
UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA


# SIMULADOR DE CELDAS FOTOVOLTAICAS CON DISPOSITIVOS MÓVILES

TESIS
PARA OBTENER EL GRADO DE
**INGENIERO EN REDES**


PRESENTA
**OSCAR IVÁN OJEDA ALDANA**

DIRECTOR DE TESIS
**DR. JAIME SILVERIO ORTEGÓN AGUILAR**

ASESORES
**DR. HOMERO TORAL CRUZ**
**M.T.I. MELISSA BLANQUETO ESTRADA**
**DR. JAVIER VÁZQUEZ CASTILLO**
**DR. JOSÉ HERNÁNDEZ RODRÍGUEZ**





Chetumal Quintana Roo, México, Diciembre de 2015

# UNIVERSIDAD DE QUINTANA ROO
## DIVISIÓN DE CIENCIAS E INGENIERÍA

**TRABAJO DE TESIS ELABORADO BAJO SUPERVISIÓN DEL COMITÉ DE ASESORÍA Y APROBADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:**

**INGENIERO EN REDES**

**COMITÉ DE TRABAJO DE TESIS**

**DIRECTOR:** _____

**DR. JAIME SILVERIO ORTEGÓN AGUILAR**

**ASESOR:** _____

**MTI. MELISSA BLANQUETO ESTRADA**

**ASESOR:** _____

**DR. HOMERO TORAL CRUZ**





Chetumal Quintana Roo, México, Diciembre de 2015

## Agradecimientos

Durante mis seis años de formación como ingeniero he conocido a personas maravillosas que me han acompañado todo este tiempo, a aquellos compañeros de clase que se convirtieron en mis amigos, a aquellas personas que me acompañaron en las desveladas para poder llegar a ser la persona que soy hoy en día les doy las gracias. También quisiera agradecer a mis profesores, tanto de mi primera institución de educación superior como de la Universidad de Quintana Roo, no habría obtenido mejor formación de otra manera, a aquellos profesores que vieron potencial en mí y me motivaron a explotarlo, en especial los profesores M.S.I Rubén Gonzales Elixavide y al Dr. Jaime Silverio Ortegón Aguilar por su guía.

A otra institución que quisiera darle mi gratitud es al Consejo Quintanarroense de Ciencia y Tecnología por el apoyo durante el programa de jóvenes investigadores en el año 2015 que me abrió un nuevo panorama para mi formación profesional.

Y, por último, pero no menos importantes, quisiera agradecer a mis padres, por enseñarme cada día a ser una mejor persona y ver que no me falte nada aun siendo un adulto.


# UNIVERSIDAD DE QUINTANA ROO

## División de Ciencias e Ingeniería


## Resumen

A lo largo de este documento se explora el desarrollo y prueba de una aplicación de simulación para dispositivos móviles. Durante el primer capítulo se provee una introducción de los objetivos y el porqué del proyecto.

En el segundo capítulo de este documento se sienta la teoría que sustenta el desarrollo de la aplicación y se hace una descripción sobre las tecnologías y librerías utilizadas para llevarlo a cabo, en este capítulo se da una breve descripción de las librerías utilizadas que servirán como referencia para otros desarrollos que lo requieran, no solamente proyectos de simulación o energías alternativas.

Para el tercer capítulo, se explica a profundidad el desarrollo teórico de la aplicación, las clases creadas y sus funciones y métodos y cómo estas interactúan para llevar a cabo la simulación. En el capítulo cuarto se encuentran las pruebas de funcionamiento, además de explicaciones de la interfaz de usuario y su despliegue como aplicación universal para dispositivos Android.

## Contenido

# CAPÍTULO 1

# CAPÍTULO 1. INTRODUCCIÓN

*"La energía solar no es una energía alternativa: es la energía"* - **Hermann Scheer**

Las energías renovables, como lo es la energía solar, provienen de recursos naturales que vienen en una cantidad ilimitada, pudiendo recurrir a estos de manera constante, contrario a las fuentes de energía convencionales que son limitadas y en algunos casos perjudiciales para el ambiente; las energías renovables como recurso limpio, producen prácticamente cero impactos gracias a la nula emisión de gases de efecto invernadero.

Los sistemas de energía fotovoltaica a gran y pequeña escala han tenido gran auge en años reciente debido a sus potenciales beneficios a largo plazo y al rápido crecimiento de la electrónica de potencia. Aunado a esto, los costos asociados a este tipo de sistemas se han reducido de manera considerable debido a diversas alternativas provistas por gobiernos para promover la energía limpia. En este sentido, el modelado y simulación de celdas fotovoltaicas es crucial en el desarrollo, prueba y análisis de generación de energía. Actualmente existen emuladores y simuladores de celdas fotovoltaicas que permiten generar curvas de corriente-voltaje y potencia-voltaje bajo ciertas condiciones ambientales dadas, pero con un costo monetario considerable.

En años recientes, la Universidad de Quintana Roo ha participado en el desarrollo de un emulador de celdas fotovoltaicas basado en procesadores ARM, dicho esto, se llega a la idea de que sistemas embebidos para la simulación pueden ser reemplazados por dispositivos móviles como los teléfonos inteligentes. A lo largo de este documento se explora el desarrollo y prueba de una aplicación de simulación para dispositivos móviles.

## Justificación

Actualmente la Universidad de Quintana Roo participa en la creación de un emulador de celdas fotovoltaicas basado en procesadores ARM, esto llevó a la idea de que es posible desarrollar una interfaz para dispositivos móviles capaz de realizar los cálculos necesarios para generar las curvas de corriente-voltaje y potencia-voltaje y que pueda ser, por tanto, portable a distintos dispositivos.

Los sistemas fotovoltaicos han tenido un gran auge en años recientes debido al rápido crecimiento y avances en la tecnología de semiconductores y electrónica de potencia. Adicionalmente, el costo asociado a este tipo de sistemas se redujo, teniendo un impacto en los costos de producción de energía eléctrica. En este sentido, el modelado y emulación de sistemas fotovoltaicos es crucial para el diseño y prueba de sistemas de energía fotovoltaicos, análisis de la generación de energía y desarrollo de sistema de control, interfaces, entre otros. Actualmente hay modelos para las celdas fotovoltaicas que permiten generar las curvas de corriente-voltaje y potencia-voltaje para los paneles fotovoltaicos bajo condiciones ambientales dadas, pero la mayoría de los emuladores son costosos.

## Objetivo General

Desarrollar un simulador de celdas fotovoltaicas para dispositivos móviles para capaz de generar curvas de corriente-voltaje y potencia-voltaje.

## Objetivos Particulares

- Portar el código existente escrito en Python a un dispositivo móvil en Java.
- Implementar una opción para usar los sensores del dispositivo móvil como datos de entrada para el simulador (luz considerando 505nm) y temperatura.
- Implementar una opción para consultar información como temperatura e irradiancia directamente de Internet.
- Realizar la depuración de la aplicación en dispositivos móviles.

## Alcance

La principal función del software desarrollado es simular el comportamiento de distintas celdas fotovoltaicas dada la información proporcionada en las hojas de datos de sus respectivos fabricantes. En este momento se hace la simulación de una sola celda y no se hará la conexión con el convertidor DC-DC necesario para realizar una emulación.

El código escrito en Python consta de diferentes librerías para poder llevar a cabo su funcionamiento, librerías matemáticas y de graficado, mismas que se buscará su equivalencia para poder realizar funciones similares en Java.

Además, para enriquecer el uso de la aplicación, se implementarán opciones para el uso de sensores y acceso al hardware para proveer de mejores y más herramientas que el desarrollo original, mientras que, a su vez, se mantendrá la opción de consulta web para obtener la información requerida para llevar a cabo la simulación.

# CAPÍTULO 2

# CAPÍTULO 2. MARCO TEÓRICO

El software de simulación que se describe en este documento fue desarrollado con base en el artículo *A high-accuracy photovoltaic emulator system using ARM processors* [1], utilizando un conjunto de funciones y librerías específicamente para su desarrollo y que se describen a continuación. En dicho artículo se menciona lo referente en cuanto a la implementación de un emulador de celdas fotovoltaicas escrito en lenguaje Python y siendo ejecutado en un sistema embebido como los es una tarjeta de desarrollo Beagle Bone, en el software mencionado, se evalúa el modelo de dos diodos descrito por Ishake y otros en el artículo *Simple, fast and accurate two-diode model for photovoltaic modules* [2].

## Modelo de Dos Diodos (Two-Diode Model)

El presente modelo de dos diodos presenta “un buen compromiso entre precisión y complejidad de operaciones computacionales” [1], en la Figura 1 se puede apreciar de manera lógica el modelo de dos diodos que fue utilizado de igual manera por Atoche y otros [1] y que servirá para el desarrollo del proyecto a través de la documentación.

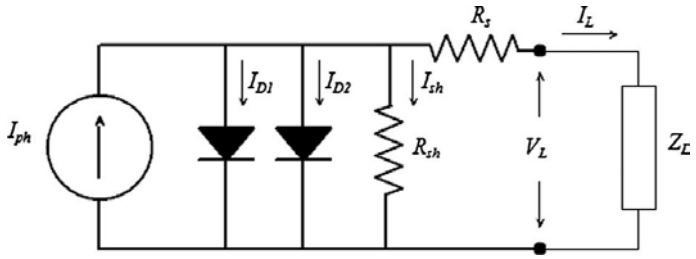



Figura 1 - Modelo de celda fotovoltaica [1]

Y es el modelo referente para llevar a cabo la simulación de celdas fotovoltaicas, para mayor información puede consultar los artículos previamente mencionados ya que el desarrollo de ello abarca más allá del ámbito de esta sección.

## Java

Java es un estándar global para desarrollo y distribución de aplicaciones móviles y embebidas, contenido basado en web, etc. Java está diseñado para permitir el desarrollo de aplicaciones portátiles de elevado rendimiento para el más amplio rango de plataformas informáticas posibles, de esta manera, desarrolladores y empresas pueden proporcionar más servicios y mejorar la productividad, las comunicaciones y la colaboración del usuario final y reducir drásticamente el costo de propiedad tanto para aplicaciones de usuario como de empresa. Java nos permite:


Figura 2 - Logo de Java

- Escribir software multiplataforma.
- Crear programas que se puedan ejecutar en un explorador web y acceder a recursos Web disponibles y locales.
- Desarrollar aplicaciones de servidor para foros en línea, almacenes, encuestas, procesamiento de formularios HTML y mucho más.
- Combinar aplicaciones o servicios que utilizan el lenguaje Java para crear aplicaciones o servicios con un gran nivel de personalización.
- Escribir aplicaciones potentes y eficaces para teléfonos móviles, procesadores remotos, microcontroladores, módulos inalámbricos, sensores, gateways, productos de consumo y prácticamente cualquier otro dispositivo electrónico [3].

En la Figura 3 se puede observar un conjunto de dispositivos y tecnologías que ejecutan Java para su funcionamiento.


Figura 3 - Ecosistema de Java

## Python

Python es un lenguaje orientado a objetos, interpretado e interactivo. Incorpora módulos, excepciones, tipado dinámico, tipos de dato de alto nivel y clases. Python combina potencia notable junto a una sintaxis clara y limpia, es decir, altamente legible para el ser humano. Tiene muchas interfaces a llamadas de sistema y librerías, así como a diversos sistemas de ventanas y es extensible en C o C++, lo que nos permite escribir fácilmente nuevos módulos dependiendo de la implementación a realizar.


Figura 4 - Logo de Python

Por último, Python es portátil, se ejecuta en muchas variantes de Unix en Mac y en PC bajo DOS, Windows, Windows NT y OS/2 [4]. A continuación, en la Figura 5 se puede observar un ejemplo de código escrito en lenguaje Python.

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

Figura 5 - Factorial en Python

## Android

Figura 6 - Logo de Android

"Android es una pila (stack) de software para dispositivos móviles en los que se incluye un sistema operativo, middleware y aplicaciones clave. El SDK de Android provee las herramientas y APIs necesarias para empezar a desarrollar aplicaciones en la plataforma Android utilizando el lenguaje Java." [5]

### Antecendentes

Figura 7 - HTC Dream

Android fue liberado al público en octubre de 2008 y fue implementado inicialmente con un solo Smartphone, el HTC Dream que se puede apreciar en la Figura 7, también conocido comúnmente como el T-Mobile G1. Desde entonces, muchos fabricantes de smartphones fueron liberando dispositivos impulsados por Android. En junio de 2010 había alrededor de 60 dispositivos compatibles con Android distribuidos en una red de colaboradores de 21 OEMs, 59 operadoras de telefonía celular y 49 países. El volumen y variedad de los dispositivos impulsados por Android continúan excediendo las expectativas aún más optimistas. [6]

Debido a su apertura y el uso del lenguaje Java, Android se volvió popular para desarrolladores de juegos móviles, según estadísticas se han descargado aplicaciones de la tienda de aplicaciones de Android unas 23,693,511,105 veces [7] al momento de ser consultados, dicho número sigue en aumento.

La siguiente es una descripción de Android y es solamente una breve introducción como está constituido el sistema y conceptos que nos introducen a términos y

notaciones clave que serán mencionados más adelante para describir el sistema de simulación de celdas fotovoltaicas.

## API (Application Programming Interface)

Un nivel de API es un valor entero que identifica de forma exclusiva la revisión del framework API que ofrece una versión de la plataforma Android.

La plataforma Android provee un framework API que las aplicaciones pueden usar para interactuar con el sistema Android. El framework API consiste de:

- Un conjunto básico de paquetes y clases
- Un conjunto de atributos y elementos XML para la declaración del archivo de manifiesto.
- Un conjunto de atributos y elementos XML para la declaración y acceso a los recursos.
- Un conjunto de Intents o intenciones.
- Un conjunto de permisos que las aplicaciones pueden solicitar, como también refuerzos de permisos incluidos en el sistema.

Cada versión sucesiva de la plataforma Android puede incluir actualizaciones del framework API de Android, además que cada versión es compatible en algunos casos con versiones de sistema anteriores mediante librerías de soporte.

A continuación, en la Tabla 1 se puede apreciar el nivel de API con su versión del sistema Android respectivamente.

Tabla 1 - Niveles de API [7]

| Versión de la plataforma | Nivel de API | Código de versión |
|---|---|---|
| **Android 6.0** | 23 | M |
| **Android 5.1** | 22 | LOLLIPOP_MR1 |
| **Android 5.0** | 21 | LOLLIPOP |
| **Android 4.4W** | 20 | KITKAT_WATCH |
| **Android 4.4** | 19 | KITKAT |
| **Android 4.3** | 18 | JELLY_BEAN_MR2 |
| **Android 4.2, 4.2.2** | 17 | JELLY_BEAN_MR1 |
| **Android 4.1, 4.1.1** | 16 | JELLY_BEAN |
| **Android 4.0.3, 4.0.4** | 15 | ICE_CREAM_SANDWICH_MR1 |
| **Android 4.0, 4.0.1, 4.0.2** | 14 | ICE_CREAM_SANDWICH |
| **Android 3.2** | 13 | HONEYCOMB_MR2 |
| **Android 3.1.x** | 12 | HONEYCOMB_MR1 |
| **Android 3.0.x** | 11 | HONEYCOMB |
| **Android 2.3.4**<br>**Android 2.3.3** | 10 | GINGERBREAD_MR1 |
| **Android 2.3.2**<br>**Android 2.3.1**<br>**Android 2.3** | 9 | GINGERBREAD |
| **Android 2.2.x** | 8 | FROYO |
| **Android 2.1.x** | 7 | ECLAIR_MR1 |
| **Android 2.0.1** | 6 | ECLAIR_0_1 |
| **Android 2.0** | 5 | ECLAIR |
| **Android 1.6** | 4 | DONUT |
| **Android 1.5** | 3 | CUPCAKE |
| **Android 1.1** | 2 | BASE_1_1 |
| **Android 1.0** | 1 | BASE |

Las aplicaciones pueden hacer uso del elemento del manifiesto proporcionado por el framework API <*uses-sdk*> que describe los niveles máximo y mínimo de API sobre los cuales son capaces de ejecutarse, así como también el nivel preferido de API para el cual fueron diseñadas [8].

Los niveles de API nos permiten a los desarrolladores hacer uso de nuevas características y funciones que se agregan conforme se actualiza la versión de sistema, un nivel de API bajo nos garantiza que la aplicación (app) pueda ejecutarse

en un mayor número de dispositivos disponibles en el mercado, mientras que un nivel más avanzado de API nos limita a unos cuantos. Para vencer esa limitante, Google libera con cada actualización de API una librería de soporte, para poder hacer uso de las nuevas funciones en versiones de sistema anteriores que no soportarían dichas funciones de forma nativa.

## Arquitectura del Sistema Android

El sistema Android es una poderosa pila (stack) de software como se describió anteriormente, y está típicamente divida en cuatro áreas como se describe en la siguiente Figura 8, una capa de aplicaciones, framework para aplicaciones, librerías y runtime y el kernel Linux, las librerías escritas en C/C++ se ubican sobre la capa del kernel Linux y son accesibles para las aplicaciones mediante el uso de librerías, lo que permite utilizar un lenguaje de alto nivel como lo es Java para acceder a recursos de hardware, como las información de los sensores del dispositivo, comunicación de red, cámaras y audio.

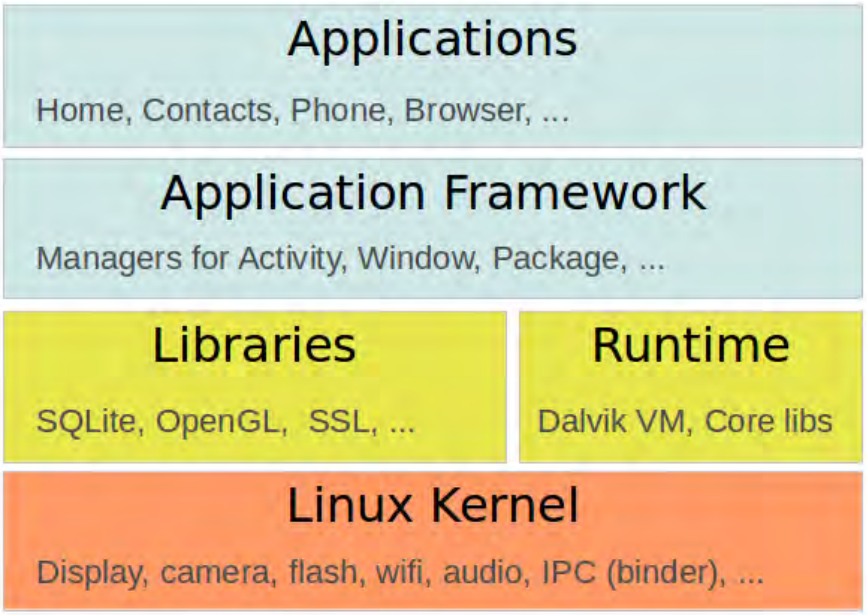



Figura 8 - Pila de software de Android [9]

Los niveles pueden ser descritos en resumen de la siguiente manera:

*Aplicaciones* – Cada dispositivo viene con un conjunto de aplicaciones preinstaladas por default, ya se sea por la capa de personalización del fabricante del dispositivo o porque forman parte del AOSP (Android Open Source Project), como por ejemplo el navegador, la cámara o el marcador. Estas aplicaciones operan en la misma capa de aplicaciones de terceros, como las que se obtienen a través de la tienda de aplicaciones de Google conocida como Google Play Store, que además de aplicaciones ofrece otro tipo de contenido.

*Framework de aplicaciones* – Es una API que provee interacción de alto nivel con el sistema Android desde las aplicaciones [9]. Esta capa contiene importantes manejadores responsables de controlar el ciclo de vida de las aplicaciones, proveer acceso a información compartida como contactos, gestión de recursos del sistema y de las aplicaciones instaladas.

*Librerias y Runtime* – Es esta capa se ubican las librerías para diversas funciones comunes, como el renderizado de gráficas, almacenamiento de datos, navegación web, etc., así como las bibliotecas de Java para ejecutar aplicaciones en Android. En el runtime se ubican las librerías que forman parte del núcleo de Android y la máquina virtual Dalvik (que ejecuta las aplicaciones mediante la interpretación de código pre-compilado) o ART (que genera binarios al momento de instalar una aplicación para ser ejecutada con mayor rapidez) dependiendo de la versión de sistema instalada.

Kernel Linux – Capa de comunicación con el hardware como comúnmente se le conoce en el ámbito de la informática. Fue desarrollado originalmente por Linus Torvalds en 1991 y constituye una pieza fundamental en Android.

Figura 9 - Logo de Linux

## Componentes de Aplicaciones

El framework de aplicaciones de Android nos permite crear aplicaciones con contenido multimedia enriquecido y de gran calidad utilizando un conjunto de componentes reutilizables de los cuales se describen a continuación los utilizados.

### Actividad

Una actividad es un componente de una aplicación que provee una pantalla donde el usuario puede interactuar con el fin de realizar alguna acción, como realizar una llamada telefónica, tomar una fotografía o enviar un email. Cada actividad está dada por una ventana que dibuja su interfaz de usuario. En la Figura 10 se puede ver un ejemplo de una actividad recién creada que ocupa toda la pantalla del dispositivo.

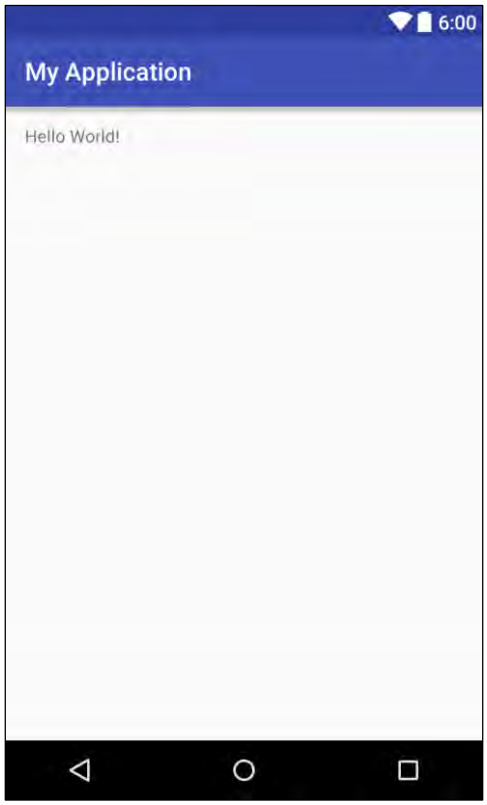



Figura 10 - Actividad en blanco

Una aplicación consiste típicamente en múltiples actividades que se encuentran enlazadas entre sí. Normalmente una de estas actividades es considerada la actividad “principal” que es la que se le presenta al usuario el ejecutar la aplicación por primera vez (no confundir con first run). Cada actividad puede iniciar a otras actividades para realizar diferentes acciones, como por ejemplo ir a una “ventana” de preferencias de la aplicación.

Cada que una nueva actividad es iniciada (Start), la actividad anterior es detenida (Stop), pero el sistema conserva esta actividad en una pila (la pila de retroceso o back stack).

Cuando una actividad es detenida a causa del inicio de una nueva actividad, esta es notificada en sus cambios de estado a través de métodos de devolución de llamadas de ciclo de vida [10]. Existen diferentes métodos que llevan el control del ciclo de vida de una actividad los cuales serán explicados más adelante en este documento.

## Fragmentos

Un fragmento (Fragment) representa un comportamiento o una porción de una interfaz de usuario en una actividad (Actividad). Se pueden combinar múltiples fragmentos en una sola actividad para construir una interfaz de usuario de configuración dinámica como se observa en la Figura 11 y reutilizar un fragmento en múltiples actividades.

Se puede pensar en un fragmento como una sección modular de una actividad, tiene su propio ciclo de vida, recibe sus propios eventos y se puede agregar o remover mientras la actividad está en ejecución.

Un fragmento siempre debe estar embebido en una actividad y es afectado por el ciclo de vida de la propia actividad que lo contiene, es decir, si la actividad que lo contiene entra en pause, entonces todos los fragmentos contenidos entrarán en pausa.

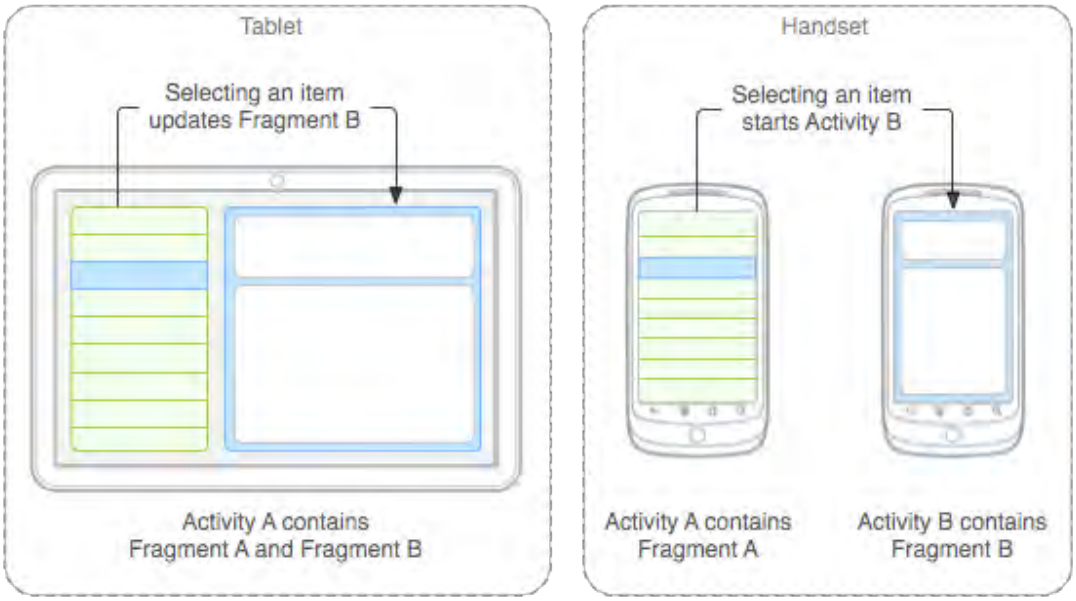



Figura 11 - Fragmentos en configuración dinámica [11]

Cuando un fragmento es agregado como parte de la interfaz de una actividad, este "vive" dentro de un ViewGroup dentro de la jerarquía del actividad que lo contiene, y este a su vez contiene su propia interfaz [11].

Un fragmento no siempre es necesario para el funcionamiento de una aplicación, puede haber actividades sin contener fragmentos, pero no puedo haber una aplicación sin actividad ni fragmento suelto, ya que depende de las actividades.

## Ciclo de Vida

Una actividad y/o un fragmento puede estar en distintos estados dependiendo de cómo está interactuando con el usuario, estos estados son descritos a continuación en la siguiente Tabla 2 - Ciclo de vida.

Tabla 2 - Ciclo de vida

| Estado | Descripción |
|---|---|
| **Running** | La Actividad es visible y el usuario está interactuando en ella. |
| **Paused** | La Actividad continua visible pero parcialmente "oscura", su instancia está siendo ejecutada, pero puede ser killed por el sistema. |
| **Stopped** | La Actividad no es visible, su instancia está siendo ejecutada, pero puede ser killed por el sistema. |
| **Killed** | La Actividad ha sido detenida por el sistema con la llamada del método finish() |

### *Métodos de ciclo de vida*

El sistema Android define un ciclo de vida para las actividades y fragmentos por medio de métodos de ciclo de vida predefinidos. Los métodos más importantes son los siguientes que se muestran en la Tabla 3.

Tabla 3 - Métodos de ciclo de vida [12]

| Método | Propósito |
|---|---|
| **onCreate()** | Llamado cuando la Actividad es creada. Es usado para inicializar la Activity, por ejemplo, para crear al interfaz de usuario. |
| **onResume()** | Llamada si la Actividad vuelve a ser visible y el usuario interactúa con la Actividad de nuevo. Es usado para inicializar campos, registrar listeners o servicios. |
| **onPause()** | Es llamado una vez que otra actividad toma el primer plano. Siempre es llamada cuando una actividad ya no es visible. Es usado para liberar recursos o para almacenar información de la |

| | |
|---|---|
| | aplicación, al igual que para remover listener y servicios de sistema. |
| **onStop()** | Es llamado una vez que la actividad ya no es visible y no se necesitará durante algún tiempo. En caso de que haya escases de recursos la actividad puede ser destruida. |

Android posee otros métodos en su ciclo de vida de los cuales no se garantiza que todo sean llamados, en la Figura 12 - Ciclo de vida de una Actividad se puede apreciar un diagrama de flujo del ciclo de vida de una Actividad.

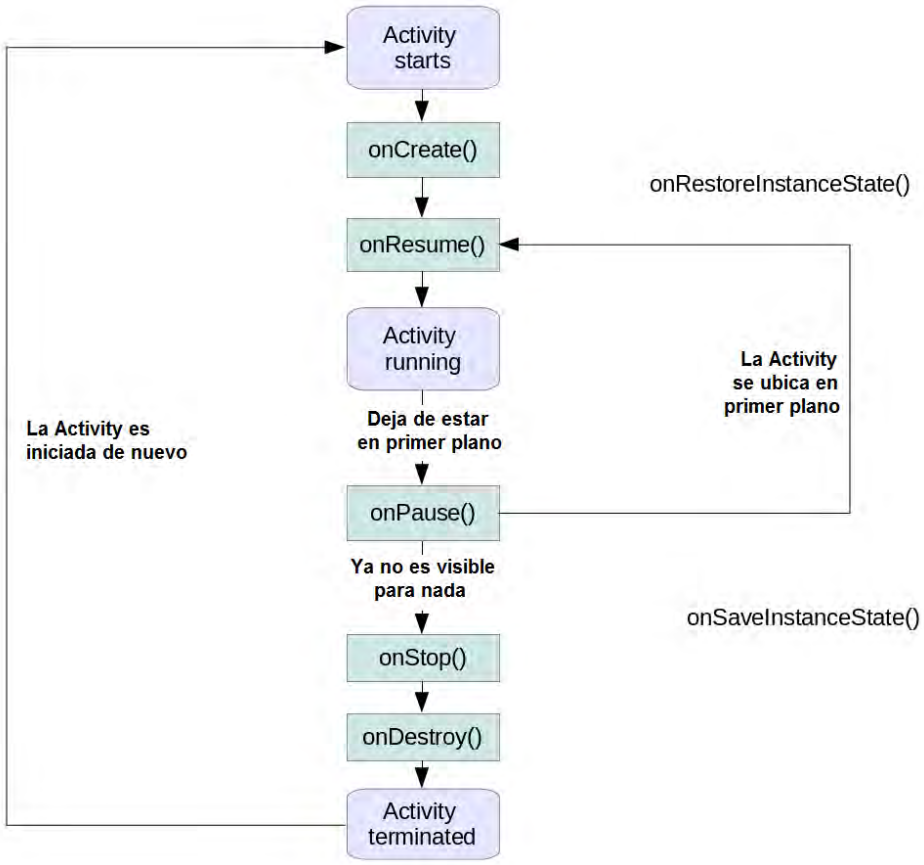



Figura 12 - Ciclo de vida de una Actividad [12]

## Librerías de Soporte

Las librerías de soporte de Android son un conjunto de librerías de código que proveen retro-compatibilidad con versiones anteriores del framework de la API de Android. Esto significa que por diseño las aplicaciones escritas con APIs recientes pueden utilizar funciones de las librerías de soporte y funcionar con versiones anteriores de Android, a partir del nivel de API 4 en adelante.

La inclusión de estas librerías en un proyecto de Android es considerada como una buena práctica para los desarrolladores de aplicaciones, dependiendo del rango de versiones a las que la aplicación está dirigida, el uso de estas librerías puede ayudar incrementando el desempeño de la aplicación y haciendo que esta llegue a más usuarios.

Las librerías de soporte proveen diferentes características para cada target de nivel de API, para poder hace un uso efectivo de estas librerías es importante considerar qué características requieren soporte y entender cuáles son compatibles con cada librería en determinado nivel de la API de Android [13].

## Interfaz Gráfica de Usuario

En Android como en cualquier aplicación de software, la interfaz gráfica de usuario es todo lo que el usuario puede ver y que permite la interacción con el software. El SDK de Android provee una variedad de componentes de interfaz gráfica de usuario prefabricados como objetos de distribución estructurada (layouts) y ciertos controles (widgets) que permiten construir interfaces gráficas para nuestras aplicaciones. Android también provee un conjunto de módulos para interfaces especiales como son los diálogos (dialogs), notificaciones (push y toast) y menús [14].

Todo elemento de interfaz de usuario es construido utilizando objetos View y ViewGroup. Un View es un objeto que dibuja algo en la pantalla con lo que el usuario puede interactuar. Un ViewGroup es un objeto que sostiene otros objetos View y ViewGroup de tal forma que definen la distribución de la interfaz de usuario, como, por ejemplo, haciendo uso de diversos fragmentos dentro de una misma actividad para formar una interfaz gráfica modular que permita adaptarse a distintas configuraciones (tamaño, densidad y orientación) de pantalla [15].

Para poder definir la interfaz gráfica, debemos tener conocimiento de dos componentes, los layouts en XML y los Widgets (también conocidos como forms) que provee los elementos dentro de los layouts.

### *Layouts*

El layout es el componente de la aplicación que define la estructura de la interfaz gráfica de usuario, tanto para una Actividad, Fragmento o Widget de pantalla de inicio, para definir un layout se requieren dos cosas:

- Definir los elementos de la interfaz gráfica en un archivo XML.
- Instancias los elementos definidos del archivo XML dentro del código.

Android provee la flexibilidad de modificar el estado de los componentes mediante código de la aplicación, por ejemplo, no es necesario declarar todos los aspectos

de un botón, como el texto o color dentro del archivo XML que define la interfaz gráfica, estos puedes cambiarse a través de código en Java de la aplicación y puede modificarse durante su ejecución de manera dinámica [16]. A continuación, podemos ver un ejemplo de la estructura de la definición de un layout en XML.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
   <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
   <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

Dicho layout está conformado por un contenedor principal de orientación vertical, un TextView que es el equivalente al JLabel de Java con el texto “Hello, I am a TextView”, seguido de un botón que lleva el texto “Hello, I am a Button”.

Entre los contenedores principales o padres utilizados en este proyecto se encuentran los siguientes:

LinearLayout – Este no permite distribuir el contenido de dos maneras, de manera vertical y de manera horizontal, se puede embeber para poder acomodar los elementos de la interfaz gráficas de la manera que uno desee, la única limitante es el tamaño de la pantalla, para eso entra en juego en otro layout padre utilizado, el ScrollView.

ScrollView – Este layout es utilizado en conjunto con LinearLayout y sirve para proveer contenido desplazable a través de la pantalla, para cuando el espacio provisto por el LinearLayout no es suficiente de manera horizontal y/o vertical.

### *Widgets*

Los widgets, también conocidos como forms o controles de entrada, son los componentes que sirven para que el usuario interactúe con la aplicación, Android provee una serie de controles de entrada predefinidos que cualquier desarrollador puede utilizar en sus aplicaciones como botones, campos de texto, radio, etc [17].

Buttons - Un botón, es un objeto de la interfaz gráfica que puede ser presionado o clicado por el usuario para disparar una acción, estos se definen de la siguiente manera en el archivo XML de la interfaz gráfica. En la Figura 13 – Botón se puede observar los distintos aspectos que puede adoptar un botón en una aplicación en Android.

```
<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/button_text"
   ... />
```


Figura 13 – Botón [18]

TextFields - Un TextField o campo de texto es un campo editable que se usa para introducir caracteres alfanuméricos en nuestra aplicación, puede utilizarse con distintas configuraciones que proveen de mecanismos para la evaluación de expresiones regulares, tanto en el archivo XML de su interfaz como en el código Java de la aplicación. Se define de la siguiente manera en el archivo XML. Se puede

observar un ejemplo de un campo de texto en la Figura 14 - Campo de texto en la aplicación para correo electrónico Gmail.

```
<EditText
   android:id="@+id/email_address"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:hint="@string/email_hint"
   android:inputType="textEmailAddress" />
```

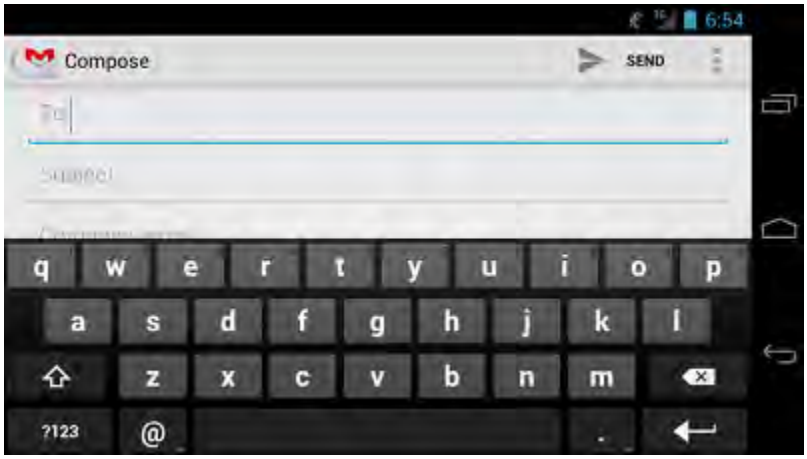



Figura 14 - Campo de texto [19]

Spinner - Un Spinner es un elemento de la interfaz gráfica que nos sirve para seleccionar un elemento de una lista desplegable, para "poblar" el Spinner se puede realizar desde el archivo XML o desde el código Java de nuestra aplicación. Su definición en XML es la siguiente, también se puede observar un ejemplo de una lista desplegable en la Figura 15 – Spinner .

```
<Spinner
   android:id="@+id/planets_spinner"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content" />
```

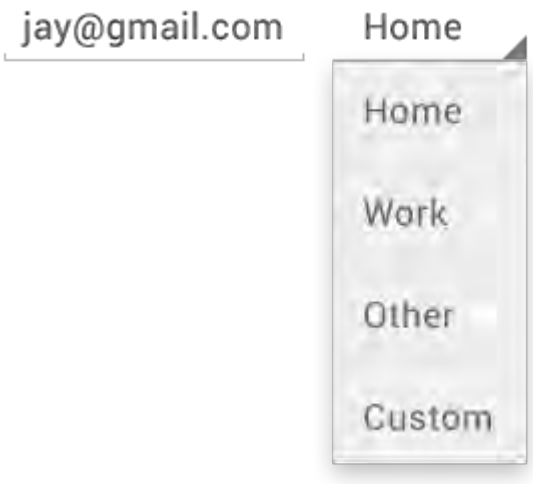



Figura 15 – Spinner [20]

## Manifest de Aplicación

Cada aplicación debe contener un archivo de manifiesto llamado AndroidManifest.xml en su directorio principal [21]. Este archivo contiene información de vital importancia para la ejecución y compilación de nuestra aplicación, entre otras cosas ese archivo contiene información sobre:

- El nombre de nuestra aplicación.
- Las Acivities, Servicios y Broadcast que la conforman.
- Los permisos de la aplicación.
- El mínimo nivel de API requerido para ejecutar nuestra aplicación.
- Y una lista de librerías externas utilizadas en nuestra aplicación.

## Android Query

Android-Query, también conocido como AQuery es una librería ligera utilizada para realizar tareas asíncronas como solicitud de imágenes, consultas json, entre otras tareas que deben realizarse de esta manera, asíncrona, ya que de otro modo puede causar el cierre inesperado de la aplicación que requiere ese contenido. Un ejemplo de implementación de la librería AQuery es el siguiente, donde se observa una solicitud de multimedia remota.

```
//fetch a remote resource in raw bitmap
String url = "http://www.vikispot.com/z/images/vikispot/android-w.png";
aq.ajax(url, Bitmap.class, new AjaxCallback<Bitmap>() {
    @Override
    public void callback(String url, Bitmap object, AjaxStatus status) {
    }
});
```

Realizar dicha solicitud requiere el uso de otra clase de la cual se requeriría amplios conocimientos, mientras que con la librería AQuery se simplifica y donde lo único que se requeriría sería el URL del objeto multimedia en cuestión.

También es utilizada para la manipulación de elementos de interfaz gráfica, logrando realizar tareas con menos líneas de código, el principal propósito de esta librería es hacer la programación en Android de una manera más simple, fácil y eficiente [22].

## Otto Event Bus

El funcionamiento de esta librería se basa en la publicación y suscripción de eventos; un EventBus es una librería que simplifica la comunicación entre diferentes partes de una aplicación, por ejemplo, envía algo desde una Actividad a un servicio, o facilita la interacción entre fragmentos de manera asíncrona, como por ejemplo seleccionar un ítem de una lista en un fragmento y que este despliegue las características de dicho ítem en otro fragmento.

## Material Dialogs

Es una librería para hacer uso del tema Material Design en diálogos emergentes a través de distintas versiones de Android, no sólo de la versión 5 (Lollipop) en adelante [23]. Lo que permite construir diálogos con temas de manera fácil y rápida. No solo permite la creación de diálogos con Material Design, sino que también diferentes implementaciones de contenido dentro de diálogos que de otra forma sería difícil de escribir. En la Figura 16 se pueden observar distintos tipos de diálogos implementados con esta librería.

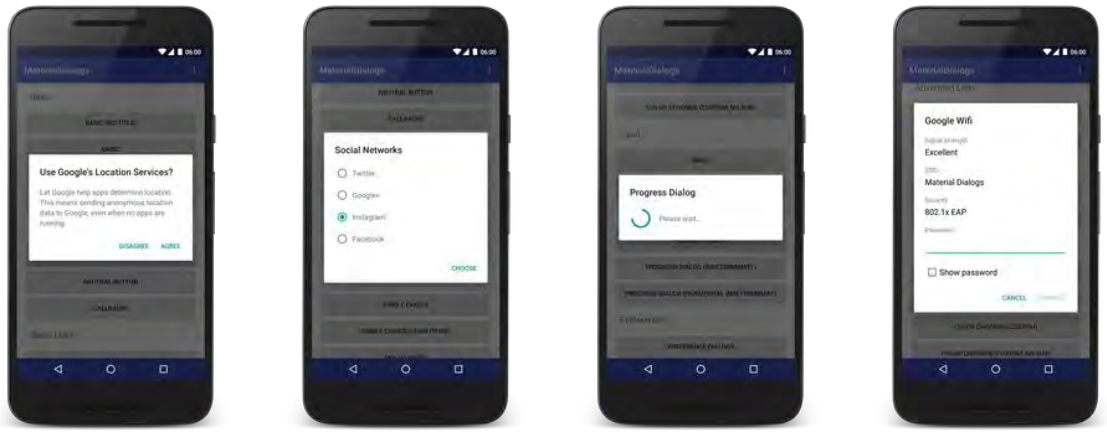



Figura 16 - Dialogos Basic, Single Choice, Progress y Custom View [24]

Los diálogos u objetos utilizados de dicha librería en este proyecto son los siguientes.

### List Dialogs

Crea un diálogo de lista donde lo único que se requiere es el arreglo de cadenas de texto que desplegará, la implementación de métodos de devolución de llamada (Callback) también se realiza de manera simple como se mostrará más adelante en este documento [24].

### Custom View

Permite la implementación del contenido del diálogo mediante el uso de un fragmento embebido en él, lo que permite, como su nombre lo dice, hace el uso de una vista completamente personalizada para el diálogo a desplegar.

### Progress Dialog Indeterminate

Permite la implementación de un diálogo de progreso en sus dos variantes, circular u horizontal, donde el diálogo se ejecutará mientras se realiza una tarea de carga o de consulta asíncrona, se puede utilizar en conjunto con otras librerías que permiten el uso de tareas asíncronas.

## MPAndroidChart

MPAndroidChart es una librería de fácil uso para la generación de gráficas en aplicaciones de Android, soporta múltiples tipos de gráficas (8 en total) entre las cuales podemos encontrar gráficas de barras, de líneas, de pastel entre otros, así como también soporta la interacción con las gráficas mediante el uso de gestos como el zoom con dos dedos que es muy conocido por estar implementado en múltiples aplicaciones o el desplazamiento por la gráfica [25]. En la Figura 17 se puede apreciar el logo de dicha librería.


Figura 17 - Logo MPAndroidChart [25]

Las gráficas generadas con esta librería soportan animaciones y selección de puntos, así como también interpolación y otras características que serán explicadas en el capítulo 3 de este documento; la librería funciona a partir de la versión 2.2 de Android (API 8) en adelante.

A continuación, se pueden ver unos ejemplos de gráficas generadas con esta librería.

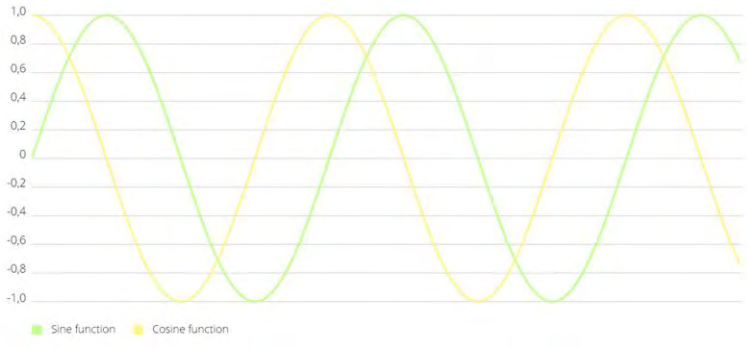



Figura 18 - Gráfica de funciones sinusoidales generadas con MPAndroidChart [25]

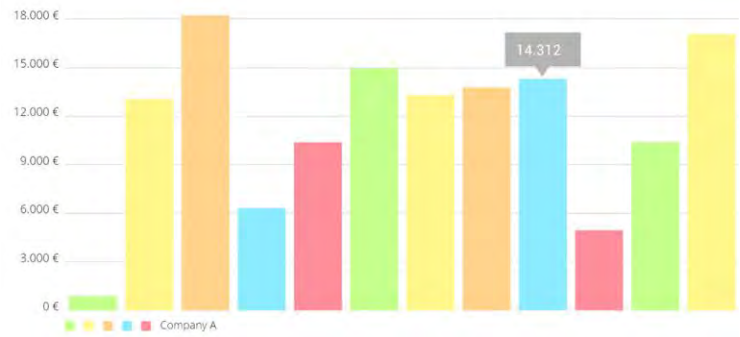



Figura 19 - Gráfica de barras generada con MPAndroidChart [25]

# CAPÍTULO 3. IMPLEMENTACIÓN

## Diagrama de clases

En el siguiente capítulo se realiza una descripción de las clases y funciones utilizadas en el desarrollo del proyecto, la clase BusProvider sirve para proporcionar un flujo de información de manera constante y rápida entre las clases que así lo requieran y la clase InformationEvent proporciona el objeto que pasa a través de la clase BusProvider para actualizar la información de manera asíncrona, el resto de las relaciones se puede apreciar en la siguiente Figura 20 y que se puede apreciar mejor en el ANEXO :

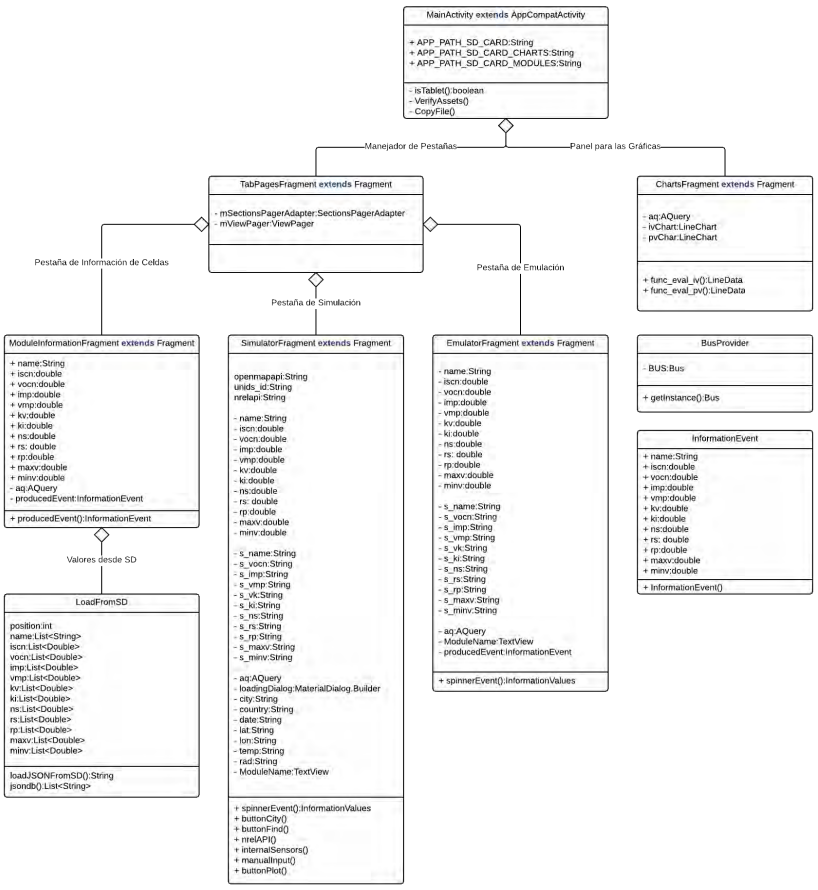



Figura 20 - Diagrama de clases

### *MainActivity*

Es la clase encargada de iniciar a la actividad principal que en este caso es sólo una, esta clase contiene métodos para la verificación de archivos necesarios en el funcionamiento del sistema, así como también métodos de restauración y de verificación de configuración de la pantalla para adaptar su contenido dependiendo si se trata de un Smartphone o una Tablet, también es la clase encargada de la barra de herramientas que está presente en todo momento y que despliega el menú de la aplicación.

#### Funciones y métodos

- **isTablet:** Esta función es utilizada para la verificación de la configuración de la pantalla, es decir, la densidad, el tamaño. La verificación de la orientación de la pantalla queda descartada ya que dependiendo de su resultado forzará a adoptar una configuración específica.

  Una vez adoptada la configuración de Smartphone o Tablet esta llamará al manejador de diseño (layout) correspondiente para sacar mayor provecho del tamaño de la pantalla. Si la función isTablet devuelve verdadero se llama a una configuración de doble panel, si es falsa, se llama a una configuración de un panel como se puede observar en la Figura 11 - Fragmentos en configuración dinámica  en el Capítulo 2.

- **VerifyAssets:** esta función realiza la verificación de la existencia de los archivos requeridos para el funcionamiento del sistema como lo son: el archivo de configuraciones de celdas fotovoltaicas, la carpeta donde se ubica el archivo y la carpeta donde se guardan las imágenes generadas por el sistema, todas estas ubicadas en la carpeta “PV Emulator” dentro de la memoria interna del dispositivo.

- **CopyFile:** este método es utilizado para la copia de archivos en caso que la función VerifyAssets lo requiera, se encarga de realizar la copia del archivo de celdas fotovoltaicas empaquetado en el APK de la aplicación.

### *BusProvider*

Esta clase está encargada de la comunicación entre los fragmentos de la aplicación, por sí sola no realiza alguna función más que la de proveer un bus entre clases, pero las clases que la ocupan hacen uso de ella en sus métodos de ciclo de vida, en los métodos onStart y onStop, y esto se repite las veces que sea necesario. A continuación, se hace una descripción de los métodos y su función relacionada a la clase onStart y onStop.

#### Métodos

- **onStart:** es el método que suscribe la clase al EventBus para que pueda escuchar o publicar eventos en el bus y que afecta a las clases donde se use dependiendo de su propio ciclo de vida o el de la actividad que la contiene.

- **onStop:** es el método utilizado para desuscribir la clase al EventBus, esto para que al momento que el fragmento se detenga y luego se restaure no cause conflictos con una suscripción previa al EventBus.

### *TabsPagesFragment*

Esta clase se encarga del comportamiento de las pestañas en la aplicación, en sí sólo contiene los métodos del adaptador utilizado para instanciar cada una de las clases de las tres pestañas.

Posee un objeto de tipo ViewPager que se encarga del comportamiento de pase de página para las pestañas y un objeto de tipo TabLayout que se encarga del comportamiento gráfico de la selección de pestañas en la aplicación.

### *ModuleInformationFragment*

Esta clase se encarga de manejar la información de las celdas fotovoltaicas extraídas del archivo modules.json, en dicha clase se manejan tres métodos además del propio onCreate que viene por defecto, los métodos onStart, onStop e InformationEvent. En esta clase también se instancia un objeto de la clase LoadFromSD, que a su vez permite localizar el archivo modules.json y acomodar la información en una cadena de texto para su procesamiento.

Contiene un objeto de tipo Spinner que es un `widget` utilizado para crear listas desplegables que muestran los nombres de las celdas fotovoltaicas como se puede observar en la Figura 21.

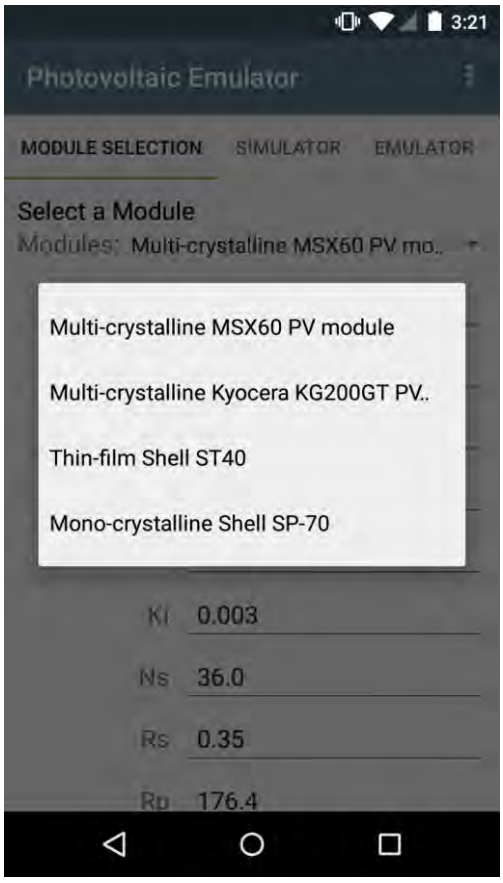



Figura 21 - Spinner de nombres de celdas fotovoltaicas

### Métodos

- **InformationEvent:** es utilizado en conjunto con la anotación @Produce y sirve para publicar eventos a fragmentos creados de manera dinámica, como, por ejemplo, a una pestaña que aún no ha sido creada ya que por el ciclo de vida dinámico que llevan los fragmentos dentro del actividad, estos afectan al bus, de manera que si se publica un evento en este fragmento y otro que aún no ha sido creado intenta acceder evento, al realizar la transición de fragmentos, el que publicó el evento ya no existe y no tiene evento al cual suscribirse.

## *LoadFromSD*

Es la clase encargada de la lectura del archivo `modules.json` donde se encuentran las especificaciones de las celdas fotovoltaicas que se requiere para el funcionamiento del sistema, esta clase consiste en dos funciones:

### Funciones

- **loadJSONFromSD:** se encarga de organizar todo el contenido del archivo `modules.json` en un string para luego pasar por un proceso de análisis que coloque toda la información en objetos de tipo `List`; esta función devuelve un objeto `String`.

- **jsondb:** esta función es utilizada para el proceso de análisis del `String` devuelto por la función loadJSONFromSD. Analiza todo el `String` que en realidad consiste en un arreglo de objetos json de los cuales obtiene diferentes listas de sus atributos como lo son el nombre, máximo voltaje, máxima corriente y otros, para luego regresar un objeto `List` donde se guardan los nombres para desplegarlos en el fragmento llamado ModuleInformationFragment, el cual también accede a las demás listas de los otros valores en esta clase.

### *SimulatorFragment*

Esta clase es quizás la que tiene más peso en cuanto a importancia en la aplicación, ya que es la encargada de realizar la simulación del comportamiento de las celdas fotovoltaicas en determinadas circunstancias y posee las funciones o métodos más complejos de toda la aplicación, cuanta con funciones como la obtención de información de clima a través de la API de OpenWeatherMap [26] y de obtención de datos de irradiancia a través de la API del National Renewable Energy Laboratory [27], además de obtención de datos mediante los sensores internos del smartphone y del procesado de cálculo para la generación de las gráficas en la simulación que se despliegan en otro fragmento. Los métodos y funciones utilizadas en esta clase son los siguientes:

#### Métodos y funciones

- **buttonCity:** este método es disparado cuando se pulsa el botón Find City en la pestaña Simulator, cuando es activado, este método toma la información escrita en el campo City y verifica que tenga algo escrito, de otra manera un diálogo emergente informará sobre la falta de información, en caso de que exista información, esta será consultada con la API de OpenWeatherMap haciendo uso de la librería AQuery, para la realización asíncrona de la tarea de buscar el contenido, ya que de no hacer esa tarea de manera asíncrona causaría el cierre inesperado de la aplicación.

  Para la realización de la consulta existen cuatro resultados posibles: el error 404 indica que no existe nombre exacto o parecido de la ciudad que se requiere, en este caso un diálogo emergente informará que verifiquemos el nombre escrito, el segundo caso, la consulta fue exitosa y se ubicará el nombre de la ciudad de manera oficial, el país al que pertenece, la longitud y latitud, además de la fecha y hora de la consulta, campos que se observan en la Figura 23. El tercer caso, muestra el error -101 que indica la falta de acceso a Internet para obtener los datos solicitados como se observa en la Figura 22.


Figura 22 - Error -101

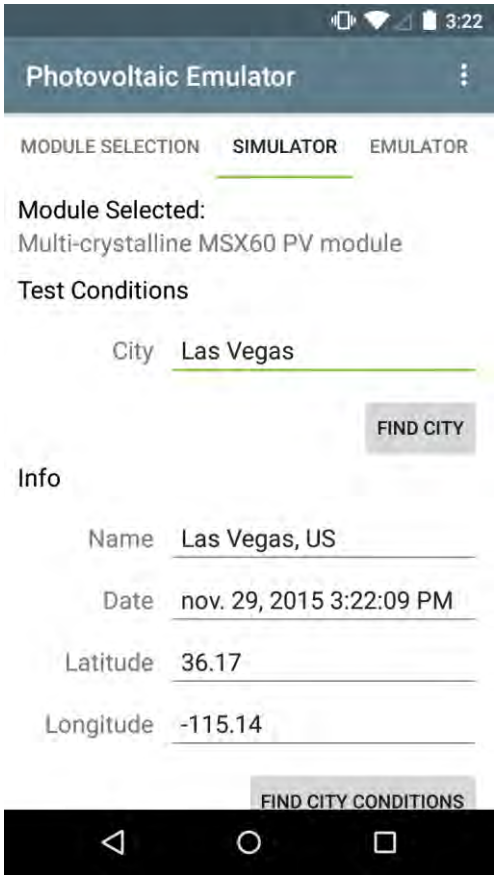



Figura 23 - Campos de nombre, fecha y ubicación

Para los demás casos, que son de error no específicos se cuenta con una simple notificación.

- **buttonFind:** este método hace uso de la librería MaterialDialogs para desplegar un diálogo emergente con opciones en la aplicación, el uso de esta librería permite una fácil personalización y una rápida implementación de un diálogo, lo único que se requiere para su implementación es un arreglo de cadenas de texto con los nombres de las opciones y un título a desplegar como se observa en la Figura 24.

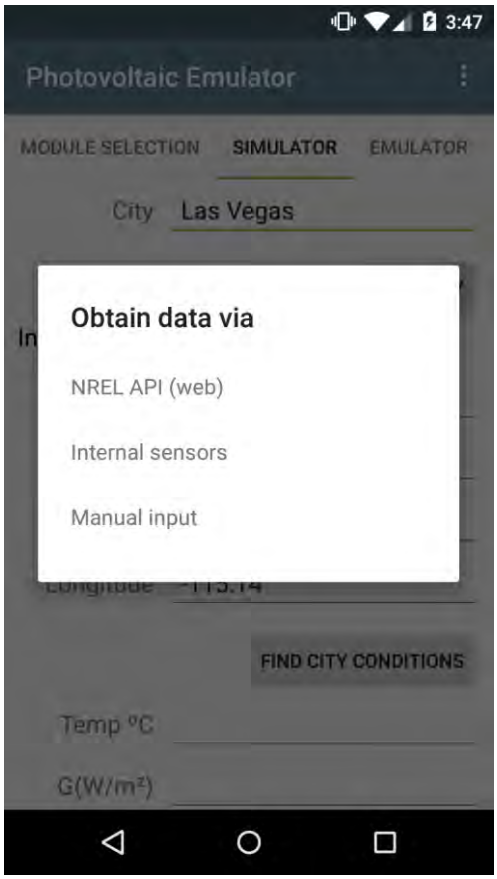



Figura 24 - Diálogo MaterialDialogs

- **nrelAPI:** este método hace uso de la API del National Renewable Energy Laboratory si y sólo si el campo country es igual a “US” dado que esta API sólo provee información de ciudades de Estados Unidos; esta opción es usada para obtener la información de irradiancia, en caso de que el campo country no sea igual a “US” el campo G(W/m^2) se queda en blanco, mientras que el campo Temp es actualizado a la temperatura obtenida del método FindCity y no puede proceder a menos de que haya información previa de una ciudad.

- **internalSensors:** este método utiliza acceso a información de los sensores del Smartphone para proveer datos como la temperatura ambiental o del

sensor de luz de propio dispositivo, en caso de encontrar información de dichos sensores, esta será reflejada en sus respectivos campos de temperatura e irradiancia, en caso contrario, mantendrá los campos en blanco y avisará mediante un diálogo emergente la ausencia de los sensores en el dispositivo.

- **manualInput:** este método activa una bandera para el estado de manualInput = true, para que, al momento de realizar los cálculos pertinentes, estos, sean obtenidos de los campos de temperatura e irradiancia, cabe mencionar que estos campos están configurados mediante el diseño de la interfaz en XML para que sólo acepten valores numéricos flotantes o enteros positivos y negativos, por lo que no es necesario hacer ninguna evaluación de expresiones regulares mediante código.

- **buttonPlot:** este método se encarga de realizar los cálculos para la representación gráfica de la simulación de acuerdo al modelo de dos diodos descrito previamente en el marco teórico de este documento, una vez realizados los cálculos para obtener el arreglo de datos, estos se mandar mediante una interfaz implementada de manera asíncrona entre la clase SimulatorFragment y la clase ChartsFragment.

### *ChartsFragment*

La clase ChartsFragment es la clase encargada de la representación gráfica de los valores del arreglo generado por el método buttonPlot de la clase SimulatorFragment, dicha clase cuenta con dos métodos, los cuales se encargan del acomodo de los valores en la gráfica. Esta clase en particular, es embebida dentro de la actividad principal en caso de que la función isTablet lo indique.

### Métodos

- **func_eval_iv:** este método se encarga de acomodar los valores del arreglo obtenido de SimulatorFragment en la curva corriente-voltaje y de refrescar la gráfica generada.

- **func_eval_pv:** este método se encarga de acomodar los valores del arreglo obtenido de SimulatorFragment en la curva potencia-voltaje y de refrescar la gráfica generada.

Como se puede observar en la Figura 25 las gráficas son generadas a partir de la información de la celda fotovoltaica de la clase ModuleInformationFragment y de los valores introducidos en los campos de temperatura e irradiancia de la clase SimulatorFragment para luego pasar los valores como argumento a la clase ChartsFragment.

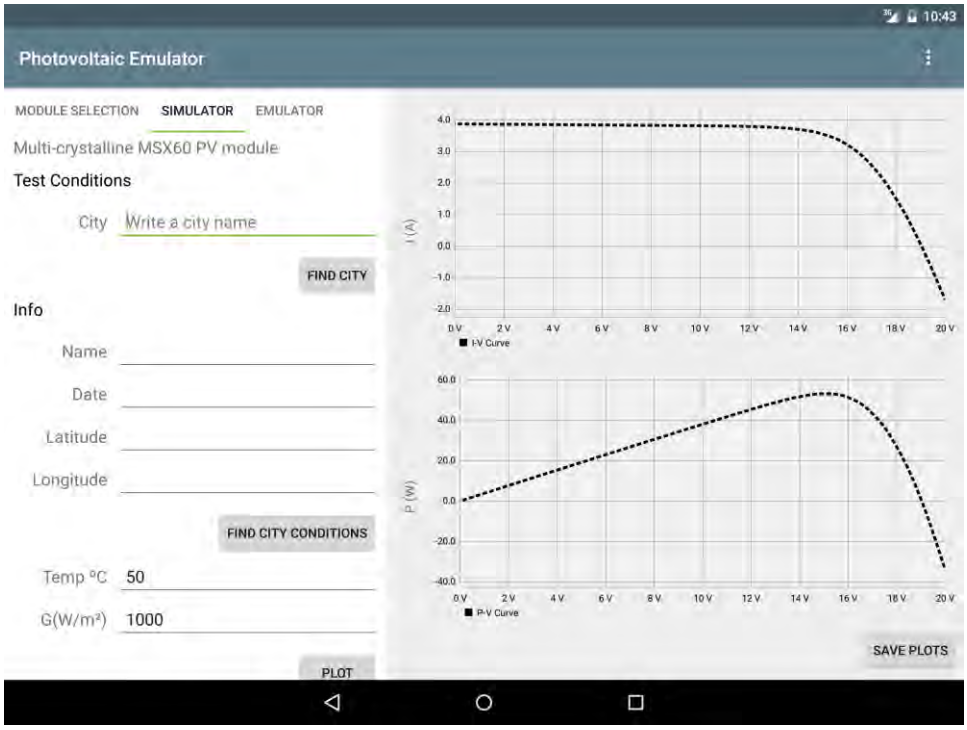



Figura 25 - Curvas IV y PV

# CAPÍTULO 4

# CAPÍTULO 4. RESULTADOS EXPERIMENTALES

Para los resultados experimentales, las pruebas consistieron en verificar el funcionamiento de los distintos módulos de la aplicación y en la comparación de la aplicación desarrollada previamente en el lenguaje Python.

Como primer experimento se llevó a cabo la verificación del funcionamiento de la aplicación clase por clase, primeramente, la clase ModuleInformationFragment con el apoyo de los registros del IDE AndroidStudio [28] que fue el utilizado para llevar a cabo el desarrollo de esta aplicación; podemos en la Figura 26 el despliegue de la clase ModuleInformationFragment la cual muestra un elemento seleccionado y en la captura de pantalla del IDE en la Figura 27 que nos muestra la información del registro que maneja de manera interna la aplicación.


Figura 26 - Pantalla de la aplicación en vista de Tablet

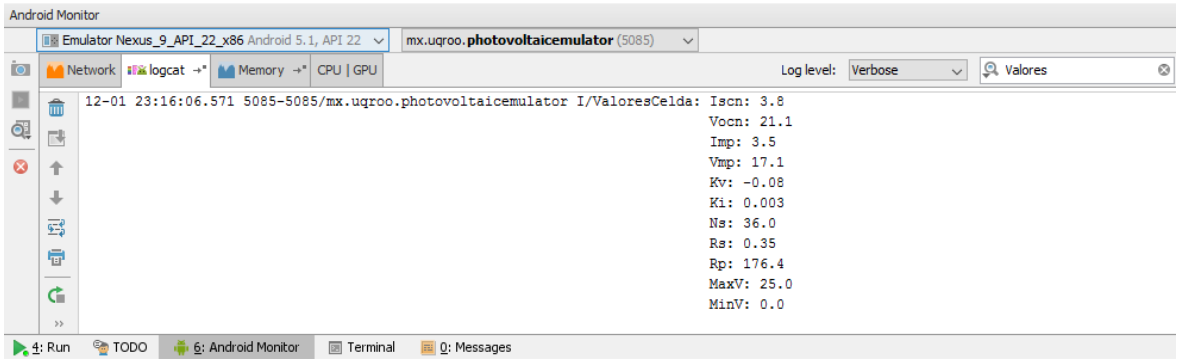



Figura 27 - Registro en el IDE

Se puede corroborar el traspaso de los valores a la clase SimulatorFragment mediante una notificación en la pestaña que la hospeda indicando que los valores han sido recibidos para su procesamiento como se observa en la Figura 28.

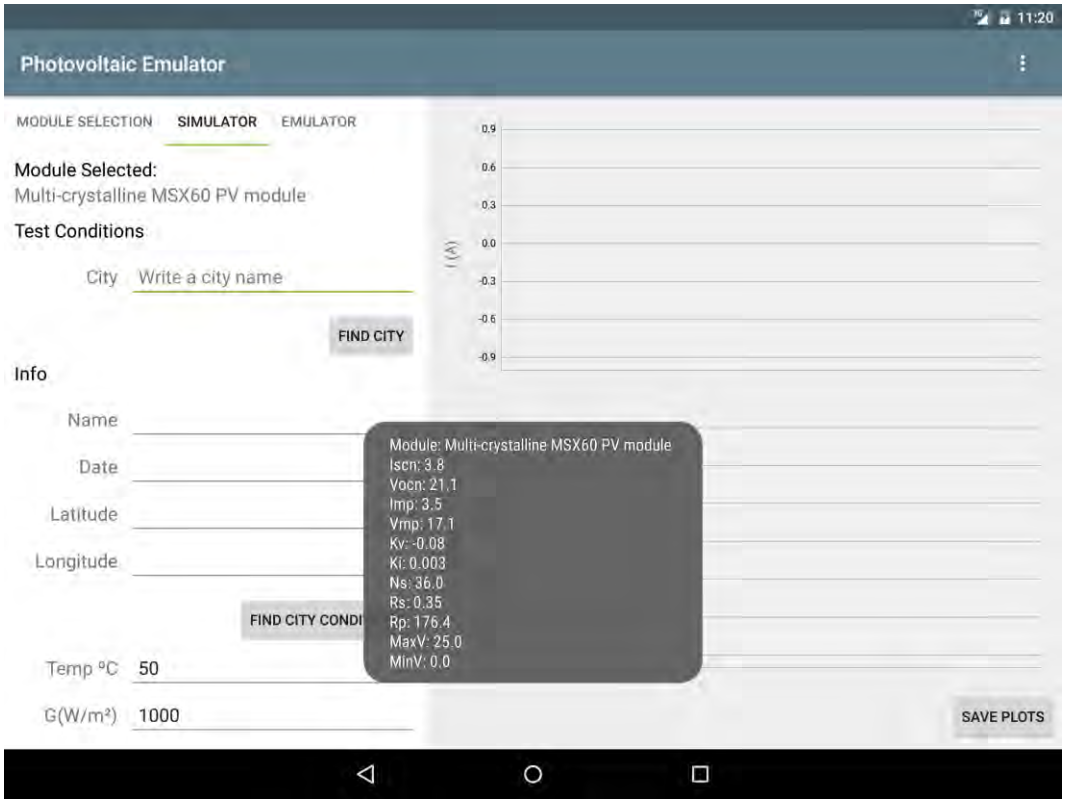



Figura 28 - Notificación Toast

Una vez recibidos los datos, podemos proceder a realizar la simulación ingresando manualmente los valores de temperatura e irradiancia, en caso de ser requeridos, estos datos pueden ser solicitados vía Internet o mediante el uso de sensores del dispositivo siempre y cuando lo permita, en la Figura 29, vemos el resultado del proceso de obtención de datos de localización de una ciudad.

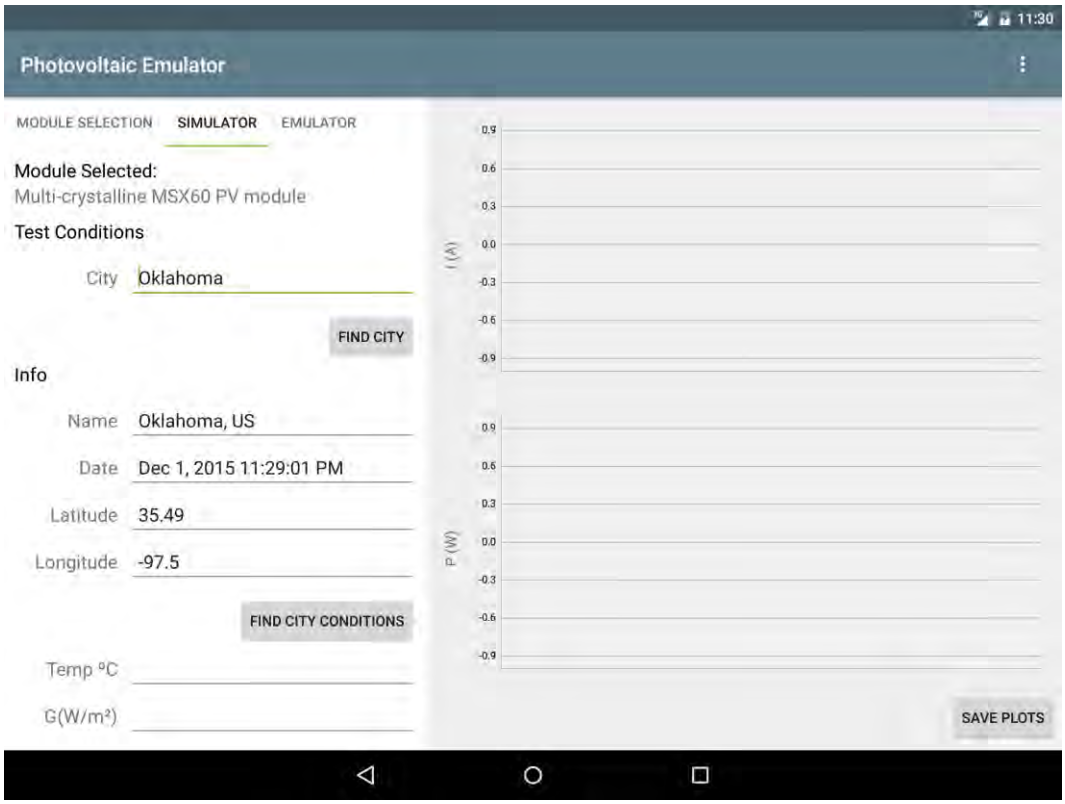



Figura 29 - Valores encontrados mediante API

Una vez obtenidos los datos de la localización de dicha ciudad, se procede a buscar sus condiciones climatológicas necesarias para la simulación del comportamiento de la celda fotovoltaicas.

El botón Find City Conditions nos provee de tres mecanismos para la obtención de datos, como se había explicado en el capítulo anterior y que se puede apreciar en la Figura 30 que se muestra continuación.

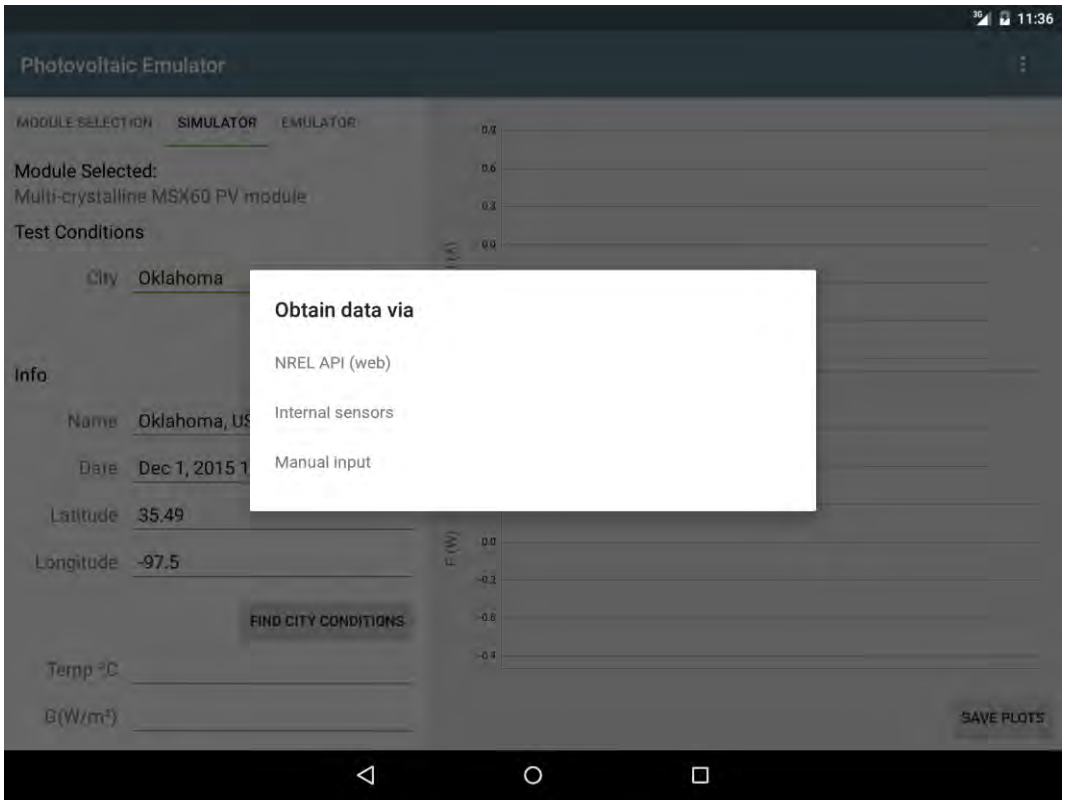



Figura 30 - Opciones para obtención de datos

En la Figura 31 se pueden observar los datos ya obtenidos mediante la API del NREL [27] de Estados Unidos.

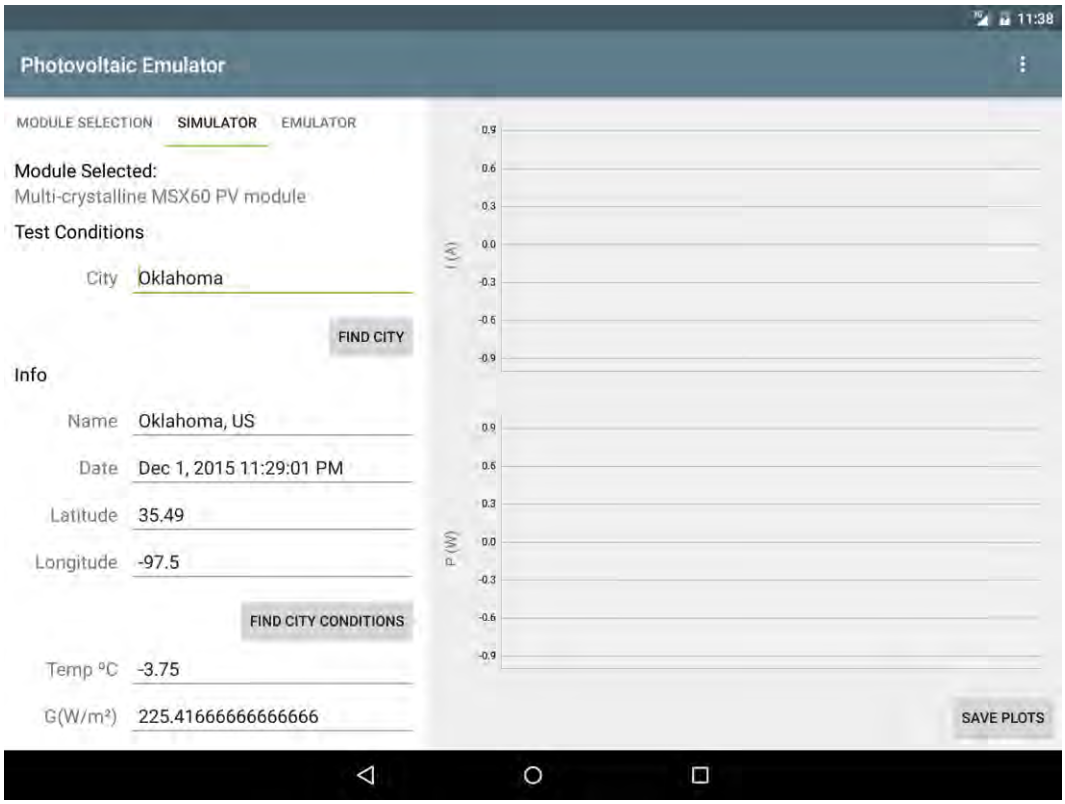



Figura 31 - Condiciones encontradas

El botón Plot no se observa en la Figura 31 porque no cabe dentro de la pantalla pero es accesible mediante un movimiento de desplazamiento con un dedo, de nuevo, gracias al registro de AndroidStudio que podemos observar en Figura 33 una vista preliminar de la información calculada de acuerdo a las ecuaciones descritas en el capítulo dos de este documento.


Figura 32 - Gráficas generadas por la aplicación

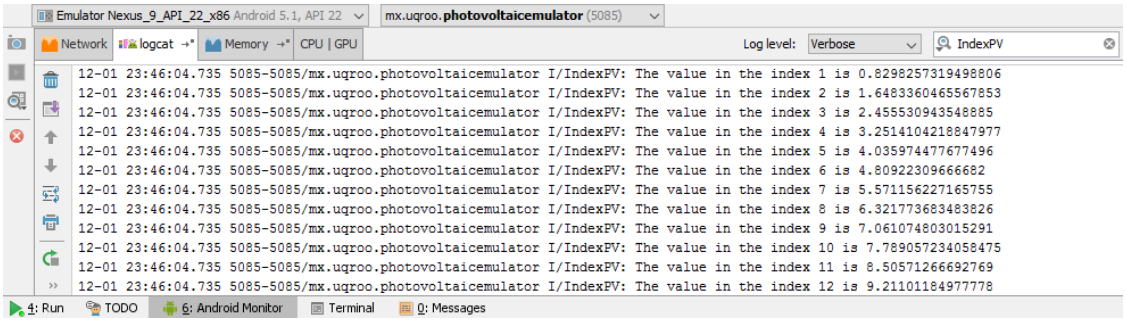



Figura 33 - Registro con valores de la gráfica

Como se había mencionado anteriormente, la clase ChartsFragment no realiza operación alguna más que ubicar los valores ya calculados en las gráficas además de permitir guardar estos en la memoria interna del dispositivo.

Dicho lo anterior, ahora se procede a comprar los resultados obtenido mediante la aplicación escrita originalmente en Python y que es ejecutaba en sistemas embebidos, contra la aplicación desarrollada en Android para dispositivos móviles.

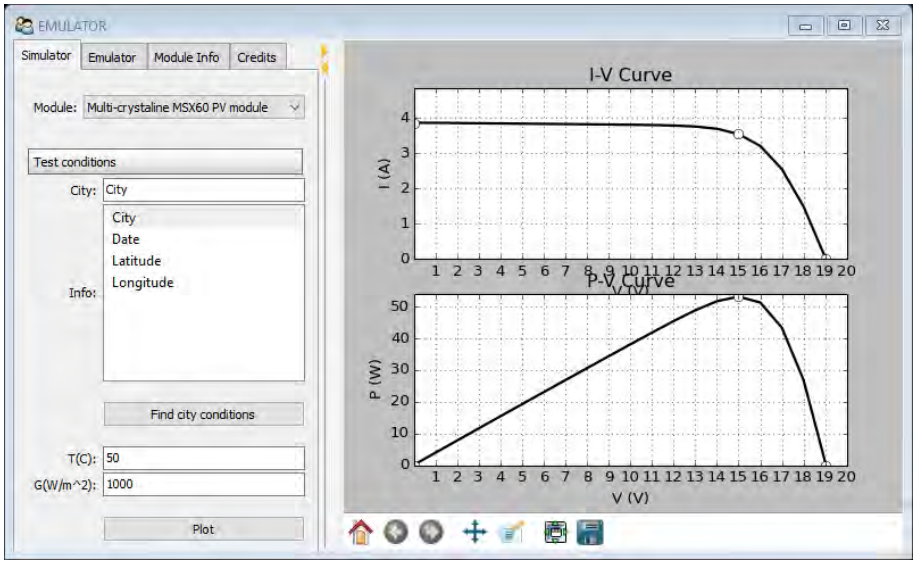



Figura 34 - Emulador de celdas fotovoltaicas en Python


Figura 35 - Simulador de celdas fotovoltaicas en Java

Podemos observar en la Figura 34 y Figura 35 al emulador para celdas fotovoltaicas y al simulador respectivamente, el simulador, al ser ejecutado sobre un dispositivo móvil corriendo el sistema Android, tiene la ventaja del acceso a ciertas tecnologías que el emulador no tiene, entre sus ventajas se destaca la interfaz con la cual se interactúa mediante la tecnología TouchScreen que lleva años en el mercado mientras que el emulador requiere de un dispositivo señalador (mouse) y un teclado para poder interactuar con él.

Otra de las ventajas que tiene el simulador para Android es que dispone de sensores que permiten obtener datos del ambiente como la intensidad de la luz o la temperatura en ciertos dispositivos. Además de contar con un conjunto de tecnologías de red como lo es el WiFi y las redes de banda ancha móvil.

En cuanto a la experimentación, se optó por evaluar dos celdas fotovoltaicas e diferentes temperaturas. Las celdas evaluadas con sus parámetros se presentan en la siguiente tabla.

Tabla 4 - Celdas fotovoltaicas

| Parámetro | Multi-crystalline solar MSX-60 | Thin-film Shell ST40 |
|---|---|---|
| **Isc** | 3.8 A | 2.86 A |
| **Voc** | 21.1 V | 23.3 V |
| **Imp** | 3.5 V | 2.41 A |
| **Vmp** | 17.1 V | 16.6 V |
| **Kv** | - 80 mV/ºC | - 100 mV/ºC |
| **Ki** | 3 mA/ºC | 0.35 mA/ºC |
| **Ns** | 36 | 36 |

En cuanto a la precisión o fidelidad de los cálculos se puede observar en la siguiente gráfica que los cálculos se sobreponen, mostrando a primera vista que los resultados son muy semejantes.

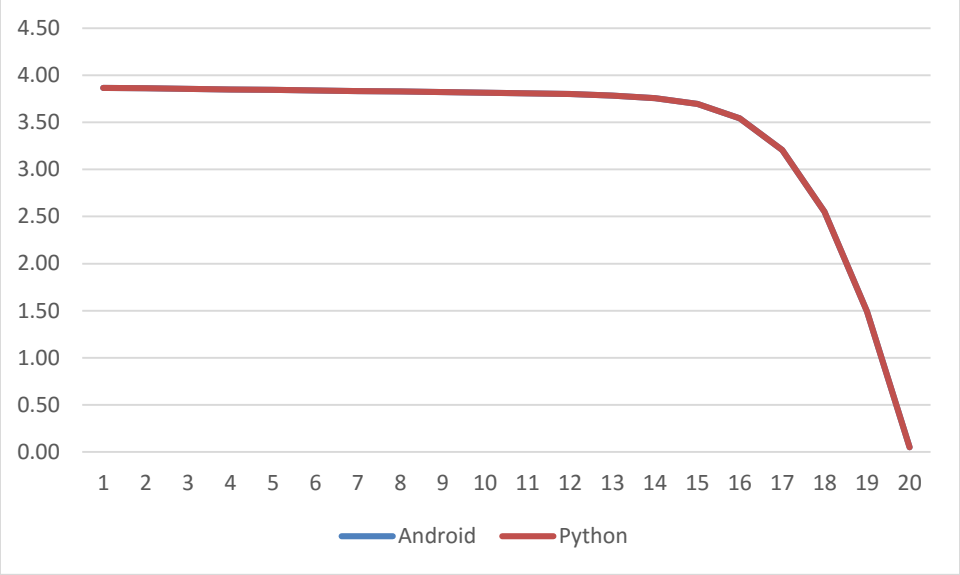



Figura 36 - Comportamiento I-V @ 50ºC 1000 W/m^2 con Multi-crystalline solar MSX-60

Esto se debe a que su diferencia es una menor a una millonésima parte como se puede observar en la Tabla tal

Tabla 5 - Comportamiento I-V @ 50ºC 1000 W/m^2 con Multi-crystalline solar MSX-60

| Índice | Android | Python | Diferencia |
|---|---|---|---|
| **0** | 3.867326631136630 | 3.867326631140000 | 0.000000000003370 |
| **1** | 3.861668705319750 | 3.861668705320000 | 0.000000000000250 |
| **2** | 3.856010442805860 | 3.856010442810000 | 0.000000000004140 |
| **3** | 3.850351312899650 | 3.850351312900000 | 0.000000000000350 |
| **4** | 3.844689936017300 | 3.844689936020000 | 0.000000000002700 |
| **5** | 3.839022708466170 | 3.839022708470000 | 0.000000000003830 |
| **6** | 3.833340175489470 | 3.833340175490000 | 0.000000000000530 |
| **7** | 3.827617433718460 | 3.827617433720000 | 0.000000000001540 |
| **8** | 3.821788662292390 | 3.821788662290000 | -0.000000000002390 |
| **9** | 3.815679388848800 | 3.815679388850000 | 0.000000000001200 |
| **10** | 3.808826159387040 | 3.808826159390000 | 0.000000000002960 |
| **11** | 3.799997069610570 | 3.799997069610000 | -0.000000000000570 |
| **12** | 3.785926017019360 | 3.785926017020000 | 0.000000000000640 |
| **13** | 3.758044919540920 | 3.758044919540000 | -0.000000000000920 |
| **14** | 3.694559426536710 | 3.694559426540000 | 0.000000000003290 |
| **15** | 3.544295547219970 | 3.544295547220000 | 0.000000000000030 |

| | | | |
|---|---|---|---|
| **16** | 3.207568361551870 | 3.207568361550000 | -0.000000000001870 |
| **17** | 2.551650437912790 | 2.551650437910000 | -0.000000000002790 |
| **18** | 1.492196259589240 | 1.492196259590000 | 0.000000000000760 |
| **19** | 0.048761697466977 | 0.048761697467000 | 0.000000000000023 |

Estos valores fueron calculados considerando una temperatura de 50ºC y 1000 W/m^2 con la celda Multi-crystalline solar MSX-60.

A continuación, la representación gráfica de la misma celda fotovoltaica pero ahora evaluada a 75ºC y 1000 W/m^2.

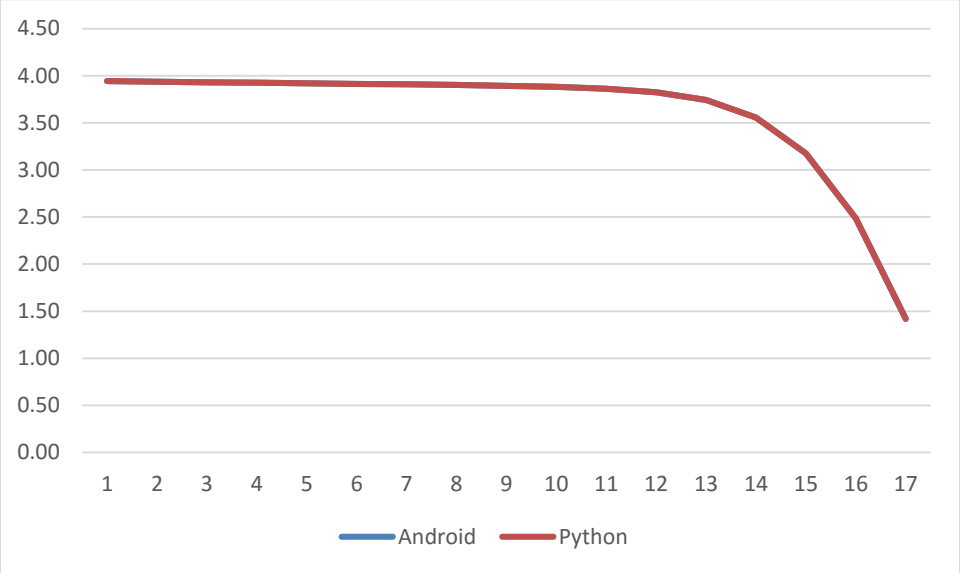



Figura 37 - Comportamiento I-V @ 75ºC 1000 W/m^2 con Multi-crystalline solar MSX-60

Tabla 6 - Comportamiento I-V @ 75ºC 1000 W/m^2 con Multi-crystalline solar MSX-60

| Índice | Android | Python | Diferencia |
|---|---|---|---|
| **0** | 3.942175864477830 | 3.942175864480000 | 0.000000000002170 |
| **1** | 3.936513533103950 | 3.936513533100000 | -0.000000000003950 |
| **2** | 3.930844811902370 | 3.930844811900000 | -0.000000000002370 |
| **3** | 3.925160727588690 | 3.925160727590000 | 0.000000000001310 |
| **4** | 3.919439526620980 | 3.919439526620000 | -0.000000000000980 |
| **5** | 3.913628251322050 | 3.913628251320000 | -0.000000000002050 |
| **6** | 3.907597497864250 | 3.907597497860000 | -0.000000000004250 |
| **7** | 3.901030081750030 | 3.901030081750000 | -0.000000000000030 |
| **8** | 3.893146930596820 | 3.893146930600000 | 0.000000000003180 |
| **9** | 3.882034361599020 | 3.882034361600000 | 0.000000000000980 |
| **10** | 3.863012267994520 | 3.863012267990000 | -0.000000000004520 |
| **11** | 3.824802551399330 | 3.824802551400000 | 0.000000000000670 |
| **12** | 3.741263643041710 | 3.741263643040000 | -0.000000000001710 |
| **13** | 3.557145703198500 | 3.557145703200000 | 0.000000000001500 |
| **14** | 3.176652952462040 | 3.176652952460000 | -0.000000000002040 |
| **15** | 2.485934442355830 | 2.485934442360000 | 0.000000000004170 |
| **16** | 1.419528289770820 | 1.419528289770000 | -0.000000000000820 |

Evaluación de la celda fotovoltaica Thin-film Shell ST40 a 50ºC y 1000W/m^2

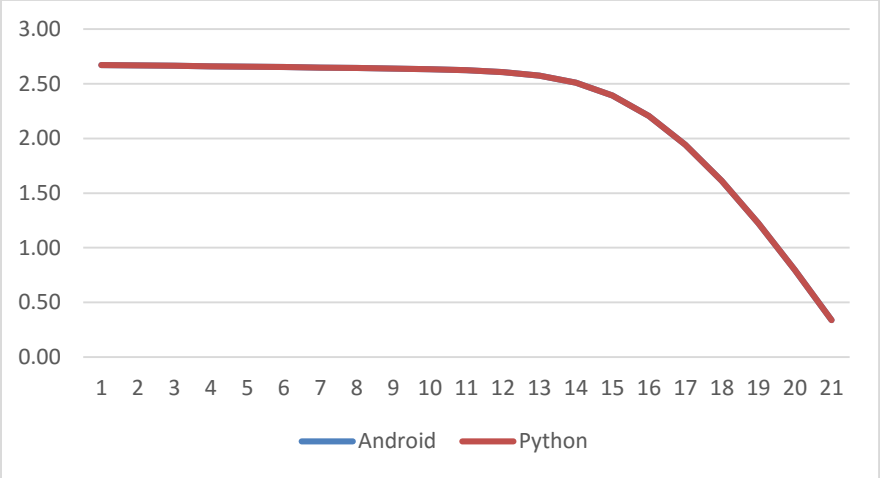



Figura 38 - Comportamiento I-V @ 50ºC 1000 W/m^2 con Thin-film Shell ST40

Tabla 7 - Comportamiento I-V @ 50ºC 1000 W/m^2 con Thin-film Shell ST40

| Índice | Android | Python | Diferencia |
|---|---|---|---|
| **0** | 2.672506992760850 | 2.672506992760000 | -0.000000000000850 |
| **1** | 2.668728270203860 | 2.668728270200000 | -0.000000000003860 |
| **2** | 2.664944913138350 | 2.664944913140000 | 0.000000000001650 |
| **3** | 2.661151063711090 | 2.661151063710000 | -0.000000000001090 |
| **4** | 2.657333380045710 | 2.657333380050000 | 0.000000000004290 |
| **5** | 2.653461392249800 | 2.653461392250000 | 0.000000000000200 |
| **6** | 2.649465362807030 | 2.649465362810000 | 0.000000000002970 |
| **7** | 2.645185476310650 | 2.645185476310000 | -0.000000000000650 |
| **8** | 2.640255635316680 | 2.640255635320000 | 0.000000000003320 |
| **9** | 2.633840525827950 | 2.633840525830000 | 0.000000000002050 |
| **10** | 2.624056453577900 | 2.624056453580000 | 0.000000000002100 |
| **11** | 2.606776106403540 | 2.606776106400000 | -0.000000000003540 |
| **12** | 2.573525207110070 | 2.573525207110000 | -0.000000000000070 |
| **13** | 2.509114913230430 | 2.509114913230000 | -0.000000000000430 |
| **14** | 2.392519148020200 | 2.392519148020000 | -0.000000000000200 |
| **15** | 2.205322165715360 | 2.205322165720000 | 0.000000000004640 |
| **16** | 1.942145509290960 | 1.942145509290000 | -0.000000000000960 |
| **17** | 1.610942739279230 | 1.610942739280000 | 0.000000000000770 |
| **18** | 1.225199468368850 | 1.225199468370000 | 0.000000000001150 |
| **19** | 0.797761456826991 | 0.797761456827000 | 0.000000000000009 |
| **20** | 0.338756617158379 | 0.338756617158000 | -0.000000000000379 |

La misma celda, pero ahora a 75ºC y 1000 W/m^2

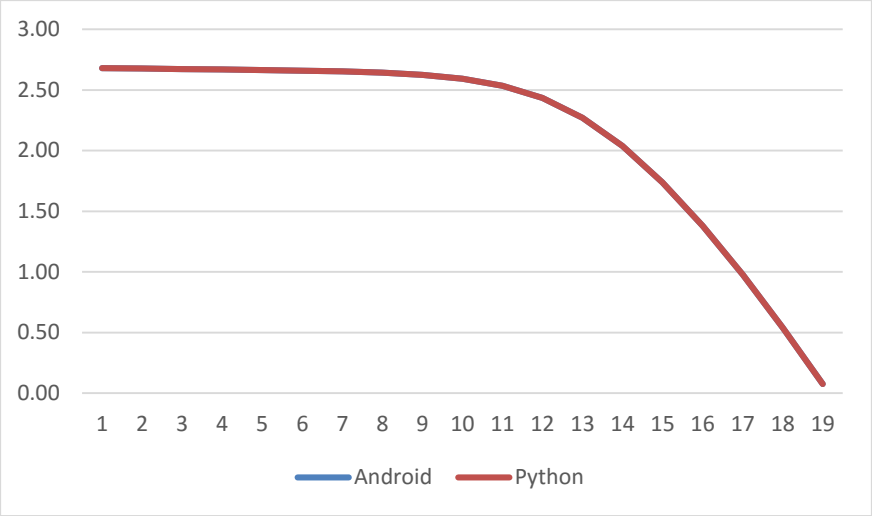



Figura 39 - Comportamiento I-V @ 75ºC 1000 W/m^2 con Thin-film Shell ST40

Tabla 8 - Comportamiento I-V @ 75ºC 1000 W/m^2 con Thin-film Shell ST40

| Índice | Android | Python | Diferencia |
|---|---|---|---|
| **0** | 2.681147385063840 | 2.681147385060000 | -0.000000000003840 |
| **1** | 2.677303669616710 | 2.677303669620000 | 0.000000000003290 |
| **2** | 2.673383172012100 | 2.673383172010000 | -0.000000000002100 |
| **3** | 2.669299105335590 | 2.669299105340000 | 0.000000000004410 |
| **4** | 2.664865842403860 | 2.664865842400000 | -0.000000000003860 |
| **5** | 2.659686453491040 | 2.659686453490000 | -0.000000000001040 |
| **6** | 2.652915372510500 | 2.652915372510000 | -0.000000000000500 |
| **7** | 2.642771016825020 | 2.642771016830000 | 0.000000000004980 |
| **8** | 2.625595409759960 | 2.625595409760000 | 0.000000000000040 |
| **9** | 2.594283718192060 | 2.594283718190000 | -0.000000000002060 |
| **10** | 2.536489077151220 | 2.536489077150000 | -0.000000000001220 |
| **11** | 2.434739326722050 | 2.434739326720000 | -0.000000000002050 |
| **12** | 2.271664170935570 | 2.271664170940000 | 0.000000000004430 |
| **13** | 2.038452483928010 | 2.038452483930000 | 0.000000000001990 |
| **14** | 1.738050140759870 | 1.738050140760000 | 0.000000000000130 |
| **15** | 1.380689588275650 | 1.380689588280000 | 0.000000000004350 |
| **16** | 0.978087000397031 | 0.978087000397000 | -0.000000000000031 |
| **17** | 0.540411110570149 | 0.540411110570000 | -0.000000000000149 |
| **18** | 0.075578934869633 | 0.075578934869600 | -0.000000000000033 |

Cabe mencionar que los resultados nos arrojan a que el poder de procesamiento y precisión brindados por dispositivos móviles con sistema Android son igual o más potentes que el dispositivo utilizado en el desarrollo original.

Con la representación de las gráficas y los valores de las tablas, podemos darnos cuenta que la diferencia de los valores es prácticamente nula y tratándose de voltaje, una parte menor a una millonésima de unidad no es significativa.

# CAPÍTULO 5

# CAPÍTULO 5. CONCLUSIONES

El desarrollo de esta aplicación trajo consigo una serie de retos que, en ciertas ocasiones, me orillaron a buscar diversas alternativas como la librería del manejo de gráficas que fue cambiada varias veces durante su desarrollo ya que se iban encontrando librerías con mejores características que le daban un plus a la aplicación como la que se quedó al final, que permite interactuar con las gráficas generadas por la simulación de una manera más natural mediante gestos que todos conocemos y usamos en nuestros dispositivos móviles. Estos movimiento o retos encontrados me llevaron a buscar alternativas para ir mejorando y rediseñando el funcionamiento de la aplicación.

Uno de los contratiempos encontrados fue que los conocimientos que poseía estaban desfasados a como lo dictan las líneas de diseño y desarrollo de Android, ya que, al ser un sistema operativo en constante y rápida evolución, se crean nuevas y mejores librerías que debemos aprender a implementar para hacer nuestras aplicaciones más compatibles con un mayor número de dispositivos. Además de la costumbre de utilizar Java durante mucho tiempo, muchas librerías que funcionan de manera estándar en la programación de software de escritorio no llevaban a cabo la misma función en Android, como, por ejemplo, para la solicitud de una cadena json en Java para escritorio toma una sola línea de código, mientras que para lograr la mista solicitud en Android eran necesarias bastantes más, este fue un hecho que me orillo a buscar soluciones en librerías desarrolladas por terceros y que cumplen de muy buena forma su función, haciendo que la programación resulta de una manera más ágil y clara.

Ahora, el proceso de traducción de código escrito en Python a Java conlleva un esfuerzo significativo ya que se requiere tener conocimiento de ambos lenguajes de programación, así como de estructuras de datos y programación orientada a objetos y que al final, la traducción o implementación de dicho código resulta más tediosa dadas sus raíces, como el manejo implícito de ciertas funciones en Java mientras que en Python todo resulta ser más dinámico.

En la aplicación desarrollada, puedo decir que se cumplió el objetivo principal que era programar una interfaz móvil para la simulación de celdas fotovoltaicas, eso como mencioné, lleva mucho esfuerzo y estoy seguro que este tipo de proyectos de software serán de relevancia para el campo de la generación de energías alternativas, y no sólo para ese campo, sino para aquellos compañeros que quieran incursionar en la programación para dispositivos móviles puedan consultar la teoría y código utilizados en el desarrollo de este proyecto.

Una meta no declarada pero que hubiera sido de mi agrado llevar a cabo, es portar el código a otras plataformas móviles como lo son Windows Phone y iOS mediante el uso de Xamarin, la implentación de C# para Android y demás sistemas.

De igual manera, quiero mencionar la importancia de las clases cursadas durante estos años de carrera, sin ellas me hubiera resultado difícil lograr este proyecto y que sientan las bases para la incursión en el desarrollo de aplicaciones y la simulación.

# REFERENCIAS BIBLIOGRÁFICAS

[1] A. Castillo Atoche, J. Vázquez Castillo, J. Ortegón-Aguilar, R. Carrasco-Alvarez, J. Sandoval Gío y A. Colli-Menchi, «A high-accuracy photovoltaic emulator system using ARM processors,» *Solar Energy,* vol. 120, pp. 389-398, 2015.

[2] Z. Kashif Ishaque, «Simple, fast and accurate two-diode model for photovoltaic modules,» *Solar Energy Materials and Solar Cells,* vol. 95, nº 2, p. 586–594, 2011.

[3] Oracle Corporation, «Conozca más sobre la tecnología Java,» [En línea]. Available: http://java.com/es/about/. [Último acceso: 18 Noviembre 2015].

[4] Python Software Fundation, «General Python FAQ,» [En línea]. Available: https://docs.python.org/2/faq/general.html#what-is-python. [Último acceso: 17 Noviembre 2015].

[5] Google Inc., «Introduction to Android | Android Developers,» [En línea]. Available: http://developer.android.com/guide/index.html. [Último acceso: 19 Noviembre 2015].

[6] A. Rubin, «Official Google Blog: Celebrating Android,» 23 Junio 2010. [En línea]. Available: https://googleblog.blogspot.mx/2010/06/celebrating-android.html. [Último acceso: 23 Noviembre 2015].

[7] AndroidLib.com, «Android Market statistics from AndroLib, Androlib, Android Applications and Games directory,» [En línea]. Available: http://www.androlib.com/appstats.aspx. [Último acceso: 19 Noviembre 2015].

[8] Google Inc., «<uses-sdk> | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/manifest/uses-sdk-element.html#ApiLevels. [Último acceso: 20 Noviembre 2015].

[9] Vogella GmbH, «Introduction to Android development with Android Studio - Tutorial,» [En línea]. Available: http://www.vogella.com/tutorials/Android/article.html#android_os_layer. [Último acceso: 20 Noviembre 2015].

[10] Google Inc., «Activities | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/components/activities.html. [Último acceso: 23 Noviembre 2015].

[11] Google Inc., «Fragments | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/components/fragments.html. [Último acceso: 23 Noviembre 2015].

[12] Vogella GmbH, «Android application and activity life cycle - Tutorial,» [En línea]. Available: http://www.vogella.com/tutorials/AndroidLifeCycle/article.html#activity_lifecycle. [Último acceso: 30 Noviembre 2015].

[13] Google Inc., «Support Library | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/tools/support-library/index.html. [Último acceso: 25 Noviembre 2015].

[14] Google Inc., «User Interface | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/ui/index.html. [Último acceso: 25 Noviembre 2015].

[15] Google Inc., «UI Overview | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/ui/overview.html. [Último acceso: 25 Noviembre 2015].

[16] Google Inc., «Layouts | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/ui/declaring-layout.html. [Último acceso: 13 Noviembre 2015].

[17] Google Inc., «Input Controls | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/ui/controls.html. [Último acceso: 30 Noviembre 2015].

[18] Google Inc., «Buttons | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/ui/controls/button.html. [Último acceso: 30 Noviembre 2015].

[19] Google Inc., «Text Fields | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/ui/controls/text.html. [Último acceso: 30 Noviembre 2015].

[20] Google Inc., «Spinners | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/ui/controls/spinner.html. [Último acceso: 30 Noviembre 2015].

[21] Google Inc., «App Manifest | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/guide/topics/manifest/manifest-intro.html. [Último acceso: 30 Noviembre 2015].

[22] P. Liu, « android-query - Android Framework - Simpler Coding for Android - Google Project Hosting,» [En línea]. Available: https://code.google.com/p/android-query/. [Último acceso: 27 Noviembre 2015].

[23] A. M. Follestad, «Aidan — Projects,» [En línea]. Available: http://aidanfollestad.com/projects. [Último acceso: 27 Noviembre 2015].

[24] A. M. Follestad, «afollestad/material-dialogs,» [En línea]. Available: https://github.com/afollestad/material-dialogs. [Último acceso: 27 Noviembre 2015].

[25] P. Jahoda, «PhilJay/MPAndroidChart,» [En línea]. Available: https://github.com/PhilJay/MPAndroidChart. [Último acceso: 27 Noviembre 2015].

[26] OWM Inc., «Openweathermap.org - current,» [En línea]. Available: http://openweathermap.org/current. [Último acceso: 1 Diciembre 2015].

[27] National Renewable Energy Laboratory, «Solar Resource Data API | NREL: Developer Network,» [En línea]. Available: http://developer.nrel.gov/docs/solar/solar-resource-v1/. [Último acceso: 2015 Diciembre 1].

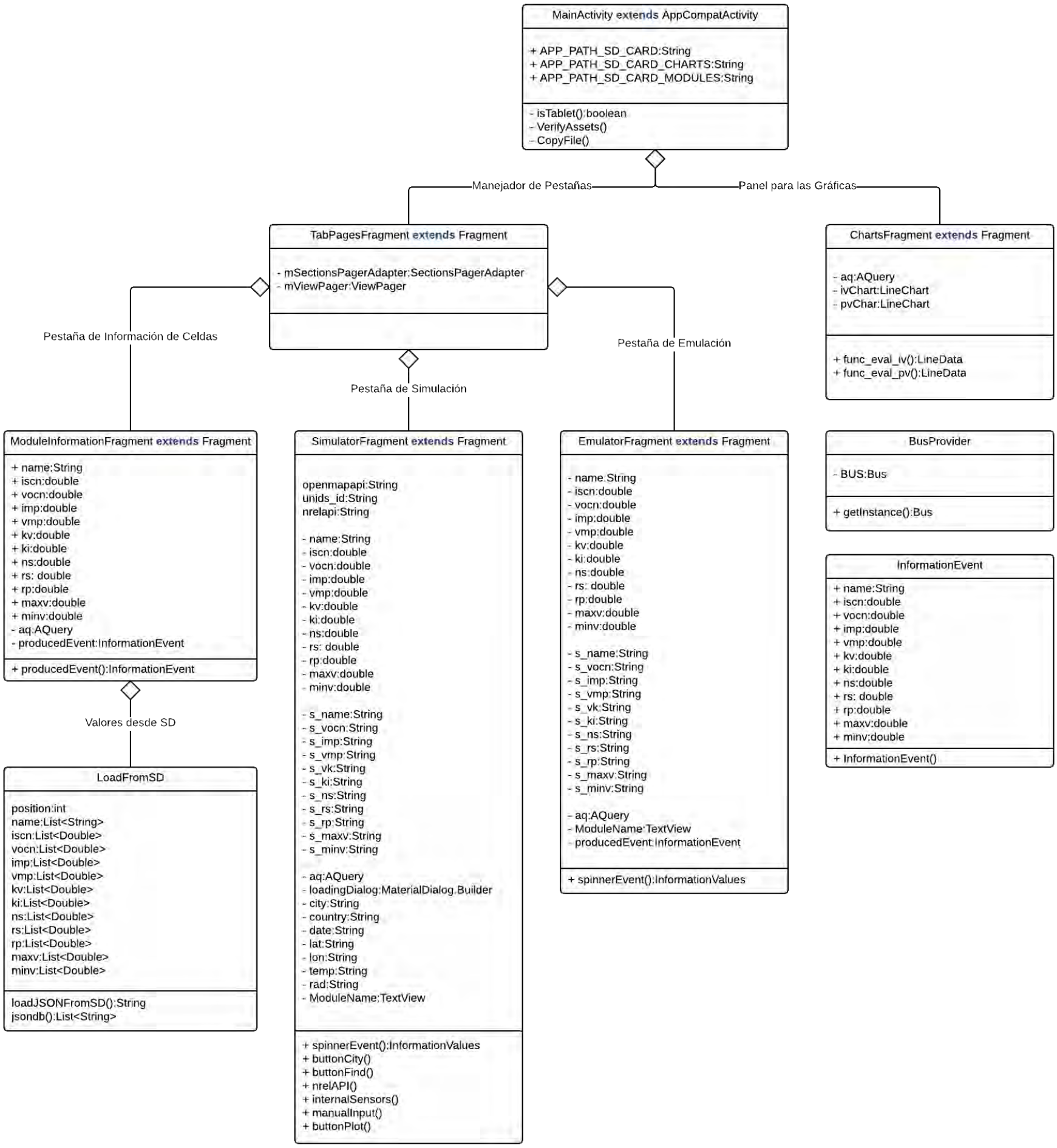
[28] Google Inc., «Download Android Studio and SDK Tools | Android Developers,» [En línea]. Available: http://developer.android.com/intl/es/sdk/index.html. [Último acceso: 2 Diciembre 2015].

# ANEXOS

## *Índice de ilustraciones*

## *Índice de tablas*

# ANEXO A


MainActivity extends AppCompatActivity
+ APP_PATH_SD_CARD:String
+ APP_PATH_SD_CARD_CHARTS:String
+ APP_PATH_SD_CARD_MODULES:String
- isTablet():boolean
- VerifyAssets()
- CopyFile()
Manejador de Pestañas
Panel para las Gráficas
TabPagesFragment extends Fragment
- mSectionsPagerAdapter:SectionsPagerAdapter
- mViewPager:ViewPager
ChartsFragment extends Fragment
- aq:AQuery
- ivChart:LineChart
- pvChar:LineChart
+ func_eval_iv():LineData
+ func_eval_pv():LineData
Pestaña de Información de Celdas
Pestaña de Simulación
Pestaña de Emulación
ModuleInformationFragment extends Fragment
+ name:String
+ iscn:double
+ vocn:double
+ imp:double
+ vmp:double
+ kv:double
+ ki:double
+ ns:double
+ rs: double
+ rp:double
+ maxv:double
+ minv:double
- aq:AQuery
- producedEvent:InformationEvent
+ producedEvent():InformationEvent
Valores desde SD
LoadFromSD
position:int
name:List<String>
iscn:List<Double>
vocn:List<Double>
imp:List<Double>
vmp:List<Double>
kv:List<Double>
ki:List<Double>
ns:List<Double>
rs:List<Double>
rp:List<Double>
maxv:List<Double>
minv:List<Double>
loadJSONFromSD():String
jsondb():List<String>
SimulatorFragment extends Fragment
openmapapi:String
unids_id:String
nrelapi:String
- name:String
- iscn:double
- vocn:double
- imp:double
- vmp:double
- kv:double
- ki:double
- ns:double
- rs: double
- rp:double
- maxv:double
- minv:double
- s_name:String
- s_vocn:String
- s_imp:String
- s_vmp:String
- s_vk:String
- s_ki:String
- s_ns:String
- s_rs:String
- s_rp:String
- s_maxv:String
- s_minv:String
- aq:AQuery
- loadingDialog:MaterialDialog.Builder
- city:String
- country:String
- date:String
- lat:String
- lon:String
- temp:String
- rad:String
- ModuleName:TextView
+ spinnerEvent():InformationValues
+ buttonCity()
+ buttonFind()
+ nrelAPI()
+ internalSensors()
+ manualInput()
+ buttonPlot()
EmulatorFragment extends Fragment
- name:String
- iscn:double
- vocn:double
- imp:double
- vmp:double
- kv:double
- ki:double
- ns:double
- rs: double
- rp:double
- maxv:double
- minv:double
- s_name:String
- s_vocn:String
- s_imp:String
- s_vmp:String
- s_vk:String
- s_ki:String
- s_ns:String
- s_rs:String
- s_rp:String
- s_maxv:String
- s_minv:String
- aq:AQuery
- ModuleName:TextView
- producedEvent:InformationEvent
+ spinnerEvent():InformationValues
BusProvider
- BUS:Bus
+ getInstance():Bus
InformationEvent
+ name:String
+ iscn:double
+ vocn:double
+ imp:double
+ vmp:double
+ kv:double
+ ki:double
+ ns:double
+ rs: double
+ rp:double
+ maxv:double
+ minv:double
+ InformationEvent()