****

**NOMBRE DEL PROYECTO**

**R E P O R T E T É C N I C O**

PARA OBTENER EL TÍTULO DE

**TÉCNICO SUPERIOR UNIVERSITARIO EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN, ÁREA SISTEMAS INFORMÁTICOS**

*P R E S E N T A*

**ANTONIO ALONSO PÉREZ**

ASESORA INDUSTRIAL:

ASESORA UNIVERSITARIA:

ORGANIZACIÓN:   
GENERACIÓN DE TERMINO:

**Autorizacion de digitalizacion**

**DEDICATORIA Y RECONOCIMIENTO**

**INDICE**

# RESUMEN

# *ABSTRACT*

# INTRODUCCIÓN

KeD construye soluciones de inteligencia de negocio implementando soluciones tecnológicas de Microsoft y SAP.

KeD es la sede líder de capacitación tecnológica desde 1991, certificado por Microsoft, cuentan con 10 reconocimientos, destacándonos como el mejor centro de capacitación y certificación Microsoft en Latinoamérica.

Capacitan más de 5,000 personas cada año con los más altos índices de satisfacción de nuestros clientes, tenemos consultores especializados en Inteligencia de negocios, nube privada y pública e hibrida, creación de portales, son la empresa más innovadora creando contenidos de distribución en línea.

La empresa cuenta con un sitio web en el que hace referencia a su información y los servicios que ofrecen, pero necesitan una plataforma para poder administrar y gestionar los cursos y capacitaciones que brinda la organización.

Esto se realizará implementando y partiendo de un sistema incompleto de gestión en el cual se innovará la parte de conexión remota por un protocolo de administración para que los datos y la integridad del cliente se encuentre segura al interactuar con el sitio.

Nativelabs Innovación Tecnológica, cuenta con más de 12 años en la industria del desarrollo del *software*, ofreciendo diversos servicios de ingeniería en sistemas, analizando en forma profunda los requerimientos, buscando la mejor solución que cumplan y se adapten a las necesidades de cada uno de sus clientes.

Llevando a cabo distintas actividades partiendo con un análisis bien estructurado, desarrollándose de la mejor manera, buscando el uso de nuevas tecnologías de *software* para que brinde un mejor servicio en los diferentes entornos como lo son en *web*/*internet*, aplicaciones móviles para los sistemas operativos *IOS* y *Android*.

Teniendo como principal virtud la aplicación de estándares, modelos ágiles alineados a los objetivos de negocio y flexibilidad en los tiempos de entrega de cada proyecto, identificando áreas de oportunidades, agregando valor a sus sistemas ya existentes para gestionar un cambio.

Por políticas de privacidad no se hará mención del nombre de la aplicación, ni el nombre de la empresa a la que se le elaboro.

A *Nativelabs* le fue solicitado realizar una aplicación móvil que les permita a los empleados de un banco la opción para consultar su situación crediticia desde cualquier lugar en el momento que lo necesite. A continuación se hace una breve descripción del contenido de los capítulos.

Capítulo 1

En este capítulo se hace mención de una solicitud al requerir de un sistema en el que se les permita a los empleados de un banco la opción de consultar su situación crediticia. Se hace una descripción de los requerimientos funcionales y no funcionales que demanda el cliente para dicha aplicación, se desarrollaron ejemplos de: casos de uso, diagrama de clases y diagrama de componentes, esto para entender el funcionamiento que podría tener dicha aplicación.

Capítulo 2

En éste capítulo se hace una breve explicación sobre el consumo de un servicio *Rest*, abarcando desde la creación de un proyecto, hasta la captura de la petición *Rest* y el pintado de los datos en la pantalla del dispositivo móvil, una explicación sobre las librerías utilizadas, la manera de comunicación que hay entre el *Frontend* con el *Backend* y otras maneras de ver los datos de respuesta por medio de un navegador web.

# OBJETIVOS

**Objetivo general**

**Objetivos específicos**

|  |
| --- |
| **UNIVERSIDAD TECNOLÓGICA DE TECÁMAC** |
| ***DIVISIÓN TIC*** |
| ***PROGRAMA DE ESTADÍAS PROFESIONALES*** |

|  |  |  |
| --- | --- | --- |
| ***PROGRAMA DE TRABAJO*** |  | |
|  | | *FECHA: 03/07/2018* |

*DATOS DEL ALUMNO*

|  |  |
| --- | --- |
| NOMBRE: | Antonio Alonso Pérez |
| DIVISIÓN: | Tecnologías de la Información y Comunicación |
| CARRERA: | Técnico Superior Universitario en Tecnologías de la Información y Comunicación, Área Sistemas Informáticos |
| MATRÍCULA: | 2517160051 |
| GENERACIÓN: |  |

*ASESORA UNIVERSITARIA*

|  |  |
| --- | --- |
| NOMBRE: |  |
| CARGO: |  |

*DATOS DE LA ORGANIZACIÓN*

|  |  |
| --- | --- |
| NOMBRE DE LA  ORGANIZACIÓN: | KeD |
| DEPARTAMENTO: | Centro de capacitación |
| ÁREA: | Desarrollo de software |
| DIRECCIÓN: | Av. Revolución No. 374 Col. San Pedro de los Pinos, C.P. 03800, CDMX. |
| TELÉFONO: | 52785560 |
| E-MAIL: | clientes@ked.com.mx |

*ASESOR INDUSTRIAL*

|  |  |
| --- | --- |
| NOMBRE: |  |
| CARGO: |  |

*PERÍODO*

|  |  |
| --- | --- |
| DURACIÓN: | 15 semanas |
| FECHA DE INICIO: | 03 de septiembre del 2018 |
| FECHA DE TERMINACIÓN: | 12 de diciembre del 2018 |
| HORARIO: | 9:00 am a 4:00 pm |

*PROYECTO*

|  |  |
| --- | --- |
| NOMBRE: |  |
| DESCRIPCIÓN: |  |
| OBJETIVO  GENERAL: |  |
| OBJETIVOS  ESPECÍFICOS: |  |
| ALCANCE(S): |  |
| META(S): |  |
| RECURSOS: | ***Hardware***  1 computadora de escritorio con las siguientes características:  Procesador *Intel Core™2 Duo*.  Memoria *RAM* de 8 *GB*  Disco duro de 150 *GB*.  ***Software***  *Windows* 10 Enterprise(S.O. de 64 *bits*).  *Sublime* text.  *Git*hub Desktop |

*PLAN DE TRABAJO*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ACTIVIDAD | | DESCRIPCIÓN | SEMANA | | FECHAS | |
| INICIO | TÉRMINO | INICIO | TÉRMINO |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

*CRONOGRAMA DE ACTIVIDADES (PROGRAMA)*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | ACTIVIDADES | CONTROL | MAYO | | | | | JUNIO | | | | JULIO | | | | AGOSTO | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | INCIO Y  ELABORACIÓN | PROG. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| REAL |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | DESARROLLO Y  PRUEBAS | PROG. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| REAL |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

*FIRMAS*

# MARCO TEORICO

Se caracteriza por tener un lenguaje teórico donde se define la disciplina a la cual pertenece el campo de estudio escogido, el o los conceptos relevantes y el fenómeno que se quiere profundizar o estudiar.

Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible.

La primera característica, orientado a objetos (*OO*), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para *OO*, una primera idea es diseñar el *software* de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el “comportamiento” (el código) y el “estado” (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables.

Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema *software*. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos.

*Android*

Es un sistema operativo basado en el núcleo *Linux*. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tabletas y también para relojes inteligentes, televisores y automóviles.

*Gradle*

*Gradle* es una herramienta de automatización de compilación de código abierto centrada en la flexibilidad y el rendimiento. Los *scripts* de compilación de *Gradle* se escriben usando *Groovy* o *Kotlin* DSL. Lea sobre las características de *Gradle* para saber qué es posible con *Gradle*.

Altamente personalizable: *Gradle* se modela de una manera que se puede personalizar y ampliar de las formas más fundamentales.

Rápido: *Gradle* completa las tareas rápidamente reutilizando las salidas de las ejecuciones anteriores, procesando solo las entradas que cambiaron y ejecutando las tareas en paralelo.

Potente: *Gradle* es la herramienta de compilación oficial para *Android* y viene con soporte para muchos lenguajes y tecnologías populares.

*Service Rest*

*Rest* (*Representational State Transfer*) es un estilo de arquitectura que abstrae los elementos de dicha arquitectura dentro de un sistema hipermedia distribuido. *Rest* es un conjunto de principios, o maneras de hacer las cosas, que define la interacción entre distintos componentes, es decir, las reglas que dichos componentes tienen que seguir. El protocolo más usado que cumple esta definición, es el protocolo *HTTP*.

Esto quiere decir, por extensión, que toda aplicación *web* bajo el protocolo *HTTP* es a su vez una aplicación *Rest*. Sin embargo, como veremos más abajo, eso no implica en absoluto que todas las aplicaciones *web* sean servicios *web* *RestFul*, ya que estas tienen que cumplir una serie de requisitos para ser consideradas tales. Existen, además, otros tipos de métodos para implementar servicios *web*, que seguramente te suenen, como *RPC* o SOAP. Sin embargo, el uso de dichos mecanismos no se suele recomendar en favor de *RestFul*, ya que *RestFul* es mucho más fácil de entender e implementar. No obstante, como siempre, la decisión sobre implementar una tecnología u otra depende en gran medida de las características del proyecto en el que estemos implicados, por lo que es recomendable siempre hacer un análisis concienzudo del proyecto y las tecnologías disponibles para decantarnos por una o por otra.

Un protocolo cliente/servidor sin estado: cada mensaje *HTTP* contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en *HTTP* utilizan *cookies* y otros mecanismos para mantener el estado de la sesión.

Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: *HTTP* en sí define un conjunto pequeño de operaciones, las más importantes son *POST*, *GET*, *PUT* y *DELETE*. Con frecuencia estas operaciones se equiparan a las operaciones *CRUD* en bases de datos (CLAB en castellano: crear, leer, actualizar, borrar) que se requieren para la persistencia de datos, aunque *POST* no encaja exactamente en este esquema.

Una sintaxis universal para identificar los recursos. En un sistema *Rest*, cada recurso es direccionable únicamente a través de su URI. El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema *Rest* son típicamente *HTML* o *XML*.

Como resultado de esto, es posible navegar de un recurso *Rest* a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

JSON

JavaScript *Object* *Notation* - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, *Standard* ECMA-262 3rd *Edition* - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, *Perl*, *Python*, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras:

* Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
* Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

*Retrofit*

Retrofit convierte su API HTTP en una interfaz Java.

Usa anotaciones para describir la solicitud HTTP:

* Reemplazo de parámetros de URL y soporte de parámetros de consulta
* Conversión de objeto a cuerpo de solicitud (p. Ej., JSON, búferes de protocolo)
* Cuerpo de solicitud multiparte y carga de archivo.

Las anotaciones sobre los métodos de interfaz y sus parámetros indican cómo se manejará una solicitud.

Todos los métodos deben tener una anotación *HTTP* que proporcione el método de solicitud y la *URL* relativa. Hay cinco anotaciones incorporadas: *GET, POST, PUT, DELETE, y HEAD.* La *URL* relativa del recurso se especifica en la anotación.

Manipulación de URL

Una *URL* de solicitud puede actualizarse dinámicamente utilizando bloques y parámetros de reemplazo en el método. Un bloque de reemplazo es una cadena alfanumérica rodeada por { y }. Un parámetro correspondiente debe ser anotado @*Path* usando la misma cadena.

En *Android*, las devoluciones de llamada se ejecutarán en el hilo principal. En la JVM, las devoluciones de llamada se realizarán en el mismo hilo que ejecutó la solicitud HTTP.

*Gson*

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

Hay algunos proyectos de código abierto que pueden convertir objetos Java a *JSON*. Sin embargo, la mayoría de ellos requieren que coloque anotaciones Java en sus clases; algo que no puedes hacer si no tienes acceso al código fuente. La mayoría tampoco respalda completamente el uso de *Java* *Generics*. *Gson* considera que ambos son objetivos de diseño muy importantes.

# METODOLOGIA

Durante el desarrollo de *App*, se implementó la metodología de trabajo llamada *SCRUM*, la cual se basa principalmente en el desarrollo de *software* ágil. *SCRUM* es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente en equipo y obtener el mejor resultado posible de un proyecto.

Esta metodología es muy eficiente ya que en ella se realizan entregas parciales y regulares del producto final. Esto se realizó durante la etapa de desarrollo, lo cual ayudó a modificar el sistema con respecto a las expectativas del cliente y tener mayor satisfacción con los resultados.

*SCRUM* está indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, por ello se desarrolló la aplicación mediante esta metodología, ya que para la empresa, la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Algunos de los motivos por los cuales se determinó esta metodología para la elaboración del proyecto son:

* Es una metodología simple, aunque requiere trabajo duro porque se basa en la adaptación continua a las circunstancias de la evolución del proyecto, esto quiere decir que se adaptó el proyecto a las peticiones del cliente.
* Es una metodología ágil, lo cual permitió desarrollar el proyecto en menos tiempo.
* Orientado a las personas más que a los procesos, se dice que está orientado en las personas ya que se toma más en cuenta la experiencia de usuario.
* Emplea la estructura de desarrollo ágil: incremental basada en iteraciones y revisiones.
* Los requerimientos se capturan como ítems de la lista *Product* *Backlog*, esta lista se realizó durante la etapa de análisis, en la cual se realizó una lista donde se obtuvieron los requerimientos del sistema.
* Se comienza con la visión general del producto, especificando y dando detalle a las funcionalidades o partes que tienen mayor prioridad de desarrollo y que pueden llevarse a cabo en un periodo de tiempo breve.

# CAPÍTULO 1. INICIO Y ELABORACIÓN DE CONSUMO DE SERVICIOS *REST* DESDE DISPOSITIVOS MÓVILES

Es importante mencionar que por cuestiones legales de privacidad y a fin de conservar la confidencialidad de ambas empresas se describe una breve historia del análisis ya implementado con anterioridad del “Consumo de servicios *Rest* desde dispositivos móviles” sólo para fines académicos e históricos.

## Antecedentes del análisis.

El *“*Consumo de servicios *Rest* desde dispositivos móviles” desarrollada en Nativelabs Innovación Tecnológica, surgió de una solicitud al requerir de un sistema en el que pudieran los empleados de un banco la opción de consultar su situación crediticia desde cualquier lugar en el momento que lo necesiten, sin tener que estar conectados en la red interna del banco.

## Identificación de requerimientos funcionales y no funcionales.

La obtención de requisitos se enfoca en la descripción del propósito de sistema, donde el cliente, los desarrolladores y los usuarios identifican un área problema y definen al sistema que resuelve al mismo, la toma de requerimiento sirve como contrato entre el cliente y los desarrolladores, estructura y formaliza durante el análisis para producir un modelo de análisis.

También son todas aquellas declaraciones en las cuales se expresa la manera en la que un sistema se debe de comportar con respecto a las entradas, salidas y detallando específicamente las situaciones particulares para el desarrollo de esta, al igual en la toma de requerimientos se enlista las acciones que se deben de detallar para el proceso de desarrollo.

### Requerimientos funcionales.

Son los procedimientos o actividades necesarias para cumplir con las expectativas necesarias que el cliente pide describiendo las interacciones entre el sistema y su ambiente.

El orden de los requerimientos es proporcionado por los analistas internos del banco, la descripción de cada requerimiento es omitido por políticas de privacidad como ya se hizo mención anteriormente.

1. **RQ01** – Ingreso a la App.
   1. **FN01**-Ingreso al sistema por clave de usuario de empleado para autenticación en LDAP.
2. **RQ02**– Opción “Mis Ofertas” Ofertas Visuales.
   1. **FN01**.- Ofertas visuales en la App en forma de carrusel de los diferentes préstamos susceptibles de contratar según capacidad de empleado con base a la información.
3. **RQ02**– Opción “Mis Ofertas” Simulación.
   1. **FN01**.- Selección de Oferta para realizar Simulación.
   2. **FN02**.- Simulación de la Oferta Seleccionada.
4. **RQ02**– Opción “Mis Ofertas” Contratación.
   1. **FN01**.- Se deberán mostrar el detalle de los datos del préstamo a contratar, solicitar clave de operación y mostrar mensaje de éxito.
5. **RQ02**– Opción “Mis Ofertas” llenado y consulta contrato.
   1. **FN01**.- Se deberá obtener la información del contrato dentro del flujo de la contratación.
6. **RQ02** – Mis Ofertas “Envió de Contrato a e*mail*”.
   1. **FN01**.- Se deberá mostrar mensaje de que el contrato ha sido enviando por mail.
7. **RQ03** – Opción “Mis Préstamos” Posición Global.
   1. **FN01**- Realizar Consulta a la capa de persistencia para mostrar la Posición Global actual del empleado con el detalle de cada Préstamos (en su caso).
8. **RQ03** – Opción “Mis Préstamos” Solicitud de Nuevo Préstamo (Contratación).
   1. **FN01**.- Podrá acceder a lista de Nuevos Préstamos a los cuales tiene derecho el empleado conforme a su capacidad.
9. **RQ03** – Mis Préstamos / Detalle.
   1. **FN02**.- Dentro de la Opción de Mis Préstamos y en el Detalle de uno que este contratado, tendrá la opción de “Liquidar, Recolocar o inclusive Adelantar Pago”.
10. **RQ03** – Mis Préstamos / Gráfica de Barras.

10.1 **FN03**.- Dentro de la Opción de Mis Préstamos se podrá ver en forma de Grafica de Barras “Línea del Tiempo” de los Préstamos Contratados y Simulados.

1. **RQ04** – Balance Quincenal Gráficas de Pastel.

11.1 **FN01**.- Vista en gráfica de pastel de los Saldos y Amortizaciones de sus Préstamos.

1. **RQ04** – Menú Hamburguesa

12.1 **FN01**- Dentro de este Menú se podrán realizar consultas relacionadas a Préstamos Hipotecarios, Preguntas Frecuentes y Consulta de Contratos de los Préstamos vigentes.

### Requerimientos no funcionales

Describen atributos sólo del sistema o del ambiente del sistema que no están relacionados directamente con los requisitos funcionales. Se refieren directamente a las propiedades emergentes como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema.

Requerimientos de interfaz.

* **RNFZ001** La interfaz debe estar diseñada en modo responsivo.
* **RNFZ002** Exponer como servicios *Rest* las funcionalidades para la integración con la aplicación móvil.
* **RNFZ003** Las interfaces deben contener un título de la función que ofrezca.

Requerimientos de personalización

* **RNFP001** En el apartado de cerrar sesión se debe de mostrar el correo respectivo a cada uno de los usuarios.
* **RNFP002** El sistema solo mostrará las funciones pertinentes (Guardar, Eliminar, Cancelar y Actualizar) que pueden realizar cada uno de los usuarios que estén registrados en la plataforma.
* **RNFP003** El sistema debe estar disponible 100% o muy cercano a esta disponibilidad durante el horario hábil laboral.
* **RNFP004** El sistema debe ser construido sobre la base de un desarrollo evolutivo e incremental, de manera tal que nuevas funcionalidades y requerimientos relacionados puedan ser incorporados afectando el código existente de la menor manera posible; para ello deben incorporarse aspectos de reutilización de componentes.
* **RNFP005** La App debe ser de fácil uso y entrenamiento por parte de los empleados de la unidad de transparencia, así como de fácil adaptación de la entidad con el mismo.
* **RNFP006** El sistema debe contar con facilidades para la identificación de la localización de los errores durante la etapa de pruebas y de operación posterior.
* **RNFP007** El sistema debe estar en capacidad de permitir en el futuro su fácil mantenimiento con respecto a los posibles errores que se puedan presentar durante la operación del sistema.

Requerimientos generales

* **RNFG001** Garantizar la confiabilidad, la seguridad y el desempeño del sistema. En este sentido la información almacenada podrá ser consultada y actualizada permanente y simultáneamente, sin que se afecte el tiempo de respuesta.
* **RNFG002** La interfaz debe ser en idioma español.
* **RNFG003** Debe permitir la visualización de toda la información requerida.
* **RNFG004** El tipo de letra de todos los formularios debe ser Arial.
* **RNFG005** Debe ser versátil en la presentación de la información: textos, íconos, etc…

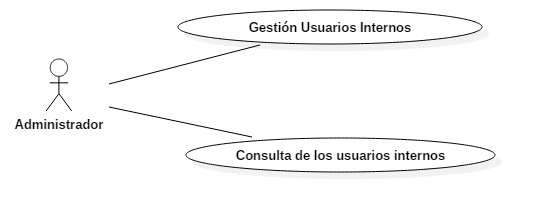
## Elaboración de casos de uso

Es la representación gráfica entre el usuario y un sistema informático, el cual conformado por dos tipos de nodos: actor (representa cualquier elemento que intercambia información con el sistema, por lo que está fuera de él) y caso de uso (es una secuencia de intercambios en dialogo con el sistema que se encuentran relacionadas con su comportamiento).

Los componentes que integran al realizar estos diagramas son:

* Actores: Los usuarios que interactúan con el sistema.
* Casos de uso: las funciones que se realiza el sistema.
* Líneas de flujo: conexiones establecidas entre los actores y caso de uso.

En la figura 1.1 se muestra el diagrama de caso de uso general que se muestra de forma didáctica correspondiente al módulo de usuarios internos, en el cual se identifican el administrador como el único actor encargado de gestionar a los usuarios del tipo *partner*, proveedor o supervisor dentro de la plataforma *web* de servicios.



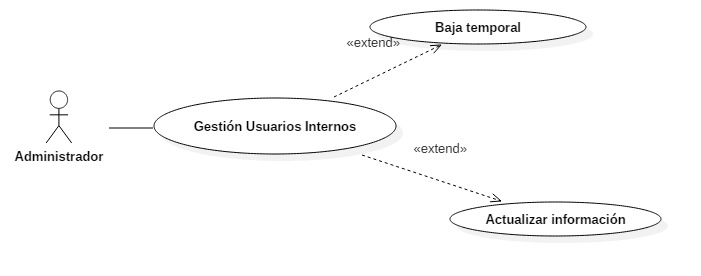
**Figura 1.1** Caso de uso general modulo Usuarios Internos.

Diagrama detalle de caso de uso

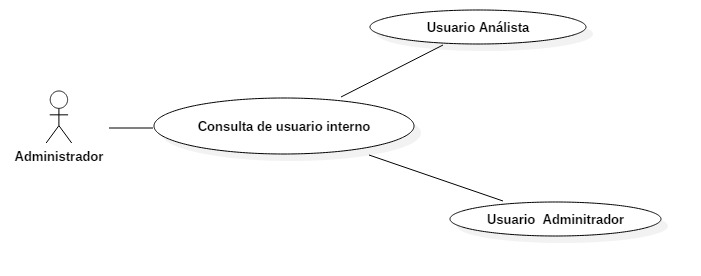
Un diagrama de casos de uso detallado se utiliza para mostrar con más profundidad las operaciones con la que interacciona el usuario. Sirven además como fichas de especificación de requisitos funcionales.

Un caso de uso detallado incluye la relación de dependencia, *include* y *extend*, las cuales son usadas para ligar gráficamente dos casos de uso cuyos flujos de eventos están unidos, normalmente en una solo sesión de usuario.

A continuación se muestran los diagramas detallados de casos de uso de manera didáctica, los cuales hacen referencia al caso de uso general del módulo general de usuarios internos (figura 1.2, 1.3).



**Figura 1.2** Caso de uso Gestionar usuarios.



**Figura 1.3** Caso de uso Consulta de usuarios internos.

Detalles de caso de uso

Es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Estos servirán para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios.

**CU001** Detalle gestionar usuarios.

A continuación se define el detalle de caso de uso, Gestionar usuarios en la figura 1.4.

|  |  |
| --- | --- |
| Caso de uso: | Gestionar usuarios |
| Descripción: | Es donde el administrador podrá realizar funciones tales como : dar de alta un nuevo usuario interno a la plataforma del tipo administrador o analista, dar de baja temporal si la cuenta está inactiva por más de 3 meses y actualizar la información de los mismo usuarios. |
| Actor: | Administrador |
| Precondición: | El administrador tiene que estar dado de alta para poder acceder al sistema. |
| Secuencia: | Flujo normal:   1. El administrador ingresa su usuario y contraseña. 2. El sistema revisa el usuario y contraseña del administrador. 3. El sistema valida que el usuario y contraseña son válidos. Ver **Flujo Alterno “Usuario o contraseña inválida”.** 4. El sistema le da acceso al administrador y le muestra el conjunto de acciones que puede realizar. 5. El administrador selecciona el módulo de usuario internos dentro del menú del sistema y ahí podrá elegir la acción a realizar. 6. Termina el CU.   Flujo alternativo: **“Usuario o contraseña invalida”.**   1. El sistema determina que el usuario o contraseña es inválido. 2. El sistema notifica al administrador que el usuario o contraseña es inválido. 3. Termina el CU.   Notas:   1. Acciones disponibles del módulo de usuarios internos. 2. Dar de alta un nuevo usuario. 3. Actualizar la información   Dar de baja temporal la cuenta de un usuario administrador o analista. |
| Postcondición: | **Flujo básico**  El acceso es permitido al administrador.  **Flujo alterno**  “Usuario o contraseña invalida”.  El acceso no es concedido al administrador. |

**Figura 1.4** Detalle Gestionar usuarios.

**CU002** Detalle consulta de usuarios internos

A continuación se define el detalle de caso de uso, Consulta de usuarios internos en la figura 1.5

|  |  |
| --- | --- |
| Caso de uso: | Consulta de usuarios internos |
| Descripción: | Es donde el administrador visualizar una consulta filtrada por el tipo de cuenta de los usuarios administrador y analista. |
| Actor: | Administrador |
| Precondición: | El administrador tiene que estar dado de alta para poder acceder al sistema. |
| Secuencia: | Flujo normal:   1. El administrador ingresa su usuario y contraseña. 2. El sistema revisa el usuario y contraseña del administrador. 3. El sistema valida que el usuario y contraseña son válidos. Ver **Flujo Alterno “Usuario o contraseña inválida”.** 4. El sistema le da acceso al administrador y le muestra el conjunto de acciones que puede realizar. 5. El administrador selecciona en el menú el módulo de usuarios internos del menú del sistema y posteriormente selecciona la pestaña de consulta de usuarios internos. 6. El sistema realiza la consulta filtrada por el tipo de usuario y la muestra la información dentro de un grid en el cual el administrador podrá elegir otra series de acciones, pero por motivos de confidencialidad no se pueden detallar. 7. Termina el CU.   Flujo alternativo: **“Usuario o contraseña invalida”.**   1. El sistema determina que el usuario o contraseña es inválido. 2. El sistema notifica al administrador que el usuario o contraseña es inválido. 3. Termina el CU.   Notas:   1. Acciones disponibles del módulo de usuarios internos. 2. Dar de alta un nuevo usuario. 3. Actualizar la información   Dar de baja temporal la cuenta de un usuario administrador o analista. |
| Postcondición: | **Flujo básico**  El acceso es permitido al administrador.  **Flujo alterno**  “Usuario o contraseña invalida”.  El acceso no es concedido al administrador. |

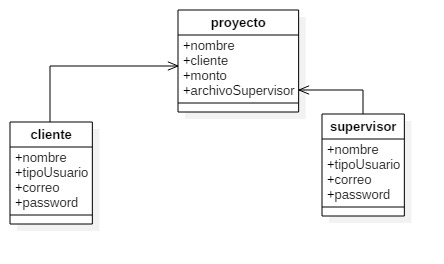
**Figura 1.5** Detalle consulta mejores Partners por zona

## Elaboración de diagrama 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, los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

* El diagrama de elementos: Clases: atributos, métodos y visibilidad.
* Relaciones: Herencia, Composición, Agregación y Uso.

El siguiente diagrama de clases está compuesto solamente de algunos elementos de manera general, ya que por motivos de confidencialidad de la empresa no se puede mostrar el diagrama de clases original que se empleó para el consumo de servicios *Rest* desde dispositivos móviles, figura 1.6.

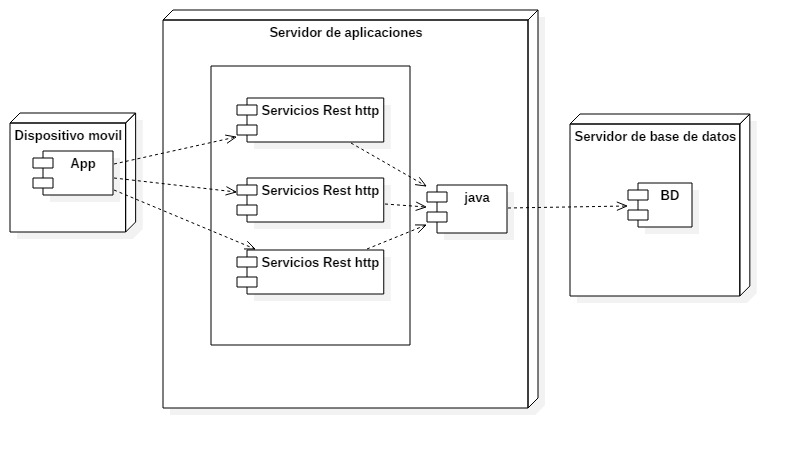


**Figura 1.6** Diagrama de clases

## Elaboración del diagrama de componentes

Un diagrama de componentes muestra los elementos de diseño de un sistema de software y se utilizan para visualizar los componentes físicos que existen en el sistema los cuales pueden ser bibliotecas, paquetes, archivos etc.

El consumo de servicios *Rest* desde dispositivos móviles, los elementos se agrupan en relación con el dominio al que pertenecen y dependiendo al modelo implementado para el desarrollo de la aplicación, el cual es el Modelo Vista Controlador, en la figura 1.7 se muestra el diagrama de componentes como muestra.



**Figura 1.7** Diagrama de componentes

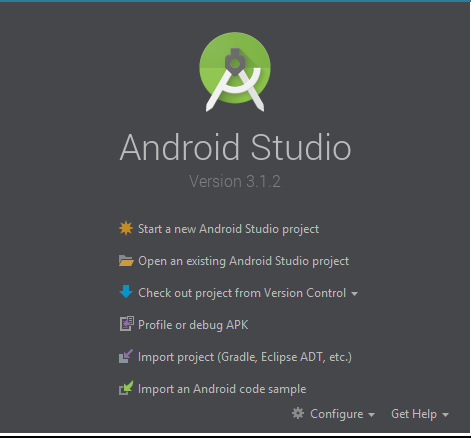
# CAPÍTULO 2. DESARROLLO Y PRUEBAS DE CONSUMO DE SERVICIOS REST DESDE DISPOSITIVOS MÓVILES.

Es importante mencionar que por cuestiones legales de privacidad y a fin de conservar la confidencialidad de ambas empresas, se dará una breve descripción de lo que es la creación de un servicio *Rest* y una aplicación desarrollada en *Android* *Studios*, describiendo el flujo de la consulta y obtención de los datos que serán pintados en la pantalla de nuestro dispositivo *Android*.

## Creación de la aplicación en *Android* *Studios*.

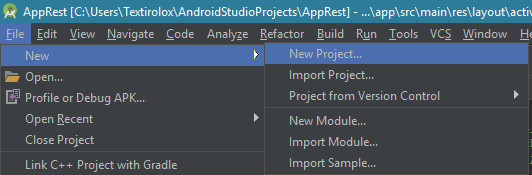
En este caso no se explicara el proceso de instalación, ni de la configuración del IDE de *Android* *Studios*, solo nos concentraremos en la creación de la aplicación (los datos mostrados en la creación son meramente representativos, no son los asignados a la aplicación desarrollada originalmente).

Una vez instalado nuestro IDE de *Android*, tenemos dos maneras de iniciar un proyecto, esto dependerá de nuestro *workspace*, si no se tiene creado ningún proyecto o que nuestro *workspace* se encuentre vacío, podremos iniciar nuestro proyecto como se muestra en la figura 2.1.



**Figura 2.1** Inicio de proyecto con workspace vacío.

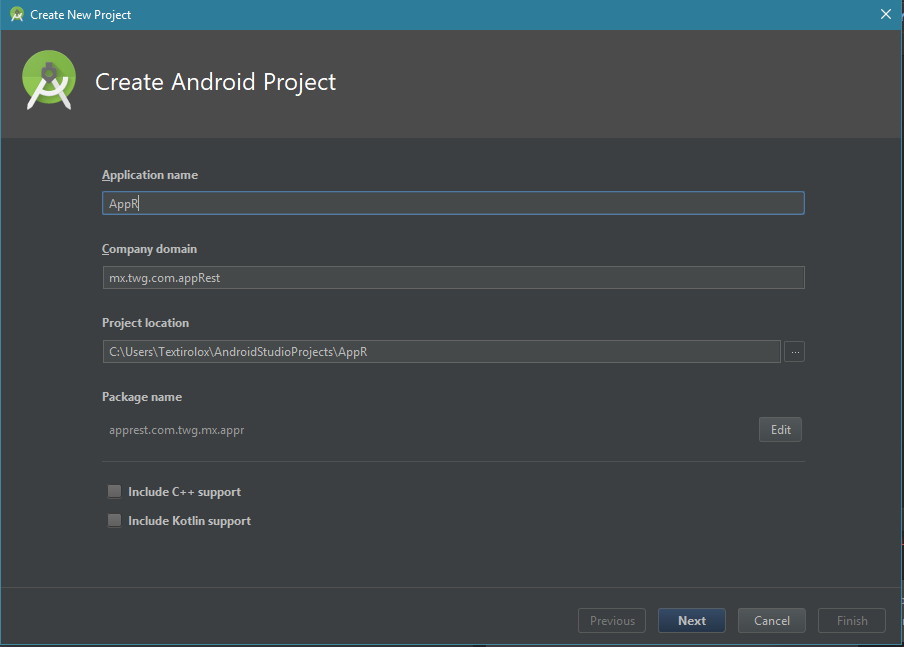
En caso de ya estar trabajando sobre un proyecto y deseamos iniciar otro, nos dirigimos a la barra de menú, dando clic en la opción *File* > *New* > *New* *Project*…, como se muestra en la figura 2.2.



**Figura 2.2** Inicio de proyecto desde a barra de menú.

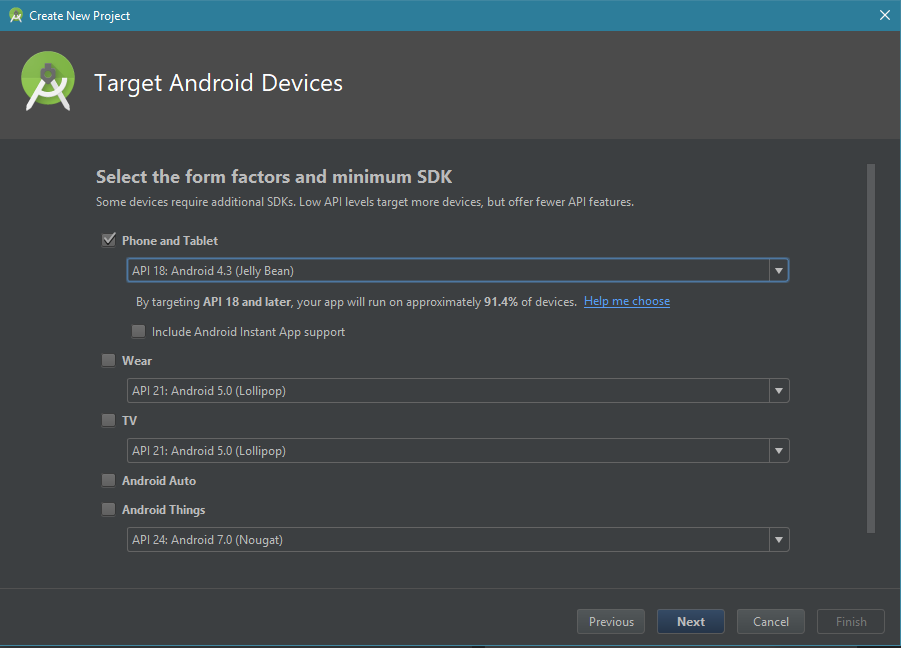
Una vez que seleccionamos la creación de un nuevo proyecto, se nos mostrara la siguiente ventana (figura 2.3) en la cual debemos de especificar las características básicas de nuestra aplicación, asignado en nombre, el dominio de la compañía y la dirección de nuestro *workspace*, ya que tenemos claro nuestras especificaciones, seleccionamos *Next*.

Cabe mencionar que en la sección del dominio de la compañía, *Android* *Studios* lo utiliza como una referencia para la creación de paquetes en nuestro proyecto, más adelante se mostrara la referencia.



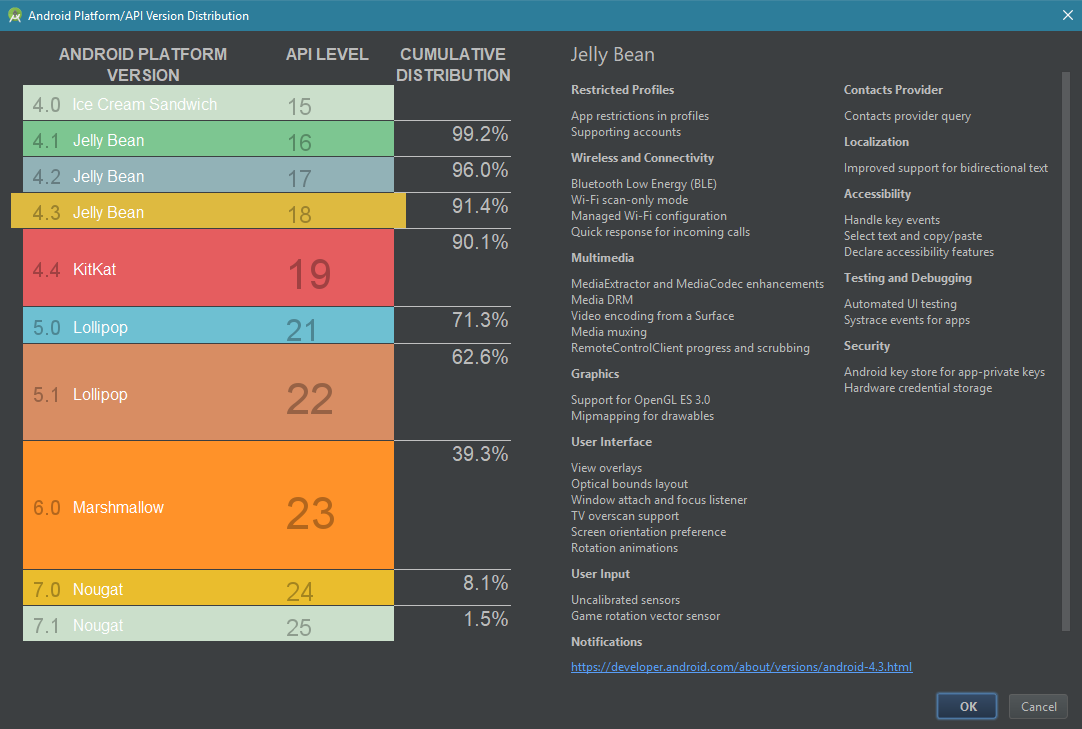
**Figura 2.3** Especificación de características.

En la siguiente ventana (figura 2.4) se debe de especificar la versión en la que la aplicación será desarrollada, tomando en cuenta que las nuevas versiones contengan tecnologías que las versiones anteriores no tengan y fallen a la hora de su instalación, se recomienda utilizar versiones más comerciales. *Android* *Studios* te puede apoyar mostrándote una tabla con los datos de las versiones más usadas (figura 2.5).



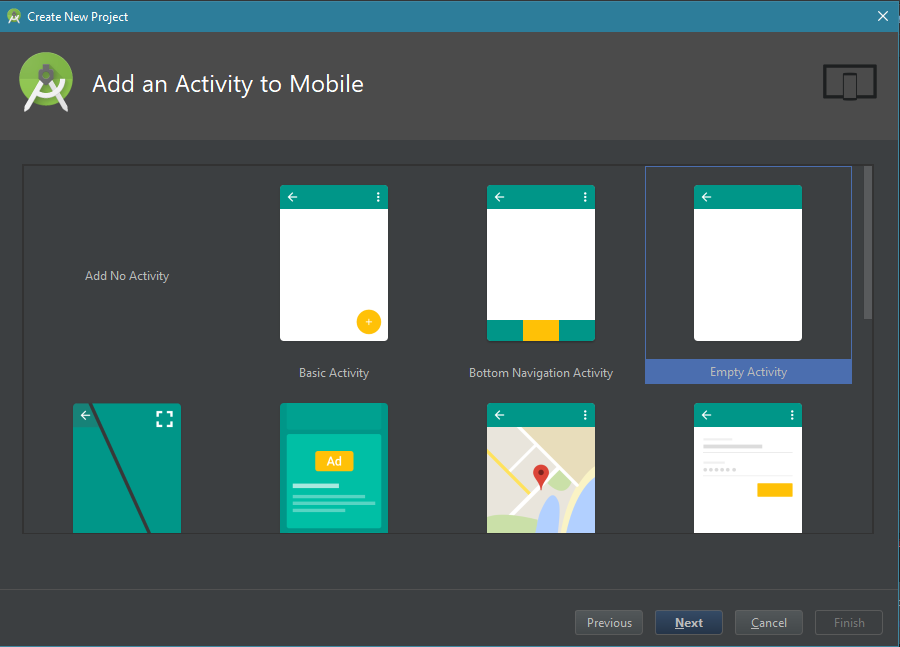
**Figura 2.4** Selección de versión para el desarrollo de la aplicación

El seleccionar la versión para trabajar ya dependerá del criterio del líder de proyecto o como el cliente lo demande de acuerdo a las tecnologías que necesite en su aplicación.



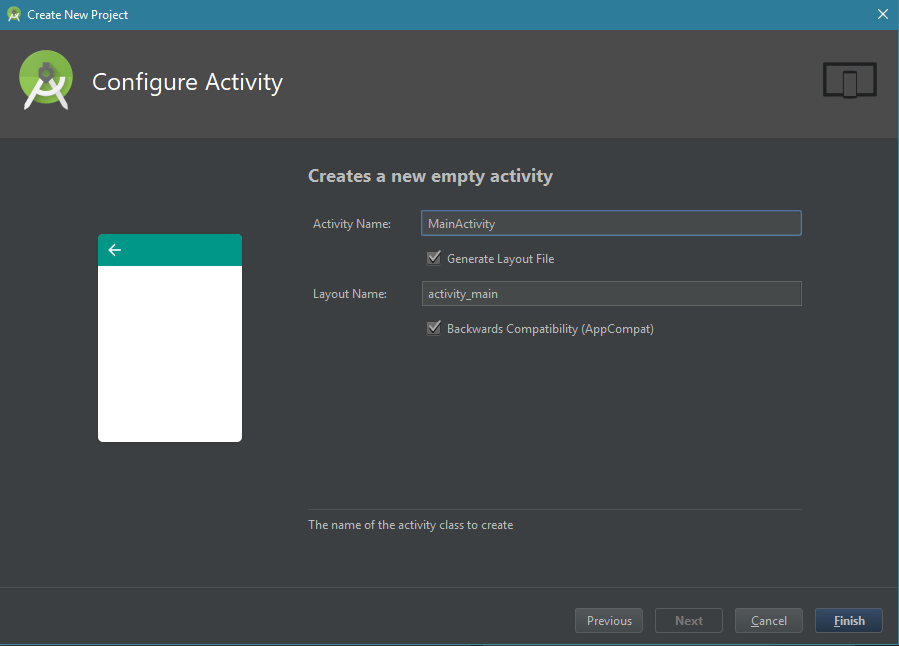
**Figura 2.5** Versiones comerciales de Android.

Ya que tenemos la versión con la cual trabajaremos, es hora de escoger una platilla, puede ser una platilla vacía o alguna ya con un estilo prediseñado, en este ejemplo se utilizó una platilla vacía (figura 2.6).



**Figura 2.6** Catalogo de plantillas

Ya para finalizar la creación de nuestra aplicación o proyecto, debemos de asignar los nombres a nuestros archivos principales tanto como del *Backend* y del *Frontend* (figura 2.7).



**Figura 2.7** Asignación de nombre principal de nuestros archivos Backend y Frontend.

### Librerías implementadas.

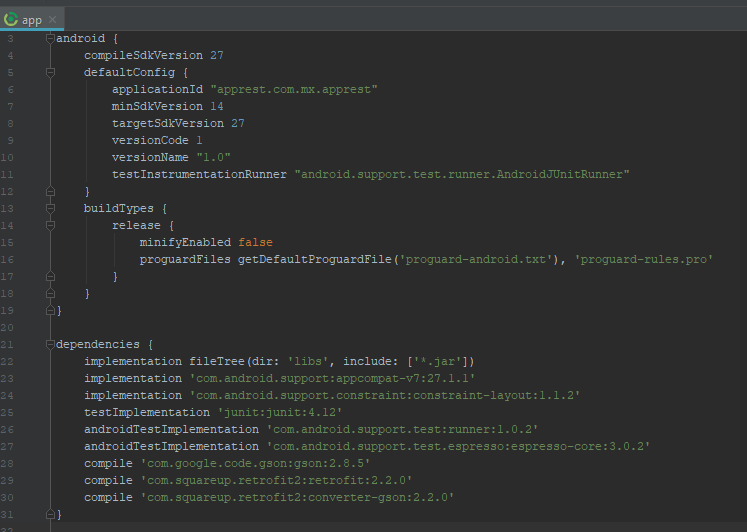
Las librerías utilizadas en la aplicación son:

'com.google.code.gson:gson:2.8.5'  
'com.squareup.retrofit2:retrofit:2.2.0'  
'com.squareup.retrofit2:converter-gson:2.2.0'

* Gson nos permite crear un objeto con los datos de respuesta de nuestro JSON.
* *Retrofit* convierte su API HTTP en una interfaz Java.

Estas librerías se agregan de manera independiente cuando se guarda el archivo Gradle después de hacer los cambios, esta es una de las ventajas que nos ofrece el control de dependencias.

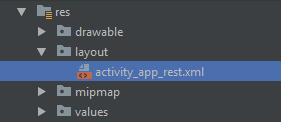
Las librerías agregadas se deben de escribir en el archivo Gradle ( Module:app ) como se muestra en la figura 2.8.



**Figura 2.8** Librerías en nuestra aplicación.

### Desarrollo *Frontend*.

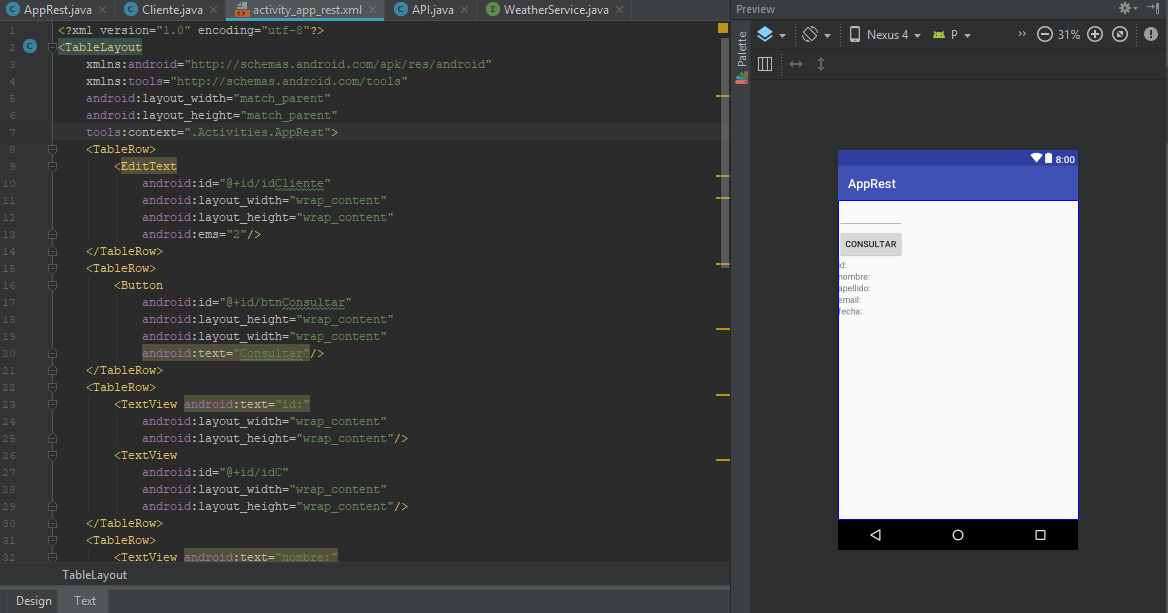
En la parte del *Frontend* (Archivo principal XML, figura 2.9), se recomienda iniciar a trabajar en esta área, así para tener identificados que datos son lo que se van a obtener para realizar nuestros procesos de consulta a nuestro servicio *Rest*.



**Figura 2.9** Archivo XML de nuestro Frontend principal.

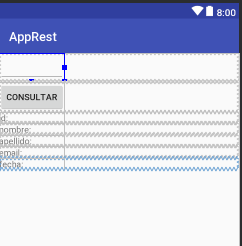
El *layout* utilizado fue *TableLayout*, ya que nos aporta un ordenado a nuestro XML por medio de tablas, dando una presentación ordenada a la aplicación como se muestra en figura 2.10, (Puedes dirigirte en la sección de Anexos del presente documento para identificar el formato XML implementado).

A cada elemento creado de entrada y salida de datos, se le debe de asignar un identificador, esto nos ayudara a que nuestras variables creadas en el *Backend* tenga una relación para la comunicación o flujo de datos. El botón de igual manera debe de tener un identificador.



**Figura 2.10** Vista Frontend con formato XML

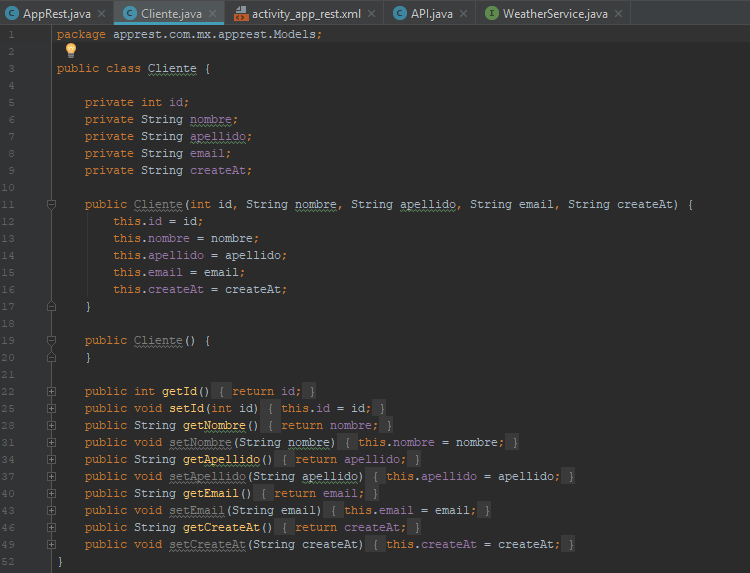
Como se puede observar en la figura 2.11, *Android* nos puede aportar información para identificar de qué manera se está creando la vista de nuestra aplicación.



**Figura 2.11** Muestra del desarrollo Frontend

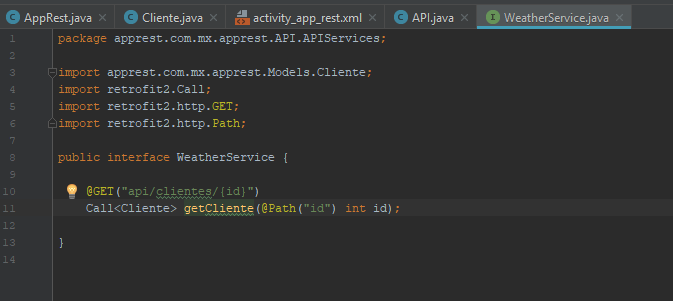
### Desarrollo *Backend*.

En primera estancia se crea nuestro modelo con nuestros métodos *Getters* y *Setters* así como nuestro constructor vacío y un constructor con todos los parámetros de entrada, que será quienes capturen los datos desde nuestra petición *Rest*, las variables utilizada en nuestra clases deben de ser del mismo tipo de dato y el mismo nombre de nuestra respuesta JSON del servicio consumido (figura 2.12).

****

**Figura 1.12** Bean de Cliente

Creamos una *Interface* donde pondremos nuestras peticiones hacia nuestro servicio *Rest* (figura 2.13), mencionando que en una interface solo se escribe la firma del método.



**Figura 2.13** Interface para peticiones HTTP

Aquí nos apoyaremos de la librería de RetroFit utilizando la anotación @GET: Explicando un poco este método como se muestra en la figura 2.13, lo que se encuentra escrito dentro del paréntesis, es la dirección la cual se va hacer la consulta de los datos sin agregar la IP del servidor, en el método GET la información de consulta se va a mostrar en la URI.

@GET(“api/clientes/{id}”)

El valor del dato {id} va a ser dinámico, este valor puede variar según la petición del cliente.

La anotación @Path nos va ayudar en el momento de que se mande a llamar el método getCliente, el dato que se está mandando lo capture y lo asigne a nuestro {“id”}.

Call<Cliente> getCliente(@Path(“id”) int id);

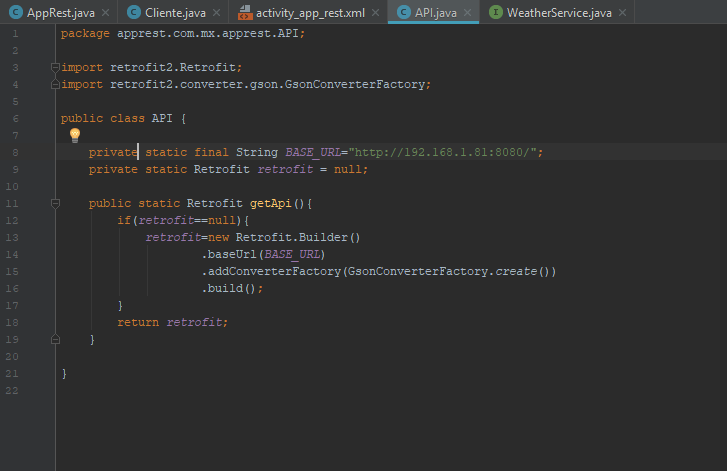
Al recibir la respuesta, la información obtenida debe de ser asignada a una clase de tipo Cliente, por eso debe de ser importante que el nombre de nuestras variables sean iguales a la de nuestra respuesta, ya que así identifica a que variable debe de asignar cada valor obtenido.

Creamos una Clase llamada API (el nombre es indefinido).

Dentro de esta clase creamos una variable privada, estática, final (no es modificable) de tipo *String* (cadena de caracteres), esta variable (BASE\_URL) tendrá el valor que va hacer la URL que no tendrá algún cambio al hacer una consulta, esto sería la IP y el puerto en donde se están exponiendo nuestro servicios, esta es la base para todas nuestras peticiones. Creamos la variable de tipo Retrofit privada y estática con un valor nulo.

Ya que tenemos nuestras dos variables, utilizamos un pequeño patrón de diseño llamado *Singleton,* esto quiere decir que solo se creara una sola instancia de nuestra clase y reusarla, esto para no crear muchas clases de la misma, cuando se manda a llamar el método que contiene su creación y si la instancia ya existe simplemente la retornamos y si no existe la creamos.

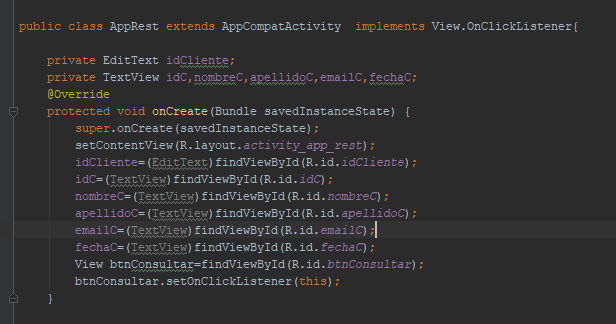
Lo que hace nuestro objeto *Retrofit* es crear nuestro cuerpo de la petición que se hará y en el momento de tener una respuesta, esta respuesta será convertida en un dato tipo GSON para su representación JSON (figura 2.14).



**Figura 2.14** Clase con la Intancia de Retrofit

En nuestra clase principal del *Backend* debemos de declarar nuestras variables globales de acceso privado, estas variables deben de ser según como se hayan asignado los elementos creados en nuestra pantalla en el *Frontend*, como pueden ser botonesy textos que son los de nuestro ejemplo (figura 2.15)*.*

En nuestro método onCreate() debemos de inicializar nuestras variables globales, esto para que nuestro *Frontend* y el *Backend* tenga comunicación por medio de la id asignada.



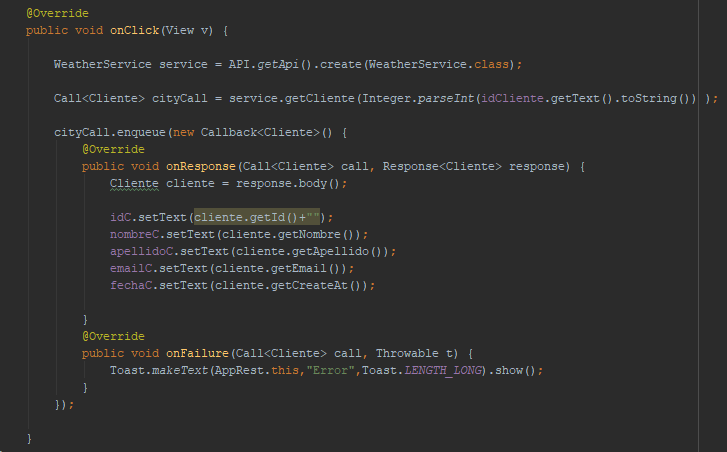
**Figura 2.15** Instancia de elementos EditText. TextView, y Botón oyente.

Nuestra clase se debe de implementar una interface View.onClickListener: *interface* que nos permite que nuestro botón tenga funcionalidad por medio del método onClick() que nos proporciona dicha *interface* al ser implementada.

En el método onClick() debemos de instanciar la *Interface WeatherServices* con nuestra clase *Singleton* para obtener nuestra BASE\_URL de la petición, complementando nuestra petición por medio del método getCliente() que tendrá como parámetro de entrada un dato de tipo entero o numérico.

Cuando realizamos una petición HTTP, debemos de sobre escribir dos métodos de interface Callback<?>, el método onResponse y onFailure: el método onResponse se utilizara cuando la respuesta sea exitosa y el método onFailure cuando la respuesta a fallado, este falla puede ser por no contar con los servicios *Rest* expuesto o una mala captura de los datos de la vista.

El contenido de ambos métodos dependerá del flujo de ejecución.



**Figura 2.16** Método onCLick con los métodos de respuesta HTTP onResponse y onFailure.

El método onResponse obtendrá nuestra respuesta JSON y será asignada a un objeto de tipo Cliente, las cuales serán reasignadas a nuestras variables declaras que pertenecen a la vista, para luego ser mostradas al usuario (figura ##).

El método onFailure solo mostrara un mensaje de error (figura ##).

### Ejecución de la aplicación.

Nos dirigimos a la barra de herramientas y buscamos el botón de ejecución como se muestra en la figura 2.17.

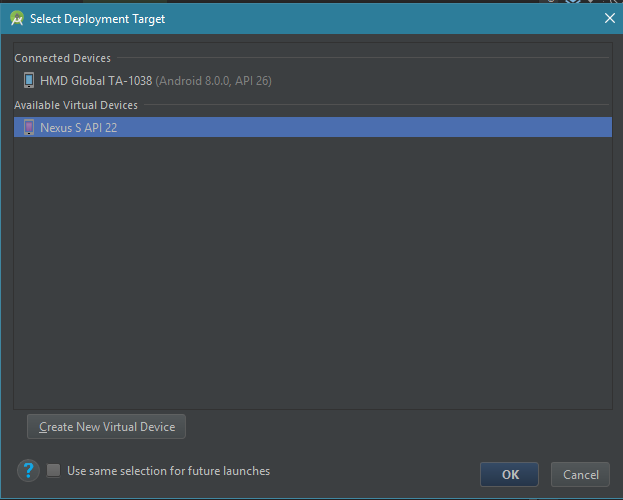


**Figura 2.17** Ejecución de la aplicación.

Android Studios nos ofrece dos maneras de probar nuestras aplicaciones en desarrollo (figura 2.18):

1. Por medio de un dispositivo físico conectado a nuestro ordenador.
2. Por medio de una máquina virtual.

En este caso se utilizó una máquina virtual.

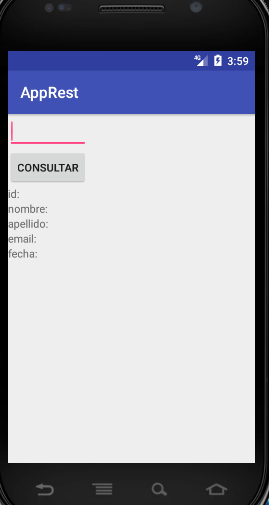


Máquina virtual

Dispositivo físico

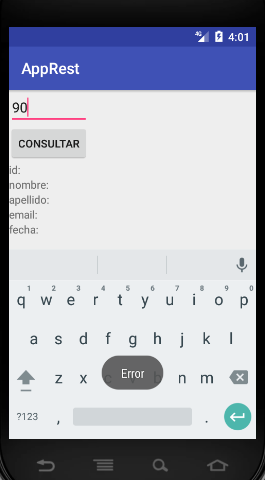
**Figura 2.18** Opción de prueba de aplicaciones.

Si nuestra aplicación esta bien diseñada, la aplicación podrá ejecutarse sin ningún inconveniente, está mostrara en la pantalla de inicio una vista sin datos de entrada o salida como se muestra en la figura 2.19.

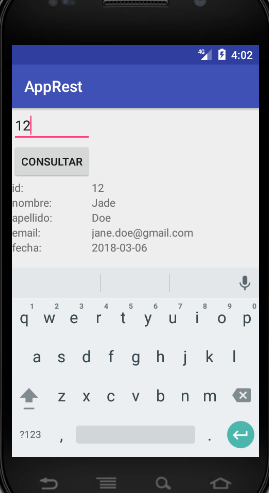


**Figura 2.19** Pantalla de inicio.

Si el servicio expuesto no cuenta con datos de la consulta requerida, mostrara un mensaje de error y los campos se quedaran vacíos (figura 2.20), si la consulta es de algún dato existente se mostraran los datos en los campos correspondientes (figura 2.21).

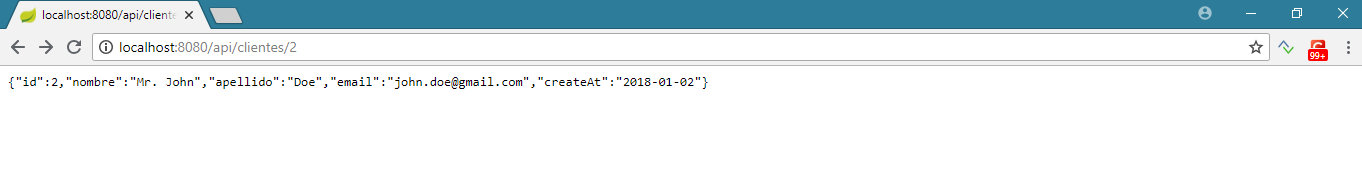


**Figura 2.20** Consulta fallida

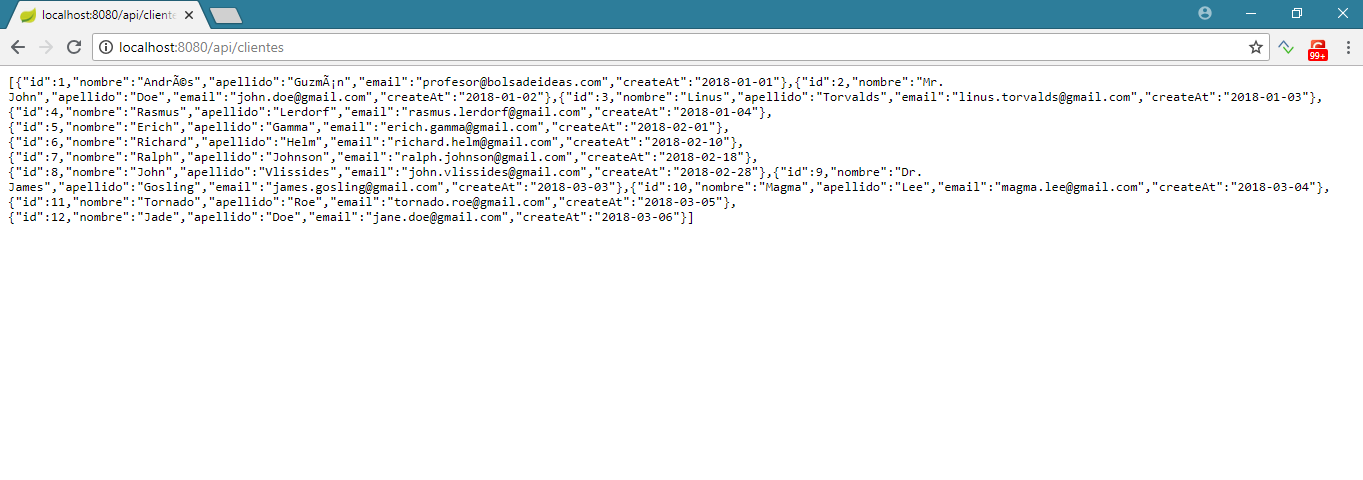


**Figura 2.21** Consulta exitosa

Otra manera de verificar los servicios que se exponen de algún proveedor, podemos escribir la URL en el buscador, como respuesta de nuestra búsqueda será una respuesta JSON como se muestra en la figura 2.22, considerando que por medio de la URL se manda un valor entero para obtener un dato especifico (también puede ser un carácter o cadena de caracteres), en la figura 2.23 se muestra una consulta general de todos los registros existentes con el proveedor, la única variante es que no se anexa un valor en la URL.



**Figura 2.22** Consulta de datos específicos



**Figura 2.23** Consulta general de datos

# CONCLUSIONES

La aplicación desarrollada ha cumplido con los requerimientos demandados por el banco, tanto en su funcionalidad como en la vista de la aplicación.

Las ventajas que nos ofrece la tecnología *Rest* puede ser aplicada en muchos ámbitos, ya que no solo se puede aplicar para la funcionalidad administrativa, también podemos ver su implementación en las redes sociales, videos juegos, entre otras aplicaciones.

El nivel de seguridad en los datos incrementa, aunque el desarrollo de una aplicación no solo se trata de escribir código, también se involucran la ética del desarrollador, en esta ocasión se desarrolló una aplicación externa al banco, así que solo se envían peticiones para transacciones. Las transacciones de los préstamos son directamente en los sistemas del banco por medio de esas peticiones. La discreción de la información es muy importante ya que el hacer mal uso de esta puede proceder a demandas, incluso en ir a prisión.

Por medio de los servicios *Rest* las aplicaciones móviles han mejorado su rendimiento debido a que la persistencia de datos no se genera en el mismo dispositivo, el utilizar un formato JSON como envió o respuesta de peticiones hace que el flujo de información sea rápido, mencionando que un servidor también se involucra en la velocidad de las respuestas.

Muchas personas utilizan estas tecnologías sin saber cómo funcionan, por medio de este reporte técnico una persona inexperta en las tecnologías *Rest* puede iniciar a comprender su funcionalidad.

Cabe mencionar que esta tecnología no solo se utiliza en el desarrollo de aplicaciones móviles, también se utilizan en otras tecnologías como C#, C++, JAVA, AngularJS, JQuery, entre otras.

# ANEXOS Y/O APÉNDICES

Presentan la información que apoya el contenido del Reporte.

## XML Del Frontend Android

<?xml version="1.0" encoding="utf-8"?>  
<TableLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".Activities.AppRest">  
 <TableRow>  
 <EditText  
 android:id="@+id/idCliente"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:ems="2"/>  
 </TableRow>  
 <TableRow>  
 <Button  
 android:id="@+id/btnConsultar"  
 android:layout\_height="wrap\_content"  
 android:layout\_width="wrap\_content"  
 android:text="Consultar"/>  
 </TableRow>  
 <TableRow>  
 <TextView android:text="id:"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 <TextView  
 android:id="@+id/idC"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 </TableRow>  
 <TableRow>  
 <TextView android:text="nombre:"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 <TextView  
 android:id="@+id/nombreC"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 </TableRow>  
 <TableRow>  
 <TextView android:text="apellido:"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 <TextView  
 android:id="@+id/apellidoC"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 </TableRow>  
 <TableRow>  
 <TextView android:text="email:"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 <TextView  
 android:id="@+id/emailC"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 </TableRow>  
 <TableRow>  
 <TextView android:text="fecha:"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 <TextView  
 android:id="@+id/fechaC"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
 </TableRow>  
</TableLayout>

# LISTADO DE SIGLAS O ACRÓNIMOS, GLOSARIO

*App*: Programa que puede ser instalado en dispositivos móviles y computadores para que el usuario realice distintos tipos de tareas.

*Backend*: Enfocado en hacer que todo lo que está detrás de un sitio *web* funcione correctamente. Toma los datos, los procesa y los envía al usuario, además de encargarse de las consultas o peticiones a la Base de Datos, la conexión con el servidor, entre otras tareas que debe realizar en su día a día.

Bean: Es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.

C #: Lenguaje fuertemente tipado. Cada variable y constante tiene un tipo, como lo hace cada expresión que evalúa un valor.

C ++: Lenguajes de programación más utilizados en el mundo. Los programas bien escritos de C ++ son rápidos y eficientes. El lenguaje es más flexible que otros lenguajes porque puede usarlo para crear una amplia gama de aplicaciones, desde juegos divertidos e interesantes hasta *software* científico de alto rendimiento, controladores de dispositivos, programas integrados y aplicaciones cliente de *Windows*.

*Cookies*: Es una pequeña pieza de información que queda almacenada en tu navegador cuando visitas algún sitio *web*.

CDDL: *Common* *Development* *and* *Distribution* *License* (Licencia Común de Desarrollo y Distribución).

*Delete*: Este método es utilizado para solicitar al servidor que elimine un archivo en una ubicación específica.

*DSL*: *Digital* *Subscriber* *Line*.

*Frontend*: Se enfoca en el usuario, en todo con lo que podemos interactuar y lo que vemos mientras navegamos. Así como en una primera cita, nuestra *web* busca causar una buena impresión y agradar al usuario, para lo cual utiliza HTML, CSS y JAVASCRIPT.

*GET*: El método *GET* significa que va a recuperar cualquier información.

*Groovy*: Lenguaje de programación orientado a objetos implementado sobre la plataforma Java.

GNU GPL: La Licencia Pública General de GNU o más conocida por su nombre en inglés GNU *General* *Public* *License* es una licencia de derecho de autor ampliamente usada en el mundo del *software* libre.

IDE: *Integrated* *Development* *Environment* (Entorno de desarrollo integrado), aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de *software*.

Instancia: Es la particularización, realización específica u ocurrencia de una determinada clase, entidad o prototipo.

Hipermedia: Término con el que se designa al conjunto de métodos o procedimientos para escribir, diseñar o componer contenidos que integren soportes tales como: texto, imagen, video, audio, mapas, etc.

HTML: Lenguaje de marcado que se utiliza para el desarrollo de páginas de *Internet*.

HTTP: Abreviatura de la forma inglesa *Hypertext* *Transfer* *Protocol*, ‘protocolo de transferencia de hipertextos’, que se utiliza en algunas direcciones de *internet*.

Embebido: Un sistema embebido o empotrado (integrado, incrustado) es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas, frecuentemente en un sistema de computación en tiempo real.

IBM: *International* *Business* *Machines* *Corporation* es una reconocida empresa multinacional estadounidense de tecnología y consultoría con sede en Armonk, Nueva York.

iOS: Sistema operativo móvil de la multinacional *Apple* *Inc*. Originalmente desarrollado para el *iPhone* (*iPhone* *OS*), después se ha usado en dispositivos como el *iPod* *touch* y el *iPad*.

IP: *Internet* *Protocol* (Protocolo de Internet), Se trata de un protocolo de comunicaciones a través de la red, especie de "matrícula" para identificarte cuando estás conectado a la red.

Java EE: *Java* *Enterprise* *Edition* (Edición Empresarial Java).

LDAP: Protocolo Ligero de Acceso a Directorios.

JavaScript: Lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, ​ basado en prototipos, imperativo, débilmente tipado y dinámico.

*Kotlin*: Lenguaje de programación de tipado estático que corre sobre la máquina virtual de Java y que también puede ser compilado a código fuente de JavaScript.

*Perl*: Lenguaje optimizado para escanear archivos de texto arbitrarios, extraer información de esos archivos de texto e imprimir informes basados ​​en esa información.

*POST*: Método se usa cuando se necesita enviar una entidad para algún recurso determinado.

*PUT*: usado para solicitar que el servidor almacene el cuerpo de la entidad en una ubicación específica.

*Product* *Backlog*: Lista de objetivos / requisitos priorizada, representa la visión y expectativas del cliente respecto a los objetivos y entregas del producto o proyecto.

Python: Lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Puerto: Numeración lógica que se asigna a las conexiones, tanto en el origen como en el destino. No tiene ninguna significación física.

RPC: Consiste en un protocolo que permite a un *software* o programa ejecutar código en otra máquina remota sin preocuparse por la comunicación, por lo regular es bastante utilizado en el paradigma cliente y servidor.

SOAP: Protocolo derivado de XML que nos sirve para intercambiar información entre aplicaciones.

*Software*: Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.

*Script*: conjunto de órdenes guardadas en un archivo de texto, generalmente muy ligero y, que es ejecutado por lotes o línea a línea, en tiempo real por un intérprete.

URI: Es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

URL: Se trata de la secuencia de caracteres que sigue un estándar y que permite denominar recursos dentro del entorno de Internet para que puedan ser localizados.

*Visual* *Age*: Es una Herramienta de desarrollo generalmente rápido de aplicaciones creada por IBM, esta herramienta permite generar aplicaciones basadas o no en *Internet* ya que esta permite al usuario importar y exportar código.

*Web*: Conjunto de información que se encuentra en una dirección determinada de internet.

Workspace: una instalación de almacenamiento de memoria para uso temporal en el desarrollo de proyectos.

*Internet*: Red informática de nivel mundial que utiliza la línea telefónica para transmitir la información.

XML: Especificación para diseñar lenguajes de marcado, que permite definir etiquetas personalizadas para descripción y organización de datos.

# BIBLIOGRAFÍA

Carlos Fontela. (2011). UML. Barcelona: Alfaomega.

Contributors. (Enero 17, 2017). JavaScript Reference. Julio 10, 2018, de Microsoft,

Sitio web: <https://docs.microsoft.com/en-us/scripting/javascript/reference/javascript-reference>.

Dougles Bell & Mike Parr. (2011). JAVA para estudiantes. México: Pearson.

Diversos Autores. (December 1999). Introducing JSON. Junio 28, 2018, de json.org Sitio web: <https://www.json.org/>.

Fco. Jvier Ceballos. (2002). C/C++. Madrid: Alfaomega.

Google. (2003). Documentation for app developers. Junio 4, 2018, de Google Sitio web: <https://developer.android.com/docs/>.

Gradle. (2018). Gradle User Manual. Junio 4, 2018, de Gradle.org Sitio web: <https://docs.gradle.org/current/userguide/userguide.html>.

IBM. (MAY, 2004). VisualAge RPG Language Reference. Julio 10, 2018, de IBM Sitio web: <https://www.ibm.com/support/knowledgecenter/en/ssw_i5_54/rzahg/rzahgrpgvisage.htm>

IBM. (2018). developerWorks. Julio 16, 2018, de IBM Sitio web: <https://www.ibm.com/developerworks/?lnk=mdev_dw_mxes&lnk2=learn>.

Jesus Tómás, Vicente Carbonéll, Miguel García, Jordi Bataller, Daniel Ferri. (2014). El gran libro de Android avanzado. México: Alfaomega.

Joel Francia. (September 25, 2017). ¿Qué es Scrum?. Mayo 22, 2018, de scrum.org Sitio web: <https://www.scrum.org/resources/blog/que-es-scrum>.

kotlinlang. (2018). Statically typed programming language for modern multiplatform applications. Julio 11, 2018, de kotlinlang.org Sitio web: <https://kotlinlang.org/>.

Margaret Rouse. (December 2017). REST (REpresentational State Transfer). Junio 19, 2018, de Search Microservices Sitio web: <https://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer>.

Oracle. (2014). Java 8. Junio 4, 2018, de Oracle Sitio web: <https://docs.oracle.com/javase/8/docs/api/>.

Paul Deitel, Harvey Deitel. (2015). Como Programar en Java Deitel. United States of American: Person.

swankjesse . (Feb 13, 2017). Retrofit. Jul 10, 2018, de Open Source Sitio web: <http://square.github.io/retrofit/>.