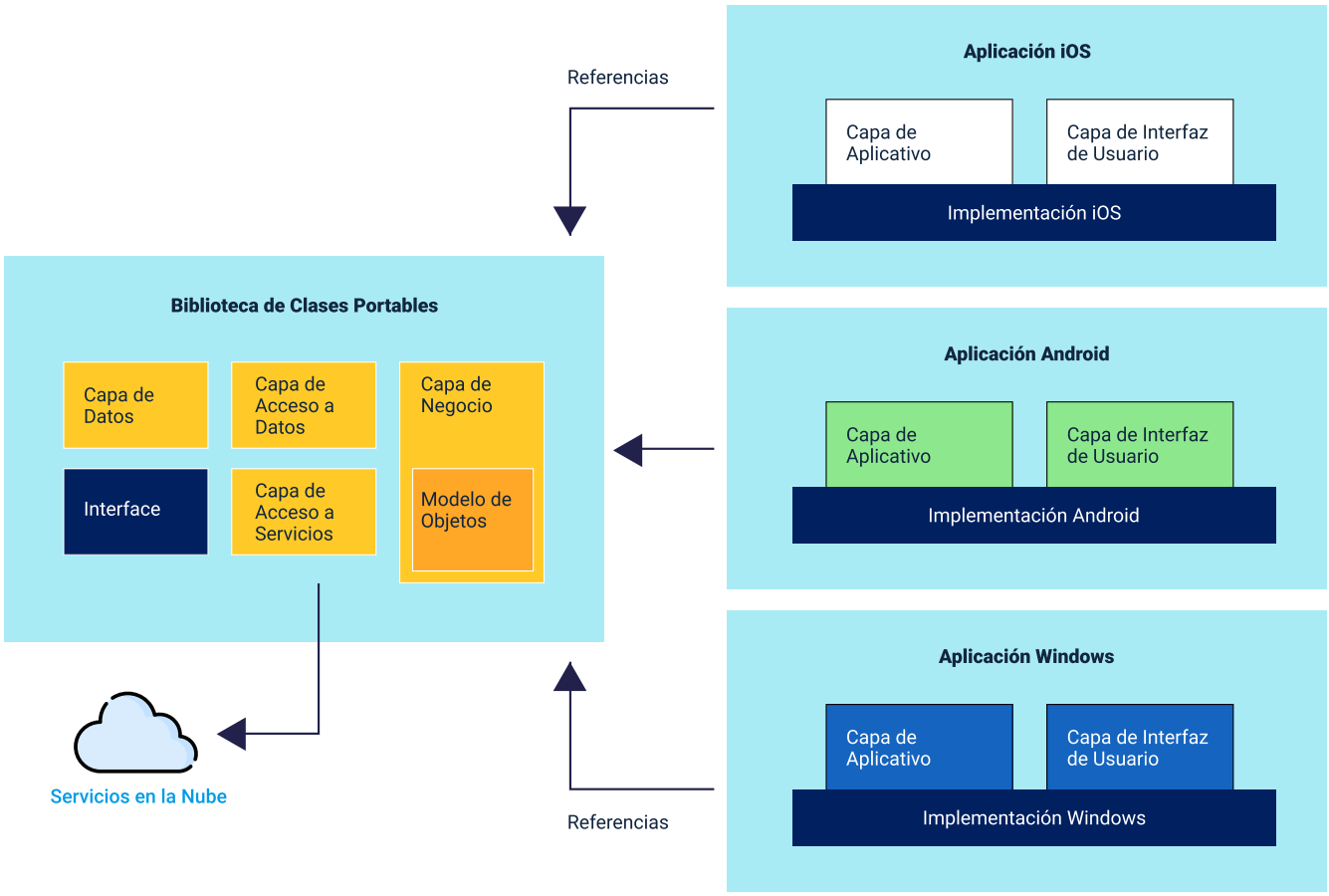
Si uno de estos elementos o apartes falla, se debe contar con una alternativa que permita dar continuidad al flujo de datos sin perjudicar la operación, implementando reglas y patrones del negocio.

Todos los componentes de negocio deben mostrar la funcionalidad de una manera transparente, así mismo, los servicios de datos para dar continuidad al flujo de trabajo. En resumen, se deben diseñar componentes que se puedan reutilizar y acoplar, manteniendo la estructura de las reglas de negocio.

## 3.7. Concurrencia y transacciones

Se debe optar por identificar cuál es el mejor modelo de comunicación y sincronización entre procesos. Tal modelo ha de permitir gestionar las transacciones de manera efectiva, por lo general, siempre se parte de los modelos conocidos: modelo optimista y modelo pesimista, como se ilustra a continuación.

**Figura 4.** Arquitectura de App multiplataforma, con biblioteca de clases




Se presenta la arquitectura de una app para los sistemas operativos iOS, Android y Windows, donde cada uno de ellos tiene la capa de aplicativo y la capa de interfaz de usuario, utilizando una biblioteca de clases portables.

- **Modelo optimista.** El primer modelo presenta un escenario ideal donde no se presentan bloqueos, no existe falla en la transacción de datos y las actualizaciones requieren un código de verificación, por lo general, contra una marca de tiempo para verificar que los datos no han cambiado desde que se recibieron por última vez.

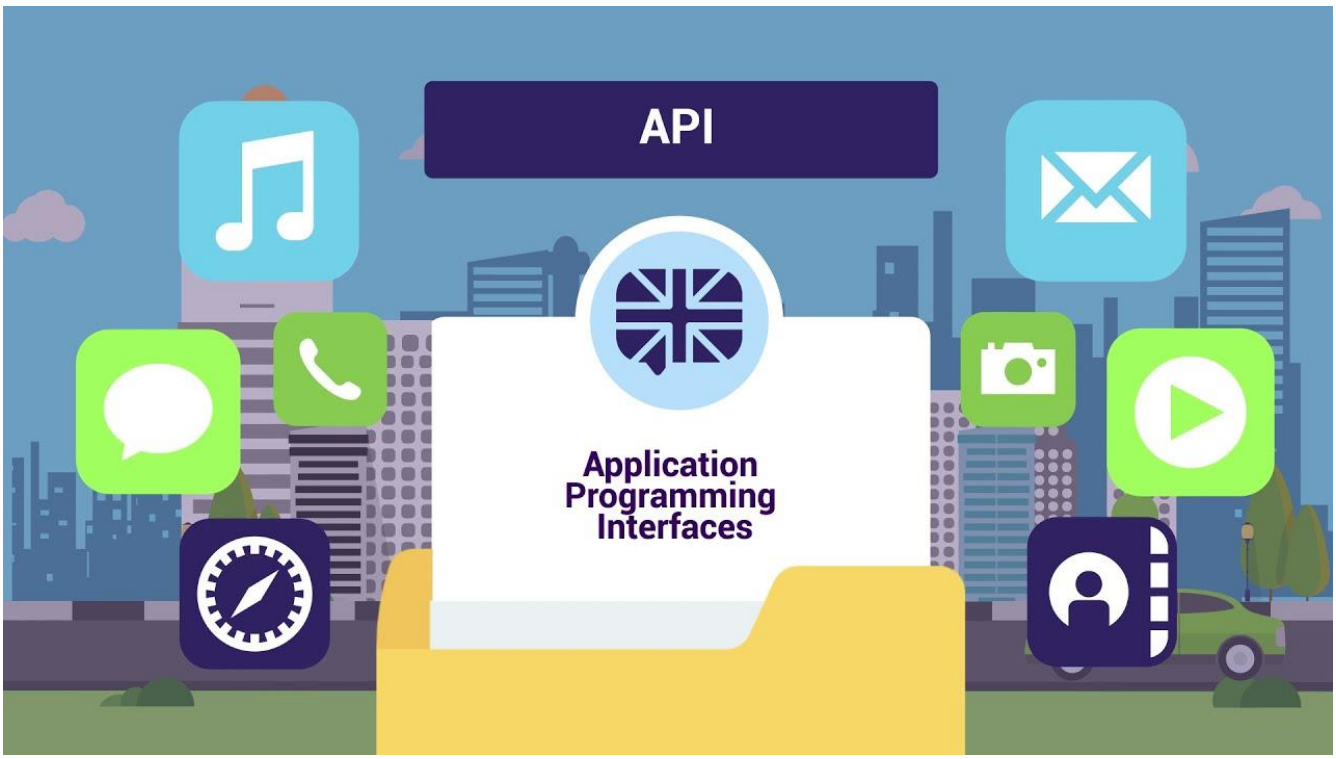
- **Modelo pesimista.** En este los datos se bloquean y no se pueden actualizar por otra operación, hasta que se libere el bloqueo. Estos dos modelos se deben alternar y ha de buscarse un equilibrio que permita solucionar cualquier inconveniente en las transacciones.

- **Capa de negocio.** Esta capa define e implementa la funcionalidad principal del sistema y se condensa la lógica del negocio, puntualizando qué es lo importante y relevante dentro del proceso de desarrollo. Por lo general, está formada por varios elementos que son comunes a casi todas las aplicaciones móviles; algunos de estos elementos se pueden reutilizar, en la mayoría de los casos se cuenta con interfaces de servicio que otros componentes y capas utilizan, por ejemplo, los patrones que algunas aplicaciones utilizan para lograr el acceso a los datos y servicios remotos y, los servicios remotos que prestan algunos terceros. Estos datos y protocolos son consumidos por la capa de negocio a través de servicios web o API.

# 4. Las API

Lo invitamos a ver el siguiente video, para conocer qué son las API.

**Video 2.** Las API




[Enlace de reproducción del video]

| **Síntesis del video: Las API** |
| --- |
| API es la abreviatura en inglés de “Application Programming Interfaces”, que significa interfaz de programación de aplicaciones.<br><br>Es un elemento que permite gestionar la comunicación entre el “software” y las aplicaciones, trabaja como intermediario o intérprete entre un “software” y otros “software” o bases de datos. |

Se presenta como un conjunto de reglas y protocolos, que se utilizan para desplegar e integrar el "software" de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de "software", por medio de un conjunto de políticas, aprobando un intercambio de información.

También se puede considerar como una interfaz de programación de aplicaciones, que apoya a los desarrolladores en el proceso de conexión de "frontend" y su "backend".

Las API utilizadas para el desarrollo móvil, son esenciales en el momento de utilizar recursos de terceros, permite ahorrar mucho tiempo de desarrollo y utilizar recursos que hacen mucho más completas las aplicaciones, de una manera sencilla.

Una API es un conjunto de definiciones y reglas que se utiliza para desplegar e integrar el "software" de dos o más aplicaciones.

Admiten que los productos o servicios de terceros se comuniquen con sus propios productos, todo esto sin necesidad de conocer ni comprender su funcionamiento. Lo único necesario es conocer el protocolo de conexión y contar con los permisos para tener acceso a los datos.

## 4.1. Historia de las API

Las API surgieron en los primeros días de la informática, mucho antes que la computadora personal; en esa época una API normalmente se usaba como biblioteca para los sistemas operativos.

- **Alojamiento.** Casi siempre estaban habilitadas localmente en los sistemas en los que operaban.
- **Intercomunicación.** Sin embargo, en algunas ocasiones pasaban mensajes entre las computadoras centrales.
- **Expansión.** Después de casi 30 años, las API se expandieron más allá de los entornos locales.
- **Integración remota.** A principios del año 2000 ya eran una tecnología importante para la integración remota de datos.

## 4.2. Tipos de API según sus permisos

El uso de una API implica diferentes modelos de permisos y, así mismo, las API son clasificadas según dichos permisos, así:

- **Privadas.** Estas API solo se usan internamente, son creadas por las empresas para su uso doméstico teniendo el control total sobre ellas.
- **“Partners”.** Estas API son desarrolladas por algunas empresas y son distribuidas por “partners” que tienen permisos especiales para vender estos servicios empresariales, ofreciendo ingresos adicionales a la empresa y a los distribuidores autorizados.
- **Públicas.** Todos tienen acceso a las API, así que otras empresas pueden desarrollar una API para generar una interacción entre ellas y así convertirse en una fuente de innovaciones, prestando un servicio libre de

restricciones. Esto permite que terceros desarrollen aplicaciones que interactúan con las API públicas.

## 4.3. ¿Por qué son importantes las API?

Si se cuenta con una API no será necesario crear desde cero un servicio, como ejemplo, se puede tomar un sistema de pagos. Es posible integrar a una aplicación una API de un servicio de pago ya existente, por ejemplo, PayPal; cuando se suministra la información de una tarjeta, la aplicación utiliza una API para compartir la información y enviar esos datos de forma remota a otro programa que verificará y validará la información.

Una vez se verifica la información se retornará una respuesta a la aplicación, indicando si los datos son válidos; esta pasarela de pago suministra un servicio muy complejo de desarrollar, que requiere altos niveles de seguridad que pueden ser aviados, al utilizar esta API que permite la integración y utilización de este servicio.

## 4.4. Tipos de API según su función

El funcionamiento de las API se establece sobre protocolos o sobre arquitecturas. Conózcalos a continuación.

- **SOAP.** Fue un protocolo muy utilizado que, paulatinamente, se ha venido convirtiendo en obsoleto. SOAP (Protocolo de Acceso a Objetos Simples) permitió la estandarización del cambio de información y datos; las API diseñadas con SOAP usaron formato XML para el diseño de sus mensajes y transacciones, recibían solicitudes a través de HTTP o SMTP.

- **Estandarización del lenguaje.** Con SOAP se facilita el proceso de comunicación estandarizando el lenguaje, reglas y políticas definidas, entre aplicaciones que funcionan en entornos distintos o están escritas en lenguajes diferentes, pero logran compartir la información.

- **Entornos descentralizados.** El protocolo SOAP se caracteriza por ser liviano y se utiliza, primordialmente, para realizar un intercambio de información en entornos descentralizados; con frecuencia se suele asociar SOAP con el lenguaje de descripción de servicios web UDDI y WSDL.

- **REST.** Por otra parte, está REST, que no es un protocolo sino una arquitectura, es la de mayor uso en la actualidad. Los denominados "Restful" son derivaciones de la arquitectura REST.

- **Arquitecturas locales y remotas.** Existen por su parte arquitecturas locales, aquellas en las que la aplicación se comunica con la API que se encuentra en el mismo ambiente o dispositivo y las arquitecturas remotas, aquellas en las que la aplicación necesita consumir una API, ubicada o alojada en servidores remotos y es necesario contar con un servicio de conexión de datos o Internet para acceder al servicio.

## API de servicios web o remotas

Estos servicios permiten que las aplicaciones conectadas se comuniquen, utilizando el protocolo HTTP que es, básicamente, un conjunto de reglas definidas entre cliente-servidor, para realizar el intercambio de información. Este es el principio fundamental que permite que las API remotas logren establecer una comunicación entre aplicaciones de distintas ubicaciones.

Los formatos más comunes de envío de datos son: Json, XML y texto plano.

Para ampliar la información, lo invitamos a ver el video ¿Qué es una API? – La mejor explicación en español, el cual se encuentra en el material complementario.

- **Estándares web.** En su gran mayoría las API están desarrolladas bajo los estándares web. Se debe comprender que no todas las API remotas se pueden considerar como API web, pero sí se concluye que todas las API web son remotas. Usan el protocolo de transferencia de información HTTP para solicitar mensajes y proporcionar una definición de la estructura de los mensajes de respuesta.

- **Formatos válidos y adaptables.** En particular, los mensajes de respuesta se transforman tomando la forma de un archivo XML o JSON, que son los formatos válidos y los que mejor se adaptan al momento de presentar los datos y la información, de una manera fácil de comprender y manipular para otras aplicaciones.

- **XML.** "Extensible Markup Language", este metalenguaje permite definir lenguajes de marcas desarrollado por el World Wide Web "Consortium" utilizado para almacenar datos en forma legible.

- **JSON.** "Javascript Object Notation" es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML se considera un formato independiente del lenguaje, es el más utilizado, hoy en día, por su fácil manejo.

- **Texto plano.** “Plain text” son aquellos archivos formados exclusivamente por texto, solo caracteres, sin ningún formato, es decir, no requieren ser interpretados para leerse.

## API de código fuente

Ofrece librerías de objetos, se usan con frecuencia para el desarrollo de una aplicación compuesta, que toma recursos y datos de varias fuentes remotas.

Las llamadas de estos servicios se ejecutan de acuerdo con los estándares del entorno de la aplicación, como por ejemplo .NET o J2EE.

## API heredadas

Se utilizan en la integración de interfaces de aplicaciones que usan reglas y protocolos de comunicación. Estas API se utilizan integrando elementos de hardware, usando protocolos de objetos remotos, por medio de archivos planos, integrando interfaces de sistemas operativos.

- **CORBA.** Uno de los ejemplos que permite comprender su funcionamiento es la API heredada CORBA (COMMON “Object Request Broker Architecture”).
- **OMG.** Es un estándar definido por el grupo de gestión de objetos (OMG) que permite el funcionamiento conjunto de componentes de “software” escritos en distintos lenguajes informáticos y que se ejecutan en distintos sistemas.
- **Asociación por medio de “token”.** Estos tipos de objetos CORBA pueden recibir muchas llamadas y repeticiones de acceso, pero el tipo de objeto

del cliente no cambia, cada instancia del tipo de objeto del cliente se asocia por medio de un “token” a sus datos.

## API de producción

Son fundamentales porque manejan la lógica comercial de las aplicaciones. Una vez desarrollada la app, esta debe recibir datos reales y es, en este momento, cuando la API de producción permite realizar esta actividad de generación de información real.

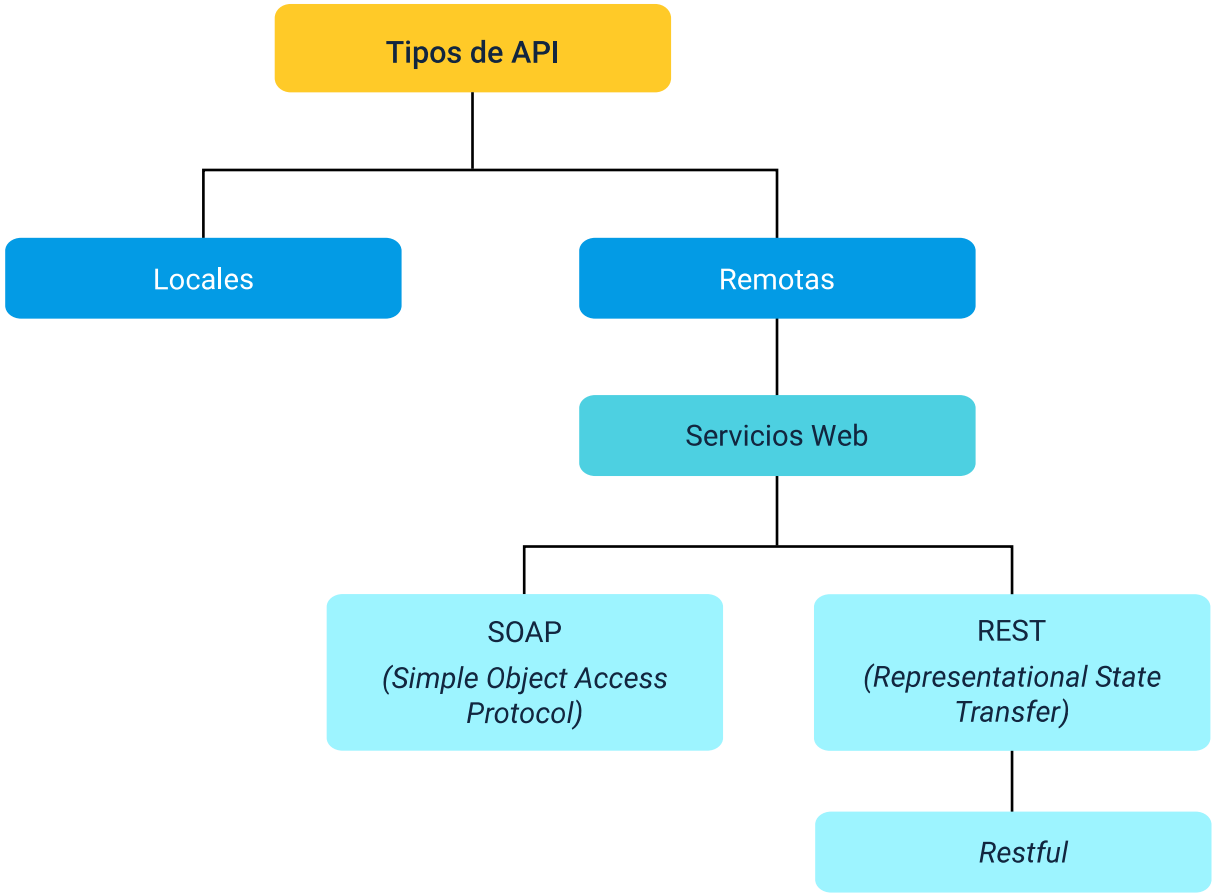
Se utiliza como un servidor que permite realizar la incisión de datos en la aplicación, permitiendo la activación de los servicios y la realización de actualizaciones de datos.

Estas API de producción no deben devolver datos a los usuarios que no estén autorizados en visualizar la información; tampoco debe permitir la actualización de la información que no debe ser modificada. Deben ser rápidas porque son las responsables de manejar, en el mundo real, todo el tráfico de la aplicación.

## API de desarrollo

También conocidas como API falsas, son utilizadas para realizar presentaciones previas al cliente demostrando el funcionamiento de la aplicación, antes de salir a producción. El objetivo de este tipo de API es realizar las pruebas de “Frontend” y “Backend”, continuar con el desarrollo, aunque no existan bases de datos o datos reales, como se muestra en la siguiente figura.

**Figura 5.** Tipos de API




Los tipos de API son:

a. Locales

b. Remotas: servicios web, los cuales son:

- SOAP ("Simple Object Access Protocol").
- REST ("Representational State Transfer").
- "Restful".

Las API de desarrollo generalmente no guardan, ni alojan datos en una base de datos, lo que se debe hacer es guardar todo en la memoria de manera temporal

mientras se realizan las pruebas. Las API web que trabajan con las limitaciones de la arquitectura REST se conocen como las API de “RESTFUL”. Una API web de “RESTFUL” no cuenta con ningún estándar oficial que genere unas reglas para su desarrollo y utilización (Fielding, 2000).

## 4.5. Arquitectura cliente-servidor REST

La arquitectura REST se compone de tres elementos: los clientes, los servidores y los recursos. De esta manera, se administran las solicitudes con HTTP; los servicios web “RESTFUL” son un grupo de parámetros que están ajustados bajo los principios de la arquitectura de transferencia de estado representacional (REST).

Los elementos de REST esencialmente utilizan la arquitectura cliente/servidor, basados en el protocolo de trasferencia HTTP, que admite la ejecución de los indicadores de recursos universales (URI).

WSDL (“Web Services Description Language”) es un formato del “Extensible Markup Language” que se usa para detallar servicios web, es un lenguaje de programación fundamentado en XML y su principio fundamental es detallar servicios web. Cuando se está programando una API y se solicita información a una API REST el servidor responde, generalmente, la solicitud con un código o códigos preestablecidos; estos son algunos de los principales:

- **2xx:** que la petición se ha procesado correctamente.
- **3xx:** señala qué se debe redireccionar, indicando la realización de una acción adicional para completar la petición.

- **4xx:** marca una solicitud inválida, se genera cuando se realizan peticiones a recursos inexistentes o no autorizados.
- **5xx:** indica fallas o errores relacionados directamente con el servidor.

## 4.6. Métodos HTTP usados en las API

Los métodos HTTP permiten la interacción con la información de la API, esto es equivalente al CRUD de las bases de datos. “Get” es el método que realiza la solicitud de información, “post” expide la nueva información registrada actualizando los datos, “put” permite la actualización de los datos cuando se realiza el cambio relevante en la información y “delete” realiza la eliminación de recursos.

Finalmente, algunas características a tener en cuenta cuando se utiliza la arquitectura “RESTFUL” son:

- **Arquitectura cliente-servidor.** La arquitectura REST administra las solicitudes realizadas por HTTP, estas solicitudes pueden provenir de los clientes, servidores y recursos.
- **Sin estado.** El contenido de los clientes no se almacena en el servidor entre las solicitudes, sino que la información sobre el estado de la sesión se queda en el cliente. En su lugar, la información sobre el estado de la sesión está en posesión del cliente.
- **Capacidad de caché.** El almacenamiento en caché puede eliminar la necesidad de algunas interacciones cliente-servidor.

- **Sistema en capas.** Las interacciones cliente-servidor pueden estar mediadas por capas adicionales, que pueden ofrecer otras funciones como el equilibrio de carga, los cachés compartidos o la seguridad. Estas capas pueden ofrecer funcionalidades adicionales como equilibrio de carga, cachés compartidos o seguridad.

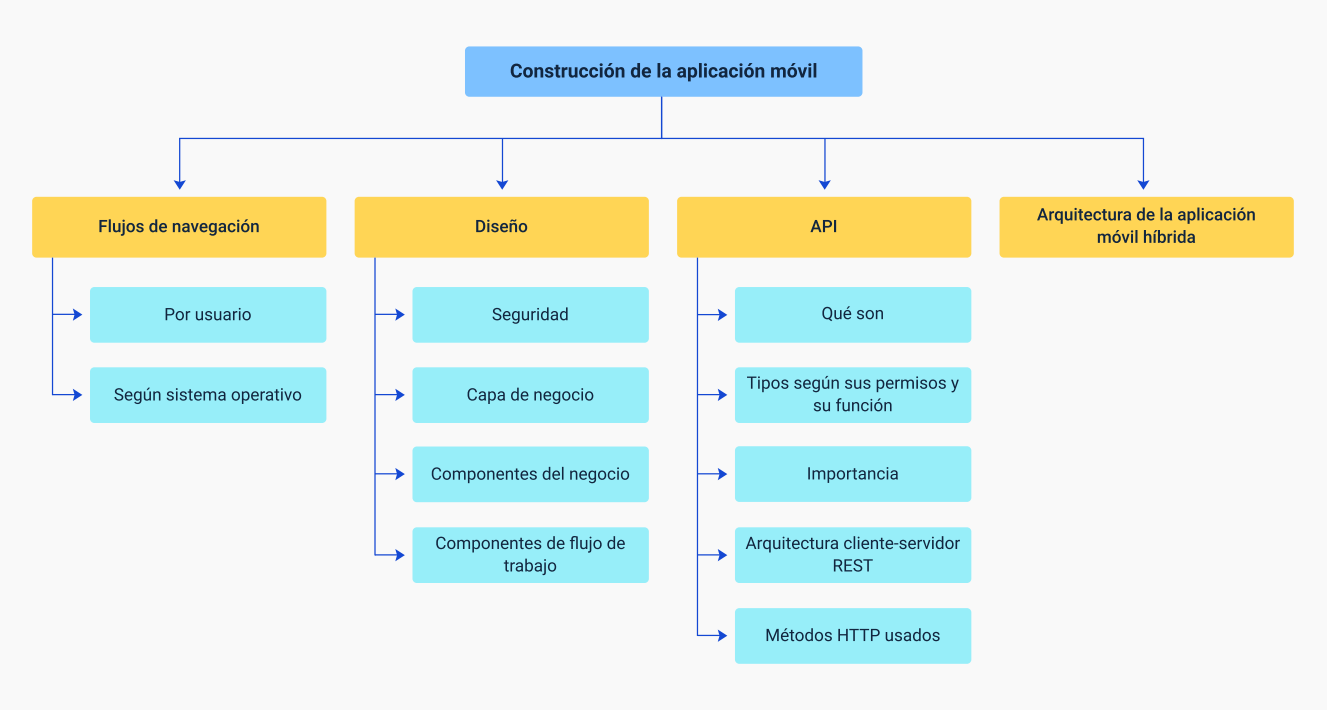
- **Código de demanda.** Los servidores pueden extender las funciones de un cliente, transfiriendo un código ejecutable.

- **Interfaz uniforme.** Esta limitación es fundamental para el diseño de las API de "RESTFUL" e incluye 4 aspectos: identificación de recursos, administración de recursos, mensajes autodescriptivos e hipermedios.

- **Identificación de recursos en las solicitudes.** Los recursos se identifican en las solicitudes y se separan de las representaciones que se devuelven al cliente.

# Síntesis

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.




El esquema presenta la síntesis de la temática estudiada en el componente formativo, comenzando por la construcción de la aplicación móvil, de donde se derivan:

- Flujos de navegación: por usuario y según sistema operativo.
- Diseño: seguridad, capa de negocio, componentes del negocio y componentes de flujo de trabajo.
- API: qué son, tipos según sus permisos y su función, importancia, arquitectura cliente-servidor REST y métodos HTTP usados.
- Arquitectura de la aplicación móvil híbrida.

# Material complementario

| Tema | Referencia | Tipo de material | Enlace del recurso |
|---|---|---|---|
| 4. Las API | Vásquez, G. (2020). NativeScript Vue introducción – creando apps móviles nativas con Vue.js [video]. YouTube. | Video | https://www.youtube.com/watch?v=kkCDDYpp4pg&t=1726s |
| 4. Las API | Bravent IT consulting company. (2019). Webinar – Desarrollando apps móviles con NativeScript [video]. YouTube. | Video | https://www.youtube.com/watch?v=qN0hwVAPA0A |
| 4. Las API | EDteam. (2019). ¿Qué es una API? – La mejor explicación en español [video]. YouTube. | Video | https://www.youtube.com/watch?v=u2Ms34GE14U&t=307s |
| 4. Las API | Nari, A. (2019). Cómo consumir una API desde NativeScript [video]. YouTube. | Video | https://www.youtube.com/watch?v=UtKZtMkIe08 |

## Glosario

**Comandos “Batching”:** esto es lo que se denomina procesamiento por lotes o “batching”. Consiste en dedicar un período de tiempo a realizar tareas similares o que requieren los mismos recursos.

**CRUD:** es el acrónimo de (“Create”, “Read”, “Update” y “Delete”) que traduce “crear, leer, actualizar y borrar”, por lo general se utiliza en las actividades básicas realizadas sobre una base de datos.

**“Partner”:** es la relación que se constituye entre dos o más empresas para obtener un mutuo beneficio. Por lo general, existe una empresa proveedora y otra que actúa como comercializadora o distribuidora de los servicios/productos.

**“String” de conexión:** se define como una cadena que se compone del código necesario para acceder a una fuente de datos (habitualmente a motor de base de datos) y poder realizar una conexión segura.

**W3C:** se define como un consorcio internacional que trabaja en el desarrollo de estándares Web, buscando mantener la idea básica de la World Wide Web.

# Referencias bibliográficas


Ainoa, L. (2014). Introducción al desarrollo de aplicaciones para Android. ICB Editores.

Fielding, R. (2000). Architectural styles and the design of network -based software architectures. Dissertations Publishing. https://www.proquest.com/openview/fc2d064044b971dda476dfb429a2b344/1?pq-origsite=gscholar&cbl=18750&diss=y

Manticore Insane Apps. (2019). Servicios REST HTTP en nativescript. Manticore-labs. https://manticore-labs.com/2019/03/01/servicios-rest-http-en-nativescript/

Nolasco, J. (2019). Desarrollo de aplicaciones con Android. Ra-Ma.

Paredes, M. (2015). Programación multimedia y dispositivos móviles. Ra-Ma.

Raboy, N. (2015). Realizar solicitudes HTTP en una aplicación móvil NativeScript [Web log post]. Blogspot. https://www.thepolyglotdeveloper.com/2015/11/make-http-requests-in-a-nativescript-mobile-application/

Raboy, N. (2016). Trabajar con datos RESTFUL en Angular 2 y NativeScript [Web log post]. Blogspot. https://www.telerik.com/blogs/working-with-restful-data-in-angular-2-and-nativescript

Raboy, N. (2018). Realizar solicitudes HTTP a servicios web remotos en una aplicación NativeScript-Vue [Web log post]. Blogspot. https://blog.nativescript.org/make-http-requests-to-remote-web-services-in-a-nativescript-vue-app/index.html

Serna, S. (2015). Diseño de interfaces en aplicaciones móviles. Ra-Ma.

# Créditos

| Nombre | Cargo | Regional y Centro de Formación |
|---|---|---|
| Claudia Patricia Aristizábal | Responsable del Ecosistema | Dirección General |
| Rafael Neftalí Lizcano Reyes | Responsable de Línea de Producción | Regional Santander - Centro Industrial del Diseño y la Manufactura |
| Wilson Andrés Cuervo Nieto | Instructor | Centro de Diseño y Metrología - Regional Distrito Capital |
| Fabián Leonardo Correa Díaz | Diseñador instruccional | Centro Agropecuario La Granja - Regional Tolima |
| Alix Cecilia Chinchilla Rueda | Evaluadora instruccional | Centro de Gestión Industrial - Regional Distrito Capital |
| Julia Isabel Roberto | Diseñadora y evaluadora instruccional | Centro para la Industria de la Comunicación Gráfica - Regional Distrito Capital |
| Ana Catalina Córdoba Sus | Metodólogo para formación virtual | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Carmen Alicia Martínez Torres | Animador y Productor Multimedia | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Wilson Andrés Arenales Cáceres | Storyboard e ilustración | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Camilo Andrés Bolaño Rey | Locución | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Yerson Fabian Zarate Saavedra | Diseñador de Contenidos Digitales | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Andrea Paola Botello De la Rosa | Desarrollador Full-stack | Centro Industrial del Diseño y la Manufactura - Regional Santander |

| Nombre | Cargo | Regional y Centro de Formación |
|---|---|---|
| Andrea Paola Botello De la Rosa | Actividad didáctica | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Daniel Ricardo Mutis Gómez | Evaluador para Contenidos Inclusivos y Accesibles | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Zuleidy María Ruíz Torres | Validador de Recursos Educativos Digitales | Centro Industrial del Diseño y la Manufactura - Regional Santander |
| Luis Gabriel Urueta Álvarez | Validador de Recursos Educativos Digitales | Centro Industrial del Diseño y la Manufactura - Regional Santander |