UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELÉCTRICA

"DESARROLLO DE UNA APLICACIÓN EN ANDROID PARA FACILITAR EL DIMENSIONAMIENTO DE UNA INSTALACIÓN FOTOVOLTAICA AISLADA"

TRABAJO FIN DE GRADO

Febrero -2021

AUTOR: Miguel Ángel García Sánchez

DIRECTOR/ES: Maria Asunción Vicente Ripoll

1 ÍNDICE

1. INTRODUCCIÓN

1.1 JUSTIFICACIÓN DEL PROYECTO

La creciente preocupación de la sociedad respecto del cambio climático y el impacto que la generación eléctrica tiene en sus efectos ha impulsado en los últimos años una rápida proliferación y desarrollo de las energías renovables a lo largo de toda la Unión Europea. Especialmente el de la energía solar fotovoltaica, ámbito en el que España ocupa una posición relevante debido a su privilegiada situación geográfica que lo sitúan como uno de los países con mayor irradiación solar de toda Europa.

Una legislación a nivel europeo y nacional cada vez más favorable y ambiciosa en cuanto a los objetivos de reducción de emisiones de efecto invernadero a corto y medio plazo, auguran un futuro muy prometedor para esta tecnología como puede extraerse de las previsiones establecidas por la UE y las principales asociaciones del sector. El creciente interés y la proliferación del número de instalaciones solares fotovoltaicas para autoconsumo, ya sea en su modalidad conectada o aislada de la red, justifica disponer cada vez más de herramientas que faciliten su cálculo y dimensionamiento y que muestren al usuario el potencial que este tipo de instalaciones tiene sobre su consumo eléctrico.

La elevada penetración en el mercado de los dispositivos móviles inteligentes [1] y del sistema operativo Android [2], lo convierten en una plataforma muy adecuada en la que implementar este tipo de herramientas.

En el siguiente proyecto se expone el desarrollo de una aplicación para dispositivos móviles Android capaz de dimensionar una instalación fotovoltaica aislada con baterías para un régimen de consumo anual a partir de los datos proporcionados por el usuario relativos a la ubicación de la instalación y características técnicas de los equipos a emplear.

La aplicación mostrará al usuario el número de captadores y baterías necesarios, así como el valor de corriente mínima de cortocircuito para seleccionar el regulador o reguladores de las baterías. La aplicación está destinada principalmente a instalaciones de autoconsumo domestico donde se ha supuesto que el consumo es continuo y constante a lo largo de todo el año. También se ha supuesto que la inclinación de los captadores es fija y la orientación sur.
El método de cálculo implementado en la aplicación es el que se recoge en el libro *Energía solar fotovoltaica. Cálculo de una instalación aislada* de Miguel Pareja Aparicio [3] que ha servido como referencia principal.

Las ecuaciones y el método de cálculo utilizados por la aplicación se exponen en el apartado Diseño del proyecto.

En el apartado Antecedentes se muestran algunas aplicaciones y webs relacionadas con la energía y cálculo de instalaciones fotovoltaicas que han servido de modelo durante el desarrollo de la aplicación.

1.2 OBJETIVOS

- Poner en práctica los conocimientos adquiridos sobre energía solar fotovoltaica durante el grado en ingeniería eléctrica de la Universidad Miguel Hernández de Elche
- Adquirir los conocimientos de programación y uso de las herramientas necesarias para el desarrollo de aplicaciones Android
- Desarrollo de una aplicación capaz de dimensionar instalaciones fotovoltaicas aisladas de la red a partir de la ubicación y características técnicas de los equipos introducidos por el usuario.

1.3 CONTEXTO

1.3.1 FOMENTO DE LAS ENERGÍAS RENOVABLES EN LA UNIÓN EUROPEA

Aunque los primeros intentos de la UE para promover el desarrollo de las energías renovables se remontan a la década de los años setenta tras la primera crisis del petróleo (1973), no es hasta los años noventa cuando el número e intensidad de las medidas y normativas en esta materia se incrementa de forma considerable a medida que también crecía la concienciación de la sociedad respecto de los problemas derivados del cambio climático y el impacto que la generación eléctrica tiene en sus efectos.

Destacan la aprobación en el año 2001 de la directiva 2001/77/CE [4] que por primera vez estaba dirigida exclusivamente a fomentar el uso de energía renovable en la producción de energía eléctrica y la directiva 2009/28/CE [5] que convertía en vinculantes los objetivos en materia energética establecidos por la Comisión Europea para la década 2011-2020 [6] . Esta directiva fijaba cuotas específicas para cada país sobre el porcentaje de energía procedente de fuentes renovables respecto de la energía total bruta disponible de forma que a nivel global se lograse el objetivo fijado del 20 % para el año 2020.

Este hecho ha marcado la política energética de Unión Europea de la última década que se ha caracterizado por la aplicación de medidas destinadas a reducir el peso en la generación eléctrica de aquellas tecnologías con mayores niveles de emisiones de CO_2 asociados, así como a garantizar el abastecimiento energético de una sociedad y economía fuertemente dependiente del exterior [7] y cada vez más vulnerable ante el incremento del coste de la energía.

Con el transcurro de la década y a medida que los objetivos están próximos a cumplirse, [8] se han establecido nuevas metas con el horizonte puesto en la próxima década. El objetivo de energía procedente de fuentes renovables se ha ampliado hasta una cuota del 32% en 2030 [9].

La aprobación en el año 2011 de la Hoja de Ruta de la Energía para 2050 [10] y su actualización en el año 2018 [11] que establece que las emisiones de efecto invernadero de la UE deberán haberse reducido entre un 80% y un 95% respecto de los niveles de 1990 en el año 2050, o más recientemente la firma del Pacto Verde Europeo de septiembre de 2020 de la Comisión, [12] reafirman las

medidas adoptadas por la UE en materia energética de los últimos años y afianzan su compromiso de reducir el impacto de su modelo energético en las emisiones de gases de efecto invernadero a la vez que avanza hacia un sistema cada vez menos dependiente del exterior.

Las políticas y medidas anteriores han supuesto un fuerte impulso al desarrollo y el fomento de las energías renovables que se han posicionado como una pieza central de la estrategia energética de la UE. La potencia instalada y la energía generada procedente de fuentes renovables no ha dejado de crecer desde el año 2000 [13]. En 2017 las energías renovables (eólica, solar, biomasa e hidráulica) produjeron por primera vez el 30% de la energía bruta de la UE [14].

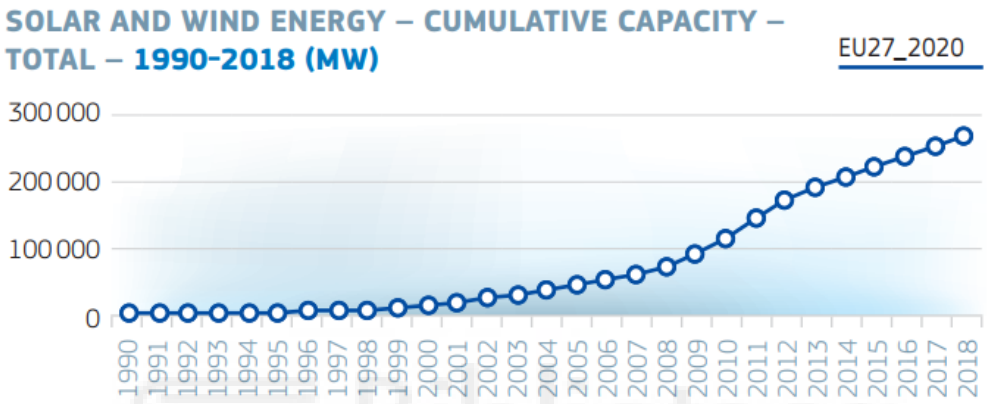


Fig. 1. Evolución de la potencia solar y eólica instalada en la Unión Europea. [13]

1.3.2 ENERGÍA SOLAR FOTOVOLTAICA EN LA UNIÓN EUROPEA

En el caso específico de la energía solar fotovoltaica, la potencia instalada tampoco ha dejado de aumentar a pesar de haberse estancando en los últimos años en algunos países, probablemente debido a un marco regulatorio nacional volátil.

A nivel global, la potencia fotovoltaica instalada durante el año 2019 se ha situado en los 16.7 GW doblando el valor del año anterior de 8.2 GW [15]. En la siguiente grafica se muestra la evolución de la potencia fotovoltaica instalada anual en la Unión Europea desde el año 2000 [15].

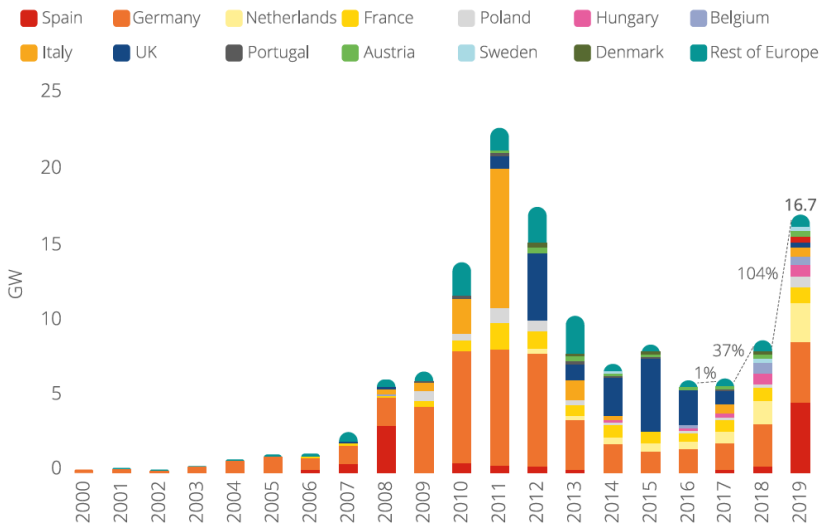


Fig. 2.Potencia fotovoltaica instalada anual en la Unión Europea por países desde el año 2000 [15]

Destacan los casos de España donde se ha producido un fuerte incremento de la potencia instalada, el mayor de la serie histórica, con 4.2 GW de forma que ha vuelto a liderar el crecimiento del mercado fotovoltaico europeo, algo que no ocurría desde el año 2008. Alemania continua con la tendencia alcista de los últimos años con 4GW instalados frente a los 2.95 GW de 2018. Los Países Bajos se han convertido en otro mercado en expansión con una potencia instalada de 2.5 GW y 1.5 GW respectivamente en los años 2019 y 2018 [15].

Las perspectivas de crecimiento para la energía solar fotovoltaica a nivel europeo son muy prometedoras. Se espera que su crecimiento continúe al alza aupado, además, de por una normativa claramente favorable a su despliegue como la comentada anteriormente a otros factores como el aumento de la competitividad por la reducción de los costes de la tecnología (baterías, módulos etc.), el incremento del autoconsumo debido a la reducción del precio de la energía fotovoltaica frente a los precios del mercado minorista o en último término, por las expectativas generadas por los planes de recuperación post-COVID anunciados hasta la fecha donde el impulso a las energías renovables se ha fijado como un pilar fundamental [15] [16].

1.3.3 ENERGÍA SOLAR FOTOVOLTAICA EN ESPAÑA

España se ha visto influenciada en gran medida por las políticas europeas de fomento de las energías renovables de las últimas décadas, hecho que se refleja en la amplia gama de normativas aprobadas desde los años noventa y motivado también por su elevado potencial en ese ámbito, especialmente en el caso de la energía solar fotovoltaica. España es uno de los países con mayor irradiación solar anual de Europa [17] lo que supone una importante ventaja competitiva que, sumada a la experiencia acumulada durante los últimos años, la convierten en unos de los principales candidatos a liderar el mercado europeo en este campo.

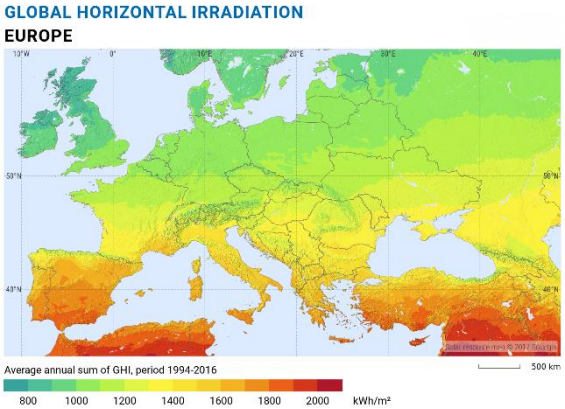


Fig. 3. Irradiación global horizontal Europa [17]

La evolución de la energía solar fotovoltaica en España se ha caracterizado por el rápido crecimiento experimentado entre los años 2005 y 2008 seguido de un largo periodo de estancamiento donde la nueva potencia instalada anual apenas se ha incrementado [18].

Uno de los factores que explican esa evolución es la aprobación de leyes como la Ley 54/1997, de 27 de noviembre, del Sector Eléctrico que definió por primera vez el denominado régimen especial vigente hasta el año 2014. Las instalaciones acogidas al régimen especial disfrutaban de una serie de incentivos o primas económicas destinadas a impulsar su competitividad y obligaba a las distribuidoras de energía eléctrica a adquirir la energía excedente de esas instalaciones siempre que fuese viable técnicamente.

Este hecho dio lugar a un fenómeno de especulación especialmente notorio en el caso de la energía fotovoltaica que culminó con la llegada de la crisis económica del año 2007. A partir de ese momento la política energética nacional se centró en controlar el déficit de tarifa acumulado producto de la política de primas, y en la aplicación de una legislación destinada a controlar dicha especulación. El efecto inmediato fue un ambiente de inseguridad jurídica en el sector que sumado a la crisis económica tuvo como consecuencia el estancamiento del crecimiento de la potencia instalada prácticamente durante la siguiente década. En la siguiente grafica se muestra la evolución de la potencia solar fotovoltaica acumulada en España desde el año 2006 donde puede apreciarse el estancamiento producido a partir del año 2008 [15].

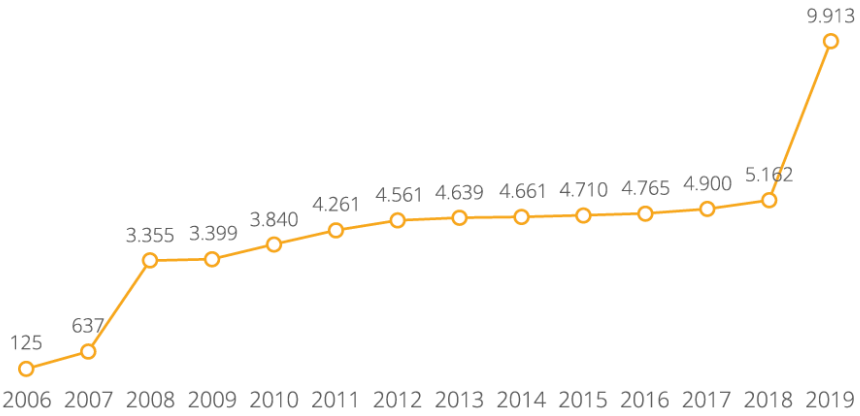


Fig. 4 Evolución de la potencia FV instalada acumulada en España en GW desde el año 2006. [15]

El estancamiento parece haberse roto en el año 2019 probablemente motivado por las subastas realizadas en el año 2017 por el gobierno en el marco de la Ley 24/2013, de 26 de diciembre que dota de una retribución específica a aquellas instalaciones de producción de energía a partir de fuentes renovables. En esa subasta, la energía solar fotovoltaica se adjudicó 3.9 GW de potencia [19]. Así, en el año 2019 se ha marcado un récord tanto en nueva potencia instalada 4.2 GW, el mayor de la serie histórica, como en generación fotovoltaica 9.223 GWh que sigue siendo de forma continuada la tercera fuente respecto de las renovables en energía generada [20]. Estas cifras no incluyen a las instalaciones de autoconsumo ya que hasta la fecha sigue sin existir un registro nacional de este tipo de instalaciones, pero se estima que la nueva legislación que simplifica y elimina las trabas económicas del marco regulatorio anterior ha impulsado su proliferación. La Unión Española Fotovoltaica (UNEF) en su informe anual del sector del año 2020 estima que la nueva potencia de autoconsumo instalada se situó en los 459 MW, casi el doble del valor estimado para el año anterior de 262 MW [15].

El impacto económico del sector sigue al alza, según cifras de la UNEF, en el año 2019 el sector aportó a la economía española 9811 millones de euros, frente a los 7811 millones del año anterior y ya supone un 0.26 % del PIB nacional [15]. De cara al futuro próximo las expectativas de crecimiento para la energía solar fotovoltaica, al igual que para el resto de renovables, siguen siendo muy prometedoras. Se espera que el crecimiento continúe al alza en los próximos años. La aprobación de la Ley de Cambio Climático y Transición Energética y su implementación a través del Plan Nacional Integrado de Energía y Clima (PNIEC) para el periodo 2021-2030 [21], ha elevado los objetivos de potencia fotovoltaica instalada para el año 2030 hasta los 39 GW lo que supone una media de 2,7 GW al año hasta 2030.

Las expectativas de crecimiento del autoconsumo también son muy prometedoras debido al elevado potencial de desarrollo en los sectores residencial, comercial o industrial como puede extraerse del análisis realizado en el informe anual para el año 2020 de la Asociación Nacional de productores de energía fotovoltaica (ANPIER) [22]. Los últimos cambios en la legislación han eliminado las trabas económicas del marco regulatorio anterior y agilizan y simplifican la tramitación expresamente para instalaciones de pequeña potencia

(< 100 KW) [23]. Esto hace de esta modalidad una opción atractiva para el consumidor medio motivado, además, por la cada vez mayor sensibilidad respecto del impacto del sistema eléctrico en el cambio climático, la reducción del tiempo de retorno de la inversión por la reducción de los costes de los equipos o las ayudas directas de la administración en forma de reducción de impuestos. Estas últimas medidas afectan principalmente a instalaciones de más de 100 kW que, además, tienen costes unitarios inferiores. En la siguiente gráfica elaborada por la UNEF se muestra la potencia fotovoltaica de autoconsumo instalada desde el año 2014 [18].

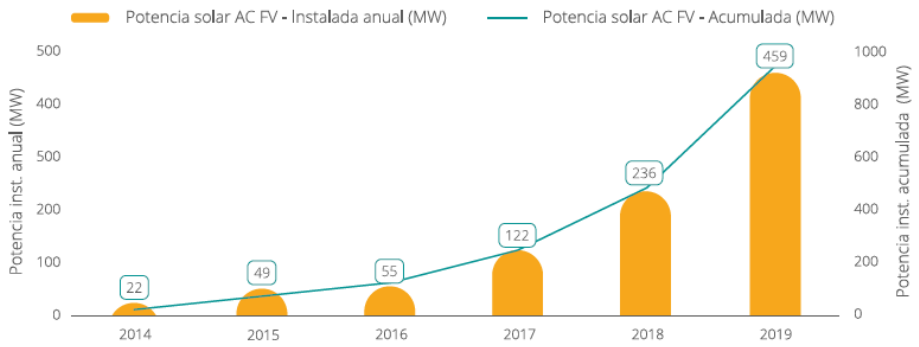


Fig. 5 Estimación de la potencia instalada de autoconsumo fotovoltaico en España desde el año 2014 [18]

2 ENERGÍA SOLAR FOTOVOLTAICA

Consiste en la transformación directa de la energía procedente del sol en forma de radiación en energía eléctrica mediante el uso de módulos fotovoltaicos. El principio de funcionamiento de estos dispositivos se basa en el denominado efecto fotovoltaico o efecto fotoeléctrico [24].

Se trata de una tecnología renovable en cuanto a que se adapta a los ciclos naturales e inagotable. Es modular, de forma que permite desde la construcción de enormes instalaciones a nivel de suelo formadas por miles de paneles para la generación y venta de electricidad a gran escala, a pequeños sistemas compuestos por muy pocos módulos instalados en tejados para autoconsumo. La disponibilidad de la energía solar en toda la superficie terrestre la convierten en una tecnología idónea para abastecer lugares remotos donde no llega el tendido eléctrico o es muy costosa su instalación.

2.1 RADIACIÓN SOLAR

La radiación solar es la radiación electromagnética emitida por el Sol producto de las reacciones de fusión que ocurren en su interior. Se divide en tres tipos dependiendo de la forma en la que incide sobre la Tierra. Esta puede ser directa, difusa o reflejada.

La radiación solar es un parámetro fundamental para poder dimensionar cualquier instalación fotovoltaica. A partir de la radiación incidente en una zona se puede determinar la energía máxima disponible.

Cuando se trata de instalaciones fotovoltaicas, su valor suele expresarse en forma de irradiancia o irradiación. La irradiancia se refiere a la potencia incidente por unidad de superficie de una zona geográfica en forma de W/m^2, mientras que la irradiación corresponde a la energía por unidad de superficie que llega a un determinado lugar en un periodo de tiempo, esta suele indicarse en Wh/m^2.

2.1.1 HORAS SOLAR PICO (HSP)

Cuando se realizan los cálculos de instalaciones fotovoltaicas también es habitual encontrar el valor de irradiación de una zona expresado en horas solar pico. Este se refiere al valor de la energía solar recibida durante un día entero en un lugar concreto en forma de un numero equivalente de horas donde hipotéticamente se hubiese recibido constantemente una irradiancia de 1000 W/m^2.

La siguiente figura ilustra el concepto de hora solar pico. El área bajo la curva corresponde a la energía solar por metro cuadrado recibida durante todo un día en una zona. Bajo la línea recta su distribución si la potencia incidente hubiese sido de forma continua 1000 W/m^2.

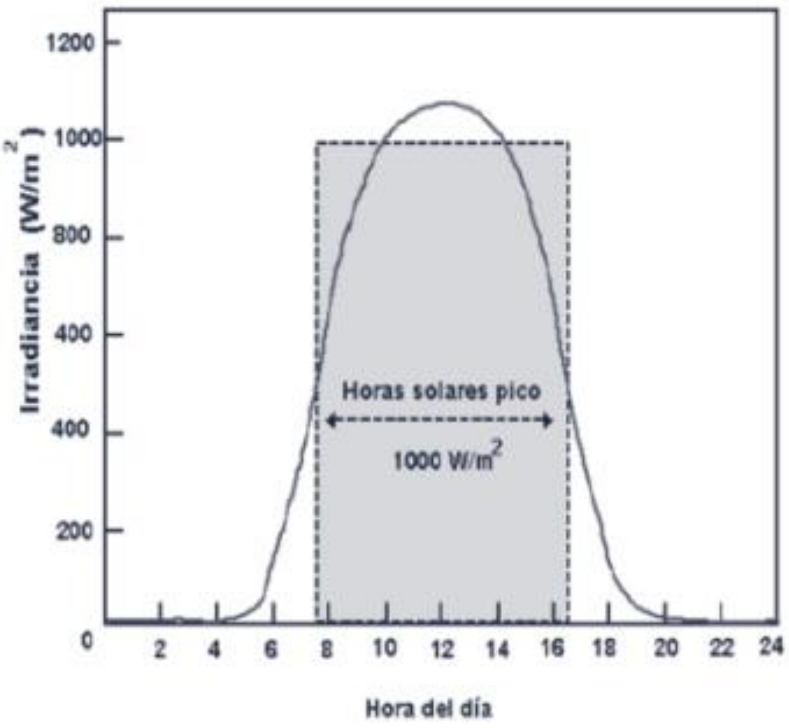


Fig. 6 Hora solar pico [3]

2.1.2 BASES DE DATOS

Lo datos de irradiación de un determinado lugar pueden consultarse en bases de datos que contienen los valores estadísticos históricos. Algunas de ellas se enumeran a continuación:

- Opensolar DB
- Atmospheric Science Data Center – NASA
- Laboratorio Nacional de Energías Renovables (NREL)
- PVGIS

2.1.3 PVGIS

Es una base de datos desarrollada por la Comisión Europea que contiene datos de la radiación solar y temperatura diarias de cualquier zona geográfica de Europa y algunas partes de Asia, América o África. Es bastante conocida y utilizada para el cálculo de sistemas fotovoltaicos. Dispone de varias herramientas, entre ellas el cálculo de la producción fotovoltaica de una instalación. Cuenta con una versión web interactiva para su uso. También dispone de una API que permite acceder y extraer los datos almacenados para usarlos en proyectos externos.

2.2 TIPOLOGÍA

Las instalaciones solares fotovoltaicas pueden clasificarse en función de su tamaño o su uso, de forma que es fácil distinguir las grandes centrales de generación de aquellas pequeñas instalaciones destinadas al consumo doméstico o autoconsumo.

Las instalaciones para autoconsumo pueden clasificarse a su vez en dos bloques:

- Conectadas a la red de distribución eléctrica. Este tipo de instalaciones tienen el respaldo de la red eléctrica en casos de fallo o cuando el recurso fotovoltaico no está disponible. Además, tienen la opción de verter a la red los excedentes de energía para su venta.
- Aisladas de la red de distribución. La demanda de energía de los receptores se abastece únicamente a través de la energía generada por la instalación. En esta modalidad es conveniente disponer de un sistema de almacenamiento que garantice la fiabilidad del suministro.

2.3 COMPONENTES DE UNA INSTALACIÓN FOTOVOLTAICA

Cualquier instalación fotovoltaica está compuesta fundamentalmente por módulos fotovoltaicos, bancos de baterías, reguladores de carga y convertidores de energía eléctrica. Dependiendo de la función de la instalación algunos de estos elementos podrán no formar parte del sistema.

2.3.1 GENERADOR FOTOVOLTAICO

Es el encargado de producir la energía eléctrica demandada por los receptores conectados a la instalación. La cantidad de energía generada por un panel fotovoltaico dependerá de sus características eléctricas, de la irradiación incidente y de su tamaño e inclinación respecto del sol.

Están formados por varias unidades denominadas células fotovoltaicas conectadas entre sí. Estas células que están compuestas generalmente de silicio puro, aunque también existen de otro tipo de materiales menos utilizados [24], generan una corriente eléctrica cuando son atravesadas por una radiación electromagnética como es el caso de la radiación solar. Este fenómeno es conocido como efecto fotovoltaico o efecto fotoeléctrico [24].

El proceso de fabricación de las células fotovoltaicas requiere de un tratamiento donde el silicio es cristalizado [24]. Dependiendo de las características cristalinas del silicio de las células se distinguen tres tipos de módulos fotovoltaicos.

- **Silicio monocristalino**. Presentan un característico color azul oscuro debido a la perfecta ordenación de los átomos que conforman la red cristalina de silicio. Son los que tienen un mayor rendimiento, pero también un coste más elevado.
- **Silicio policristalino.** Los átomos de silicio que componen la red cristalina se encuentran ordenados por regiones separadas entre si dando

lugar a irregularidades. A simple vista se reconocen por los tonos grises y azules de las células fotovoltaicas.

- **Amorfo.** Los átomos de silicio no se disponen formando una red cristalina y presentan múltiples impurezas en su estructura interna. Su proceso de fabricación es menos costoso que el de los anteriores, pero a su vez el rendimiento también es inferior.

El comportamiento de un módulo fotovoltaico puede caracterizarse a través de la curva I-V que relaciona los valores de tensión y corriente del panel obtenidos experimentalmente bajo unas condiciones constantes de irradiación y temperatura. Las condiciones estándar de medida aceptadas a nivel internacional establecen una irradiación de 1000 W/m^2, 25ºC de temperatura y una masa de aire de 1,5. Este estándar es usado por los fabricantes para indicar las características eléctricas de los captadores. Los parámetros que definen la curva I-V son los siguientes:

- Intensidad de cortocircuito I_{SC}. Corresponde a la máxima corriente que el panel puede suministrar. Se obtiene cuando se cortocircuitan los bornes del módulo.
- Tensión en circuito abierto V_{OC}. Es la máxima tensión que puede medirse en un módulo y que se produce cuando no está conectado a ningún receptor.
- Tensión nominal V_N. Valor de la tensión de trabajo para el que se ha diseñado el panel.
- Potencia máxima P_M. Valor de la máxima potencia que puede suministrar el módulo.
- Tensión máxima V_M. Valor de la tensión del panel medida en el punto de máxima potencia.
- Corriente máxima I_{MP}. Valor de la máxima corriente medida en el punto de máxima potencia.

La siguiente figura muestra la curva I-V de un panel genérico con los parámetros definidos anteriormente.

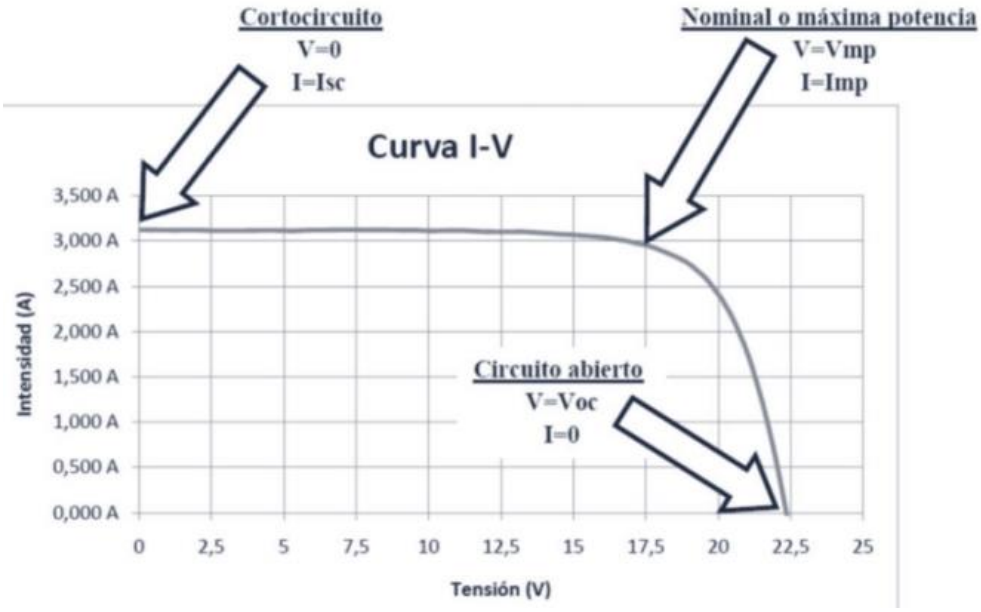


Fig. 7 Curva I-V captador solar [3]

Las condiciones de trabajo reales de los módulos fotovoltaicos pueden ser distintas a las obtenidas bajo las condiciones estándar por lo que es muy útil conocer los efectos en la curva I-V de las variaciones de irradiación y temperatura.

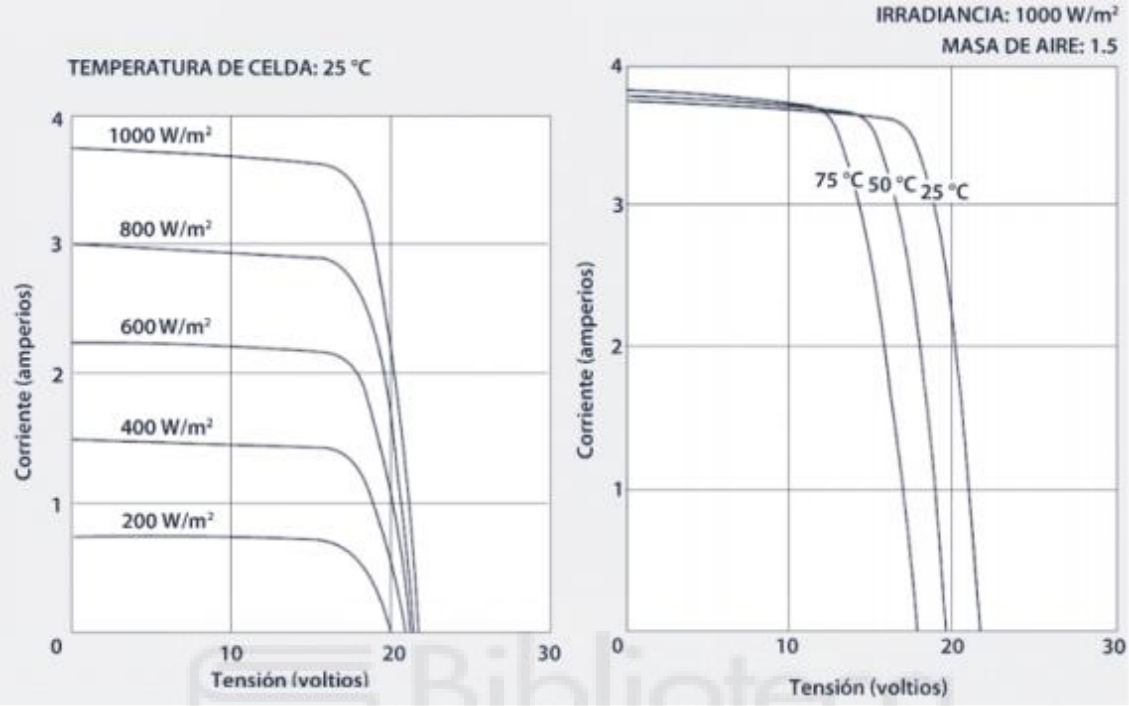


Fig. 8 Curvas I-V dependientes de la irradiación y temperatura [3]

Conocer los valores de orientación e inclinación óptima de los captadores es de suma importancia ya que de ello depende en gran medida la cantidad de energía producida por la instalación. Como norma general, en el hemisferio norte los capadores deben estar orientados hacia el sur geográfico, aunque son admisibles desviaciones de hasta el 20%.

La orientación de un captador fotovoltaico se expresa por medio del ángulo azimut (α). Así, un captador con un azimut de 0º estará orientado en dirección sur y de 90º si está orientado hacia el oeste.

La captación solar del módulo será máxima cuando la dirección de la radiación solar sea perpendicular a su superficie. El ángulo de inclinación de un captador fotovoltaico se indica con la letra β, de forma que si su valor es igual a 0º este tendrá una posición completamente horizontal respeto del suelo y de 90º si está en vertical.

El ángulo de incidencia de la radiación solar varía en función de la hora del día y el periodo del año. Algunas instalaciones están equipadas con dispositivos de seguimiento solar que modifican la inclinación de los captadores a lo largo del día y el año, de forma que la radiación incide siempre perpendicularmente sobre la superficie de estos. En el caso de instalaciones destinadas a autoconsumo doméstico, como es el caso de las instalaciones contempladas en este proyecto, la inclinación suele ser fija por lo que se escoge un ángulo que permite obtener un valor máximo de irradiación media anual. En la bibliografía pueden

encontrarse distintos métodos para obtenerlo. En este proyecto el valor óptimo de inclinación anual se obtendrá por medio de la herramienta PVGIS.

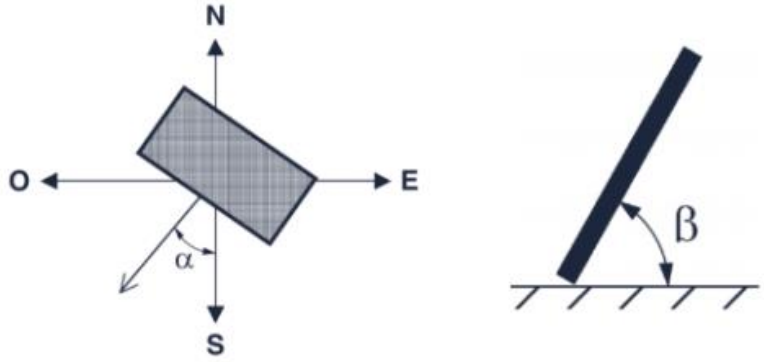


Fig. 9 Ángulos de orientación e inclinación de los captadores [3]

2.3.2 BATERÍAS

Almacenan la energía eléctrica generada por los captadores fotovoltaicos en forma de energía química mediante reacciones de oxidación-reducción. Se clasifican en función del tipo de electrodo y electrolito utilizados. A continuación, se enumeran algunas de las más comunes.

- Plomo-acido. Son las más baratas. Los electrodos son de plomo y el electrolito una solución de agua y ácido sulfúrico. Pueden encontrarse varios tipos (Monobloc, gel y AGM)
- Níquel-cadmio. El material del cátodo es níquel y el del ánodo cadmio. Se usan en instalaciones donde se requiere un alto nivel de fiabilidad ya que apenas varían su nivel de tensión durante la descarga. Soportan bastante bien las bajas temperaturas, incluso la congelación del electrolito, por lo que suelen usarse en entornos con temperaturas extremas.
- Litio. Tienen una larga vida útil y soportan mejor las descargas profundas. Su densidad energética es elevada, lo que significa que proporcionan más energía en menos tamaño, aunque también son las más caras.

Las características técnicas de una batería definen sus condiciones de trabajo y funcionamiento. Es importante tenerlas en cuenta a la hora de diseñar una instalación fotovoltaica.

- Tensión nominal. Valor de tensión de trabajo para el que ha sido diseñada la batería.
- Capacidad nominal. La cantidad de energía eléctrica que puede suministrar la batería. Se mide en Ah y suele especificarse para un determinado tiempo o régimen de descarga. Así, una batería de 1 Ah para un régimen de descarga de 100 horas (C_{100}) es capaz de proporcionar una corriente constante de 100 A durante 100 horas o 50 A durante 200 horas.
- Régimen de carga y descarga. Es el cociente entre la capacidad nominal de la batería y el valor de la corriente de carga y descarga para el que ha sido diseñada. Su valor se expresa en horas.

- Ciclo de vida. Se refiere al número de veces que puede producirse la carga y descarga de la batería durante toda su vida útil. Cada carga y descarga completas de la batería se denomina ciclo de carga. A media que estos ciclos transcurren la batería va perdiendo algunas de sus propiedades, como su capacidad máxima, debido al deterioro que sufre en cada uno de ellos.
- Autodescarga. Es el proceso de descarga que ocurre en las baterías de forma lenta y persistente debido a las reacciones químicas que se están produciendo de forma constante en su interior. Es recomendable mantener las baterías cargadas si no van a utilizarse durante un largo periodo de tiempo para evitar descargas profundas.
- Profundidad de descarga. Es el porcentaje de la capacidad total de la batería que puede utilizarse durante un ciclo de carga. Las baterías pueden clasificarse en función de este valor en superficiales, admiten hasta un 20% de la descarga de la capacidad nominal, y en descargas profundas cuando admiten descargas de hasta un 80 % de la capacidad nominal.

2.3.3 REGULADOR

Es el dispositivo que regula los procesos de carga y descarga del banco de baterías. Interrumpe la corriente desde las baterías hacia la instalación cuando se han descargado para evitar descargas profundas o desde el campo fotovoltaico hacia las baterías para evitar sobrecargas. Existen dos tipos de reguladores.

- Reguladores PWM. Son los más sencillos y baratos. Actúan como un interruptor entre los módulos fotovoltaicos y las baterías. En las instalaciones con este tipo de reguladores, la tensión de los paneles debe coincidir con la del banco de baterías ya que no pueden modificar la tensión de entrada. El voltaje del banco de baterías no suele coincidir con el punto de máxima potencia de los paneles lo que produce una pérdida de potencia que puede alcanzar el 30%.
- Reguladores MPPT. Cuentan con un convertidor CC-CC y un seguidor del punto de máxima potencia en su interior por lo que pueden modificar la tensión de entrada del campo fotovoltaico para que a la salida coincida con la del banco de baterías. De esta manera, los paneles pueden trabajar siempre en el punto de máxima potencia.

2.3.4 CONVERTIDOR

Transforma la tensión y corriente que recibe de las baterías de forma que sean adecuadas para los equipos conectados a la instalación. Se pueden encontrar varios tipos CC-CC, CA-CC, CC-CA, CA-CA. El más habitual en el caso de instalaciones solares fotovoltaicas es el convertidor de corriente continua a alterna o inversor. El inversor transforma la corriente continua recibida de los captadores o las baterías a corriente alterna y eleva la tensión al nivel de tensión de trabajo de los receptores.

A la hora de seleccionar un inversor es importante que sea capaz de soportar la potencia máxima de la instalación. También debe tenerse en cuenta si se van a verter los excedentes a la red eléctrica, en ese caso el inversor deberá reproducir de la forma más fiel posible las formas de onda de la red.

2.3.5 TIPOS DE CONFIGURACIÓN

Los componentes descritos anteriormente pueden conectarse entre sí adoptando múltiples configuraciones dependiendo de las necesidades de la instalación[26]. En este proyecto se han considerado aquellas instalaciones destinadas a autoconsumo en su modalidad asilada de la red. La configuración más habitual en este tipo de instalaciones es la que se muestra en la figura 10.

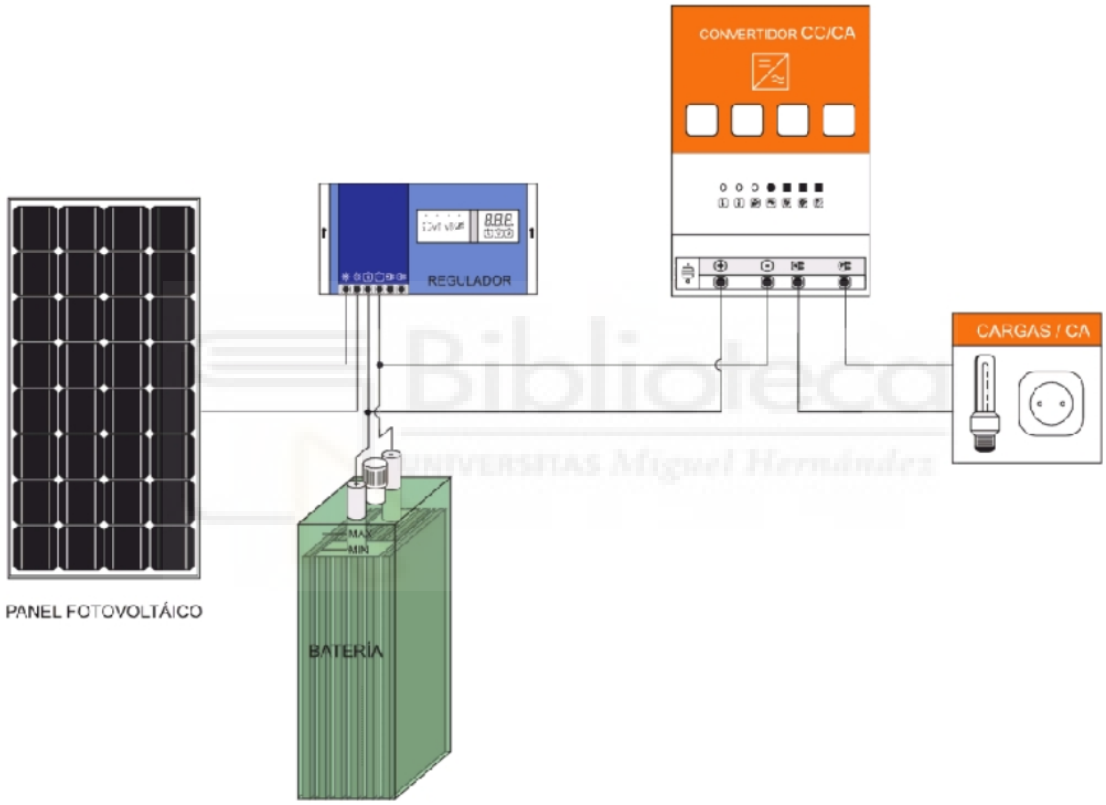


Fig. 10 Configuración típica para sistemas fotovoltaicos aislados con acumulación [26]

3 ANTECEDENTES: APLICACIONES Y PORTALES WEB PARA EL CÁLCULO DE LAS INSTALACIONES FOTOVOLTAICAS

En el siguiente apartado se muestran algunas de las webs o aplicaciones para el dimensionamiento de instalaciones fotovoltaicas disponibles en la App Store de Google que han servido como referencia durante el proceso de desarrollo.

3.1 SOLAR CT

Es una de las más completas y con un diseño más cuidado. Dispone de una amplia gama de funciones que cubren la mayoría de los aspectos relacionados con el dimensionamiento.

La opción *Cálculo paso a paso* se asemeja bastante a los objetivos que se pretenden lograr en este TFG. En ella a partir de los datos introducidos por el usuario se obtiene el número de equipos necesarios de la instalación (módulos, baterías, reguladores y tamaño del inversor). Una sus limitaciones es que necesita que el usuario introduzca de forma manual los datos de irradiación en forma de horas de pico solar (HPS). Dispone también de otras opciones donde cada uno de los pasos del dimensionamiento pueden calcularse de forma independiente y con más detalle. También cuenta con una función específica para instalaciones de bombeo fotovoltaico.

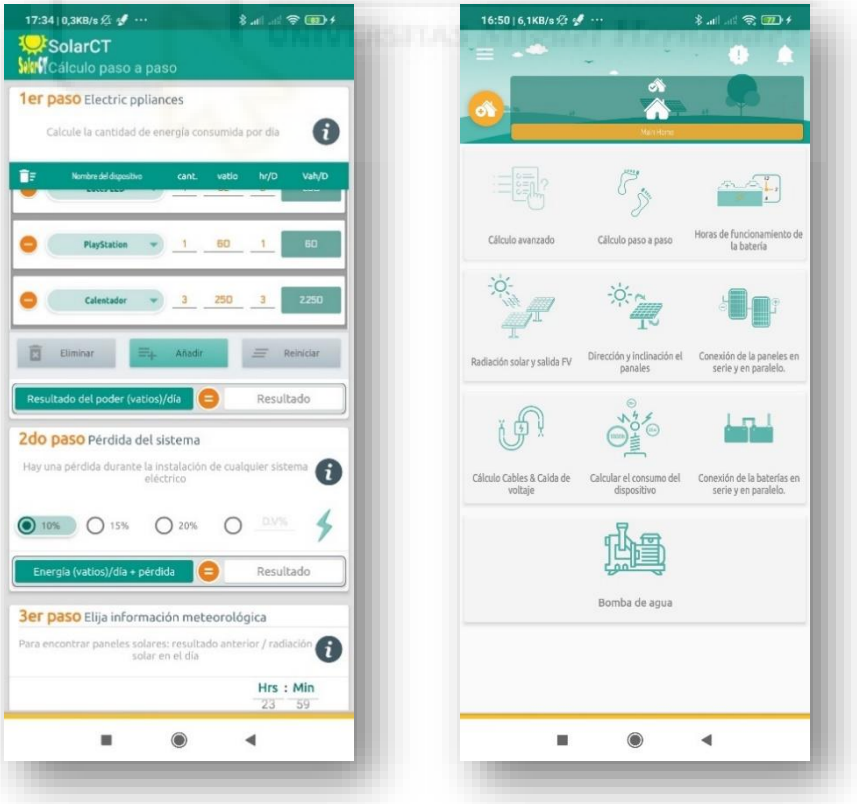


Fig. 11 Solar CT

3.2 AUTOCONSUMO SOLAR

Permite dimensionar instalaciones destinadas a autoconsumo ya sean aisladas o conectadas a la red, así como instalaciones destinadas a bombeo fotovoltaico. Una de las características de la aplicación es que obtiene los datos de irradiación necesarios para el cálculo a partir de la ubicación introducida por el usuario y empleando, según indican sus desarrolladores, los datos del atlas de radiación solar elaborado por el Ministerio de agricultura, alimentación y medio ambiente de España, la CFE de México o los proporcionados por la base de datos de la NASA. Además del dimensionamiento, permite obtener una estimación del coste de la instalación y el periodo de retorno de la inversión a partir del ahorro en el consumo eléctrico.

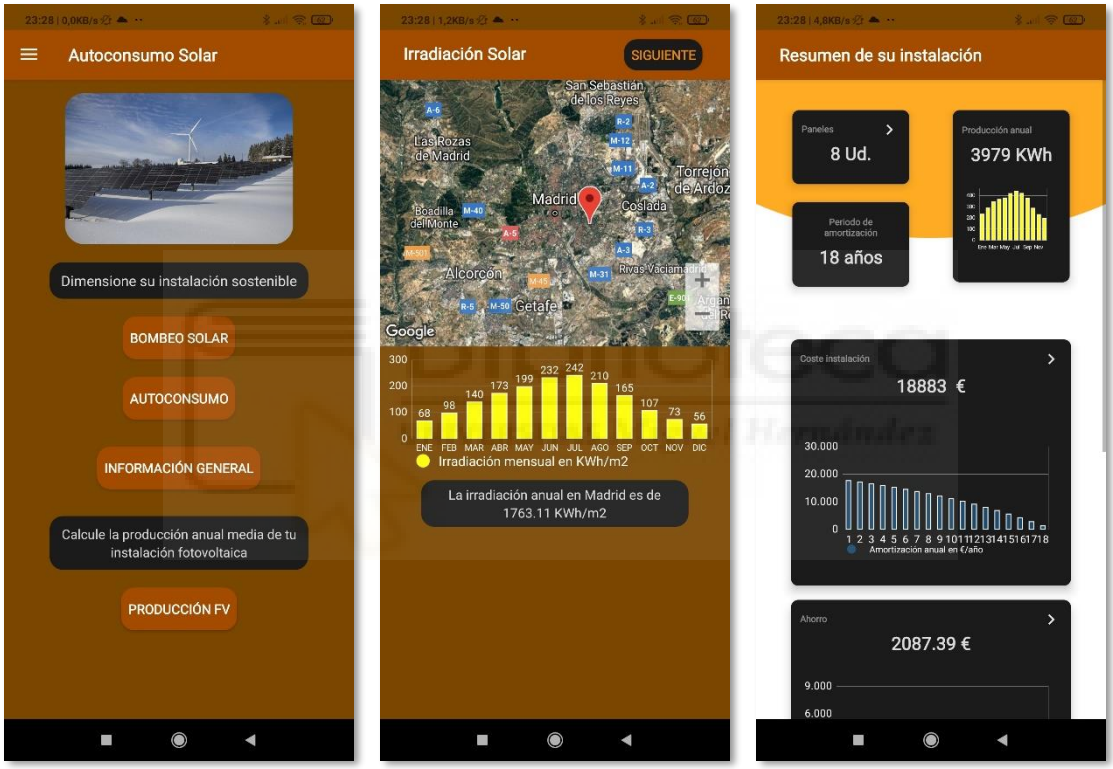


Fig. 12 Autoconsumo solar

3.3 CALCULADORA SOLAR AUTOCONSUMO

Permite calcular de forma independiente los parámetros de un sistema fotovoltaico a partir de los datos introducidos por el usuario. A través de varios botones y de forma separada pueden calcularse el número de módulos fotovoltaicos o de baterías introduciendo sus parámetros eléctricos y datos de consumo de las cargas conectadas. Dispone de una opción para obtener las horas de pico solar en función de la provincia donde este situada la instalación. También permite dimensionar la sección del cableado teniendo en cuenta la caída de tensión, la temperatura y el material de los conductores.

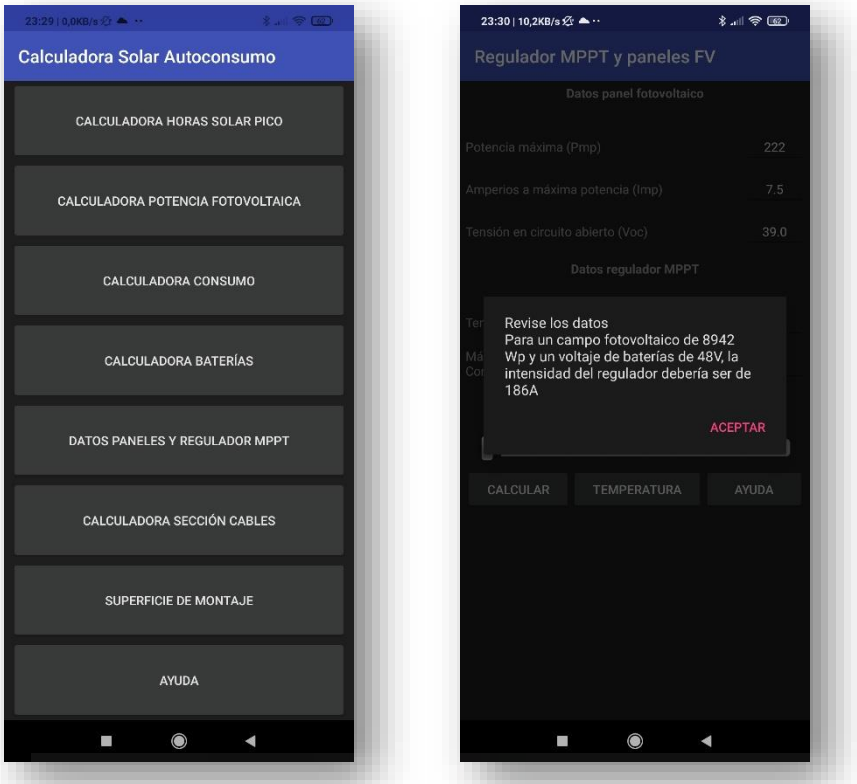


Fig. 13 Calculadora solar autoconsumo

3.4 CALCULATIONSOLAR.COM

Se trata de una web que permite dimensionar un sistema fotovoltaico a partir de la ubicación, datos de consumo, periodo de uso y orientación de los captadores. Dispone de una lista de componentes a nivel interno de forma que elige automáticamente el que más adecuado en función de la información introducida. Muestra el número de componentes necesarios en la instalación y sus características. Permite una vez realizados los cálculos modificar los modelos de los componentes seleccionados automáticamente por otros de la base de datos interna, pero no por otros introducidos por el usuario manualmente.

4 DISEÑO

4.1 SELECCIÓN DE LOS PASOS DEL DISEÑO DE LA INSTALACIÓN FOTOVOLTAICA

En el siguiente apartado se definen los parámetros necesarios para el dimensionamiento de instalaciones fotovoltaicas aisladas y de las ecuaciones necesarias para llevarlo a cabo. Estas ecuaciones serán las mismas que se utilicen en la aplicación.

4.1.1 DIMENSIONAMIENTO

El concepto de dimensionamiento consiste en el cálculo del número óptimo de elementos que componen la instalación teniendo en cuenta diversos factores como la demanda, el coste o la fiabilidad. El criterio de fiabilidad es esencial cuando se trata de instalaciones aisladas ya que no cuentan con el respaldo de la red eléctrica en caso de fallo. Existe una relación directa entre el tamaño de los equipos y la fiabilidad del sistema, de forma que se debe llegar a una solución óptima entre la energía producida y la energía almacenada. Cuanto mayor sea el tamaño del sistema de almacenamiento y los captadores más fiable será la instalación, pero a su vez también se incrementará su coste.

4.1.2 MÉTODOS DE DIMENSIONAMIENTO

En la bibliografía se recogen multitud de métodos para el dimensionado de instalaciones fotovoltaicas, algunos de ellos se basan en cálculos complejos que requieren de avanzados sistemas informáticos para su ejecución.
En este proyecto se ha optado por seguir un método sencillo que sea fácil de implementar en la aplicación, pero que permita obtener resultados lo suficientemente fiables a partir de los datos introducidos.

El método de cálculo seleccionado es el que está descrito en el libro *Energía solar fotovoltaica. Cálculo de una instalación aislada* de Miguel Pareja Aparicio [3]. Está basado en el denominado método *Amperio-hora* que se ha modificado de forma que cumpla con los cálculos del método definido por el IDAE en su documento *Pliego de condiciones técnicas de instalaciones aisladas de red* [25].

El método de cálculo puede resumirse en los pasos enumerados a continuación:

- Cálculo de la energía media diaria anual consumida por la instalación.
- Cálculo del coeficiente de pérdidas
- Cálculo del número de módulos fotovoltaicos.
- Cálculo del número de baterías.
- Cálculo de la corriente mínima del regulador.
- Cálculo de la potencia mínima del inversor.

4.1.3 ENERGÍA MEDIA DIARIA DE LA INSTALACIÓN

Es la energía que debe proporcionar el sistema fotovoltaico para abastecer la demanda de los receptores. Es un factor fundamental a la hora de dimensionar una instalación fotovoltaica aislada ya que junto con los días de autonomía determina el tamaño de esta. Para obtenerla es necesario conocer el número y la potencia de los receptores y su tiempo de funcionamiento medio. El tiempo de funcionamiento puede estimarse conociendo el uso que se va a dar a la instalación o bien extraerse de tablas de referencia elaboradas por diversas fuentes.

$$E_d = \sum n \cdot P_i \cdot T$$

- E_d Energía máxima diaria de la instalación expresado en Wh/día
- n Numero de cada tipo de receptor conectado a la instalación
- P_i Potencia de cada uno de los receptores conectados a la instalación
- T Tiempo medio en horas diarias de demanda de cada receptor

4.1.4 PÉRDIDAS DE LA INSTALACIÓN

Aquellas relacionadas con cada uno de los elementos que forman parte del sistema (baterías, captadores, regulador, inversor), así como a factores no contemplados. Estas pérdidas pueden cuantificarse en forma de coeficiente a partir de la siguiente ecuación:

$$\boldsymbol{K_T = [1 - (K_B + K_C + K_R + K_X)] \cdot [1 - \frac{(K_A \cdot D_{aut})}{P_d}]}$$

- $\boldsymbol{K_T}$ Pérdidas totales de la instalación
- $\boldsymbol{K_A}$ Pérdidas debidas a la autodescarga diaria de la batería a 20 ºC
- $\boldsymbol{K_B}$ Pérdidas debidas al rendimiento de la batería
- $\boldsymbol{K_C}$ Pérdidas debidas al rendimiento del inversor
- $\boldsymbol{K_R}$ Pérdidas debidas al rendimiento del regulador
- $\boldsymbol{K_X}$ Pérdidas no contempladas como el efecto Joule o caídas de tensión
- $\boldsymbol{D_{aut}}$ Días de autonomía de la instalación
- $\boldsymbol{P_d}$ Profundidad de descarga de la batería

En la aplicación las perdidas por la autodescarga de la batería y de las pérdidas no contempladas tendrán un valor por defecto del 0,5% y 10% respectivamente al tratarse de valores típicos recogidos en [3].

4.1.5 NUMERO DE MÓDULOS FOTOVOLTAICOS

Para determinar el número total de módulos fotovoltaicos es necesario conocer el valor de la energía media diaria de la instalación que se ha calculado en el

primer apartado, las características eléctricas de los módulos que se van a utilizar y los datos de irradiación de la instalación.

$$N_{modulos} = \frac{E_d}{0{,}9 \cdot P_{modulo} \cdot HPS \cdot K_T}$$

- $N_{modulos}$ Número total de captadores fotovoltaicos de la instalación
- E_d Energía media diaria de la instalación expresada en Wh/día
- P_{modulo} Potencia máxima del captador
- HPS Horas de pico solar
- K_T Coeficiente de pérdidas de la instalación

El valor de 0,9 que aparece en el denominador de la ecuación anterior corresponde al rendimiento del captador fotovoltaico que oscila según la bibliografía que se consulte entre el 85% y el 95%. Se ha escogido el 90% por ser el que está recogido en [3].

El valor para las horas de pico solar será el del mes más desfavorable del año y con la inclinación optima de los captadores.

El valor obtenido con la ecuación anterior, en caso de tratarse de un valor decimal, debe redondearse al entero superior más cercano.

Una vez se tiene el total de módulos necesarios, puede calcularse su disposición en la instalación.

4.1.6 CAPTADORES EN SERIE

$$N_{m,s} = \frac{V_{instalación}}{V_{m,nominal}}$$

- $N_{m,s}$ Numero de módulos conectados en serie
- $V_{instalación}$ Tensión nominal de la instalación que coincide con el del banco de baterías en el caso de reguladores PWM.

En el caso de reguladores MPPT la tensión de la instalación no coincide con el de las baterías ya que esta se adapta a la tensión de funcionamiento del campo de captadores en cada momento para obtener la máxima potencia. Por simplicidad, para el desarrollo de la aplicación se ha considerado que los reguladores utilizados son del tipo PWM.

4.1.7 CAPTADORES EN PARALELO

$$N_{m,p} = \frac{N_{modulos}}{N_{m,s}}$$

- $N_{modulos}$ Número total de módulos de la instalación
- $N_{m,p}$ Número de módulos conectados en serie

4.1.8 CAMPO DE BATERÍAS

Cuando se dimensiona una instalación fotovoltaica aislada es necesario disponer de un sistema de baterías que ajuste la curva de generación del sistema a la curva de demanda de los receptores acumulando la energía generada en momentos de baja demanda y la suministre cuando no se dispone del suficiente recurso fotovoltaico.

Para calcular la capacidad del banco de baterías debe tenerse en cuenta la energía media diaria demandada por la instalación cuando no se dispone de la suficiente irradiación solar, los días de autonomía y la profundidad de descarga de las baterías utilizadas.

Los días de autonomía dependerán del perfil de uso que vaya a darse en la instalación y de las condiciones climatológicas en función de su ubicación, es decir, la media del número de días nublados al año. En la bibliografía pueden encontrarse múltiples métodos y criterios para obtenerlos.

Capacidad mínima del campo de baterías

$$C_{bat} \geq \frac{E_d \cdot D_{aut}}{P_d \cdot K_T}$$

- C_{bat} Capacidad mínima del banco de baterías
- E_d Energía media diaria de la instalación
- P_d Profundidad de descarga de la batería
- D_{aut} Días de autonomía
- K_T Coeficiente de pérdidas de la instalación

4.1.9 BATERÍAS EN PARALELO

$$N_{b,p} = \frac{E_d \cdot D_{aut}}{C_{nom,bat} \cdot V_{instlación} \cdot P_d \cdot K_T}$$

- $C_{nom,bat}$ Capacidad nominal del modelo de batería para un régimen de descarga de 100 horas
- $V_{instlación}$ Tensión nominal de la instalación

En caso de obtener un valor decimal este debe redondearse al entero superior más cercano.

4.1.10 BATERÍAS EN SERIE

$$N_{b,s} = \frac{V_{instalación}}{V_{nom,bat}}$$

- $V_{nom,bat}$ Tensión nominal de la batería seleccionada

4.1.11 CORRIENTE MÍNIMA DEL REGULADOR

El regulador deberá ser capaz de soportar la corriente máxima del campo fotovoltaico debida a un cortocircuito. El valor mínimo de la corriente del regulador se obtiene con la siguiente ecuación.

$$I_{regulador} \geq F_{SR} \cdot I_{CC,modulo} \cdot N_{m,p}$$

- $I_{regulador}$ Intensidad del regulador
- F_{SR} Coeficiente de seguridad
- $I_{CC,modulo}$ Intensidad de cortocircuito de los módulos fotovoltaicos
- $N_{m,p}$ Numero de módulos fotovoltaicos conectados en paralelo

4.1.12 POTENCIA MÍNIMA DEL INVERSOR

Debe ser capaz de soportar la suma de la potencia nominal de todas las cargas de la instalación. Como es difícil que se produzca un momento en el que todos los receptores estén funcionando al mismo tiempo, puede emplearse un coeficiente de simultaneidad. Se ha fijado un valor del 80% que se corresponde con el utilizado en [libro].

$$P_{inversor} \geq F_{sim} \cdot \sum n \cdot P_i$$

- $P_{inversor}$ Potencia mínima del inversor
- F_{sim} Coeficiente de simultaneidad
- n Numero de cada tipo de receptor conectado a la instalación
- P_i Potencia de cada uno de los receptores de la instalación

4.2 DISEÑO DE LAS FUNCIONALIDADES DE LA APLICACIÓN

Antes de comenzar con la programación de la aplicación debe realizarse un análisis previo donde se definan cuáles van a ser las funcionalidades y aspectos básicos de diseño. Este paso es fundamental ya que en él se determinan las bases sobre las que se asentará el desarrollo y diseño posteriores. Sirve también para entender a grandes rasgos su funcionamiento.

El objetivo fundamental de la aplicación es indicar al usuario el número de captadores y baterías necesarios de una instalación fotovoltaica aislada para un régimen de consumo anual a partir de los datos de consumo y características eléctricas de los equipos a utilizar. Debe contar con una interfaz gráfica donde se puedan introducir los datos necesarios para realizar el cálculo que se ejecutara empleando las ecuaciones descritas en el apartado anterior. Esa misma interfaz podrá servir también para mostrar los resultados obtenidos. Dado que para llevar a cabo el dimensionamiento se debe disponer de los valores de irradiación de la instalación, se desarrollará una funcionalidad que permita obtener esos valores a través de una base de datos externa. En este caso se ha optado por la base de datos PVGIS, una de las más conocidas y utilizadas para el cálculo de instalaciones fotovoltaicas. El valor escogido será el valor de

irradiación medio diario del mes más desfavorable del año y con la inclinación optima de los paneles fotovoltaicos. Estos valores se mostrarán junto con el resto de los resultados. También se empleará el sistema de mapas de Google junto con el sistema GPS del dispositivo móvil para obtener los datos de la ubicación de la instalación que servirán a su vez para descargar los valores de irradiación desde PVGIS.
A continuación, se enumeran las funcionalidades principales de la aplicación:

- Dimensionar una instalación fotovoltaica aislada de la red
- Obtener los datos de latitud y longitud de la instalación a partir de la localización automática del dispositivo o del sistema de mapas de Google
- Obtener los datos de irradiación e inclinación óptima a partir de la ubicación de la instalación y PVGIS.

4.2.1 METODOLOGÍA DE DESARROLLO

Existen múltiples formas de estructurar y organizar el desarrollo de aplicaciones móviles dependiendo de las características y contexto propios del proyecto. La metodología escogida debe ser ágil y adaptarse al tiempo y recursos disponibles. Algunos ejemplos de metodologías son el desarrollo en cascada o el desarrollo ágil. En este caso, al tratarse de un proyecto académico donde uno de los objetivos consiste en adquirir los conocimientos para el desarrollo básico de aplicaciones Android, se va a optar por una metodología rápida de desarrollo mediante prototipos funcionales cuya complejidad se irá incrementando a través de nuevas versiones. El modelo de desarrollo en cascada también es adecuado ya que desde el inicio están claramente definidas las funcionalidades. Estas funcionalidades no se van a modificar y cada una de ellas puede realizarse de forma secuencial.

El desarrollo de la aplicación se va a estructurar en dos fases o versiones. Una primera versión básica cuyo objetivo principal será obtener un prototipo funcional. Servirá para poner a prueba el método de cálculo seleccionado y los conocimientos de programación adquiridos. El usuario introducirá en un mismo *layout* o pantalla todos los datos necesarios para el cálculo, incluidos el coeficiente de pérdidas y el valor de irradiación. En esa misma pantalla también se mostrarán los resultados obtenidos. En esta primera versión no se hará uso del sistema de mapas ni de la base de datos PVGIS.

En una segunda fase o versión avanzada se implementará el sistema de mapas para la localización de la instalación y la descarga de los datos de irradiación desde PVGIS. La aplicación se dividirá en varias pantallas donde localizar la instalación, introducir los datos y mostrar los resultados.
En los siguientes apartados se muestran de forma gráfica los aspectos generales del diseño y funcionamiento de las dos versiones mediante maquetas o *mockups*.

4.3 ESTRUCTURA DE LA APLICACIÓN

4.3.1 VERSIÓN SIMPLE: INSERCIÓN DE DATOS MANUAL – CALCULADORA SOLAR BÁSICA

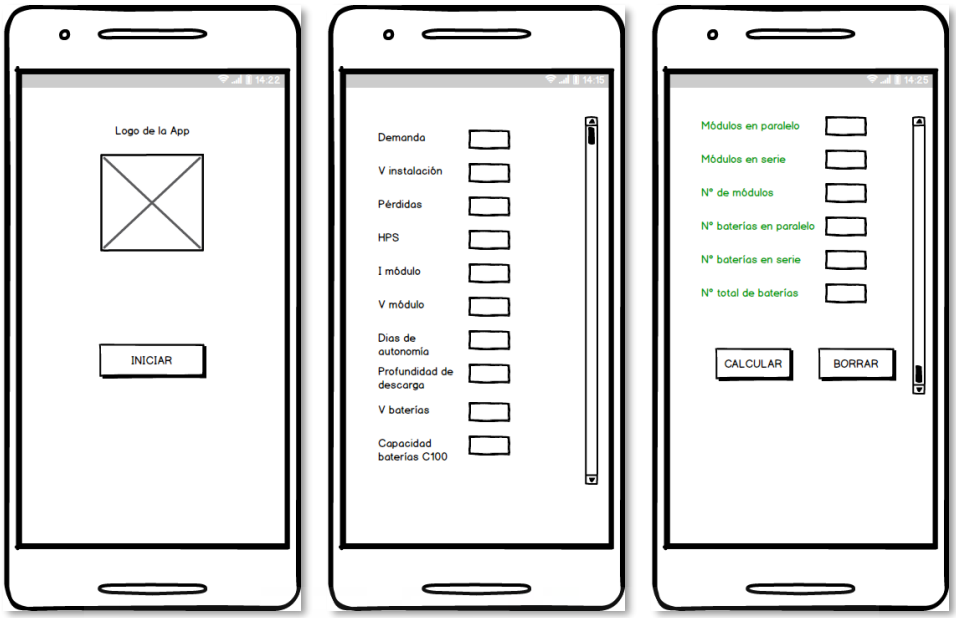


Fig. 15 Disposición general de las pantallas de la versión simple

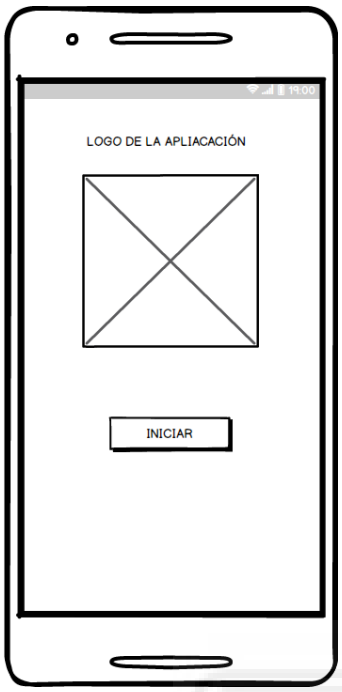
PANTALLA INICIAL

Incluirá el logo de la aplicación y un botón de inicio que conducirá a la pantalla donde se introducirán los datos necesarios para el dimensionamiento del sistema.

PANTALLA DE DATOS

En esta pantalla el usuario introducirá los datos para realizar el dimensionamiento de la instalación. Rellenará los valores de consumo y tensión del sistema junto con las características eléctricas de los paneles y baterías. También introducirá los días de autonomía, las pérdidas y los datos de irradiación en forma de horas de pico solar. Esta pantalla contará con un *scroll* para poder desplazarse entre los distintos campos. También se presentarán los resultados obtenidos. Se mostrará la cantidad total de captadores y baterías necesarios, así como su distribución en paralelo y en serie. En la parte inferior habrá dos botones. Uno para realizar el cálculo y otro para limpiar los valores y campos en el caso de realizar un nuevo dimensionamiento ya que en cada inicio de la aplicación permanecerán los datos introducidos anteriormente.

4.3.2 VERSIÓN AVANZADA: DETECCIÓN DE UBICACIÓN Y ACCESO A PVGIS – CALCULADORA SOLAR AVANZADA

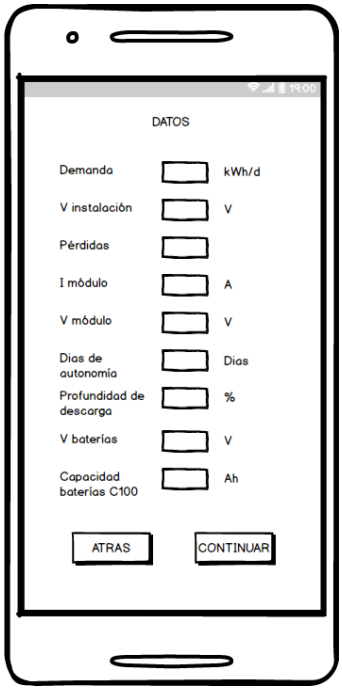


PANTALLA INICIAL

Contiene el logo de la aplicación y el botón de iniciar que conducirá a la siguiente pantalla.

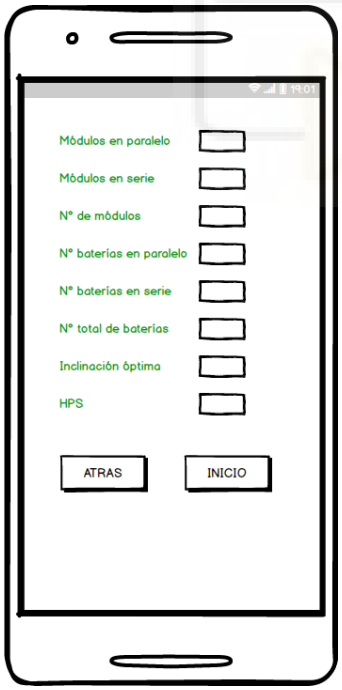
LOCALIZACIÓN

En esta pantalla el usuario indicará la ubicación de su instalación fotovoltaica. Contiene un mapa sobre el que se puede navegar y colocar un marcador pulsando sobre la pantalla. También dispone de la opción de autolocalización para obtener sus coordenadas actuales a través del GPS del dispositivo móvil. Para ello tendrá que presionar sobre el botón localizar. Esta función servirá para obtener y almacenar los valores de latitud y longitud de la instalación que son necesarios para descargar los datos de irradiación desde PVGIS. Al pulsar en continuar la aplicación se conectará a PVGIS a través de su API y descargará los valores de irradiación de los doce meses del año. Internamente se procesarán esos datos para pasarlos a horas de pico solar y se almacenará el del mes más desfavorable. También se parará a la pantalla siguiente.



DATOS

En esta pantalla se introducirán los datos de la instalación y los equipos para poder realizar el dimensionamiento. En la parte inferior habrá dos botones. El botón atrás permitirá retroceder a la pantalla anterior para modificar la ubicación. El botón continuar conducirá a la pantalla donde se muestran los resultados.



RESULTADOS

En esta pantalla se mostrarán los resultados obtenidos. Se indicarán la cantidad total de módulos y baterías del sistema, así como su distribución en serie y paralelo. Igualmente se mostrará la tensión mínima que deberá soportar el regulador de las baterías. Este valor se habrá calculado a partir de la corriente de cortocircuito y el número de captadores conectados en paralelo. También se indicarán las horas de pico solar del mes más desfavorable de la instalación y la inclinación optima de los captadores obtenido de PVGIS.

5 PROGRAMACIÓN

5.1 PLATAFORMA ANDROID STUDIO

Android Studio es la plataforma de desarrollo o IDE oficial para el desarrollo de aplicaciones para el sistema operativo Android y que a su vez está basado en el entorno Intellij IDEA. Dispone de múltiples herramientas de asistencia que mejoran la eficiencia y la productividad durante el desarrollo como su potente analizador de código. Este último permite mediante sugerencias completar el código teniendo en cuenta el contexto durante la programación o corregir las posibles incoherencias o errores que puedan producirse. Algunas de sus ventajas son las siguientes:

- Ejecuta las compilaciones de forma muy rápida.
- Permite visualizar en tiempo real los cambios en las pantallas o *layouts* permitiendo realizar el diseño del apartado gráfico incluso sin necesidad de código, simplemente arrastrado elementos.
- Cuenta con un emulador en el que visualizar el estado de la aplicación en una gran variedad de dispositivos y resoluciones.
- Los cambios en el diseño pueden realizarse directamente editando un archivo escrito en XML.
- Está disponible para las principales plataformas Microsoft Windows, macOS y GNU/Linux y es completamente gratuito.

5.1.1 ESTRUCTURA PROYECTOS ANDROID STUDIO

Cualquier proyecto creado en Android Studio está estructurado en una serie de carpetas y archivos que se muestran en la ventana de herramientas situada en su lado izquierdo. Hay que diferenciar la carpeta de proyecto, que es la carpeta principal que engloba a todos los elementos del proyecto, de las carpetas denominadas módulos. Los módulos representan aplicaciones distintas, componentes o diferentes versiones de una misma aplicación. Lo habitual es trabajar con un proyecto que contiene un solo modulo correspondiente a la aplicación que se está desarrollando. El módulo principal se estructura a su vez en dos niveles. Dentro del primer nivel se encuentran las carpetas y archivos relacionados con el código fuente de la aplicación, sus recursos y configuración mientras que en el segundo nivel la carpeta External References contiene las referencias a las librerías de Android y Java necesarias para llevar a cabo la compilación.

El código fuente regula el comportamiento lógico de la aplicación y en función del lenguaje escogido por el desarrollador estará escrito en Java o Kotlin estructurándose según las reglas y normas de este tipo de lenguajes. Los recursos son archivos escritos en formato XML que definen el aspecto de las ventanas principales, de los textos o los menús. En las carpetas de recursos también pueden localizarse las imágenes o archivos de sonido que se vayan a utilizar.

5.1.2 INTERFAZ GRAFICA

La interfaz gráfica de Android Studio está dividida en varias barras o ventanas.

La ventana del editor es el área donde se escribe el código de la aplicación. El aspecto de la ventana de código varía en función del archivo que se esté tratando, si se trata de un archivo de diseño en XML o de programación en Java. La ventana de navegación muestra la ruta de los archivos que están activos durante el desarrollo del programa. La barra de herramientas permite realizar múltiples acciones entre ellas la ejecución del código.

Los botones de herramientas permiten acceder a las ventanas de herramientas que contienen tareas específicas como la búsqueda o acceder al árbol de carpetas descrito en el apartado anterior. La barra de estado muestra el estado del IDE y de la aplicación lanzando advertencias en caso de que se produzcan problemas o errores.

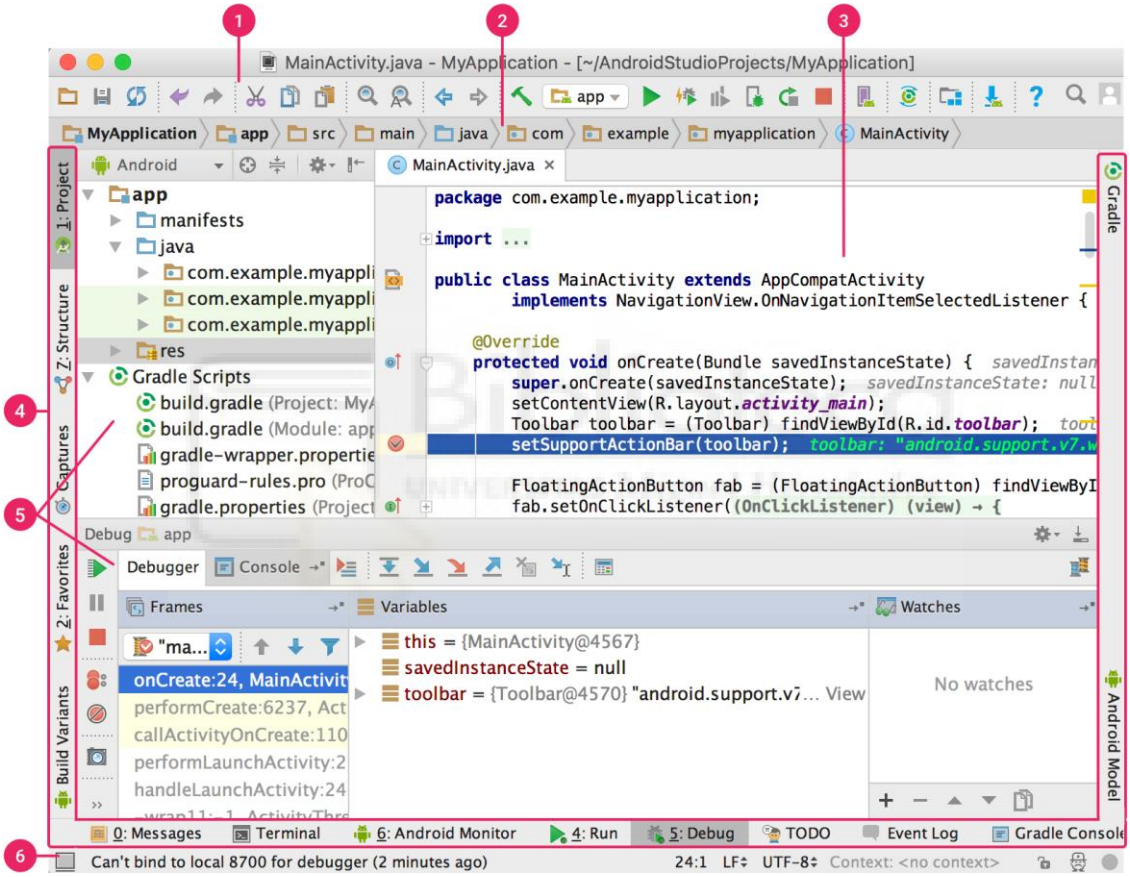


Fig. 16 Ventana principal de Android Studio [27]

5.2 LENGUAJES UTILIZADOS

Android es un sistema operativo desarrollado por Google inicialmente para dispositivos móviles con pantalla táctil, aunque hoy día está presente en todo tipo de dispositivos como televisiones, relojes inteligentes e incluso vehículos. Desde su lanzamiento y hasta hoy Java ha sido el lenguaje oficial para el desarrollo de aplicaciones, aunque a partir del año 2017 también es posible programar empleando el lenguaje Kotlin creado con el objetivo de reemplazar a Java en un futuro próximo.

Java es un lenguaje de programación creado en el año 1995 por la empresa Sun Microsystems. Es un lenguaje orientado a objetos (POO) con el objetivo de que los desarrolladores pudiesen ejecutar sus programas en cualquier dispositivo sin necesidad de tener que reescribirlos cada vez. Ofrece las funcionalidades y características de otros lenguajes como C, pero es menos confuso y sencillo, además dispone de una gran cantidad de bibliotecas y herramientas para el desarrollo. Es un lenguaje que hace un uso muy eficiente de la memoria del sistema ya que elimina objetos que no están referenciados correctamente. Es seguro y robusto ya que provee de herramientas de comunicación segura que protegen la privacidad de los datos y también permite la ejecución de tareas simultaneas en un mismo programa mejorando el rendimiento y la velocidad. Todas estas características lo han convertido en uno de los lenguajes más populares y utilizados de los últimos 15 años.

En Android, Java es el encargado de regular la parte lógica de las aplicaciones mientras que la parte gráfica correspondiente al aspecto de las distintas pantallas se hace a través del lenguaje de marcas XML.

Las siglas XML hacen referencia a *Extensible Markup Language* o lenguaje de marcas extensible. Se trata de un lenguaje pensado para estructurar e intercambiar información entre aplicaciones. Esto se hace a través de ficheros de texto donde la información se estructura o delimita mediante pares de etiquetas. Se trata de un sistema de intercambio de información abstracto independientemente de la plataforma o el sistema operativo utilizados. Podría definirse más que como un lenguaje como un conjunto de reglas para tratar la información. Su estructura es fácilmente legible ya que utiliza una sintaxis simple. También permite ampliar y actualizar continuamente la información que contiene añadiendo nuevas marcas o etiquetas siempre que se mantenga la estructura y reglas.

5.3 ESTRUCTURA (ACTIVIDADES)

Cualquier aplicación desarrollada en Android se compone de múltiples pantallas que contienen la información y los elementos que permiten al usuario interactuar con ella como botones, imágenes, textos, menús, etc.

Cada una de estas pantallas es a su vez una activity o actividad. Una actividad contiene las partes lógica y grafica que regulan su comportamiento y apariencia. Como ya se ha comentado en los apartados anteriores esto se hace por medio de archivos escritos en Java y XML. Cuando se habla de actividades es conveniente hacerlo también de su ciclo de vida. El ciclo de vida de una actividad se refiere a los diferentes estados por los que esta pasa durante la ejecución de la aplicación. El principal estado de una actividad es cuando esta activa, es decir, cuando está en primer plano y el usuario puede interactuar con ella. En cada uno de los estados pueden llevarse a cabo distintas funciones como entrar y salir de la aplicación o dejarla en segundo plano, esto se hace a través de una serie de métodos que se describen a continuación:

- Método **onCreate()**: Lleva a cabo la inicialización de la interfaz de usuario y de las estructuras de datos que componen a la actividad. También se

inician tareas y subprocesos que cumplen funciones específicas dentro de la aplicación. Este método solo se ejecuta una vez durante el ciclo de vida.

- Método **onStart()**: Se ejecuta a continuación del método **onCreate()**, en ese momento la actividad está a punto de volverse visible al usuario.
- Método **onResume()**: Se ejecuta después del método **onStart()** y es en este momento cuando la actividad se vuelve visible al usuario de forma que puede interactuar con ella a través de los elementos mostrados en la pantalla. Este método no solo se ejecuta al inicio de la actividad, sino también cuando se vuelve a ella después de haberla minimizado.
- Método **onPause()**: Se ejecuta cuando la actividad deja de estar en primer plano, bien porque el usuario la ha minimizado o porque se muestra otra ventana de dialogo de forma que la actividad pasa a mostrarse en el fondo. Cuando se ejecuta este método ciertas tareas que hayan sido definidas dentro del mismo se detienen momentáneamente a la espera de reanudarse de nuevo. Cuando este método es ejecutado la actividad puede tomar dos caminos en función de la interacción del usuario. La aplicación puede volver de nuevo al primer plano llamando al método **onResume()** o dejar de ser visible mediante el método **onStop()**.
- Método **onStop()**: La actividad deja de ser visible para el usuario y se detienen todas las tareas que se estén ejecutando. Al mismo tiempo se liberan los recursos empelados por esas tareas. Una vez que la actividad ha sido detenida puede volver a reanudarse a través del método **onRestart()** o ser destruida definitivamente con el método **onDestroy()**.
- Método **onRestart()**: Se ejecuta después del método **onStop()** en caso de que el usuario decida reanudar la actividad en primer plano.
- **onDestroy()**: Se ejecuta cuando el usuario da por finalizada su interacción con la actividad o porque el sistema detecta un cambio en la configuración como podría ser la rotación del dispositivo. En este momento se liberan todos los recursos que se estén utilizando por la actividad.

En general, cualquier actividad puede pasar por cuatro estados distintos. Inexistente, pausada, detenida o activa. Dependiendo del estado en el que se encuentre empleará recursos de la memoria del sistema y será o no visible para el usuario.

ESTADO	USO DE MEMORIA	VISIBLE	EN PRIMER PLANO
Inexistente	No	No	No
Pausada	Si	Parcialmente	No
Detenida	Si	No	No
Activa	Si	Si	Si

En la siguiente figura se muestra de forma esquemática cada uno de los distintos estados por los que pasa una actividad durante su ciclo de vida en la ejecución de una aplicación en Android y los métodos ejecutados en cada uno de ellos.

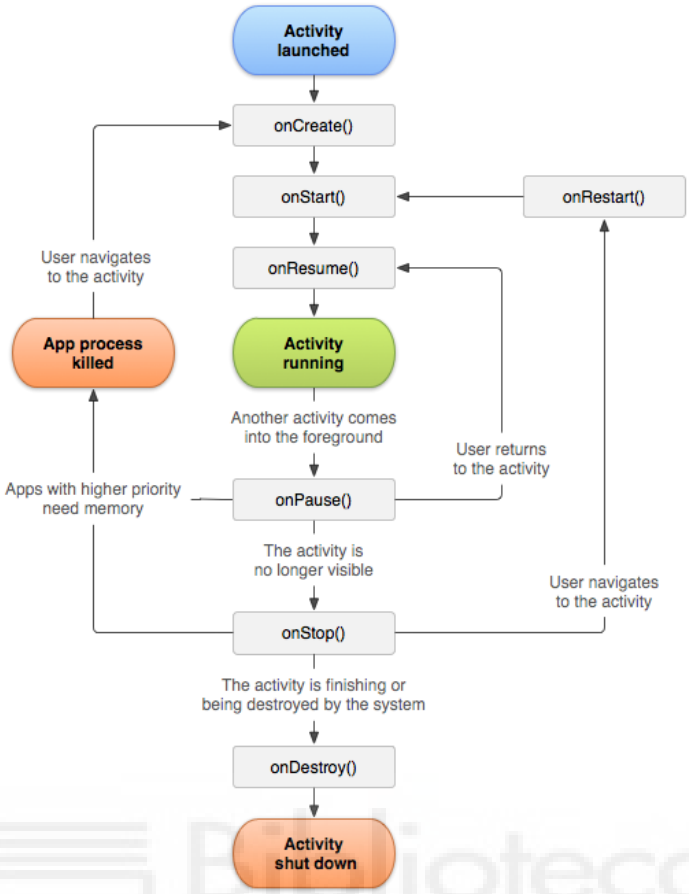


Fig. 17 Ciclo de vida de una actividad en Android [27]

5.4 PERMISOS Y MANIFEST

Android cuenta con un sistema de seguridad que evita que las aplicaciones puedan acceder a determinados recursos del sistema a menos que se las autorice específicamente para ello informando al mismo tiempo al usuario. Para acceder a esos recursos es necesario que durante el desarrollo se indique en un archivo denominado *manifest* o manifiesto los permisos de los que dispone la aplicación. Existe una amplia variedad de permisos que puede consultarse en la documentación oficial de Android. Algunos ejemplos de los más habituales son los siguientes:

- INTERNET: Permite a las aplicaciones establecer la conexión y acceder a internet.
- CAMERA: Permite a la aplicación acceder y usar la cámara del dispositivo.
- ACCESS_COARSE_LOCATION y ACCESS_FINE_LOCATION: estos dos permisos hacen referencia a la localización del dispositivo a través de los sistemas GPS o de telefonía móvil y wifi.
- WRITE_EXTERNAL_STORAGE: Permite el acceso y modificación de archivos de la memoria externa del dispositivo.
- BLUETOOTH: Permite a la aplicación utilizar el sistema bluetooth para enviar y recibir información.

Como se ha comentado, los permisos necesitan ser declarados en el archivo Androidmanifest.xml. Este archivo se crea al iniciar un nuevo proyecto en Android Studio y es uno de los más importantes del sistema ya que en él se declaran además de los permisos una serie de parámetros y condiciones que permiten el funcionamiento correcto de la aplicación. En él están contenidas las referencias a las bibliotecas y dependencias que serán usadas. Está escrito en XML de forma que la información está estructurada mediante etiquetas. Algunas de ellas se definen a continuación:

- **<manifest>**: Es la etiqueta raíz del archivo **Androidmanifest** que a su vez encierra al resto de elementos. Contiene dos atributos principales. **Xmlns** que define el espacio de nombres de Android y **package** que define el nombre del paquete de Java de la aplicación y que la identifica de forma única, distinguiéndola de otras en cualquier dispositivo o en la tienda de aplicaciones Google Play.

- **<application>**: Dentro de esta etiqueta se introducen todos los atributos de los componentes de la aplicación como las actividades, servicios y bibliotecas. Algunos de los atributos que contiene en su interior son el nombre y la descripción de las funcionalidades, los identificadores para el icono, la ruta de los temas de las distintas pantallas o los permisos en caso de que otras aplicaciones hagan uso de partes de la misma.

- **<uses-sdk>**: En esta etiqueta se especifica la compatibilidad de la aplicación con una o varias versiones del sistema operativo. Sus dos atributos principales son **minSdkVersion** donde se indica el nivel mínimo de la Api de Android necesario para ejecutar la aplicación y que evita que se instale en versiones inferiores. Y **targetSdkVersion** que determina en qué nivel de API fue desarrollada la aplicación. Este último atributo es opcional.

- **<uses-permission>**: Dentro de esta etiqueta se indican los permisos de la aplicación comentados anteriormente y que permiten para acceder y hacer uso de determinados recursos del sistema que inicialmente están restringidos.

- **<activity>**: Esta contenida dentro de la etiqueta **application**. Es necesaria para poder lanzar cada una de las actividades que conforman la aplicación. Algunos de sus atributos son el nombre de la clase que implementa a la actividad, es decir, la ruta completa de la misma o la orientación donde se indica si la interfaz gráfica se mostrará al usuario en forma vertical o horizontal. Muchos de estos atributos son opcionales y tienen valores por defecto, aunque no sean declarados.

Debido al aumento de la complejidad y tamaño de las aplicaciones actuales algunos de los parámetros declarados en el manifiesto han quedado anticuados teniendo que declararse dentro de la herramienta Gradle integrada en Android Studio. Gradle es un sistema que permite automatizar la compilación de algunas partes del código mediante scripts cuando se cumplen ciertas condiciones. Esto

permite reducir el tamaño y la complejidad de las aplicaciones. Un ejemplo ilustrativo del uso de Gradle sería el caso de una aplicación que contase con dos versiones, una gratuita y otra de pago. Gradle puede usarse para que en función del contenido desbloqueado por el usuario se compilasen solo aquellas funciones para las que tiene acceso sin necesidad de hacerlo con las demás.

Fig. 18 Ejemplo archivo manifest en Android Studio

5.5 ARCHIVOS XML Y JAVA

Como se indicó en el primer apartado, cualquier proyecto en Android Studio está organizado en una serie de carpetas que contienen los archivos con el código que regula el comportamiento de los aspectos lógico y gráfico de las aplicaciones.

Los archivos que tienen que ver con la apariencia de las distintas pantallas se localizan dentro de la carpeta res o recursos del árbol de proyecto y están escritos en XML. El archivo *Strings* contenido en la subcarpeta *values* almacena todos los textos que aparecerán en las vistas, menús y botones de la aplicación. Dentro de este archivo cada cadena de texto se idéntica con la etiqueta **<String>** y un nombre como atributo. Gestionar los textos de la aplicación de esta manera puede resultar muy útil en el caso de aplicaciones multilenguaje ya que pueden crearse distintos archivos con los textos en diferentes idiomas y seleccionar el adecuado según la necesidad.

Los archivos dentro de la subcarpeta layout controlan el diseño de las actividades. En él están representados todos los widgets o contenedores que aparecen en las pantallas de la aplicación. La opción de Android Studio que permite añadir directamente los componentes arrastrándolos a las pantallas facilita el desarrollo ya que al hacerlo el código XML se genera automáticamente

dentro de este archivo. Una etiqueta o nodo principal determina como se organizan los widgets dentro de una actividad. Por defecto la etiqueta raíz es **<RelativeLayout>** que indica que los widgets se distribuirán en la pantalla según posiciones relativas respecto de otros elementos de forma que se adaptarán a cualquier tipo de pantalla. Dentro de esta etiqueta pueden definirse multitud de atributos como los tamaños de los contenedores y textos de las actividades o el *padding* que es el espacio lateral que existe entre contenido de contenedor y sus límites.

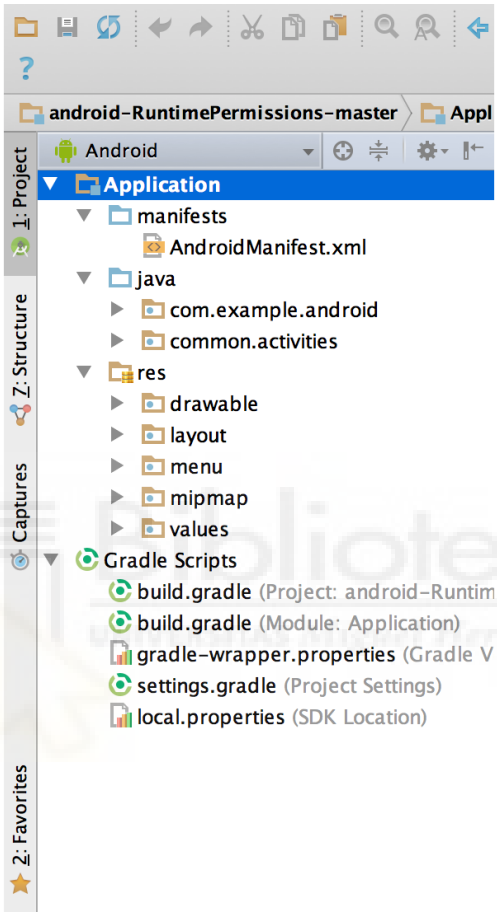


Fig. 19 Estructura carpetas del árbol de proyectos de Android Studio [27]

Los archivos relacionados con la parte lógica de la aplicación se encuentran dentro de la carpeta Java de árbol de proyecto de Android Studio. Existe al menos un archivo Java por cada actividad, aunque pueden existir tantos como sean necesarios. Estos contienen código adicional y pueden ser llamados para realizar tareas específicas dentro de aplicación.

Java es un lenguaje orientado a objetos (POO), esto significa que los programas o aplicaciones se construyen a base de una serie de componentes denominados objetos que cumplen una función específica dentro del programa y que pueden comunicarse entre sí. Este tipo de programación permite estructurar el código en módulos y reutilizar partes o la totalidad del mismo para otros propósitos. La base fundamental de POO son las clases. Estas son plantillas que sirven para crear los objetos. Las clases se dividen en variables y métodos. Un método no es más que un conjunto de instrucciones para realizar una tarea, esto es equivalente al

comportamiento de un objeto mientras que las variables corresponden a su estado Los métodos trabajan a su vez con los atributos o variables definidos dentro de la clase del tipo de objeto al que pertenecen. Puede ser que este requiera de unos parámetros de entrada o que devuelva datos a la salida. La sintaxis general de una clase en Java es la siguiente.

```
class NombredelaClase{

    //Constructor

    //variables

   //métodos

}
```

El constructor no es más que un conjunto de instrucciones que permiten inicializar los atributos de la clase a partir de la que se crea un objeto. Tienen el mismo nombre que la propia clase y no arrojan valores de salida.
A partir de una clase pueden crearse infinidad de objetos que comparten las mismas características, aunque algunos de sus atributos pueden diferir en su valor.

La estructura general del archivo java de una actividad en Android Studio es la siguiente:

```
package com.example.myapplicationaaaaaa;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

En la primera línea **package** o paquete es un identificador que sirve para localizar a un conjunto de clases agrupadas en Java. También sirve para establecer el tipo de acceso a esas clases desde el exterior. Esto está relacionado con el concepto de encapsulamiento de la POO.

A continuación, se indican las sentencias **import** que importan las librerías básicas de Android y Java que permiten crear las aplicaciones y tras ellas una clase principal denominada **MainActivity** que contiene los métodos relacionados

con el ciclo de vida de la actividad como el método **onCreate()**. Todo lo que se declare dentro de este método se ejecutará automáticamente cuando se lance la actividad. La estructura de funcionamiento de este archivo es similar a la de cualquier otro programa escrito en Java. Pueden declararse variables, clases o interfaces. Establecer el ámbito de estos, públicos o privados, indicar con comentarios aclaraciones relativas al código, etc.

5.6 LIBRERÍAS Y APIS

Las librerías son fragmentos de código ya escrito a los que se puede recurrir dentro del propio código del desarrollador con el objetivo de hacer más rápido y fácil la programación de algunas funcionalidades. Una librería está compuesta por clases, funciones, variables, etc. Un ejemplo del uso de librerías en el siguiente proyecto es la conexión de la aplicación y el tratamiento de los datos descargados de la base de datos externa PVGIS. Para hacerlo es necesario importar varias librerías relacionadas con los protocolos de conexión como HTML o JSON en el caso de la extracción y procesamiento de los datos una vez descargados. Recurrir a estas librerías permite no tener que programar desde cero todo lo relacionado con estas funcionalidades. Simplemente se importan las librerías y se hace uso de los fragmentos de código programados para tal fin. Esto no solo ahorra tiempo, sino que permite realizar fácilmente tareas que de otra forma resultaría muy compleja.

Una API es un conjunto de instrucciones sobre cómo usar una librería o servicio. Contiene los protocolos y definiciones para poder realizar la comunicación entre distintas aplicaciones. Para el proyecto actual se van a utilizar las APIs de Google Maps y PVGIS para poder acceder a los datos de ubicación e irradiación de las instalaciones que se van a dimensionar. Las librerías y APIs pueden encontrarse en la carpeta *External Libraries* del árbol de proyecto de Android Studio.

5.6.1 API PVGIS

El acceso a los datos de PVGIS se realiza por medio de una API web. Para ello es necesario componer, mediante una serie de parámetros, una URL que conduce al lugar donde están alojados los datos a los que se quiere acceder. Algunos de esos parámetros son de uso obligatorio por lo que es imprescindible especificarlos, mientras que otros contienen valores por defecto que no es necesario indicar a menos que se quieran modificar. Estos pueden consultarse en la documentación de PVGIS dentro del apartado *Non-interactive service* [28]. Los datos a los que se puede acceder desde la API de PVGIS se citan a continuación:

- Rendimiento de sistemas fotovoltaicos conectados a red
- Rendimiento de un sistema fotovoltaico con seguimiento
- Rendimiento de un sistema fotovoltaico aislado
- Datos mensuales de irradiación
- Datos promedio de irradiancia
- Datos horarios de radiación
- Año meteorológico típico

En este proyecto únicamente se hará uso de los valores de irradiación media mensual. En la siguiente figura se muestra el aspecto general con los parámetros mínimos de la URL para acceder a los datos de irradiación media mensual de PVGIS.

```
https://re.jrc.ec.europa.eu/api/MRcalc?lat=45&lon=8&horirrad=1
```

Esta URL se compone en primer lugar, del protocolo y el nombre del host de PVGIS seguida de la palabra **api** que especifica una ubicación dentro del servidor donde están alojados los datos. El valor **MRcalc** hace referencia a la herramienta de cálculo de la irradiación media mensual mientras que **lat** y **lon** a los valores de latitud y longitud respectivamente. **horirrad=1** indica que se muestren los datos de la irradiación recibida sobre el plano horizontal. En este proyecto la estructura de la URL se ha modificado añadiendo los parámetros adicionales **optrad=1** que indica que en la entre la información mostrada tienen que aparecer los valores de irradiación recibida sobre el plano de inclinación óptimo y **outputformat=json** para que el formato de salida de los datos sea del tipo JSON. También se ha especificado el año inicial de los datos consultados, el parámetro **startyear**. En este caso únicamente se han tomado los datos de irradiación media mensual del año 2016, el último año disponible de la base de datos.

En la siguiente captura aparece la estructura final de la URL utilizada en el proyecto para una ubicación especifica.

```
https://re.jrc.ec.europa.eu/api/MRcalc?lat=38&lon=-
0.85&horirrad=1&startyear=2016&optrad=1&outputformat=json
```

5.7 DESCRIPCIÓN DE LA VERSIÓN SIMPLE

En este apartado se describen las características y funcionamiento de la versión simple de la aplicación a través de capturas y secciones del código en Java y XML.

El objetivo principal de esta primera versión es obtener un prototipo funcional y validar las ecuaciones del modelo para dimensionar una instalación fotovoltaica aislada de la red.

5.7.1 PANTALLA INICIAL

En la pantalla inicial aparece el logo de la aplicación y un botón que permite pasar a la actividad o pantalla siguiente donde se introducirán los datos para el cálculo de la instalación.

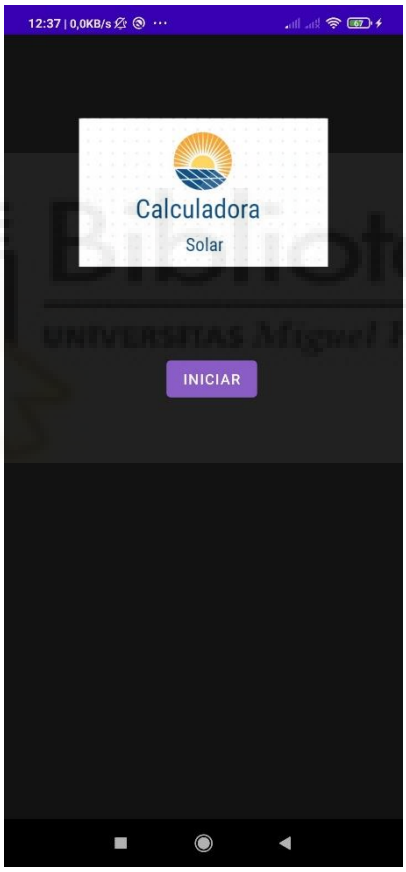


Fig. 10 Captura de la pantalla inicial de la versión simple

5.7.2 JAVA

La siguiente captura muestra el código de parte lógica de esta pantalla. En primer lugar, aparece el nombre del paquete de la aplicación seguido de las líneas que importan las clases y librerías que se van a utilizar. La clase denominada **MainActivity** identifica a la actividad de la pantalla actual respecto de las demás. Este tipo de clase se crea por defecto al crear cualquier nueva actividad y contiene al método **onCreate()** que es el encargado de la iniciarla. Dentro de la clase **MainActivity** también se ha definido el método **iniciar** que establece el funcionamiento de la aplicación cuando se pulsa el botón iniciar. Al pulsar sobre este botón pasamos a la siguiente pantalla o actividad. Esto se hace por medio de un objeto de tipo **intent** que se ha llamado iniciar y que recibe dos parámetros de entrada, el contexto de la actividad actual y el nombre de la clase principal de la actividad a la que nos queremos dirigir. Tras crear el objeto, el método **startActivity**, que recibe como parámetro al propio objeto, inicia la siguiente actividad o pantalla de la aplicación.

```
package com.example.Solcarc;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    //Método botón iniciar
    public void Iniciar (View view){
        Intent iniciar = new Intent(this, MapsActivity.class);
        startActivity(iniciar);
    }
}
```

5.7.3 XML

En la siguiente captura se muestra el código que regula el apartado gráfico de la pantalla y los elementos que aparecen en ella. Las etiquetas **Button** e **ImageView** hacen referencia a la apariencia del botón y la imagen del logo de la aplicación como sus tamaños y posiciones relativas. Dentro del botón, la línea **android:onClick="Iniciar"** relaciona la parte lógica definida en el archivo Java de la actividad con la parte gráfica. En este caso, indica que cuando se pulse el botón iniciar se ejecute el método **iniciar()** definido previamente.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="160dp"
        android:layout_marginLeft="160dp"
        android:layout_marginTop="56dp"
        android:onClick="Iniciar"
        android:text="@string/btn_iniciar"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="245dp"
        android:layout_height="208dp"
        android:layout_marginTop="52dp"
        android:contentDescription="@string/referencia"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@mipmap/logo2" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

5.7.4 PANTALLA DE DATOS Y RESULTADOS

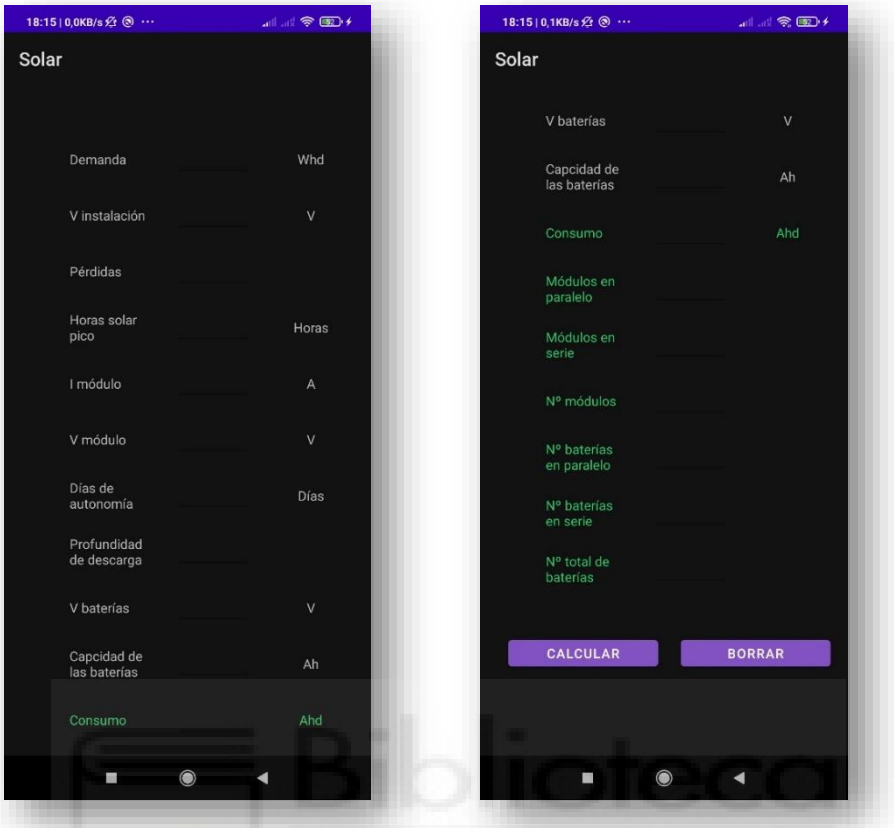


Fig. 11 Capturas de la pantalla de datos y resultados de la versión simple

En esta pantalla aparecen los parámetros que tiene que introducir el usuario para que la aplicación pueda realizar el dimensionamiento de la instalación fotovoltaica. Internamente este se llevará a cabo empleando las ecuaciones definidas en el apartado de diseño del proyecto. Esta pantalla cuenta con un *scroll* para poder desplazarse entre los distintos parámetros. En verde figuran aquellos campos donde se mostrarán los resultados una vez realizado el dimensionamiento. En la parte inferior aparecen dos botones. El botón calcular permite realizar el cálculo una vez introducidos todos los datos. Al pulsarlo, automáticamente aparecerán los resultados obtenidos en los campos habilitados para ello. El botón borrar permite reiniciar los valores de la pantalla de datos introducidos hasta ese momento.

5.7.5 JAVA

```
package com.example.solar;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

import java.net.InterfaceAddress;

public class MainActivity extends AppCompatActivity {

    private EditText et_demanda, et_vinstalacion, et_perdidas, et_consumo,
et_hsp, et_imodulo, et_vmodulo, et_rp, et_rs, et_nmodulo, et_Daut,
et_Pdesc, et_Vbateria, et_Cbateria, et_npbaterias, et_nsbaterias,
et_nbaterias;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et_demanda = (EditText)findViewById(R.id.et_demanda);
        et_vinstalacion = (EditText)findViewById(R.id.et_vinstlacion);
        et_perdidas = (EditText)findViewById(R.id.et_perdidas);
        et_consumo = (EditText)findViewById(R.id.et_consumo);
        et_hsp = (EditText)findViewById(R.id.et_hps);
        et_imodulo = (EditText)findViewById(R.id.et_imodulo);
        et_vmodulo = (EditText)findViewById(R.id.et_vmodulo);
        et_rp = (EditText)findViewById(R.id.et_Rp);
        et_rs = (EditText)findViewById(R.id.et_Rs);
        et_nmodulo = (EditText)findViewById(R.id.et_nmodulos);
        et_Daut = (EditText)findViewById(R.id.et_Daut);
        et_Pdesc = (EditText)findViewById(R.id.et_Pdesc);
        et_Vbateria = (EditText)findViewById(R.id.et_Vbateria);
        et_Cbateria = (EditText)findViewById(R.id.et_Cbateria);
        et_npbaterias = (EditText)findViewById(R.id.et_npbaterias);
        et_nsbaterias = (EditText)findViewById(R.id.et_nsbaterias);
        et_nbaterias = (EditText)findViewById(R.id.et_nbaterias);

        SharedPreferences datos = getSharedPreferences("datos",
Context.MODE_PRIVATE);
        et_demanda.setText(datos.getString("demanda", ""));
        et_vinstalacion.setText(datos.getString("vinstalacion", ""));
        et_perdidas.setText(datos.getString("perdidas", ""));
        et_hsp.setText(datos.getString("hsp", ""));
        et_imodulo.setText(datos.getString("imodulo", ""));
        et_vmodulo.setText(datos.getString("vmodulo", ""));
        et_Daut.setText(datos.getString("daut", ""));
        et_Pdesc.setText(datos.getString("pdesc", ""));
        et_Vbateria.setText(datos.getString("vbateria", ""));
        et_Cbateria.setText(datos.getString("cbateria", ""));

    }
```

En la captura anterior aparece la primera parte del código correspondiente a la parte lógica de la pantalla de datos y resultados. Se identifican en primer lugar las líneas con el nombre del paquete de la aplicación y las relacionadas con la importación de librerías, así como la clase principal con el método **onCreate()**. También se han declarado los objetos del tipo **EditText** que recogerán los datos introducidos por el usuario. Se han declarado tantos objetos de este tipo como parámetros aparecen en la pantalla incluidos los que mostraran los resultados del cálculo. En el interior del método **onCreate()**, tras las líneas por defecto que sirven para iniciar la actividad, se han escrito las líneas de código que sirven para relacionar o conectar los objetos declarados anteriormente de tipo **EditText** con la parte grafica de la pantalla. Esto se hace por medio de **findViewById** donde el atributo **id** que aparece en su interior identifica a cada uno de los widgets que aparecen en la pantalla y donde quieren mostrarse o capturar valores. Este identificador se establece dentro del archivo XML del apartado grafico de la actividad.

A continuación, se hace referencia a un objeto de la clase **SharedPreferences** llamado **datos**. Este objeto permite almacenar los valores introducidos por el usuario en la pantalla y mostrarlos en inicios posteriores de la aplicación.
En caso de no existir valores almacenados, los campos de la pantalla de datos y resultados aparecerán en blanco. Esto se consigue haciendo uso de los métodos **setText()** que se escribe a continuación del nombre establecido para los **EditText** correspondientes y **getString()** del objeto datos que es recibido como parámetro.

```
public void Calcular (View view){

    String demanda_string = et_demanda.getText().toString();
    String vinstlacion_string = et_vinstalacion.getText().toString();
    String perdidas_string = et_perdidas.getText().toString();
    String hsp_string = et_hsp.getText().toString();
    String imodulo_string = et_imodulo.getText().toString();
    String vmodulo_string = et_vmodulo.getText().toString();
    String Daut_string = et_Daut.getText().toString();
    String Pdesc_string = et_Pdesc.getText().toString();
    String Vbateria_string = et_Vbateria.getText().toString();
    String Cbateria_string = et_Cbateria.getText().toString();

    Double demanda_Double = Double.parseDouble(demanda_string);
    Double vinstalacion_Double = Double.parseDouble(vinstlacion_string);
    Double perdidas_Double = Double.parseDouble(perdidas_string);
    Double hsp_Double = Double.parseDouble(hsp_string);
    Double imodulo_Double = Double.parseDouble(imodulo_string);
    Double vmodulo_double = Double.parseDouble(vmodulo_string);
    Double Daut_double = Double.parseDouble(Daut_string);
    Double Pdesc_double = Double.parseDouble(Pdesc_string);
    Double Vbateria_double = Double.parseDouble(Vbateria_string);
    Double Cbateria_double = Double.parseDouble(Cbateria_string);
    //Operaciones
    //Consumo Ahd
    Double consumo_Double =
((demanda_Double/vinstalacion_Double))/(perdidas_Double);
    String consumo_String = String.valueOf(consumo_Double);
```

```
    //MÓDULOS FOTOVOLTAICOS
    //E producida por el modulo fotovoltaico
    Double e_modulo = 0.9*imodulo_Double*hsp_Double;
    //Numero modulos en paralelo
    Double np_double = consumo_Double/e_modulo;
    int np_int = (int)Math.ceil(np_double);
    String np_string = String.valueOf(np_int);
    //Numero de modulos en serie
    Double ns_double = vinstalacion_Double/vmodulo_double;
    int ns_int = (int)Math.ceil(ns_double);
    String ns_string = String.valueOf(ns_int);
    //Numero total de modulos
    int nmodulos_int = np_int*ns_int;
    String nmodulos_string = String.valueOf(nmodulos_int);

    //BATERÍAS
    //Energía que se debe almacenar en la batería
    Double Ebateria = (consumo_Double*Daut_double)/(Pdesc_double);

    //Nº Baterías en paralelo
    Double npbateria = Ebateria/Cbateria_double;
    int npbateria_int = (int)Math.ceil(npbateria);
    String npbateria_string = String.valueOf(npbateria_int);

    //Nº baterías en serie
    Double nsbateria = vinstalacion_Double/Vbateria_double;
    int nsbateria_int = (int)Math.ceil(nsbateria);
    String nsbateria_string = String.valueOf(nsbateria_int);

    //Nº total de baterías
    int nbaterias_int = npbateria_int * nsbateria_int;
    String nbaterias_string = String.valueOf(nbaterias_int);


    et_rp.setText(np_string);
    et_rs.setText(ns_string);
    et_consumo.setText(consumo_String);
    et_nmodulo.setText(nmodulos_string);
    et_npbaterias.setText(npbateria_string);
    et_nsbaterias.setText(nsbateria_string);
    et_nbaterias.setText(nbaterias_string);

    //Almacenamiento de datos
    SharedPreferences datos = getSharedPreferences("datos",
Context.MODE_PRIVATE);
    SharedPreferences.Editor Obj_editor = datos.edit();
    Obj_editor.putString("demanda", et_demanda.getText().toString());
    Obj_editor.putString("vinstalacion",
et_vinstalacion.getText().toString());
    Obj_editor.putString("perdidas", et_perdidas.getText().toString());
    Obj_editor.putString("hsp", et_hsp.getText().toString());
    Obj_editor.putString("imodulo", et_imodulo.getText().toString());
    Obj_editor.putString("vmodulo", et_vmodulo.getText().toString());
    Obj_editor.putString("daut", et_Daut.getText().toString());
    Obj_editor.putString("pdesc", et_Pdesc.getText().toString());
    Obj_editor.putString("vbateria", et_Vbateria.getText().toString());
    Obj_editor.putString("cbateria", et_Cbateria.getText().toString());
    Obj_editor.commit();
```

En la sección de código anterior aparece el método definido para el botón calcular. En primer lugar, se declaran las variables de tipo **Double** y **String** que servirán para almacenar los resultados obtenidos por las ecuaciones y para mostrarlos en la pantalla respectivamente. Tras la declaración de variables se han escrito, en el formato adecuado para el lenguaje de programación, las ecuaciones del modelo de cálculo seleccionado y que pueden consultarse en el apartado de diseño. Por medio de estas ecuaciones se calcula el número total, en serie y en paralelo de captadores fotovoltaicos y baterías. Los resultados se muestran en pantalla mediante el método **setText()** disponible para los objetos de tipo **EditText** definidos al inicio. En último lugar mediante un objeto del tipo **editor** y el método **putString()** se almacenan los valores obtenidos dentro del objeto **datos** de la clase **SharedPreferences**.

En el siguiente fragmento de código se muestra el método definido para el botón borrar. Mediante los métodos **getText()** y **clear()** de los **EditText** pueden borrarse los valores introducidos por el usuario.

```
    public void Borrar (View view){
        et_demanda.getText().clear();
        et_vinstalacion.getText().clear();
        et_perdidas.getText().clear();
        et_hsp.getText().clear();
        et_imodulo.getText().clear();
        et_vmodulo.getText().clear();
        et_Daut.getText().clear();
        et_Pdesc.getText().clear();
        et_Vbateria.getText().clear();
        et_Cbateria.getText().clear();
    }
}
```

5.7.6 XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:layout_editor_absoluteY="60dp">

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="100dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/tv_demanda"
            android:layout_width="90dp"
            android:layout_height="40dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentBottom="true"
            android:layout_marginStart="70dp"
            android:layout_marginLeft="70dp"
            android:layout_marginBottom="0dp"
            android:gravity="center|start"
            android:text="@string/tv_demanda" />

        <EditText
            android:id="@+id/et_demanda"
            android:layout_width="80dp"
            android:layout_height="40dp"
            android:layout_alignTop="@+id/tv_demanda"
            android:layout_marginStart="24dp"
            android:layout_marginLeft="24dp"
            android:layout_marginTop="0dp"
            android:layout_toEndOf="@+id/tv_demanda"
            android:layout_toRightOf="@+id/tv_demanda"
            android:autofillHints=""
            android:ems="10"
            android:hint="@string/hint"
            android:inputType="numberDecimal"
            android:textSize="14sp" />
```

En esta sección de código se muestra una captura del contenedor **ScrollView** que permite al usuario desplazarse de forma vertical por la pantalla de datos y resultados. En su interior están contenidas el resto de los elementos que aparecen en la vista gráfica. Tan solo aparecen los primeros de ellos, no se ha incluido la totalidad del código debido al elevado número de líneas de programación ya que superan las setecientas, aunque sus parámetros y funcionamiento es idéntico a lo mostrado en otras secciones similares. Los elementos contenidos dentro de un **RelativeLayout** ocupan posiciones relativas unos respecto de otros de forma que el orden de estos no es especialmente importante. Sin embargo, los objetos colocados en el interior de un **LineaLayout** se disponen unos a continuación de los otros en función del valor definido en el atributo **orientation** que puede ser horizontal o vertical. En este caso se han empleado los **LinearLayouts** para crear una serie de filas dispuestas verticalmente. Estas filas a su vez contienen un **RelativeLayout** donde se distribuyen los textos y widgets de cada uno de los parámetros de la pantalla.

5.7.7 MANIFEST

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.solar">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Solar">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Al no hacerse uso de las bases de datos externas o el sistema de geolocalización no se han definido permisos dentro del manifiesto de la aplicación. Este mantiene el estado por defecto al iniciar el proyecto en Android Studio. Entre otros parámetros aparecen el nombre, la ubicación del archivo con el tema de la aplicación o de su icono.

5.8 DESCRIPCIÓN DE LA VERSIÓN AVANZADA

5.8.1 PANTALLA INICIAL

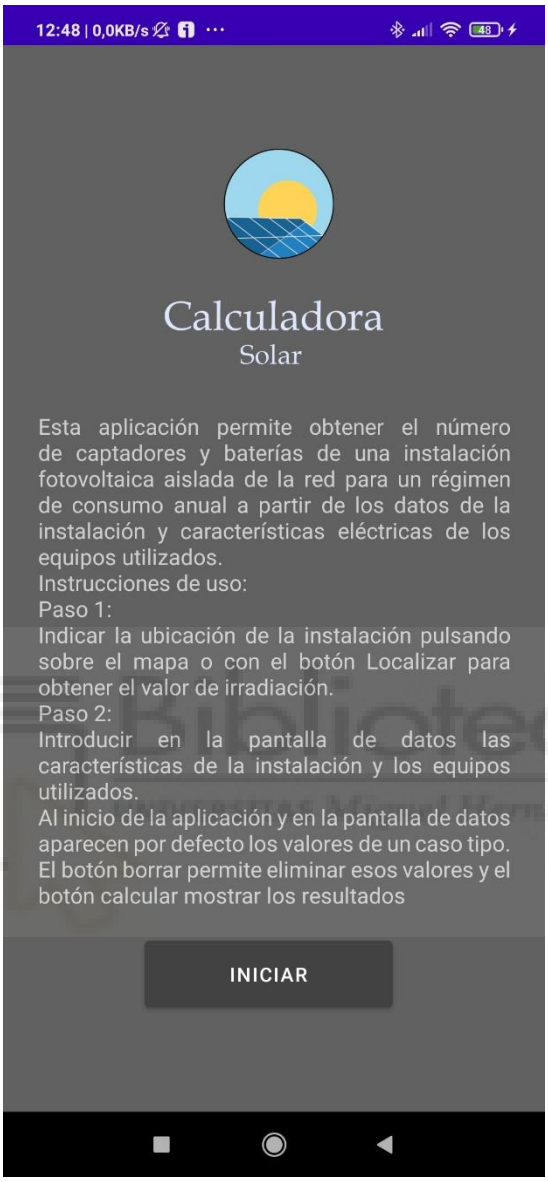


Fig. 12 Captura de la pantalla inicial de la versión avanzada

Esta pantalla es similar a la pantalla inicial de la versión simple. Contiene el logo de la aplicación y un botón de inicio que pasa a la siguiente actividad. También contiene un texto acerca del funcionamiento de la aplicación y unas breves instrucciones de uso. En los siguientes apartados se incluyen los fragmentos de código de las partes lógica y grafica.

5.8.2 JAVA

```
package com.example.Solcarc;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    //Método botón iniciar
    public void Iniciar (View view){
        Intent iniciar = new Intent(this, MapsActivity.class);
        startActivity(iniciar);
    }
}
```

5.8.3 XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="160dp"
        android:layout_marginLeft="160dp"
        android:layout_marginTop="56dp"
        android:onClick="Iniciar"
        android:text="@string/btn_iniciar"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView" />
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="245dp"
        android:layout_height="208dp"
        android:layout_marginTop="52dp"
        android:contentDescription="@string/referencia"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@mipmap/logo2" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

5.8.4 PANTALLA DE UBICACIÓN

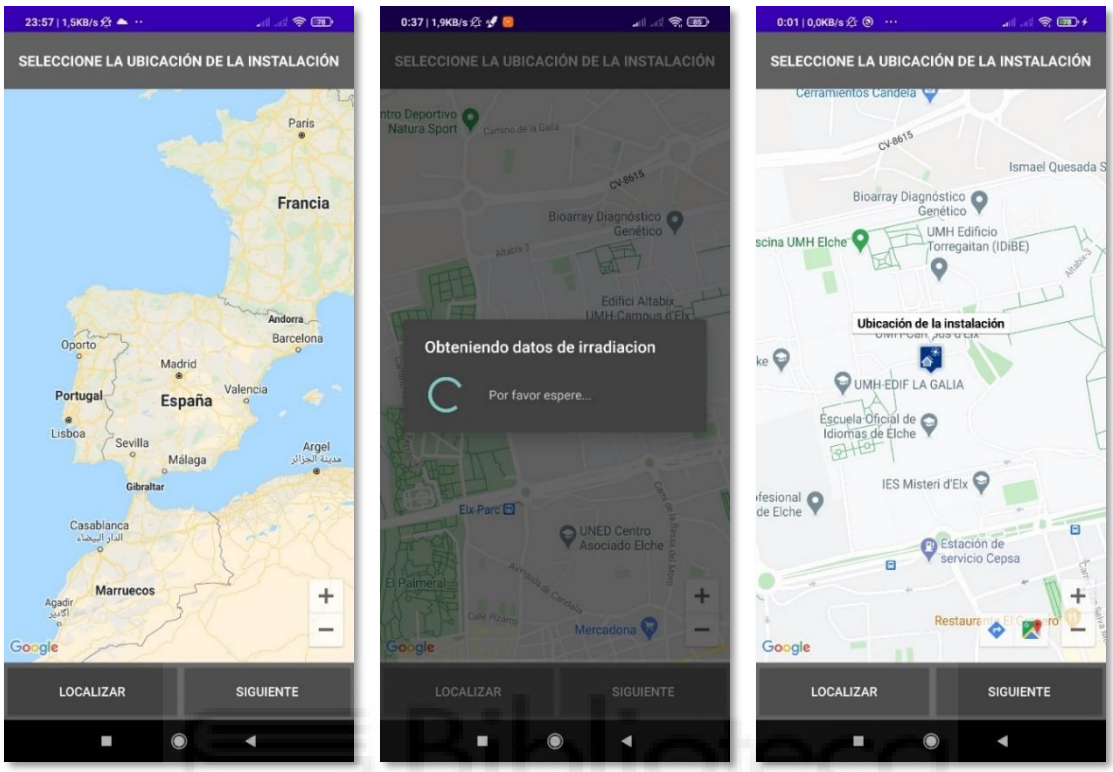


Fig. 13 Capturas de la pantalla de ubicación de la versión avanzada

En esta pantalla el usuario debe indicar el emplazamiento de su instalación fotovoltaica. La finalidad de esta actividad es obtener las coordenadas de latitud y longitud de la instalación que son necesarias para poder descargar los valores de irradiación haciendo uso de la API de PVGIS. La ubicación puede indicarse añadiendo directamente un marcador sobre el mapa al pulsar sobre la pantalla del dispositivo. El mapa tiene habilitados los controles de zoom y desplazamiento de forma que se puede navegar libremente por él. El emplazamiento también puede especificarse pulsando sobre el botón localizar. Este botón utiliza la geolocalización del dispositivo para obtener las coordenadas del usuario y añade automáticamente el marcador sobre su posición. El botón siguiente permite pasar a la siguiente actividad e iniciar el acceso y procesamiento de los datos de irradiación de PVGIS. Este proceso se realiza en segundo plano mostrando al usuario un cuadro de dialogo hasta que finaliza. En la parte superior de la pantalla se ha colocado un texto que indica al usuario que debe indicar la ubicación de su instalación.

5.8.5 JAVA

La captura de código siguiente muestra la primera parte de la clase principal de la actividad **MapsActivity**. En esta actividad se recogen los métodos y atributos relacionados con la implementación del sistema de mapas, así como de la conexión con la base de datos de PVGIS. La clase **MapsActivity** implementa dos interfaces. **OnMapReadyCallback** que sirve para cargar el mapa en la pantalla del dispositivo y **GoogleMap.OnMapClickListener** para detectar cuando el usuario interacciona con la vista donde se encuentra. Dentro del método **onCreate()** se ha declarado la variable **permissionCheck** para comprobar si el permiso **ACCESS_FINE_LOCATION** está declarado dentro del manifiesto de la aplicación y declararlo en caso contrario mediante un condicional. Este permiso es necesario para poder usar el sistema de geolocalización del dispositivo. También se han declarado los objetos **mMap** de la clase **GoogleMaps** que representa a los mapas de Google, coordenadas del tipo **LatLng** que servirá para almacenar las coordenadas extraídas del mapa y **cuadroprogreso** de la clase **ProgressDialog** para configurar el dialogo de progreso que se muestra al usuario cuando se está realizando una tarea en segundo plano.

```
public class MapsActivity extends FragmentActivity implements
OnMapReadyCallback, GoogleMap.OnMapClickListener {
    private GoogleMap mMap;
    public LatLng coordenadas;
    private ProgressDialog cuadroprogreso;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is
ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        //Comprobación de permisos
        int permissionCheck = ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION);

        if (permissionCheck == PackageManager.PERMISSION_DENIED){
            if
(ActivityCompat.shouldShowRequestPermissionRationale(this,Manifest.permiss
ion.ACCESS_FINE_LOCATION)){

            } else {
                ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
            }

        }
    }
```

En el fragmento de código siguiente están representados los métodos **onMapReady()** y **onMapclick()** contenidos en la clase principal.
El método **onMapReady()** sirve para inicializar el mapa. En su interior el valor **true** de las propiedades **setZoomControlsEnabled** y **setMapToolbarEnabled** del objeto **mMap** creado al inicio de la clase indican que sobre el mapa deben aparecer los controles de zoom y la barra de herramientas. El método **movecamera()** sirve para centrar el mapa en una ubicación especifica cuando se abre por primera vez, para ello se le deben indicar unas coordenadas y un nivel de zoom. En este caso, se le han indicado mediante el objeto de tipo **LatLng** las coordenadas de la ciudad de Madrid y se ha establecido un valor de zoom de 5, de esta forma aparecerá centrado sobre la península ibérica.

mMap.setOnMapClickListener(this) sirve para asignar al objeto **mMap** el evento **OnMapListener ()** que se activa cuando el usuario pulsa sobre la vista del mapa. El método **onMapClick()** determina lo que ocurre cuando el usuario pulsa sobre el mapa. Dentro de este el método **addMarker()** del objeto **mMap** se utiliza para crear y añadir un marcador al mapa. En sus atributos se especifica la ruta donde se encuentra la imagen del marcador y el texto que aparecerá al pulsar sobre él. El atributo **draggable(true)** indica que el marcador podrá desplazarse por la pantalla si se arrastra mientras se mantiene pulsado al mismo tiempo. El parámetro **position(latlng)** recibe las coordenadas de la posición seleccionada por el usuario al pulsar sobre la pantalla. Ese valor se guarda al mismo tiempo dentro de las variables coordenadas, latitud y longitud almacenadas dentro de la clase **Global**. Esta clase se ha creado específicamente para guardar todas las variables que van a pasar de unas actividades a otras**.** El método **mMap.clear()** sirve para borrar los marcadores que ya existen en pantalla al pulsar de nuevo sobre ella. El objetivo fundamental del uso de mapas en la aplicación es obtener las coordenadas de latitud y longitud de la instalación fotovoltaica. Este valor es imprescindible para poder descargar posteriormente los datos de irradiación de PVGIS.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    mMap.getUiSettings().setZoomControlsEnabled(true);
    mMap.getUiSettings().setMapToolbarEnabled(true);

    LatLng Madrid = new LatLng(40.4165, -3.70256);
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(Madrid, 5));

    mMap.setOnMapClickListener(this);

}

@Override
public void onMapClick(LatLng latlng) {
    mMap.clear();
    mMap.addMarker(new
MarkerOptions().position(latlng).draggable(true).title("Ubicación de la
instalación").icon(BitmapDescriptorFactory.fromResource(R.drawable.marcado
r)));
    Global.coordenadas = latlng;
    Global.latitud = Double.toString(latlng.latitude);
    Global.longitud = Double.toString(latlng.longitude);

}
```

La sección de código siguiente contiene el método definido para el botón localizar que permite registrar la ubicación actual del dispositivo móvil en caso de que se quiera ubicar la instalación fotovoltaica en la posición actual. Esto se consigue por medio de un objeto de la clase **LocationManager** y una interfaz del tipo **LocationListener**. La interfaz **locationlistener** implementa una serie de métodos que controlan los cambios de ubicación que se producen en todo momento y que son registrados por el objeto **locationmanager**. De todos ellos únicamente se va a definir el que denomina **onLocationChanged()**. Este método es llamado cuando se registra una variación de la posición actual. El resto se utilizan cuando se producen variaciones de otros parámetros que carecen de utilidad en esta aplicación por lo que se dejan en blanco. El método **onLocationChanged()** recibe como parámetro de entrada una variable de tipo **Location** que permite acceder a los valores de latitud y longitud registrados. Estos valores se almacenan en las variables correspondientes de la clase **Global**. En caso de haber indicado previamente la posición seleccionando manualmente en pantalla la ubicación de la instalación, los valores de las variables para las coordenadas almacenados serán sustituidos por los nuevos. También se eliminarán los marcadores registrados en la pantalla con el método **clear()** y se añadirá uno nuevo a la posición actual mediante **addmarker()**. Al igual que para la inicialización del mapa el método **movecamera()** centra el mapa en la posición actual, pero en este caso con un nivel de zoom de 15. El método **removeUpdates()** del objeto **locationmanager** detiene el servicio de registro de la ubicación, ya que no tiene sentido seguir haciéndolo una vez

obtenida. El método **requestLocationUpdates()** establece el tiempo y distancia mínimas para registrar una nueva ubicación.

```
public void Localizar (View view){
    LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
    LocationListener locationListener = new LocationListener() {
        @Override
        public void onLocationChanged(@NonNull Location location) {
            LatLng lat = new LatLng(location.getLatitude(),
location.getLongitude());
            Global.coordenadas = lat;
            Global.latitud = Double.toString(location.getLatitude());
            Global.longitud = Double.toString(location.getLongitude());
            mMap.clear();
            mMap.addMarker(new
MarkerOptions().position(lat).draggable(true).title("Ubicación de la
instalación").icon(BitmapDescriptorFactory.fromResource(R.drawable.marcado
r)));
            mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(lat, 15));
            locationManager.removeUpdates(this);

        }
        public void onStatusChanged(String provider, int status, Bundle
extras){}

        public void onProviderEnabled(String provider){}

        public void onProviderDisabled(String provider){}

    };

    int permissionCheck = ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION);


locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
0, 0, locationListener);

}
```

En las siguientes capturas del código se explica el funcionamiento de la descarga y procesamiento de los datos obtenidos de PVGIS. Esto se realiza por medio de una clase que se ha llamado **Obtenerirradiacion**. Esta clase hereda a su vez las propiedades de la clase **AsyncTask** que implementa varios métodos que permiten realizar tareas en segundo plano. Realizar tareas en segundo plano permite evitar bloqueos de la aplicación en momentos donde tiene que ejecutarse una parte del código que requiere mucho tiempo de procesamiento. En esos momentos la aplicación no se ha bloqueado realmente, pero si transmite esa sensación al usuario produciendo una mala experiencia. El concepto de segundo plano tiene que ver con el concepto de multitarea en programación que significa la ejecución al mismo tiempo de partes distintas del código. Cada una de esas tareas que se realizan simultáneamente recibe el nombre de hilo. El número de hilos que una aplicación puede ejecutar al mismo tiempo está directamente relacionado con las propiedades del hardware del dispositivo y el concepto de concurrencia. Lo habitual en una aplicación cuando se hace uso de

la multitarea es contar con dos hilos. Un hilo principal que suele encargarse de todas las tareas relacionadas con la interfaz gráfica que se muestra al usuario como las vistas, los botones o los menús. Y un hilo secundario que es el encargado de llevar a cabo aquellas tareas más pesadas. Este hilo secundario se ejecuta al mismo tiempo que el principal, pero es invisible para el usuario a menos que se le muestre un cuadro de dialogo indicándole el progreso de la tarea que se está llevando a cabo. En primer lugar, para comenzar la descarga de datos de irradiación y hacer uso de esa clase es necesario pulsar el botón siguiente cuyo código se muestra en la siguiente captura.

Cuando se pulsa el botón, en primer lugar, se lleva a cabo una comprobación del valor de la variable **coordenadas** contenido en la clase **Global**. Si está vacío, se muestra al usuario un mensaje que le indica que debe especificar la ubicación de su instalación. Hasta que no lo haga no puede realizarse otra tarea. Una vez ha indicado en el mapa donde está su instalación y la variable **coordenadas** contiene un valor, se crea un objeto llamado **oi** de la clase **Obtenerirradacion**. Este objeto por medio del método **execute()** inicia la ejecución de la tarea en segundo plano, en este caso el acceso y descarga de datos desde PVGIS. Este método tiene como parámetro de entrada la dirección URL donde están los datos de irradiación. Puede observarse que esa URL es una cadena de texto que tiene concatenadas dos variables que corresponden a la latitud y la longitud. Esto es así, porque estas varían cuando se indica una nueva ubicación sobre el mapa.

```
public void Siguiente (View view){
    if (Global.coordenadas != null){
        //Intent datos = new Intent(this, Datos.class);
        //startActivity(datos);
        Obtenerirradiacion oi = new Obtenerirradiacion();
        oi.execute("https://re.jrc.ec.europa.eu/api/MRcalc?lat=" + Global.latitud
+ "&lon=" + Global.longitud +
"&horirrad=1&startyear=2016&optrad=1&outputformat=json");

    } else {
        Toast.makeText(this, "Por favor, indique la ubicación de su instalación",
Toast.LENGTH_SHORT).show();
    }
}
```

En la siguiente captura aparece el código correspondiente a la clase **Obtenerirradiacion**. En su interior aparecen tres métodos, estos están relacionados con las tareas en segundo plano. De los tres, únicamente el denominado **doInBackgroud** se refiere a las instrucciones del hilo secundario, los otros dos se ejecutan en el hilo principal. El método **onPreExecute()** se ejecutará antes de que empiece la descarga de datos y sirve para configurar e iniciar el dialogo de progreso que verá el usuario. En primer lugar, se inicializa por medio del constructor, el termino **new** del código, el objeto **cuadrodeprogreso** del tipo **ProgresDialog** que se declaró al inicio de la clase principal **MapsActivity**. A continuación, los métodos **SetTitle()** y **setMessage()** establecen el nombre del cuadro de progreso y el mensaje que mostrará. **SetCancelable(false)** impide que el cuadro de dialogo se cierre al pulsar fuera del mismo, mientas que el método **show()** sirve para mostrarlo en pantalla.

El método **onPostExecute()** se ejecuta cuando finaliza la tarea en segundo plano. Únicamente se encarga de eliminar de la pantalla el cuadro de dialogo si este se está mostrando mediante el método **dismis()**. También conduce a la siguiente pantalla de la aplicación a través de un objeto de tipo **intent** que se ha nombrado **datos**.

El método **doInBackground()** es el que descarga los datos de irradiación en segundo plano desde PVGIS. Para ello emplea un objeto de la clase **ComunicaciónPVGIS**. Esta clase se ha creado específicamente en un archivo localizado dentro de la carpeta java del árbol de proyecto. En esa clase se definen todos los atributos y métodos para poder acceder y descargar la información. La información descargada desde PVGIS se almacena en la variable **pvgis_str** que la almacena en forma de texto, pero estructurado en formato **JSON**. Las siglas **JSON** hacen referencia a **JavaScript Object Notation** y se refieren a un tipo formato para almacenar e intercambiar información. La estructura general de un archivo **JSON** esta divida entre dos elementos principales. Las **keys** o llaves **{}** que contienen cadenas de texto entre comillas, y los valores o **arrays** que pueden almacenar datos de tipo numérico, booleano, objeto, texto o nulo. Cada uno de ellos puede contener a su vez otras llaves o **arrays**. En su interior la información se almacena en forma de pares clave-valor esto es, un nombre entre comillas para identificar al dato almacenado y el propio dato. Cada una de estas parejas clave-valor se distinguen unas de otras por una coma que las separa. Todos los archivos **JSON** comienzan y terminan con llaves **{}**.

Una vez descargados los datos y si estos no son nulos, se crea un objeto llamado **pvgisjson** del tipo **JSON** que permitirá extraer la información que se va a usar en la aplicación ya que, de toda la información descargada, únicamente son necesarios los datos de irradiación mensual media anual y valor de inclinación óptima. Para ello se van creando sucesivos objetos del tipo **JSONObject** y **JSONArray** hasta llegar a los valores buscados. El valor de inclinación óptima se almacena en la variable de la clase **Global ioptima** mientras que los datos de irradiación media de cada uno de los meses se almacenan dentro del vector **irradiación_valor** a través de un bucle **for**. A través de un nuevo bucle **for** se recorre el vector con los datos de irradiación buscando el valor mas bajo de todos, este se almacena dentro de la variable **irradiación_menor** de la clase **Global**. A continuación, se busca ese valor entre todos los del vector y en función de su posición dentro del mismo se determina el mes correspondiente que se almacena en la variable **mes**. Esta acción se lleva a cabo usando una estructura **switch-case**. Dado que el valor necesario en las ecuaciones es el valor medio diario mas bajo y los valores descargados de PVGIS están referidos al valor medio mensual, el valor más bajo obtenido se divide entre los días totales del mes al que corresponda. Esto se realiza por medio de una estructura condicional que compara el valor de la variable **mes**. Todas estas acciones están dentro de una estructura **try-catch** que capturara los errores que puedan producirse. Esos errores se registrarán por medio de la variable **TAG** en el **Logcat** de Android Studio.

```
private class Obtenerirradiacion extends AsyncTask<String, Void, Void>{
    private final String TAG = Obtenerirradiacion.class.getSimpleName();
    @Override
    protected void onPreExecute(){
        //super.onPreExecute();
        // Mostrar el cuadro de dialogo
        cuadroprogreso = new ProgressDialog(MapsActivity.this);
        cuadroprogreso.setTitle("Obteniendo datos de irradiacion");
        cuadroprogreso.setMessage("Por favor espere...");
        cuadroprogreso.setCancelable(false);
        cuadroprogreso.show();

    }
    @Override
    protected Void doInBackground(String ... url){

        ComunicacionPVGIS pvgis = new ComunicacionPVGIS();
        String pvgis_str = pvgis.makeServiceCall(url[0]);
        Log.e(TAG, "Respuesta de la URL: " + pvgis_str);

        if (pvgis_str != null){
            try {
                List<Double> irradacion_valor = new ArrayList<Double>();
                JSONObject pvgisjson = new JSONObject(pvgis_str);
                JSONObject outputs = pvgisjson.getJSONObject("outputs");
                JSONArray meses = outputs.getJSONArray("monthly");

                //Obtener inclinacion optima
                JSONObject inputs = pvgisjson.getJSONObject("inputs");
                JSONObject plane = inputs.getJSONObject("plane");
                JSONObject fixed = plane.getJSONObject("fixed_inclined_optimal");
                JSONObject slope = fixed.getJSONObject("slope");
                Global.ioptima = Double.toString(slope.getDouble("value"));

                for (int i = 0; i<12; i++){
                    JSONObject j = meses.getJSONObject(i);
                    irradacion_valor.add(j.getDouble("H(i_opt)_m"));
                }
                Global.irradiacion_menor = irradacion_valor.get(0);

                //Obtención del valor mas bajo de irradiación mensual
                for (int i = 0; i<irradacion_valor.size(); i++){
                    if (irradacion_valor.get(i)< Global.irradiacion_menor){
                        Global.irradiacion_menor = irradacion_valor.get(i);
                    }
                }

                int mes_int = irradacion_valor.indexOf(Global.irradiacion_menor) +
1;
                String mes = "";

                switch (mes_int){
                    case 1:
                        mes = "Enero";
                        break;
                    case 2:
                        mes = "Febrero";
                        break;
                    case 3:
                        mes = "Marzo";
                        break;
                    case 4:
                        mes = "Abril";
                        break;
```

```
                case 5:
                    mes = "Mayo";
                    break;
                case 6:
                    mes = "Junio";
                    break;
                case 7:
                    mes = "Julio";
                    break;
                case 8:
                    mes = "Agosto";
                    break;
                case 9:
                    mes = "Septiembre";
                    break;
                case 10:
                    mes = "Octubre";
                    break;
                case 11:
                    mes = "Noviembre";
                    break;
                case 12:
                    mes = "Diciembre";
                    break;
            }

            Global.mes = mes;

            if (mes == "Enero" || mes == "Marzo" || mes == "Mayo"
                    || mes == "Julio" || mes == "Agosto" || mes == "Octubre"
                    || mes == "Diciembre"){

                Global.irradiacion_menor = Global.irradiacion_menor/31;

            } else if(mes == "Abril" || mes == "Junio" || mes == "Septiembre"
                    || mes == "Noviembre"){

                Global.irradiacion_menor = Global.irradiacion_menor/30;

            } else {

                Global.irradiacion_menor = Global.irradiacion_menor/28;
            }

            Global.irradiacion_masbaja =
Double.toString(Global.irradiacion_menor);

        }catch (JSONException e){
            Log.e(TAG, "Error JSON: " + e.getMessage());
            //Vuelve al hilo principal y muestra el mensaje de error del JSON
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(getApplicationContext(),
                            "Error en el JSON: " + e.getMessage(),
                            Toast.LENGTH_LONG)
                               .show();
                }
            });
        }

    }
    return null;
}
```

```
@Override
protected void onPostExecute(Void resultado){
    if (cuadroprogreso.isShowing()){
        cuadroprogreso.dismiss();
    }
    Intent datos = new Intent(MapsActivity.this, Datoss.class);
    startActivity(datos);

}
```

```
{
        "contactos": [
                {"nombre": "Antonio",
                 "teléfono": "77777"
                 "email": "aaa@hmail.com"
                },
                {"nombre": "Antonio",
                 "teléfono": "77777"
                 "email": "aaa@hmail.com"
                },
]
}
```

Ejemplo de archivo en formato JSON

En la siguiente captura se muestra el código de la clase **ComunicacionPVGIS** y se explican algunas de sus líneas más importantes. Los objetos de esta clase son los encargados de realizar la conexión y descarga de datos desde PVGIS. El método **makeservicecall()** se encarga de todo lo relacionado con la conexión de la aplicación con el exterior. Dentro de este, el objeto **URL** contiene la dirección url de **PVGIS** donde están alojados los datos que se quieren descargar. Como la comunicación con **PVGIS** se va a realizar utilizando el protocolo **HTML**, se trata de la conexión con una web, es necesario crear un objeto de la clase abstracta **HttpURLConnection**. Este es el objeto **conexión** que se crea a partir de **url** y el método **openConnection()**. El método **setRequestMethod("GET")** especifica que la conexión que se va a realizar va a extraer datos. La conexión propiamente dicha no se realiza hasta que se ejecuta la línea **InputStream inps = new BufferedInputStream (conexion.getInputStream())** que es la que establece la conexión y almacena los datos en un objeto que se ha nombrado como **inps**.

Una vez disponemos de los datos, estos deben procesarse para poder extraer aquellos de los que hará uso la aplicación, para ello es necesario convertirlos a texto. Esto se hace a través del método **convertStreamToString()**. Este método convierte en texto la información descargada desde la **URL** de **PVGIS**. Esto lo hace utilizando el objeto **sb** de tipo **Stringbuilder**. En ambos métodos las instrucciones se han incluido dentro de estructuras **try-catch**. Estas sirven para controlar posibles errores que puedan ocurrir durante la ejecución indicando

específicamente el tipo de error producido escribiéndolo en el registro de Android Studio. Para eso se ha creado una variable especial que se llama **TAG** al principio de la clase.

```
package com.example.geolocal;
import android.nfc.Tag;
import android.util.Log;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.ProtocolException;
import java.net.URL;

public class ComunicacionPVGIS {
    private static final String TAG = ComunicacionPVGIS.class.getSimpleName();
    //Constructor
    public ComunicacionPVGIS(){
    }
    public String makeServiceCall(String urlpvgis){
        String datos = null;
        try{
            URL url = new URL(urlpvgis);
            HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
            conexion.setRequestMethod("GET");
            InputStream inps = new BufferedInputStream
(conexion.getInputStream());
            datos = convertStreamToString(inps);
        } catch (MalformedURLException e){
            Log.e(TAG, "MalformedURLException: " + e.getMessage());
        } catch (ProtocolException e) {
            Log.e(TAG, "ProtocolException: " + e.getMessage());
        } catch (IOException e) {
            Log.e(TAG, "IOException: " + e.getMessage());
        }
        return datos;
    }
    private String convertStreamToString(InputStream is){
        BufferedReader bf = new BufferedReader(new InputStreamReader(is));
        StringBuilder sb = new StringBuilder();
        String line;
        try {
            while ((line = bf.readLine()) != null){
                sb.append(line).append("\n");
            }
        } catch (IOException e){
            e.printStackTrace();
        } finally {
            try {
                is.close();
            } catch (IOException e){
                e.printStackTrace();
            }
        }
        return sb.toString();
    }
}
```

5.8.6 PANTALLA DE DATOS

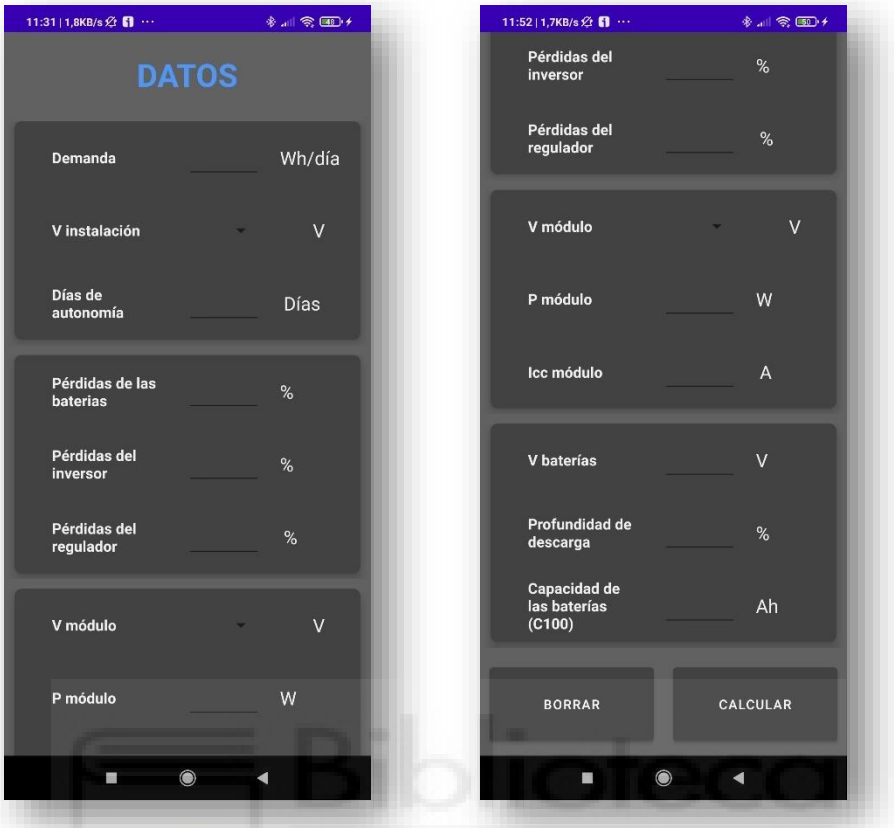


Fig. 14 Captura de la pantalla de datos de la versión avanzada

En esta pantalla el usuario tiene que introducir los datos de la instalación y de los equipos que se van a utilizar. Estos aparecen agrupados en una estructura de tarjetas. En la primera de ellas se introducen los datos relativos a la instalación como el consumo medio diario, la tensión en corriente continua a la entrada del inversor y los días de autonomía de los que contará. En la siguiente tarjeta se introducen los datos correspondientes al sistema de baterías, inversor y regulador que se introducen en forma de porcentaje. En las últimas dos tarjetas aparecen agrupados los datos de las características eléctricas de los captadores y las baterías. Deben introducirse los datos de tensión normalizada, potencia y corriente de cortocircuito de los módulos y la tensión nominal, capacidad y profundidad de descarga de las baterías. Se han limitado los valores que pueden introducirse en algunos de los campos para evitar posibles errores y cierres inesperados. No pueden introducirse valores negativos o iguales a cero en la mayoría de los campos. En el caso de las tensiones normalizadas de la instalación y los captadores estas se han limitado a los valores típicos de 12, 24 y 48 V. En el caso de las pérdidas estas están acotadas en un rango que oscila entre el cero y el veinte por ciento ya que la ecuación de perdidas deja de tener sentido fuera de este rango. En caso de que el usuario introduzca valores no validos la aplicación muestra un mensaje con información sobre los valores aceptados.

5.8.7 JAVA

En esta sección se muestran algunas partes del código correspondientes a la pantalla de datos de la aplicación. Las siguientes dos capturas representan el control de valores que el usuario puede introducir en los campos correspondientes a la demanda de la instalación y a las pérdidas del inversor. Esto se consigue por medio del método **TextWatcher**. Este método detecta cuando se han producido cambios en el texto introducido en alguno de los **EditText** establecidos como parámetros. Este método cuenta con tres eventos que se activan cuando tienen lugar alguna de sus circunstancias. En este caso solo va a hacer uso del evento **afterTextChanged** que se activa cuando se ha cambiado el texto del campo establecido como parámetro. Dentro del evento se ha incluido un condicional que controla lo que ocurre cuando se activa. En el caso del campo demanda, si el valor introducido empieza por cero o por un punto se elimina ese valor y se indica al usuario mediante un mensaje flotante que debe introducir valores superiores a uno. En el caso del parámetro inversor, se ha establecido una estructura condicional de forma que únicamente se aceptan valores comprendidos entre cero y veinte, ambos incluidos. Al igual que en el parámetro demanda, si se introduce un valor no valido se muestra un mensaje al usuario indicándole el rango aceptado. Para el resto de los parámetros se han establecido controles similares a los mostrados en las capturas.

```
et_demanda.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count,
int after) {}
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int
count) {}
    @Override
    public void afterTextChanged(Editable s) {
        if (et_demanda.getText().toString().equals("")){
        }
        else if (et_demanda.getText().toString().startsWith(".")) {
            et_demanda.setText(null);
        }
        else if(Float.parseFloat(et_demanda.getText().toString()) > 0){
        }
    }
});
```

```
et_pinversor.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count,
int after) {}
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int
count) {}
    @Override
    public void afterTextChanged(Editable s) {
        if(et_pinversor.getText().toString().equals("20")){

        } else if (et_pinversor.getText().toString().equals("")){
        }
        else if (et_pinversor.getText().toString().startsWith(".")){
            et_pinversor.setText(null);
        }
        else if (Float.parseFloat(et_pinversor.getText().toString()) >
20){
            et_pinversor.setText(null);
            Toast.makeText(Datos.this, "Las pérdidas del inversor suelen
oscilar entre el 5% y el 20%", Toast.LENGTH_SHORT).show();
        }
    }
});
```

Los valores contenidos en los campos tensión de la instalación y de los captadores fotovoltaicos se han limitado a los valores normalizados de 12, 24 y 48 V. Esto se hecho utilizando un objeto del tipo **spinner**. En la siguiente captura se muestran las líneas de código correspondientes al **spinner** de la tensión de la instalación. Las dos primeras líneas representan la declaración y conexión del **spinner** con la parte grafica de la pantalla. A continuación, se declara un vector que contiene los valores de las distintas opciones. Este vector se asocia al **spinner** por medio de un objeto **ArrayAdapter** y el método **setAdapter** que recibe como parámetros el contexto de la pantalla datos, el tipo de **spinner**, y el vector con los valores que se van a mostrar.

```
private Spinner spvinstalacion;
spvinstalacion = (Spinner)findViewById(R.id.sp_vinstalacion);
String [] tensiones_instalacion = {"","12", "24", "48"};
ArrayAdapter <String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, tensiones_instalacion);
spvinstalacion.setAdapter(adapter);
```

En la siguiente captura se indica la parte correspondiente a las ecuaciones del modelo de dimensionamiento que se ejecuta cuando se pulsa el botón calcular de la pantalla datos y el método definido para regular el comportamiento del botón borrar.

```
public void Calcular (View view){
    //Extraccion de los datos desde la UI
    String demanda_string = et_demanda.getText().toString();
    String vinstlacion_string =
spvinstalacion.getSelectedItem().toString();
    String pbateria_string = et_pbateria.getText().toString();
    String pinversor_string = et_pinversor.getText().toString();
    String pregulador_string = et_pregulador.getText().toString();
    String vmodulo_string = spvmodulo.getSelectedItem().toString();
    String pmodulo_string = et_pmodulo.getText().toString();
    String iccmodulo_string = et_iccmodulo.getText().toString();
    String Daut_string = et_Daut.getText().toString();
    String Pdesc_string = et_Pdesc.getText().toString();
    String Vbateria_string = et_Vbateria.getText().toString();
    String Cbateria_string = et_Cbateria.getText().toString();

    //Si no hay ningun campo vacio se ejecutan las operaciones
    if (!"".equals(demanda_string) && !"".equals(vinstlacion_string) &&
!"".equals(pbateria_string) && !"".equals(vmodulo_string) &&
!"".equals(pmodulo_string) && !"".equals(Daut_string) &&
!"".equals(Pdesc_string) && !"".equals(Vbateria_string) &&
!"".equals(Cbateria_string)) {

        Double demanda_Double = Double.parseDouble(demanda_string);
        Double vinstalacion_Double =
Double.parseDouble(vinstlacion_string);
        Double pbateria_Double =
(Double.parseDouble(pbateria_string))/(100);
        Double pinversor_Double =
(Double.parseDouble(pinversor_string))/(100);
        Double pregulador_Double =
(Double.parseDouble(pregulador_string))/(100);
        Double vmodulo_double = Double.parseDouble(vmodulo_string);
        Double pmodulo_double = Double.parseDouble(pmodulo_string);
        Double iccmodulo_double = Double.parseDouble(iccmodulo_string);
        Double Daut_double = Double.parseDouble(Daut_string);
        Double Pdesc_double = (Double.parseDouble(Pdesc_string))/(100);
        Double Vbateria_double = Double.parseDouble(Vbateria_string);
        Double Cbateria_double = Double.parseDouble(Cbateria_string);

        //Spinner tension instalacion
        if (vinstlacion_string.equals("12")){
            vinstalacion_Double = 12.0;
        } else if (vinstlacion_string.equals("24")){
            vinstalacion_Double = 24.0;
        } else {
            vinstalacion_Double = 48.0;
        }

        //Spinner tension modulos
        if (vmodulo_string.equals("12")){
            vmodulo_double = 12.0;
        } else if (vmodulo_string.equals("24")){
            vmodulo_double = 24.0;
        } else {
            vmodulo_double = 48.0;
        }
```

```
            //Operaciones
            //Consumo Ahd
            Double consumo_Double = demanda_Double  ;
            String consumo_String = String.valueOf(consumo_Double);

            //pérdidas
            Double kt;
            kt = (1-(pbateria_Double + pinversor_Double +
preguladorDouble + 0.10)) * (1-((0.005 * Daut_double)/(Pdesc_double)));

            //Numero total de captadores
            Double nmodulos_d =
(consumo_Double)/(0.9*pmodulo_double*Global.irradiacion_menor*kt);
            if (nmodulos_d > 0 && nmodulos_d < 1){
                nmodulos_d = Math.ceil(nmodulos_d);
            }
            else if (nmodulos_d == 0){
                nmodulos_d = nmodulos_d;
            }
            else{
                nmodulos_d = Math.rint(nmodulos_d);
            }

            //Captadores en serie
            Double nsmodulos_d = vinstalacion_Double/vmodulo_double;
            nsmodulos_d = Math.rint(nsmodulos_d);
            Global.modulosserie = String.valueOf((int)
Math.rint(nsmodulos_d));

            //Captadores en paralelo
            Double npmodulos_s = Math.rint(nmodulos_d/nsmodulos_d);
            nmodulos_d = nsmodulos_d *npmodulos_s;
            if (nmodulos_d == 0){
                nsmodulos_d = nsmodulos_d * 0;
            }
            Global.modulostotal = String.valueOf((int)
Math.rint(nmodulos_d));
            Global.modulosparalelo = String.valueOf((int)
Math.rint(npmodulos_s));
            Global.modulosserie = String.valueOf((int)
Math.rint(nsmodulos_d));

            //BATERÍAS
            //Energía que se debe almacenar en la batería
            Double Ebateria = (consumo_Double * Daut_double) /
(Pdesc_double);

            //Nº Baterías en paralelo
            Double npbateria = (Ebateria) / (Cbateria_double * kt *
vinstalacion_Double);
            if (npbateria > 0 && npbateria < 1){
                 npbateria =  Math.ceil(npbateria);
            } else if (npbateria == 0){
                npbateria = npbateria;
            } else {
                 npbateria =  Math.rint(npbateria);
            }
            Global.bateriasparalelo = String.valueOf((int)
Math.rint(npbateria));
```

```
            //Nº baterías en serie
            Double nsbateria = vinstalacion_Double / Vbateria_double;
            nsbateria =  Math.rint(nsbateria);
            Global.bateriasserie = String.valueOf((int)
Math.rint(nsbateria));

            //Nº total de baterías
            Double nbaterias = npbateria * nsbateria;
            if (nbaterias == 0){
                nsbateria = nsbateria * 0;
            }
            Global.bateriastotal = String.valueOf((int)
Math.rint(nbaterias));
            Global.bateriasserie = String.valueOf((int)
Math.rint(nsbateria));

            Intent resultados = new Intent(Datos.this, Resultados.class);
            startActivity(resultados);

            //Corriente minima del regulador
            Global.iccregulador = String.valueOf(1.25 * iccmodulo_double *
npmodulos_s);

        } else{
            Toast.makeText(this, "Por favor, rellene todos los campos",
Toast.LENGTH_SHORT).show();
        }
    }

    //Método para el botón Borrar
    public void Borrar (View view){
        et_demanda.getText().clear();
        spvinstalacion.setSelection(0);
        et_pbateria.getText().clear();
        et_pinversor.getText().clear();
        et_pregulador.getText().clear();
        spvmodulo.setSelection(0);
        et_pmodulo.getText().clear();
        et_iccmodulo.getText().clear();
        et_Daut.getText().clear();
        et_Pdesc.getText().clear();
        et_Vbateria.getText().clear();
        et_Cbateria.getText().clear();
    }


}
```

En la siguiente captura se muestra el método definido para el botón borrar. Este es similar al mostrado en la versión simple, por medio de los métodos **getText** y **clear** se borran los valores contenidos en los **EditText**. En el caso de los **spinners** estos se muestran en blanco seleccionando el valor nulo que ocupa la posición cero mediante el método **setSelection(0)**.

```
public void Borrar (View view){
    et_demanda.getText().clear();
    spvinstalacion.setSelection(0);
    et_pbateria.getText().clear();
    et_pinversor.getText().clear();
    et_pregulador.getText().clear();
    spvmodulo.setSelection(0);
    et_pmodulo.getText().clear();
    et_iccmodulo.getText().clear();
    et_Daut.getText().clear();
    et_Pdesc.getText().clear();
    et_Vbateria.getText().clear();
    et_Cbateria.getText().clear();
}
```

5.8.8 PANTALLA DE RESULTADOS

Fig. 15 Captura de la pantalla de resultados de la versión final

En esta pantalla se muestran los resultados obtenidos para el dimensionamiento de la instalación. En la parte superior se muestran el número total de captadores y baterías junto con el número de estos en paralelo y en serie. En la parte central se muestra el dato de irradiación promedio diaria del mes más desfavorable del año en horas de pico solar junto con el mes correspondiente y el valor de la inclinación optima extraído de PVGIS. En la parte inferior se muestra el valor de la corriente máxima de cortocircuito del campo fotovoltaico que es la del regulador.

5.8.9 JAVA

En la siguiente captura se muestra la parte del código correspondiente al método **oncreate()** de la pantalla de resultados. Los valores en pantalla se muestran por medio del método **setText** de los objetos **textView** y los valores de los resultados obtenidos almacenados en la clase Global.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_resultados);

    tv_numero_modulos = (TextView)findViewById(R.id.tv_numero_modulos);
    tv_modulo_serie = (TextView)findViewById(R.id.tv_modulo_serie);
    tv_modulos_paralelo =
(TextView)findViewById(R.id.tv_modulos_paralelo);
    baterias_numero = (TextView)findViewById(R.id.baterias_numero);
    tv_baterias_serie = (TextView)findViewById(R.id.tv_baterias_serie);
    tv_baterias_paralelo =
(TextView)findViewById(R.id.tv_baterias_paralelo);
    tv_irradiacion = (TextView)findViewById(R.id.tv_irradiacion);
    tv_meshsp = (TextView)findViewById(R.id.tv_meshsp);
    tv_inclinacion = (TextView)findViewById(R.id.tv_inclinacion);
    tv_regulador_corriente =
(TextView)findViewById(R.id.tv_regulador_corriente);


    tv_numero_modulos.setText(Global.modulostotal);
    tv_modulo_serie.setText(Global.modulosserie);
    tv_modulos_paralelo.setText(Global.modulosparalelo);
    baterias_numero.setText(Global.bateriastotal);
    tv_baterias_paralelo.setText(Global.bateriasparalelo);
    tv_baterias_serie.setText(Global.bateriasserie);
    tv_meshsp.setText(Global.mes);
    tv_irradiacion.setText(Global.irradiacion_masbaja);
    String ioptima = Global.ioptima + "º";
    tv_inclinacion.setText(ioptima);
    String iccregulador = Global.iccregulador + " A";
    tv_regulador_corriente.setText(iccregulador);


}
```

6 CONCLUSIONES

6.1 RESULTADOS

La finalidad de este proyecto era desarrollar una aplicación Android completamente funcional capaz de dimensionar una instalación solar fotovoltaica aislada a partir de los datos de consumo y equipos introducidos por el usuario. Para ello ha sido necesario en primer lugar, reforzar y ampliar los conocimientos relacionados con la energía solar fotovoltaica y su aplicación práctica. Se ha tenido que llevar a cabo una extensa búsqueda bibliográfica relativa a los métodos y formas de cálculo y dimensionamiento, además de un análisis de la evolución, estado actual y futuro del sector.

En cuanto a la programación, ha sido necesario adquirir una gran cantidad de conocimientos teóricos relacionados con las herramientas de desarrollo y estructura de Android, el lenguaje de programación Java, XML y su puesta en práctica. Teniendo en cuenta que al inicio del proyecto no se contaba con conocimiento alguno relacionado con la programación en Android, el resultado final ha sido bastante satisfactorio ya que además de desarrollar una aplicación que cumple con los objetivos básicos fijados al inicio del proyecto, esto se ha logrado haciendo uso de algunos recursos algo más avanzados como la geolocalización, el sistema de mapas de Google o la conexión y extracción de datos desde fuentes externas. Aunque en la mayoría de los aspectos la aplicación tenga funcionalidades similares a otras ya existentes, la implementación de la base de datos de PVGIS para la obtención de los datos de irradiación es original, ya que, aunque algunas de las aplicaciones similares existentes analizadas implementan esta función, ninguna lo hace empleando esta base de datos que además es de las más utilizadas y conocidas. El proyecto ha servido como una gran fuente de aprendizaje y puesta a prueba de los conocimientos obtenidos.

En el siguiente apartado se indican algunas mejoras que podrían implementarse en posibles actualizaciones futuras de la aplicación.

6.2 PROPUESTAS DE MEJORA DE LA APLICACIÓN

6.2.1 ESTIMACIÓN DEL COSTE ECONÓMICO DE LA INSTALACIÓN Y PERIODO DE RETORNO DE LA INVERSIÓN

En una versión posterior se podría integrar una funcionalidad que permita estimar el coste económico de la instalación a partir del precio de los equipos. Para ello será necesario añadir en la parte de la programación las ecuaciones correspondientes. También habrá que añadir un campo especifico en la parte de los datos donde el usuario pueda indicar el coste de cada uno de ellos. Si se opta por añadir una base de datos con modelos específicos, esta podría contener junto con las características eléctricas el precio.
Esta funcionalidad también debe permitir estimar el periodo de retorno de la inversión a partir del coste medio de la electricidad. Este dato podría ser introducido por el usuario directamente si dispone de él o se podría hacer una estimación del coste medio del kWh durante la programación. Una vez realizada la previsión, junto con el periodo de retorno de la inversión indicado en años, podría mostrarse de forma gráfica el ahorro anual estimado y acumulado de la instalación.

6.2.2 INTRODUCCIÓN DE DATOS DE CONSUMO

En la versión actual, el usuario introduce directamente el valor del consumo medio diario anual en Wh/día. En una futura versión, este dato podría indicarse especificando la potencia y el tiempo medio de funcionamiento diario de los receptores conectados al sistema mediante un sistema de iconos o listas. De esta forma seria más fácil y claro para el usuario determinar la demanda media de su instalación. Al disponer de la potencia de los receptores, también sería posible determinar las características del inversor o inversores necesarios.

6.2.3 ESPECIFICAR DISTINTOS PERIODOS DE FUNCIONAMIENTO

La versión actual realiza el cálculo suponiendo un consumo similar y constante durante todo el año, pero dado que el consumo de una instalación puede variar respecto de los meses de invierno y verano o bien porque el uso de esta puede estar limitado únicamente a un periodo específico del año, por ejemplo, el caso de una vivienda de uso estacional. Sería conveniente que la aplicación permitiese seleccionar y realizar el dimensionamiento únicamente en el periodo indicado.

6.2.4 INCLINACIÓN DE LOS CAPTADORES FOTOVOLTAICOS

La versión actual realiza el dimensionamiento suponiendo una inclinación optima de los módulos fotovoltaicos en función de la ubicación. Dado que en algunas ocasiones puede ser que el usuario no tenga la posibilidad de emplear esa inclinación, sería conveniente que al igual que se indican otros parámetros se pudiese especificar el valor de la inclinación e incluso comparar los resultados obtenidos con el valor óptimo.

6.2.5 BASE DE DATOS INTERNA

Podría implementarse una base de datos interna con modelos y características específicas de equipos, de forma que en vez de ser introducidas directamente por el usuario este pueda seleccionar modelos concretos de captadores o baterías.

BIBLIOGRAFÍA

[1] Hootsuite, 2020. “DIGITAL 2020 SPAIN” [En línea] Disponible en: https://wearesocial.com/es/digital-2020-espana [Accedido: Noviembre, 2020]

[2] S. Fernández "Android supera el 90% de cuota en España mientras que iOS cae por debajo del 9%, según Kantar," *Xataka*, [En línea]. Disponible en: https://www.xatakamovil.com/mercado/android-supera-90-cuota-espana-ios-cae-debajo-9-kantar [Accedido: Noviembre, 2020]

[3] M. Pareja Aparicio, *Energía solar fotovoltaica. Calculo de una instalación aislada*, 3a ed., Barcelona: Marcombo, 2016.

[4] Unión Europea. Directiva (UE) 2001/77/CE del parlamento europeo y del consejo, de 27 de septiembre de 2001, relativa a la promoción de la electricidad generada a partir de fuentes de energía renovables en el mercado interior de la electricidad

[5] Unión Europea. Directiva (UE) 2009/28/CE del parlamento europeo y del consejo, de 23 de abril de 2009, relativa al fomento del uso de energía procedente de fuentes renovables y por la que se modifican y se derogan las Directivas 2001/77/CE y 2003/30/CE.

[6] Unión Europea, COM (2006) 848 final, Programa de trabajo de la energía renovable. Las energías renovables en el siglo XXI: construcción de un futuro más sostenible, Junio 2007

[7] Eurostat, Junio 2020. *Producción e importaciones de energía* ISSN 2443-8219 [En línea] Disponible en: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Energy_production_and_imports/es [Accedido: Noviembre, 2020]

[8] European Environment Agency, EEA, *Trends and projections in Europe 2020. Tracking progress towards Europe’s climate and energy targets*, Luxembourg, Publications Office of the European Union, 2020. doi:10.2800/830157

[9] Unión Europea: Comisión Europea, *Comunicación de la Comisión al Parlamento Europeo, al Consejo, al Comité Económico y Social Europeo, al Comité de las Regiones y al Banco Europeo de Inversiones. Energía limpia para todos los europeos*, 30 Noviembre 2016, COM (2016) 860 final, [En línea]. Disponible en: https://eur-lex.europa.eu/resource.html?uri=cellar:d2648a37-c626-11e6-a6db-01aa75ed71a1.0004.02/DOC_1&format=PDF [Accedido: Noviembre, 2020]

[10] Unión Europea: Comisión Europea, *Comunicación de la Comisión al Parlamento Europeo, al Consejo, al Comité Económico y Social Europeo y al Comité de las Regiones. Hoja de Ruta de la Energía para 2050*, 15 Diciembre 2011, COM (2011) 885 final, [En línea]. Disponible en: https://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:52011DC0885&from=HR [Accedido: Noviembre, 2020]

[11] Unión Europea: Comisión Europea, *Comunicación de la Comisión al Parlamento Europeo, al Consejo, al Comité Económico y Social Europeo, al Comité de las Regiones y al Banco Europeo de Inversiones. Un planeta limpio para todos. La visión estratégica europea a largo plazo de una economía próspera, moderna, competitiva y climáticamente neutra*, 28 Noviembre 2018, COM (2018) 773 final, [En línea]. Disponible en: https://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:52018DC0773&from=es [Accedido: Noviembre, 2020]

[12] Unión Europea: Comisión Europea, *Comunicación de la Comisión. El Pacto Verde Europeo*, 11 Diciembre 2019, COM (2019) 640 final, [En línea]. Disponible en: https://eur-lex.europa.eu/legal-content/ES/TXT/HTML/?uri=CELEX:52019DC0640&from=EN [Accedido: Noviembre, 2020]
[13] European Union, UE, *EU Energy in figures*, Luxembourg, Publications Office of the European Union, 2020. doi: 10.2833/20911
[14] Agora Energiewende and Sandbag (2018): The European Power Sector in 2017. State of Affairs and Review of Current Developments, [En línea]. Disponible en: https://ember-climate.org/wp-content/uploads/2018/01/EU-power-sector-report-2017.pdf
[15] Unión Española Fotovoltaica, UNEF, *El sector fotovoltaico hacia una nueva era.* 2020. pp.28-38
[16] B. De Miguel, "Bruselas concentra en construcción y renovables el plan urgente contra la crisis de la pandemia," *El País*, [En línea]. Disponible en: https://elpais.com/economia/2020-05-04/bruselas-concentra-en-construccion-y-renovables-el-plan-urgente-contra-la-crisis-de-la-pandemia.html
[17] "Solargis". [Internet] Disponible en: https://solargis.com/es/maps-and-gis-data/download/europe
[18] Unión Española Fotovoltaica, UNEF, *El sector fotovoltaico hacia una nueva era.* 2020. pp.38-82
[19] C. Monforte, "Estas son las principales empresas que han logrado los 5.000 MW en la subasta de renovables," *Cinco Días*, [En línea]. Disponible en: https://cincodias.elpais.com/cincodias/2017/07/26/companias/1501065002_978339.html
[20] Red Eléctrica Española, REE, *Las energías renovables en el sistema eléctrico español 2019,* 2020. pp.36-45
[21] Ministerio para la Transición Ecológica y el Reto Demográfico, MITECO, Plan Nacional Integrado de Energía y Clima 2021-2030. 2020. p.12
[22] Asociación Nacional de Productores de Energía Fotovoltaica, AMPIER, *Anuario Fotovoltaico 2020.* 2020. pp.104-130
[23] Real Decreto 244/2019, de 5 de abril, por el que se regulan las condiciones administrativas, técnicas y económicas del autoconsumo de energía eléctrica.
[24] M. Pareja Aparicio, "Energía fotovoltaica," *Radiación solar y su aprovechamiento energético*. Barcelona: Marcombo, 2010. Cap. 2
[25] Instituto para la Diversificación y Ahorro de la Energía, IDAE, 2009. *Pliego de condiciones técnicas de instalaciones aisladas de red*. [En línea]. Disponible en: https://www.idae.es/uploads/documentos/documentos_5654_FV_Pliego_aisladas_de_red_09_d5e0a327.pdf. [Accedido: Nov, 2020]
[26] A. Martínez Jiménez, *Dimensionado de instalaciones solares fotovoltaicas*, España: Paraninfo, 2012.
[27] «Android developers,» [En línea]. Disponible en: https://developer.android.com/studio/intro?hl=es-419. [Accedido: Noviembre 2020].
[28] «EUSCIENCEHUB,» [En línea]. Disponible en: https://ec.europa.eu/jrc/en/PVGIS/docs/noninteractive. [Último acceso: Noviembre 2020].