ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA DEL SOFTWARE

ShadedPath: aplicación Android para calcular rutas peatonales con mínima exposición solar

(ShadedPath: Android application to compute pedestrian routes with minimal sun exposure)

Realizado por

Antonio Manuel Rivas Fuentes

Tutorizado por

Dr. José Francisco Chicano García

Departamento

Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, OCTUBRE 2015

Fecha de defensa:
El Secretario del Tribunal

Resumen: El trabajo de fin de grado consiste en realizar una aplicación móvil basada en *Android* [7]. Para calcular la ruta más sombreada desde un punto de origen a un punto de destino. Dicha aplicación móvil realizará peticiones a un servidor que estará ejecutando un programa que realizará el cálculo de rutas conforme a los parámetros que elegirá el usuario desde su móvil. Estos parámetros pueden ser: la hora de partida, la fecha y la importancia de la sombra a la hora de calcular la ruta. Siempre podremos elegir entre una ruta más corta o una más sombreada, según la importancia que le quiera dar el usuario.

La lógica de la aplicación del lado del servidor que realiza el cálculo está basada en *OpenTripPlanner*, el cual usa algoritmos de búsqueda para encontrar el camino más corto. Sirviéndonos de dicho algoritmo realizaremos las modificaciones e implementaciones necesarias para calcular la ruta más sombreada.

Palabras Clave: *algoritmo de búsqueda, open trip planner, sombra, aplicación móvil, Android*

Abstract: *This project focuses on developing an Android mobile application that calculates the most shaded route between an origin and target location. The user can set preferences through this mobile application. The application will make requests to a server that will be executed in an external machine. The preferences are: departure time, date and the importance that a shadow route might be to the user.*

The server-side application that computes routes is based on OpenTripPlanner platform. This application uses search algorithms to find the shortest path between two locations. We extend the functionality of this algorithm to the needs of our project.

Keywords: *search algorithm, open trip planner, shadow, mobile application, Android*

Índice general

Capítulo 1

Introducción

1.1. Aclaraciones previas

Dado a que aún no existen datos de ***modelos de superficie*** [1] para la ciudad de *Málaga*, el proyecto ha tenido que desarrollarse para otra ciudad, en este caso hemos elegido el centro de *Barcelona*. En un futuro no se descarta implementarlo para la ciudad de *Málaga* siempre y cuando el ***Instituto Geográfico Nacional*** (*IGN*)[2] u otro organismo, proporcione un modelo digital de superficie para *Málaga*.

Para más información se puede dirigir a la página del *IGN* y, en concreto, a la del ***Plan Nacional de Ortografía Aérea*** (*PNOA*)[3]. En esta página se pueden observar las zonas en las que existen actualmente modelos digitales. También es posible encontrar información sobre la previsión de dar soporte a otras zonas en un futuro.

Otro aspecto a tener en cuenta es lo referente a la heurística del tiempo de exposición solar. La heurística se define como una función que busca un mayor rendimiento en el cálculo estimado de la solución general o la solución óptima. El problema radica en que no podemos determinar con exactitud una posible heurística para encontrar la solución óptima a nuestro problema de encontrar la ruta con menor tiempo de exposición solar. Por este

[1] Definición de los modelos de superficie y terreno: `https://en.wikipedia.org/wiki/Digital_elevation_model`

[2] Instituto Geográfico Nacional: `http://ign.es`

[3] Plan Nacional Ortográfico Aéreo *PNOA*: `http://pnoa.ign.es/es`

motivo hemos tenido que prescindir de esta implementación. Para más información sobre este concepto diríjase a las siguientes secciones: ***Cálculo de la ruta*** (3) y ***Algoritmo de búsqueda*** (3.2).

1.2. Definición

En las estaciones con mayor período de sol en nuestro país tenemos un clima caluroso. Esto conlleva que, a veces, cueste bastante moverse por la ciudad según determinadas horas del día. En el caso de los turistas de países nórdicos podría llegar incluso a ser molesto ya que no están acostumbrados. Incluso para nosotros tanta exposición solar puede ser dañino para la piel.

Con el paso de los años y la adaptación de las nuevas tecnologías se ha desarrollado un nuevo concepto: *Smart City* [11]. Las tecnologías de la información tienen un rol fundamental para ofrecer servicios a los ciudadanos para mejorar su calidad de vida. Existen ya numerosas aplicaciones que ayudan a moverse por la ciudad.

En este trabajo de fin de grado pretendemos introducir una nueva aplicación que facilite la vida a los usuarios que deban desplazarse entre dos puntos de la ciudad.

Hoy por hoy las aplicaciones de cálculo de rutas van incorporando poco a poco distintas variables, dependiendo de las preferencias del usuario, pero casi siempre estas variables están relacionadas con el tiempo requerido o la distancia recorrida. Esta aplicación incorporará un aspecto relacionado con el confort de la ruta: el usuario podrá indicar sus preferencias entre una ruta rápida y una ruta con más sombra. Hay que tener en cuenta que la ruta ha sido desarrollada para desplazarse a pie.

Consideramos solución aquella ruta que minimiza una combinación lineal de tiempo recorrido y tiempo de exposición al sol durante el mismo.

La decisión de implementar dicho proyecto como aplicación para *Smartphone* viene motivada por la gran penetración de este tipo de dispositivos en el mercado y por la utilidad que tiene conocer en cualquier momento y lugar una ruta de las características menciona-

das. Por otro lado, hemos escogido *Android* como plataforma móvil para el desarrollo por tratarse del sistema operativo para móviles más ampliamente utilizado hoy en día entre los Smartphones en un 81,5 % en 2014 según *International Data Corporation*[4].

1.3. Preliminares

Para un buen entendimiento del proyecto debemos introducir y aclarar ciertos conceptos relacionados con los elementos que se usarán para la realización del proyecto.

1.3.1. Modelos digitales de terreno y superficie

El cálculo de la ruta se hace en base a la altura del terreno, los edificios, la dirección y la elevación del sol. Para el primer caso necesitamos usar **modelos digitales de terreno y superficie** [6], *MDT* y *MDS*, respectivamente.

Se denomina *MDT* al modelo digital que representa en *3D* de forma aproximada la superficie de un terreno. El *MDT* equivale a la superficie del terreno aplicado y el *MDS* es lo mismo pero con todos los objetos que contiene dicho terreno como pueden ser edificios y vegetación. Este punto es muy importante ya que en la mayoría de las ciudades, por lo general, nos desplazaremos por zonas donde hay edificios y árboles. Es decir, que sólo con el *MDT* no podríamos medir la altura de los mismos. Ambos modelos son complementarios para el propósito de la aplicación.

[4] *International Data Corporation*: `http://www.idc.com`

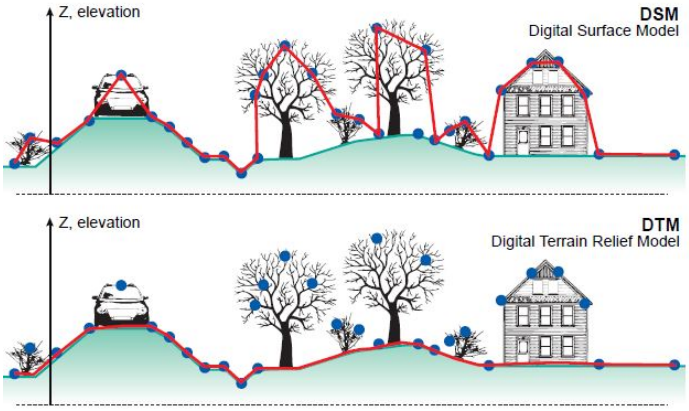

Figura 1.1: Explicación gráfica del *MDT* y *MDS*. En el primer caso se tiene en cuenta el terreno y los objetos para el *MDS*, en el segundo sólo el terreno para el *MDT*.

Para la elaboración de estos modelos digitales se ha hecho uso de la tecnología *Laser Imaging Detection and Ranging (LIDAR)* [5]. Esta tecnología mide la distancia a través de un haz de luz hacia un objetivo. Es popularmente usada para realizar mapas de alta resolución y tiene su uso extendido en disciplinas tan diversas como: geografía, geología, sismología, etc...

El **Instituto Geográfico Nacional** (*IGN*) es el organismo encargado de realizar estos modelos digitales. Se realizan vuelos por determinadas zonas de la Península Ibérica haciendo mediciones de la elevación del terreno con una resolución de $5m \times 5m$ para los *MDT* y con una resolución de *0,5 puntos/*m^2 para los *MDS*.

1.3.2. Sistemas de coordenadas Universal Transversal Mercator

El cálculo de rutas con mínima exposición solar se realiza a través de unos datos de entrada de los que se nutre esta aplicación. Los datos de alturas de los edificios y vegetación están dispuestos en un ***tablero bidimensional*** para la zona en que vamos a utilizar el cálculo de rutas. Cada recuadro de este tablero representa un área, que para el caso que nos toca, tendrá un tamaño de $5m \times 5m$ y estará representada por tres valores: la coordenada geográfica X, la Y y como último valor la altura.

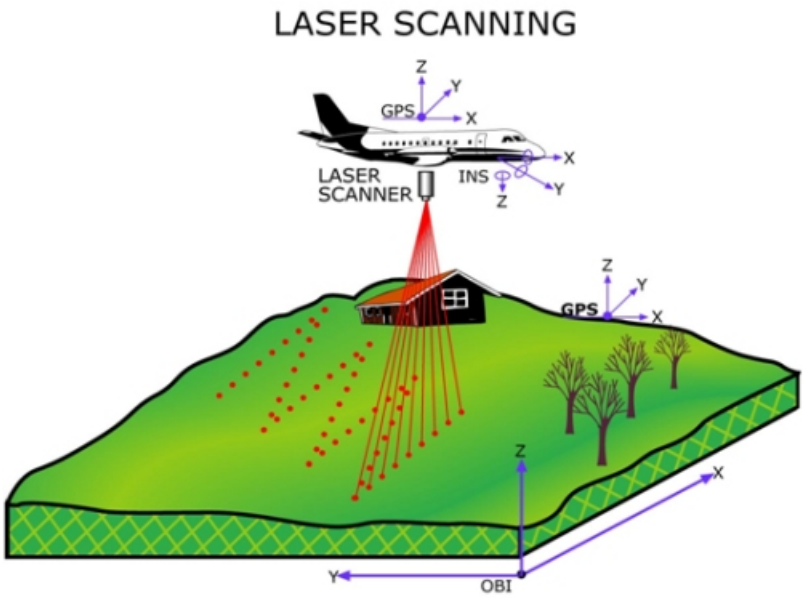

Figura 1.2: Ejemplo del funcionamiento *LIDAR* de medición de la distancia entre el sensor y el terreno.

Los dos primeros valores representan una coordenada en el formato *Universal Transversal Mercator* (*UTM*) [4]. *UTM* usa un sistema de coordenadas cartesiano en *2* dimensiones para determinar una localización en todo el globo. Poseen una representación horizontal, esto quiere decir, que es independiente de la posición vertical para identificar la localización. Algunas de las características referentes al formato *UTM* son las siguientes:

1. *Es una proyección cilíndrica*: Se obtiene proyectando el globo terráqueo sobre una superficie cilíndrica.

2. *Es una proyección transversal*: El cilindro es tangente a la superficie terrestre según un meridiano. El eje del cilindro coincide, pues, con el eje ecuatorial central (el que se corresponde con la latitud y longitud *<0,0>*.

3. *Es una proyección conforme*: Mantiene el valor de los ángulos. Si se mide un ángulo sobre la proyección coincide con la medida sobre el elipsoide terrestre.

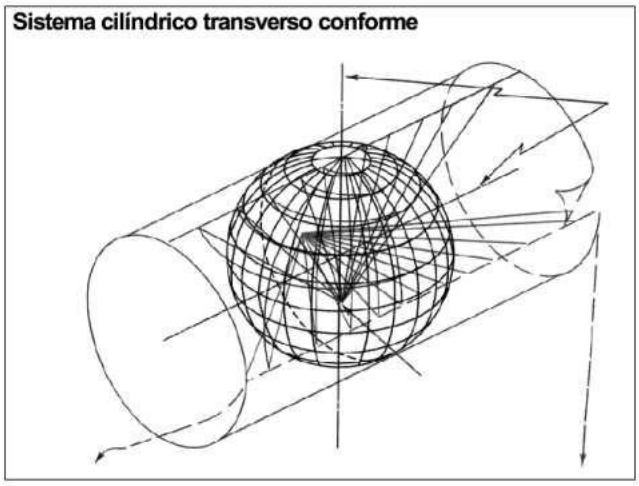

Figura 1.3: Las figura muestra el resultado de la proyección *UTM* sobre el meridiano de *Greenwich* (en la parte superior el hemisferio correspondiente al antimeridiano 180^o y en la parte inferior el hemisferio correspondiente al propio meridiano 0^o).

Sin embargo surgen algunos inconvenientes con este formato y es que no existe una uniformidad en la escala de distancias. Esto quiere decir que a medida que nos alejamos del punto de tangencia *esfera-cilindro* en la dirección perpendicular al cilindro, las distancias se agrandan como podemos observar en la Figura 1.3.

Este problema se soluciona con la introducción de los ***husos***. Se procede a dividir la superficie terrestre en *60* husos o zonas iguales de *6* grados de longitud. Con esto se tiene *60* proyecciones iguales, pero cada una con su respectivo meridiano central. Los husos se numeran del *1* al *60* comenzando desde el antimeridiano de *Greenwich* (180^o) hacia el este. De este modo el huso comprendido entre 180^o *W* y 174^o *W* es el primero. El huso comprendido entre 6^o *W* y 0^o *E* es el 30, en el que queda el cuadrante *nororiental* de la península ibérica. Esto se puede apreciar mejor en la Figura 1.4.

A su vez dentro de cada huso se establece una división en zonas. Cada zona posee 8^o de latitud y 6^o de longitud. Se designa cada una con el número de su huso y una letra mayúscula. El resultado final es una cuadrícula como la que se muestra en la Figura 1.5.

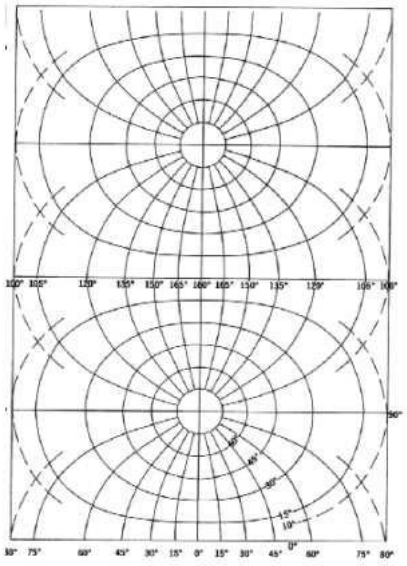

Figura 1.4: Las figura muestra el resultado de la proyección *UTM* sobre el meridiano de *Greenwich* (en la parte superior el hemisferio correspondiente al antimeridiano *180º* y en la parte inferior el hemisferio correspondiente al propio meridiano *0º*).

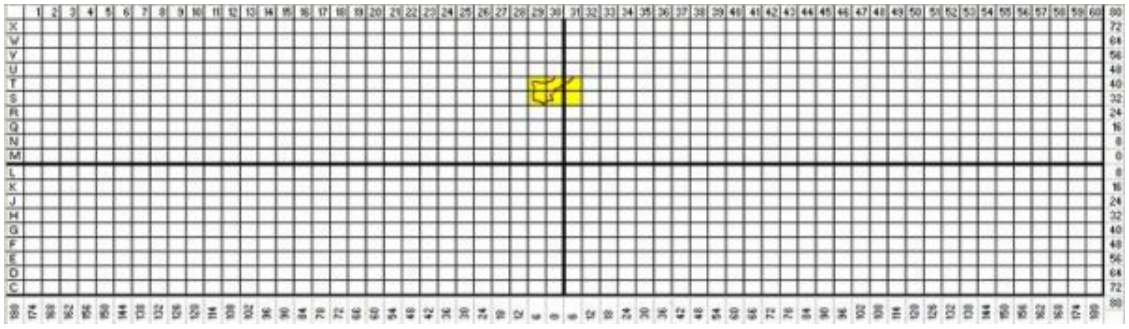

Figura 1.5: Cuadrícula *UTM*

Para denominar las zonas se usa, como se dijo, una letra mayúscula. Para ello se ha seguido la dirección de Sur a Norte partiendo de la letra C siguiendo el orden alfabético a excepto de las vocales y las letras que pueden confundirse con un número: la B, la O y la letra P. Las zonas entre la M y la X corresponden al hemisferio Norte, y las zonas entre la C y la L al hemisferio Sur. Como excepción, la zona X posee *12º* de latitud y se extiende desde los *72º* N hasta los *84º* N.

En la imagen anterior se observa que la Península Ibérica abarca *6* zonas: *29T*, *30T*, *31T*, *29S*, *30S* y *31S*.

Ahora para definir la ***geometría del huso*** consideraremos a modo de ejemplo el huso *31* donde se ubica la Península Ibérica. Este huso *31* se prolonga desde los 0^o hasta los 6^o. En general todos los husos poseen un meridiano central que los divide en *2* partes iguales con una longitud de 3^o *E*. El meridiano central se utilizará en la proyección *UTM* de cada huso.

La proyección *UTM* no recoge latitudes superiores a los 84^o *N* ni a los 80^o *S*. La primera zona de letra *X* aparece entre los 84^o *N* y los 72^o *N* de latitud. La última aparece con la letra *C* entre los 72^o *S* y los 80^o *S*.

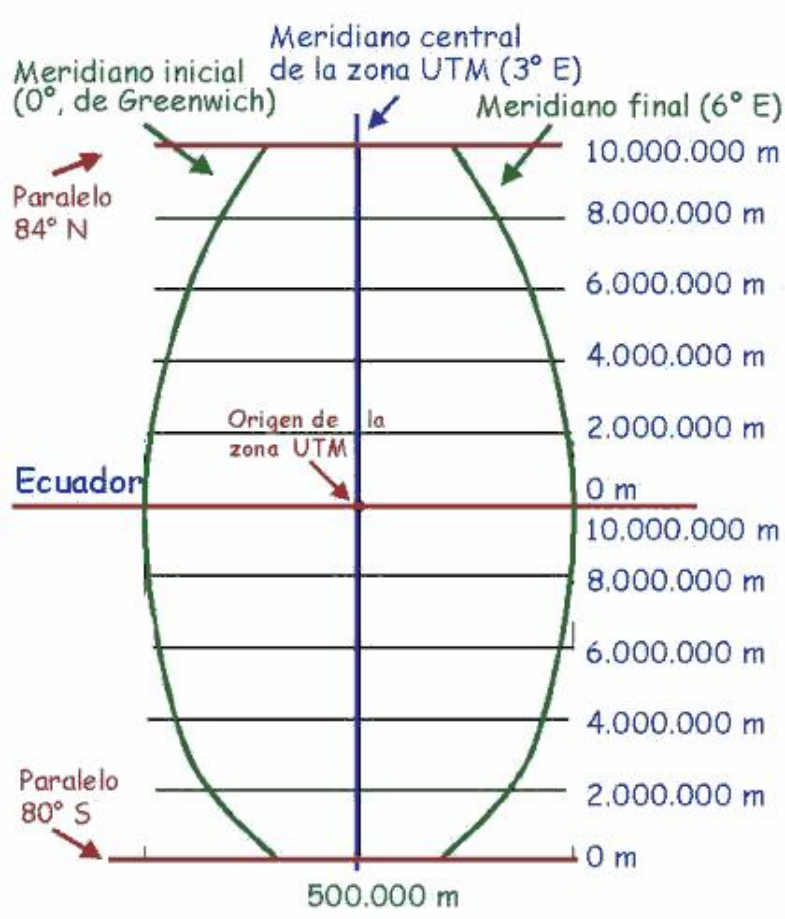

Figura 1.6: Geometría del huso

Como se ilustra en la Figura 1.6, este sería el resultado de proyectar el huso *31* según su meridiano central (*3º E*). Como se observa, éste lo divide en dos partes iguales. Esto permite establecer dos ejes cartesianos X e Y sobre el huso, de tal manera que el eje X es el ecuador y el eje Y el meridiano central. Estos ejes cartesianos permiten determinar puntos sobre el huso haciendo uso de dos coordenadas rectangulares X e Y que se denominan ***coordenadas UTM***.

El origen del sistema de *coordenadas UTM* se encuentra por tanto, en la intersección del Ecuador con el antimeridiano de *Greenwich*. Cada huso posee su propio origen de coorde-

nadas donde el paralelo del Ecuador es el origen Y y la coordenada X corresponde con el meridiano central del huso, que en la Figura 1.6 vemos que es *500.000* (entre *0º E* y *6º E*). La idea de las coordenadas UTM es que sus dos valores X e Y sean siempre positivos. Por ello no se ha elegido las coordenadas X=0 e Y=0 para el origen.

Cada zona *UTM* es expresada por el número de huso del *1* al *60* y una letra de zona de *C* a *X*. Se descompone a su vez en regiones rectangulares de *100km* de lado por lo tanto eso da una superficie de *10.000km*^{2}.

Cada cuadrado de *100km* de lado se designa mediante una pareja de letras mayúsculas. Dando lugar a una cuadrícula *hectokilométrica*, que en el caso de la península ibérica tendría el siguiente aspecto:

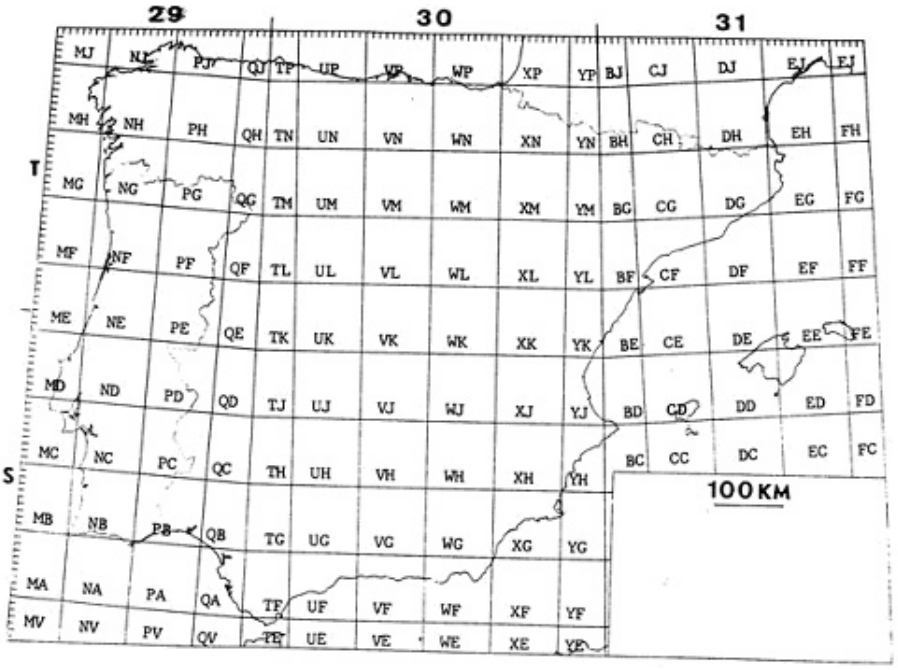

Figura 1.7: Cuadrícula *UTM* España

La primera letra de la designación de los cuadrados de *100km* expresa la posición a lo largo de un meridiano en el huso. La segunda letra expresa la posición del cuadrado a lo largo de un paralelo. Así, la designación *30T UN* permite identificar un cuadrado de

100km de lado en la superficie terrestre. La letra *U* expresa la posición del cuadrado en la dirección *Este-Oeste* y la letra *N* en la posición *Norte-Sur*. El siguiente cuadrado de *100km* a la derecha de *UN* será *VN*. El cuadrado de *100km* al norte será *UP*, mientras el que se encuentra al sur será *UM*.

El ***Instituto Nacional de Geografía*** (*IGN*) dispone de mapas topográficos, en los cuales se puede observar con más detenimiento el uso de este formato.

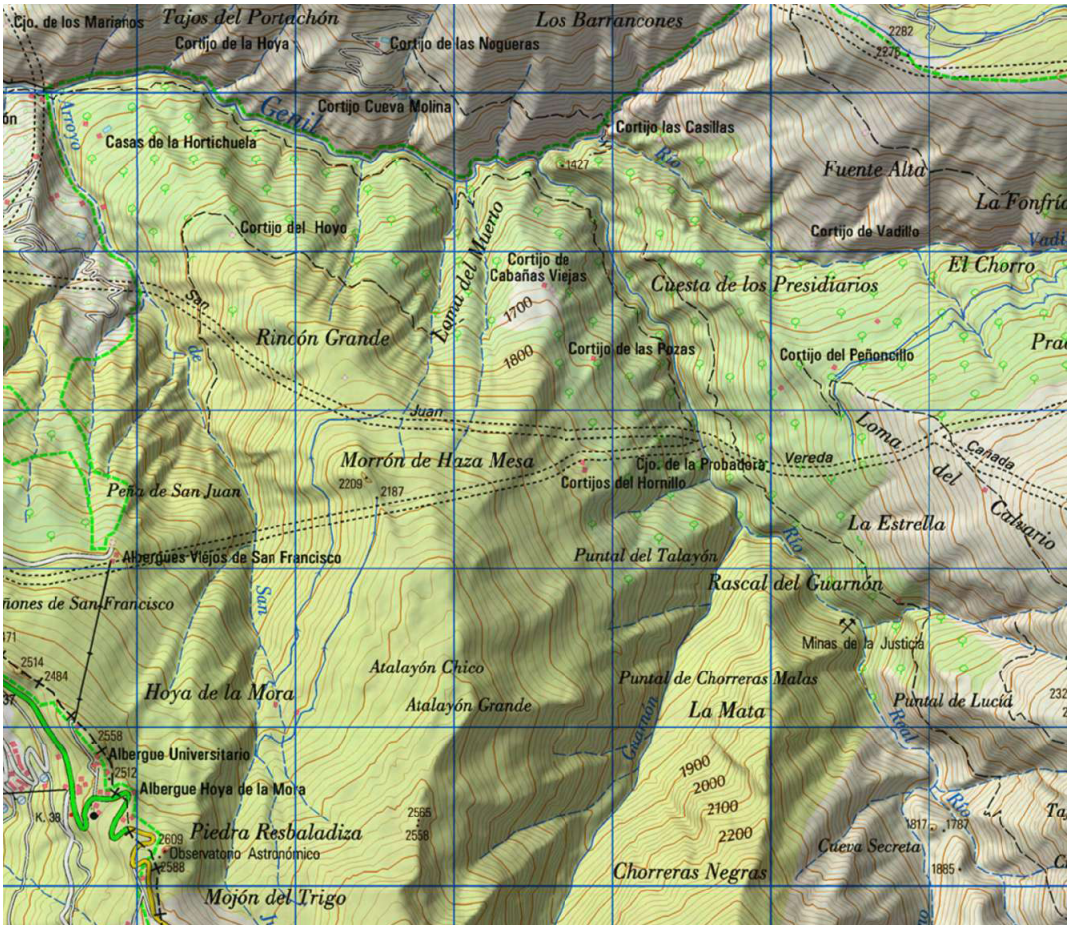

Figura 1.8: Ejemplo de mapa topográfico del *IGN*

Concretamente, en la Figura 1.8, se observa la cuadrícula *UTM*. Cada cuadrado posee un área de $1km^2$.

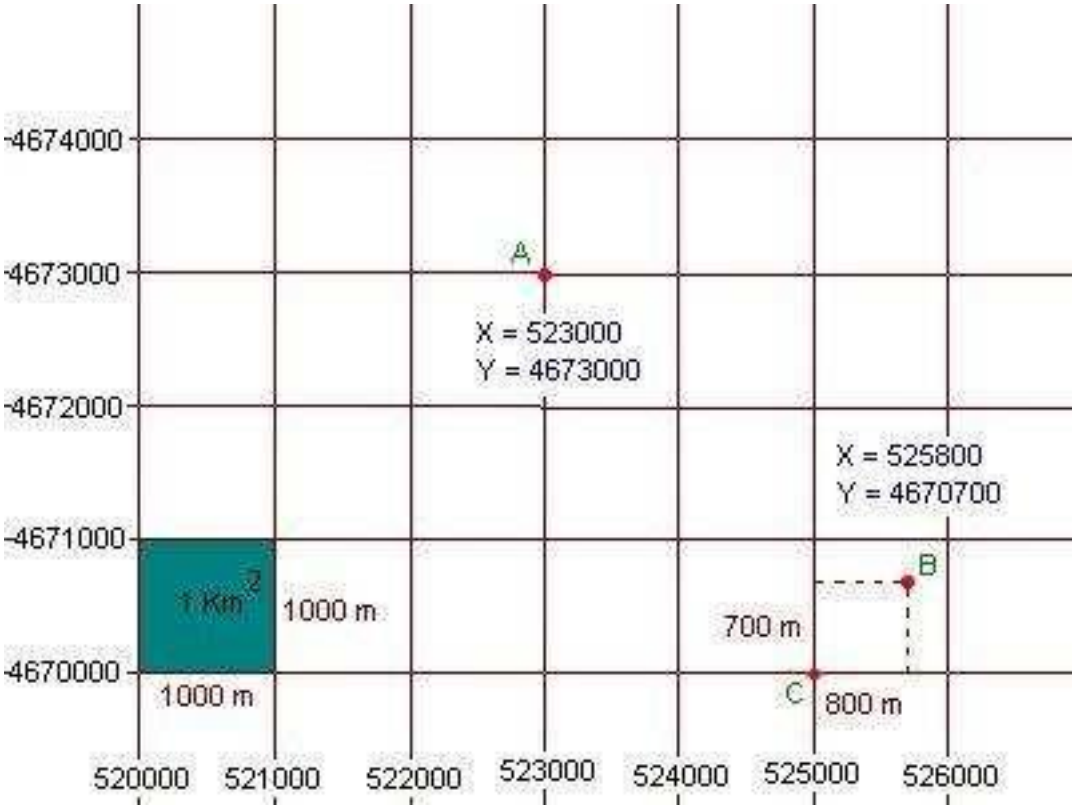

Figura 1.9: Detalle de una cuadrícula *UTM*

Para hacer referencia a cada punto de la cuadrícula *UTM*, se usan dos valores llamados *coordenadas*. Existe una coordenada X que expresa un valor en metros sobre la horizontal, mientras que la coordenada Y hace lo propio sobre la vertical del plano.

En la Figura 1.9 se representa un segmento de cuadrícula *UTM*. Cada cuadrado representa una extensión de $1km^2$. La coordenada X representa una distancia sobre la horizontal y va tomando valores en metros: *520.000*, *521.000*, *522.000*, etc... a intervalos de *1.000m* (*1km*). La coordenada Y representa una distancia sobre la vertical y va tomando los valores en metros *4.670.000*, *4.671.000*, *4.672.000*, etc... a intervalos de *1.000m* (*1km*). La posición del punto A se expresa mediante las coordenadas X e Y de su intersección sobre la cuadrícula. Este es el caso más sencillo pero no el más frecuente.

Lo más común sería encontrarnos un punto como el B, el cual no se sitúa en ningún vértice de la cuadrícula. En este caso si nos fijamos en el punto C de coordenadas X=*525.000* e Y=*4.670.000* en el vértice de la cuadrícula deberemos medir la distancia horizontal y vertical hacia el punto B. Si para esta distancia horizontal se obtienen *800m*, la coordenada X será *525.000+800=525.800*, mientras que para la Y será *4.670.000+700=4.670.700*. Por consiguiente la coordenada X aumenta hacia el Este y la coordenada Y aumenta hacia el Norte[5].

Ya hemos definido el formato UTM el cual define las coordenadas X e Y. Existe un tercer valor que es la coordenada Z y es el que nos otorgará la elevación en ese punto geográfico. Este nuevo valor expresa su cota o altitud con respecto al nivel del mar en metros. Por ejemplo si el punto A se halla a *872m* sobre el nivel del mar, equivaldrá a Z=*872*. Este valor estará en formato WGS84 que detallaremos en la sección que viene a continuación.

1.3.3. World Geodetic System (WGS84)

El formato *World Geodetic System (WGS84)* [3] se basa en un patrón matemático en *3* dimensiones que busca representar la tierra por medio de un elipsoide. Es un sistema de referencia terrestre único para referenciar las posiciones y vectores. Se estableció utilizando observaciones *Doppler* al sistema de satélites de navegación *NNSS* o *Transit*, de tal forma que se adapta lo mejor posible a toda la Tierra. Se define así mismo como un sistema cartesiano geocéntrico de la siguiente manera:

- Origen, centro de masas de la Tierra, incluyendo océanos y atmósfera.
- Eje Z paralelo a la dirección del polo *CIO* o polo medio definido por el *BIH*, época *1984* con una precisión de *0,005″*.
- El eje X es la intersección del meridiano origen, *Greenwich*, y el plano que pasa por el origen y es perpendicular al eje Z, el meridiano de referencia coincide con el meridiano cero del *BIH* en la época *1984* con una precisión de *0,005″*. Realmente el meridiano origen se define como el *IERS Reference Meridian (IRM)*.

[5]Sistema de coordenadas geográficas *UTM*: `http://www.aristasur.com/contenido/sistema-de-coordenadas-geograficas-utm`

- El eje Y ortogonal a los anteriores, pasando por el origen.

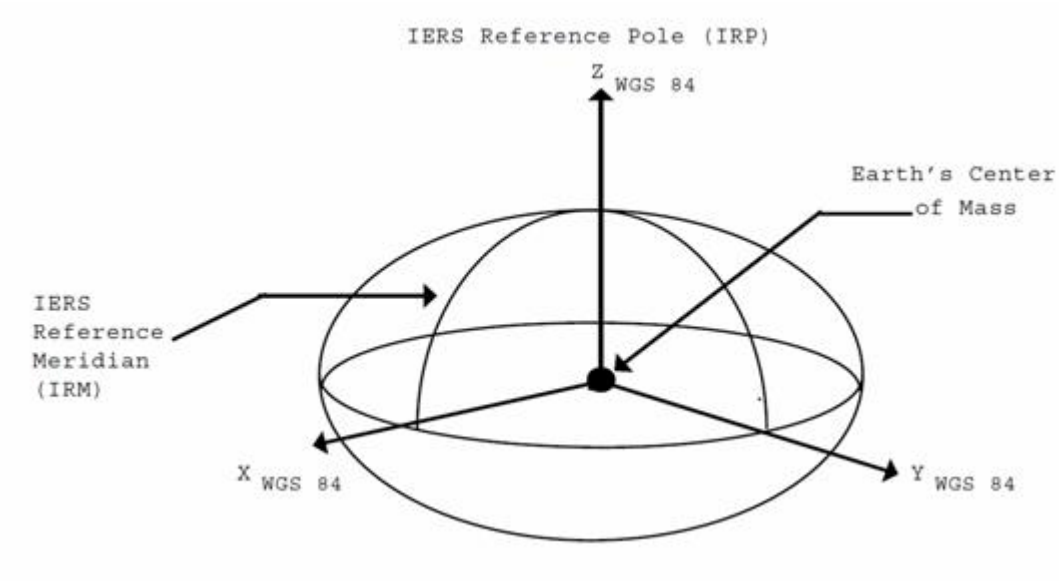

Figura 1.10: Definición de *WGS84* (Fuente: *NIMA*)

La altura, que representa la coordenada Z en el *UTM* anteriormente descrito, está tomada con dicho formato. Es decir, la altura con respecto a un determinado elipsoide con semieje mayor y menor bien definidos y cuya superficie casa bien con la placa continental europea. Para el caso que nos toca, en España, el elipsoide se puede encontrar por encima o por debajo del nivel del mar, por lo que hay que tener en cuenta que pueda haber una pérdida de precisión a la hora de obtener las alturas de los archivos en formato *UTM*[6].

1.3.4. Conceptos astronómicos

Otro aspecto fundamental a tener en cuenta es la posición y elevación del sol en todo momento durante la ruta elegida. Para comprenderlos tenemos que introducir los conceptos de ***acimut*** y de ***elevación*** del sol [8].

[6]Sistemas geodésicos de referencia: `http://www.ign.es/ign/layoutIn/actividadesGeodesiaStmagd.do`

1.3.4.1. Acimut

En astronomía el ***acimut*** es el ángulo o longitud de arco medido sobre el horizonte celeste que forman el punto cardinal **norte** y la proyección vertical del astro sobre el horizonte del observador, el cual se sitúa en una determinada latitud. Para nuestro caso, el sol es el cuerpo celeste. El *acimut* se mide en grados y determina la posición del sol en todo momento del día moviéndose en el sentido de las agujas del reloj.

El *acimut* depende completamente de la posición concreta del observador, es decir, el sol puede ser visto bajo diferentes coordenadas horizontales por diferentes observadores situados en puntos diferentes del globo.

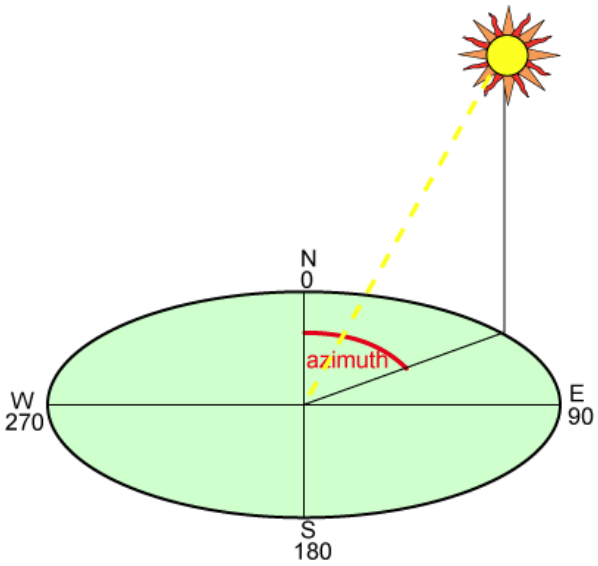

Figura 1.11: Representación del *acimut*.

Como observamos en el dibujo, el *acimut* en el Este valdría *90º*, al Sur *180º*, al Oeste *270º* y culminando en el norte con *0º/360º*.

Este concepto es muy importante dado que nos dice en todo momento la dirección que toma la proyección de los rayos del sol. De esta manera sabemos las coordenadas que debemos visitar en cada paso durante la ruta para ver las alturas de los edificios que corresponderán a la línea formada por la proyección solar.

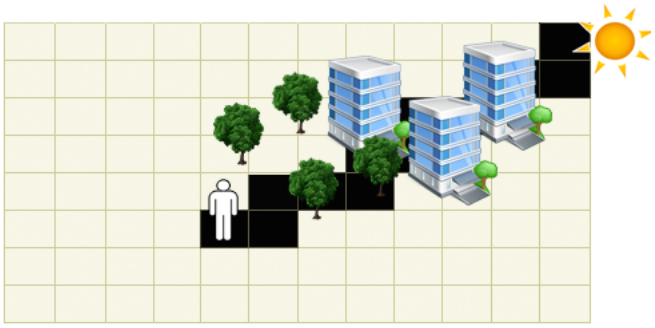

Figura 1.12: Representación del *acimut* en los datos de entrada. Cada recuadro en el tablero posee una coordenada, por lo que trazamos la línea formada por la dirección del sol desde donde nos encontramos hasta los bordes del tablero, de forma que podemos ir consultando recuadro por recuadro de la línea la altura de los edificios y vegetación.

1.3.4.2. Elevación del sol

La elevación del sol es la distancia angular vertical que hay entre el sol y el horizonte local del observador, o también llamado, plano local del observador. Diremos que el sol tiene *12º* de elevación cuando su centro geométrico está situado a *12º* sobre el horizonte o plano local del observador.

En las Figuras 1.13 y 1.14 se muestran la elevación del sol respecto a dos posiciones diferentes del observador.

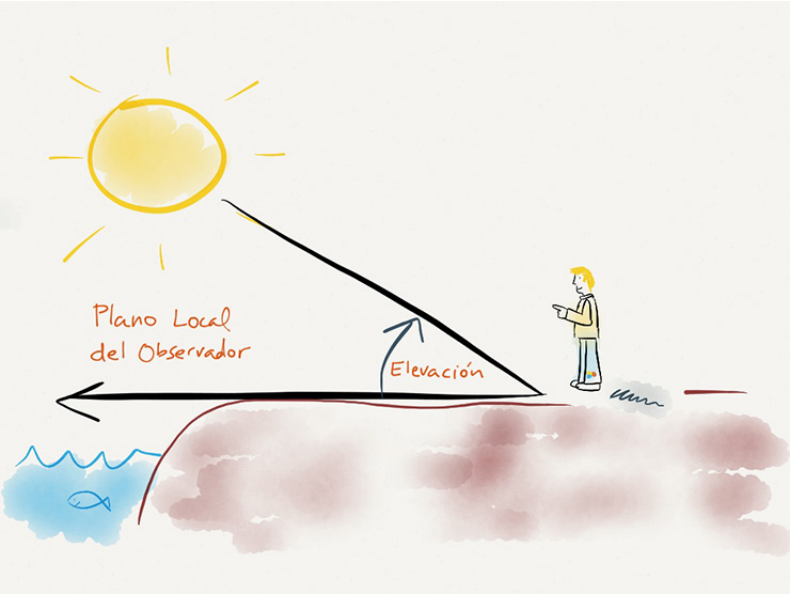

Figura 1.13: Observador situado en el nivel del mar, plano local y elevación del sol.

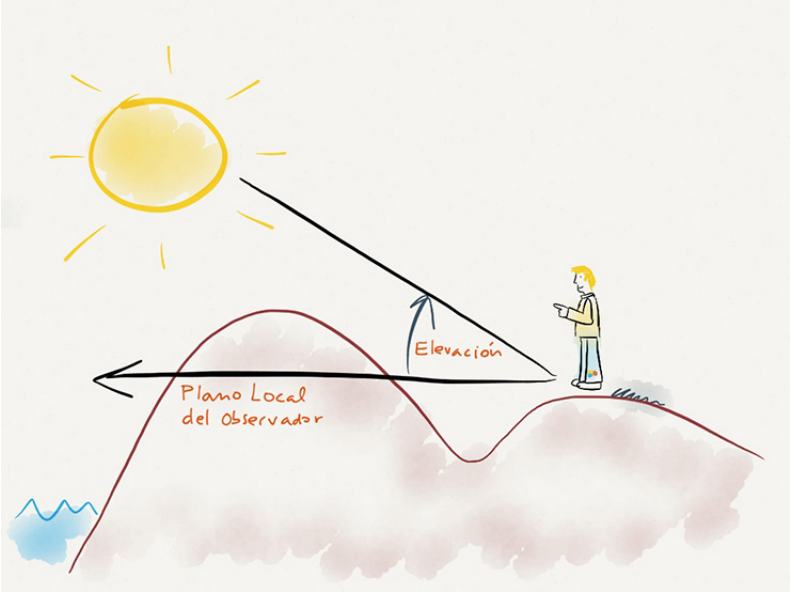

Figura 1.14: Observador situado en la cima de una montaña, plano local y elevación del sol.

Para la ejecución de los cálculos de posición y elevación del sol disponemos de una una serie de librerías destinados al cálculo en determinadas áreas de ciencias llamado *JScience*[7]. En concreto la que usaremos para nuestros propósitos será la librería ***SunRelativePosition*** que recibe como parámetros de entrada una posición geográfica, fecha y hora y que devolverá el *acimut* y elevación solar para el punto geográfico determinado.

1.4. Fases del proyecto

El proyecto se ha dividido en las siguientes fases:

1. *Análisis del problema y los recursos utilizados*
   Identificado y realizado un análisis del problema que queremos resolver, encontramos unos recursos para poder abordar el problema y que también serán sometidos a análisis para poder ver sus alcances así comos sus limitaciones.

[7] *Java* ^{TM} *Tools and Libraries for the Advancement of Sciencies.*: `http://jscience.org/`

2. *Diseño e implementación de la aplicación*
   Mediante el análisis realizado procederemos a la estructuración de la aplicación enfocándonos en sus funcionalidades principales en pos de modularizar y desarrollar de forma eficiente la aplicación.

   Las funcionalidades son: el cálculo de alturas de los modelos digitales, el cálculo de alturas angulares a través de la posición del sol, el cálculo de la ruta a través del algoritmo de búsqueda proporcionado por la plataforma *OpenTripPlanner* y el desarrollo de la aplicación móvil.

3. *Pruebas*
   Trabajando por objetivos para cada funcionalidad, al final de cada ciclo se han realizado las pruebas pertinentes para verificar las funcionalidades de manera que mantengamos una correcta implementación en todo momento y así evitar problemas y errores en el desarrollo a posterior.

4. *Informes y Documentación*
   En cada fase del proyecto iremos documentando las implementaciones y avances que vayamos realizando, de esta forma podremos tener acceso a un histórico en aras de poder hacer una modificación o corrección si lo viéramos necesario de una forma más eficiente.

1.5. Métodos materiales utilizados

Para la realización del proyecto se han utilizado los siguientes recursos:

1. Ordenador personal *MacBook Pro* con el sistema operativo *Mac OS X*.

2. El *framework* de desarrollo *Android Studio* para el desarrollo de la aplicación móvil y la realización de pruebas.

3. Uso de los *modelos de superficie* y *modelos de terreno* sacados del *IGN*.

4. Uso de la plataforma *OpenTripPlanner* para el cálculo de rutas basado en *Java* [2].

5. Uso de las librerías para cálculos de la posición del sol y la elevación mediante las librerías suministradas por la página *JScience*[8], en concreto, la librería *SunRelativePosition*.

6. Elaboración de la memoria del TFG con LaTeX.

7. Uso del *IDE Intellij IDEA* [9] para el desarrollo de la parte del servidor.

8. Móvil *Galaxy Nexus* para desplegar la aplicación y la realización de pruebas.

9. PC con procesador *i7-4790k* con *8GB* de memoria con el *SO Windows 8.1* como servidor web ejecutando el servidor para recibir las peticiones.

[8] *JScience*: `http://jscience.org/`
[9] *Intellij IDEA IDE* `https://www.jetbrains.com/idea/`

Capítulo 2

Análisis

En nuestra etapa de análisis procedemos a la captura de requisitos que conforman la aplicación. La principal funcionalidad radica en el cálculo de la ruta más sombreada desde una coordenada origen a otra destino. De éste se ramificarán los demás requisitos funcionales, los cuales darán la posibilidad de que el usuario elija las opciones pertinentes para establecer la ruta de acuerdo a sus necesidades en ese momento, así como información detallada de la ruta en cada etapa de la misma.
Empezamos a detallar la captura de requisitos en los siguientes apartados.

2.1. Requisitos funcionales

A continuación se describe los requisitos funcionales de los que dispondrá la aplicación:

Identificador del requisito	**Nombre**	**Descripción**
REQF-01	Seleccionar hora y fecha.	La aplicación debe dar la posibilidad al usuario de seleccionar la hora y la fecha de partida de la ruta.
REQF-02	Establecer la importancia de la sombra en la ruta.	El usuario podrá establecer mediante un intervalo de *0* a *10*, la relevancia o peso que tendrá la sombra en la ruta.
REQF-03	Establecer el origen y el destino.	El usuario podrá establecer el origen y el destino de la ruta en el mapa de la aplicación.
REQF-04	Mostrar la ruta.	Una vez elegido el origen y el destino, la aplicación calculará la ruta y la mostrará en pantalla.

REQF-05	Recalcular la ruta.	Una vez calculada la ruta, el usuario podrá de nuevo recalcularla. Funcionalidad introducida por si el usuario desea modificar los parámetros de entrada para la misma ruta.
REQF-06	Mostrar información de la ruta.	Una vez realizado el cálculo y mostrada la ruta, también se mostrará información general de la ruta: tiempo del recorrido, tiempo de exposición solar y distancia.
REQF-07	Mostrar información de etapas de la ruta.	Una vez calculada la ruta, el usuario podrá acceder a la información de las distintas etapas de la misma. Cada etapa tendrá el tiempo y el porcentaje de exposición solar.

2.2. Requisitos no funcionales

Los requisitos no funcionales irán enfocados principalmente al rendimiento del sistema, la arquitectura y la usabilidad en sí.

Identificador del requisito	**Tipo de requisito no funcional**	**Nombre**	**Descripción**
REQNF-01	Requisito de proceso.	Formato del archivo de entrada de coordenadas y alturas.	Los datos que recibe de entrada la aplicación deben cumplir el formato *XYZ* en el sistema *UTM*.
REQNF-02	Requisito de producto.	Desarrollo en *Android*.	La aplicación será desarrollada para la plataforma *Android*.
REQNF-03	Requisito de usabilidad.	Notificación de errores e información.	La aplicación notificará al usuario en todo momento si se ha producido un error al cargar los datos o todo lo que tenga que ver también con el proceso de cálculo de la ruta.
REQNF-04	Requisito de arquitectura.	Servidor web.	La aplicación en *Android* realizará las peticiones de cálculo a una máquina externa que estará ejecutando un servidor web, el cual procesará las peticiones y devolverá una respuesta.
REQNF-05	Requisito de recursos.	Cálculo de rutas mediante *OpenTripPlanner*.	En la máquina externa que ejecute el servidor web habrá una instancia de la plataforma *OpenTripPlanner*, el cual se encargará de realizar los cálculos de ruta.
REQNF-06	Requisito de interfaz.	Cálculo de ruta sobre *Google Maps*.	En la aplicación *Android* se usará *Google Maps* para la visualización de la ruta.
REQNF-07	Requisito de rendimiento.	Recursos de la máquina servidor.	El servidor web gozará de las características idóneas a nivel de procesador y memoria, así como conexión, para satisfacer el cálculo de las rutas y procesamiento de peticiones y respuestas en un tiempo óptimo.

Capítulo 3

Cálculo de la ruta

En esta sección se detallarán en profundidad los procesos para el cálculo de la ruta. Se explicarán los diferentes pasos para llegar a este fin como son:

1. El cálculo de alturas a partir de los modelos digitales de terreno (MDT) y superficie (MDS).

2. El cálculo de la altura angular en una coordenada a raíz de las alturas de dichos modelos.

3. El algoritmo de búsqueda para el cálculo de la ruta.

4. La plataforma ***OpenTripPlanner*** que usaremos de base para dicho fin, que desplegará el servidor y que modificaremos y añadiremos las funcionalidades para nuestro propósito.

3.1. Datos de entrada

Los cálculos de la ruta se realizan en base a unos ficheros de entrada en el que cada línea representa una coordenada en UTM seguida de la elevación del terreno o superficie respecto del nivel del mar. Antes de poder usar estos datos en bruto, tenemos que realizar una serie de modificaciones en estos datos para poder usarlos en la aplicación.

3.1.1. Los datos de alturas a partir de los modelos digitales

Un aspecto importante para el desarrollo de la aplicación es el cálculo de las alturas de los edificios y la vegetación con el objetivo de comprobar si tenemos o no sombra en un punto geográfico. Para hacer esto usaremos los *MDT* y *MDS*.

Dichos modelos los podemos encontrar en la página del **Instituto Geográfico Nacional** (*IGN*) para casi todo el territorio peninsular. En concreto, los modelos *MDT* nos dan la altura respecto al nivel del mar de todo el relieve del terreno con una resolución de $5m \times 5m$ dispuestos en una cuadrícula, mientras que los *MDS* nos dan los mismos datos sin estar organizados en una cuadrícula, y en este caso contará con la altura de los edificios y vegetación sobre el relieve del terreno. Los *MDS* poseen una resolución de *0,5 puntos/*m^2. Cada archivo cubre un área de $2km \times 2km$.

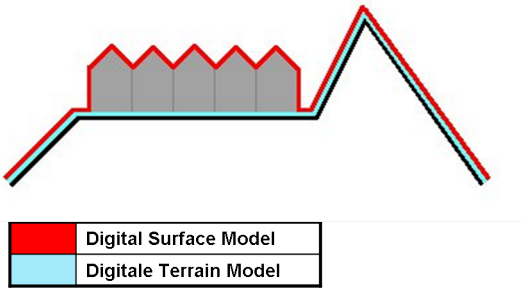

Figura 3.1: Representación del *MDS* y *MDT*

Para el *MDT* tenemos un formato de archivo *ASCII Grid* que se compone de una matriz bidimensional que será convertido a un formato más amigable que simplifica la posterior manipulación de los datos. El mismo *IGN* nos ofrece una herramienta[1] para poder convertir el archivo *ASCII* al formato *XYZ* , que está formado por filas de tres valores: *X*, *Y* y *Z*. Las dos primeras columnas, *X* e *Y*, se refieren a las coordenadas en formato *UTM*, mientras que la tercera columna, *Z*, es la altura de dicho punto en el sistema *WGS84*.

Una vez realizado el cambio de formato el archivo tendrá el aspecto siguiente:

[1]Herramientas del IGN: `http://www.ign.es/ign/layoutIn/herramientas.do`

...

428235 4580000 20.091

428240 4580000 20.382

428245 4580000 20.473

...

Una vez obtenido el archivo *MDT* correspondiente procedemos a manipular el archivo *MDS*. El archivo *MDS* posee una extensión *.las ó .laz*, por lo que han de ser convertidos también a un formato amigable para poder manipular los datos. *IGN* no ofrece ninguna herramienta para poder manejar este tipo de archivos pero la *suite* de herramientas *LAS-Tools* [2] nos ayudará para dicho fin. En concreto usaremos la herramienta llamada *laszip* que permite descomprimir el archivo *.laz* a un formato *XYZ*.

[2]Suite de herramientas de *LASTools*: `http://rapidlasso.com`

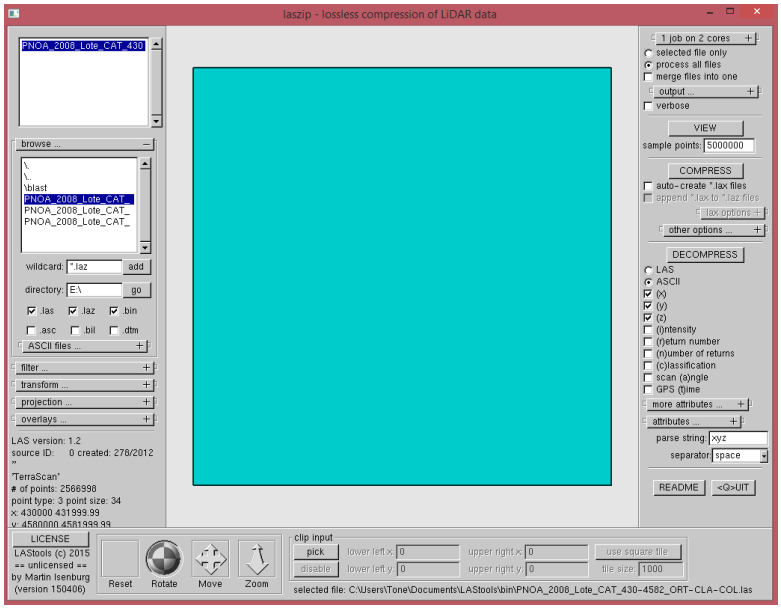

Figura 3.2: Interfaz de la aplicación *laszip*

Esto generará un archivo *txt* que contiene todos los puntos en el siguiente formato:

...

430011.43 4581792.94 25.51

430013.09 4581793.21 27.22

430014.88 4581793.33 25.66

...

Como ya dijimos anteriormente, los *MDS* no están dispuestos en una cuadrícula y tienen una resolución de *0,5 puntos/m^2*. Esto genera los resultados que observamos arriba, por lo que a la hora de combinarlos con los datos del *MDT*. Estos últimos tienen una resolución de $5m \times 5m$ y se nos hace bastante difícil y para nada trivial calcular la diferencia entre ambos modelos para obtener la altura de los edificios y vegetación, por lo que por propósitos de eficiencia al realizar los cálculos, se normalizará el *MDS* a la misma resolución del *MDT*.

3.1.2. Cálculo y ajuste de los datos de los modelos digitales de superficie

Para poder usar los datos del *MDS* con el *MDT*, utilizaremos una *Distribución Gaussiana* o **distribución normal**.

Cada punto del *MDS* lo redondearemos y normalizaremos a múltiplos de *5*, de acuerdo con la resolución del *MDT*, y haremos una media ponderada de dichos puntos cuyo peso será mayor mientras más cerca se hallen del centro del recuadro que representa la coordenada. Determinamos el centro como el punto más exacto aunque esto puede dar cierta pérdida de precisión pues el usuario no tiene que estar precisamente en el centro de un recuadro puesto que el área de una celda es de $25m^2$. Situaremos el centro del recuadro a una distancia de *2,5m* de cada lado, ya que la resolución es de $5m \times 5m$. De esta forma a valores cercanos al centro, las alturas en dicho punto tendrán mayor importancia mientras que aquellos que se acerquen a los bordes del recuadro, tendrán menos.

Con este concepto redondeamos X e Y de cada coordenada del *MDS*. Realizamos la diferencia entre el valor redondeado y el valor original. Con el resultado podemos determinar la relevancia que tendrá ese punto en el recuadro. Siendo el valor redondeado el centro, si la diferencia es cercana a 0, tendrá mayor relevancia que aquellos con una diferencia mayor. Esto lo conseguimos mediante **interpolación**[3]. Por lo que al final para cada recuadro del *MDT* tendríamos una lista de puntos de altura con sus ponderaciones correspondientes basados en su distancia al centro de ese recuadro. Con esto se realiza la media ponderada para cada recuadro de la cuadrícula, se ordena y se obtiene al final el archivo *MDS* normalizado para poder utilizarlo junto al *MDT*.

[3]Definición de interpolación: `https://es.wikipedia.org/wiki/Interpolacion`

Para interpolar y obtener así el peso o importancia del punto del MDS a su recuadro más próximo según redondeo usaremos la siguiente fórmula de **interpolación lineal**:

$$x_2 = \frac{(y_2 - y_1) * (x_3 - x_1)}{y_3 - y_1} + x_1 \tag{3.1}$$

Donde cada variable:

- x_1 y x_3 toman el valor *0* y *1* respectivamente que representa el intervalo de ponderación y siendo x_2 el valor de ponderación devuelto por la ecuación asociado al punto que estamos evaluando que se encuentra dentro de este intervalo.
- y_3 e y_1 toma el valor de la distancia al centro (*2,5m*) y *0m* respectivamente e y_2 será un valor dentro de este intervalo. Será la diferencia entre el valor redondeado del punto a un múltiplo de *5* y el valor original del mismo, que están asociados a los valores X e Y de la coordenada.

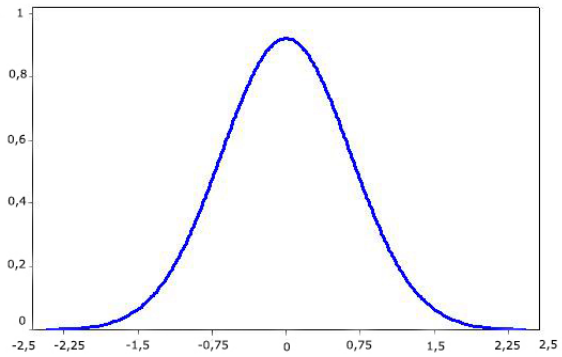

Figura 3.3: Representación de la distribución *Gaussiana*. En el eje X, en ambos extremos, se representaría la distancia en *2,5m*. En el eje Y se representan los pesos que es mayor a medida que nos dirigimos al centro. Hay que tener en cuenta que las diferencias entre el valor original y el valor redondeado siempre se hacen en valor absoluto, por eso aunque resulte en valor negativo la diferencia de la distancia, se tomará siempre en positivo.

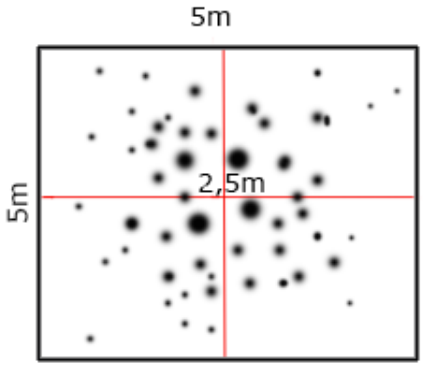

Figura 3.4: Representación de un recuadro $5m \times 5m$ donde los puntos de mayor volumen y más cercanos al centro tienen más importancia en el *MDS*. Para todos los puntos del *MDS* correspondientes a este recuadro, se realizará una media ponderada dando como resultado la altura normalizada con la misma resolución del *MDT*. De esta manera ya podremos usar los modelos correctamente al tener la misma resolución para todos los recuadros de la cuadrícula.

3.1.3. Cálculo de la altura angular de las coordenadas

Una vez normalizado los datos del *MDS*, y habiendo introducido los conceptos del *acimut* y la elevación del sol, ya estamos próximos a calcular si tenemos sombra en un punto geográfico.

La elevación del sol se mide en grados y los valores de las alturas de los edificios y vegetación están medidos en metros. Para saber si tenemos sombra en un punto basta con calcular la **altura angular** en la coordenada que queremos averiguar.

Como ya especificamos, el *acimut* es un ángulo que se forma tomando como referencia el norte con la proyección vertical del observador hacia el sol dependiendo en qué latitud se encuentre. De esta manera para cada *acimut* distinto en una posición geográfica, la elevación del sol irá cambiando, por lo que de acuerdo a la dirección del sol y su elevación en ese momento tenemos que ver en esa dirección todos los edificios y vegetación que podrían darnos o no sombra. En la Figura 1.12 mostramos la representación del *acimut*

y como se refleja en la cuadrícula. Visitando los recuadros contiguos de la línea, como observamos en la figura, para hallar la altura tenemos que calcular la diferencia de la altura del recuadro intermedio del *MDS* y la altura a nivel del suelo del recuadro donde nos encontramos sacado del *MDT* y calcular la **distancia euclídea** a dichos recuadros. Con distancia y altura podemos calcular la altura angular que no es más que el ángulo formado por el observador y el punto más alto del edificio mediante la típica fórmula de los triángulos en trigonometría [9] para obtener la tangente:

$$\tan(\alpha) = \frac{altura\ del\ edificio}{distancia\ euclidea} \tag{3.2}$$

El cual hallando la arcotangente del valor obtenido obtendremos el ángulo y por tanto la altura angular, pero no sólo es suficiente con obtener la altura angular de cada uno de los edificios y accidentes geográficos que nos encontremos por el camino, sino que hay que quedarse con el máximo en la dirección del sol en ese momento. Si dicha altura angular máxima es mayor que la elevación del sol sabremos que en ese punto tendremos sombra, de lo contrario, estaremos expuestos al sol.

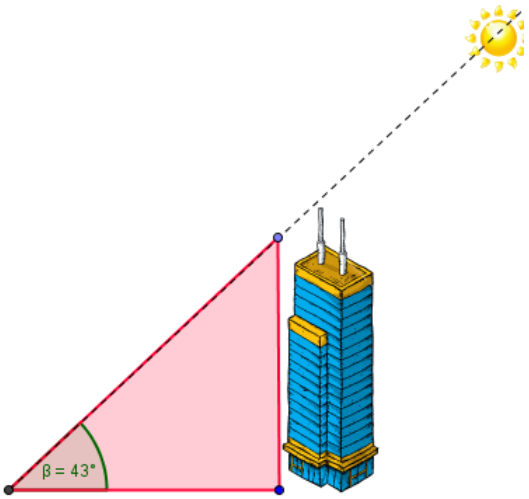

Figura 3.5: Representación gráfica de la altura angular. La altura angular dada por la elevación del sol y la altura angular del edificio las cuales son calculadas respecto a la perspectiva del observador. En este ejemplo nos demuestra que no tenemos sombra.

Por lo tanto, hay que calcular para cada celda la altura angular para todas las posibles direcciones, o *acimuts*, que tome el sol. Hay que tener en cuenta que una variación mínima del ángulo del *acimut* se traduce en un cambio sugerente de la línea desde el observador al sol por lo que tenemos que establecer un incremento del acimut mínimo para guardar dichas alturas máximas angulares. Este incremento lo estableceremos en 5^o por lo que si el ciclo completo solar para un día son 360^o entonces tendríamos 72 valores para cada recuadro de la cuadrícula el cual reduciría bastante el tamaño de los datos.

Para obtener dicho archivo con las alturas angulares usaremos **la ecuación de la recta** la cual nos determinará las líneas a trazar desde nuestra posición relativa hasta los bordes de la cuadrícula, siendo y el valor de la fila y x el valor de la columna procederíamos a calcular los puntos de y de la siguiente manera:

$$y_{destino} - y_{origen} = m(x_{destino} - x_{origen}) \tag{3.3}$$

donde m sería el valor de la pendiente que corresponde a la tangente del *acimut*. Con

esto iteraríamos por cada recuadro calculando la altura angular para el *acimut*. Guardaremos siempre la altura angular máxima por lo que al final de la iteración, correspondiendo con los bordes de la cuadrícula, tendremos guardado la altura angular máxima para ese recuadro o punto geográfico con ese *acimut*.

El aspecto final del archivo de entrada tendrá el siguiente formato:

...

428600 4581840 160.0 26.830187836478 78

428600 4581840 165.0 28.87946235735453

428600 4581840 170.0 30.051413013140785

...

428600 4581815 75.0 46.596100616712256

428600 4581815 80.0 43.62143118671359

428600 4581815 85.0 40.26161038620566

...

Donde los *2* primeros valores son la coordenada en formato UTM, el *3º* valor el *acimut* en esa coordenada y el *4º* la altura angular máxima para esa coordenada y *acimut*.

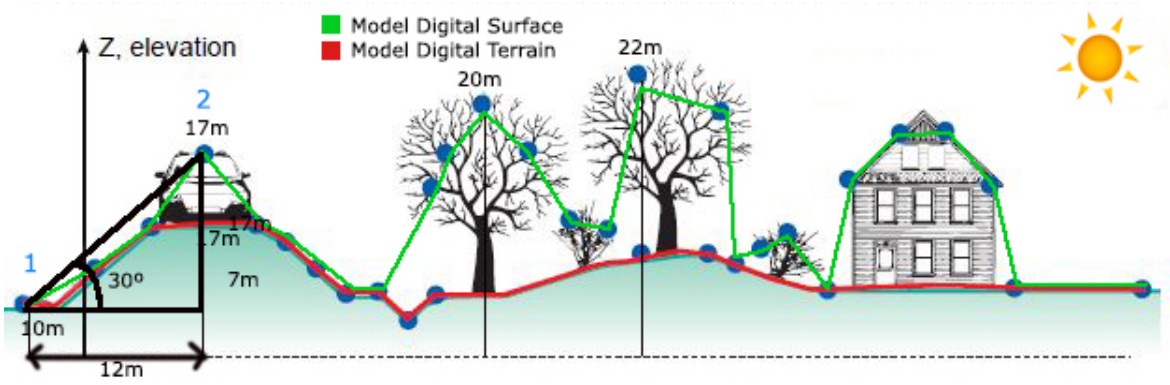

Figura 3.6: Ejemplo gráfico del cálculo de altura angular para un *acimut*. Suponiendo que el sol se encuentra al Este, con un *acimut* de *90º*, y todos los elementos del dibujo con sus respectivas alturas están situados en esta misma dirección, como resultado nos encontraremos en la posición *1*. A partir de ahí tenemos que obtener la altura angular máxima para esa posición. La obtención de este valor se hará iterando por los distintos recuadros calculando la altura angular y comparando con el máximo encontrado hasta ahora. Al final del cálculo tendríamos la altura angular máxima que en este caso correspondería con el coche encima de la colina de la posición *2*. Esta altura angular se ha calculado haciendo la diferencia entre el valor del recuadro del *MDT* de la posición *1* con el valor del *MDS* de cada uno de los recuadros de la línea. Es decir, el cálculo de la altura será *17m* - *10m* = *7m* y calculando la distancia euclídea a ese punto, *12m* en este caso, podemos calcular la tangente y la arcotangente del resultado para obtener la altura angular máxima que en este caso es *30º*.

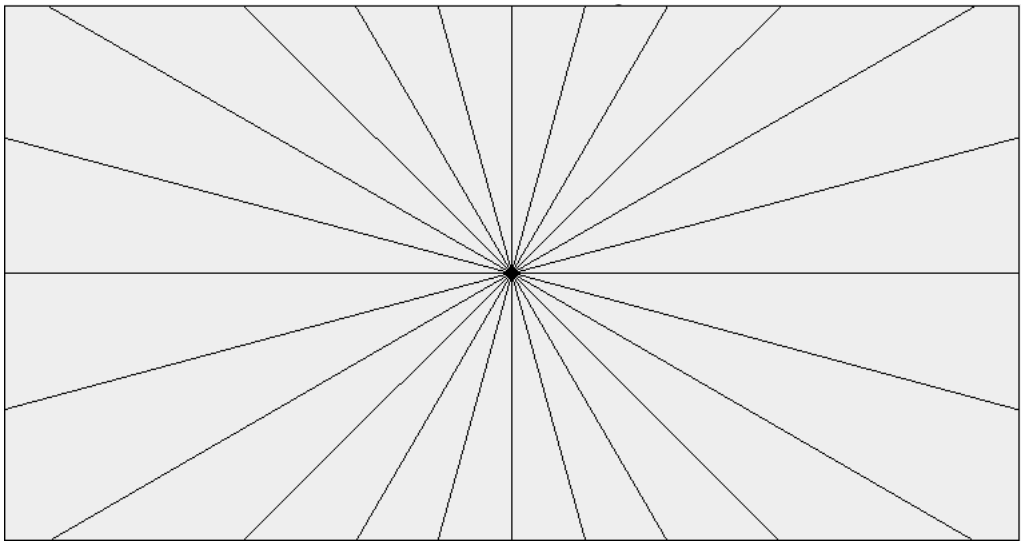

Figura 3.7: Representación de las líneas dado por los distintos *acimuts* en un punto concreto de la cuadrícula. En este caso, para propósitos de presentación se ha tomado los ángulos de los *acimuts* de *15º* en *15º* para mostrar la ecuación de la recta como trazaría las líneas. En este caso, la posición relativa del observador se encuentra en el centro, por lo que todas las *líneas* parten del mismo. Cada línea tendrá su altura angular máxima que se guardará para cada recuadro o posición geográfica.

3.2. Algoritmo de búsqueda

Una vez preparados los datos de las alturas angulares para cada una de las coordenadas geográficas procedemos al algoritmo de búsqueda que utilizará estos datos para calcular la ruta con más sombra.

Para este cálculo se usará un **algoritmo de búsqueda**, en concreto, mediante el ***algoritmo de búsqueda A**** [13] el cual se clasifica como un algoritmo de búsqueda en grafos. El método consiste en buscar un camino en un grafo desde un nodo inicial hasta un nodo final. A diferencia de otros algoritmos similares, $A*$ no se guía fundamentalmente por el resultado de la función ***heurística*** asignada. Esta podría indicar el camino con el coste más bajo, o por el coste real de desplazarse de un nodo a otro. Se podría dar el caso de que sea necesario realizar un movimiento de coste mayor para alcanzar la solución. Es por ello bastante intuitivo el hecho de que un buen algoritmo de búsqueda informada debería

tener en cuenta sendos factores: el valor heurístico de los nodos y el coste real del recorrido hasta ese momento.

La aplicación trabajará con un grafo del área de la ciudad que queremos cubrir. Cada nodo representará la intersección de las calles y las aristas el tránsito entre estos nodos, el cual tendrá un peso específico. Este peso puede ser el tiempo en transitar o el tiempo de exposición solar o una ponderación de ambas de un nodo a otro. Este peso o coste se minimizará siempre y será en función de la velocidad del usuario al desplazarse para calcular el tiempo.

Por lo tanto, el objetivo del algoritmo $A*$ es conseguir soluciones óptimas y ganar en eficiencia reduciendo así el espacio de búsqueda. Para lograr esto, se asigna a cada nodo n un valor por la siguiente ecuación:

$$f(n) = g(n) + h(n) \tag{3.4}$$

Donde cada función significa:

- *g(n)* sería el coste del camino hasta el nodo n.
- *h(n)* sería la heurística del nodo o estimación del coste de un camino óptimo desde el nodo n hasta un estado final.
- *f(n)* sería la estimación del coste total de una solución óptima que pasa por el nodo n.

Otro aspecto a tener en cuenta es lo referente a la heurística del tiempo de exposición solar. La heurística se define como una función que busca un mayor rendimiento en el cálculo estimado de la solución general o la solución óptima. El problema radica en que no podemos determinar con exactitud una posible heurística ideal para estimar el coste de tiempo de exposición solar desde un nodo hasta el nodo final para encontrar la solución óptima a nuestro problema de encontrar la ruta con menor tiempo de exposición solar. Esto es debido a que hay que tomar en cuenta demasiadas variables y cálculos no triviales para estimar la posición del sol e inclinación en un punto intermedio de la ruta para estimación del coste, por lo que al final sólo usaremos una heurística asociada a estimar el coste de la distancia hay. Por este motivo hemos tenido que prescindir de esta implementación.

Esta heurística de la distancia sólo tendrá más relevancia en función de las preferencias del usuario por lo que será máxima si el usuario prefiere una ruta más corta a otra con más sombra. Si por el contrario prefiere una ruta con más sombra, la heurística será casi *0*. Sin embargo, nunca permitiremos que esto ocurra. La razón de que nunca hagamos nula esta heurística para la ruta con más sombra es debido a que una heurística no nula evita expansiones innecesarias en la búsqueda de la ruta desde un nodo cualquiera al nodo de destino. Si lo hacemos nulo ($h(n) = 0$), el algoritmo se comportará como un ***Algoritmo de Dijkstra*** buscando todos los posibles caminos desde un nodo, empeorando de esa manera, el rendimiento del algoritmo. Los costes de las aristas del grafo serán medidos siempre en unidades de tiempo, ya sea el tiempo de distancia o de exposición solar. La heurística será calculada mediante la ***distancia euclídea*** dividido por la velocidad del viandante que supondremos por defecto, *1,33km/h*.

La distancia euclídea es la distancia que habría entre dos puntos en un espacio euclídeo. Por lo tanto, la distancia euclídea entre un punto $P = (p_1, p_2, ..., p_n)$ y $Q = (q_1, q_2, ..., q_n)$ en un espacio euclídeo n-dimensional se define como:

$$d_E(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} \tag{3.5}$$

Nuestro espacio cartesiano se define en $\mathbb{R}^2$, ya que trabajamos con coordenadas geográficas, latitud y longitud de una posición.

3.2.1. Funcionamiento del Algoritmo A* en el grafo

El grafo que usaremos para calcular la ruta está basado en la plataforma ***OpenStreetMap*** *(OSM)*[4]. Esta plataforma posee una base de datos con más de *30GB* de datos geográficos vectoriales de información muy diversa y detallada (carreteras, caminos, edificios, restaurantes, parques naturales...). Otras páginas, como es Mapzen[5], ofrece un servicio online de porciones de datos de dicha base de datos semanalmente. Los datos *OSM* son capturados mediante el uso de dispositivos *GPS*, así como algunos de ellos permiten grabar trazas de rutas. Estas trazas se pueden cargar en el servidor de *OpenStreetMap*.

El formato *OSM* es un formato de archivo *XML* propio de *OpenStreetMap*. En dicho formato se establece al principio los límites del área así como los nodos con sus respectivos identificadores y posiciones geográficas.

Dicho esto, vamos a presentar el procedimiento en pseudocódigo de como se comportaría el *algoritmo A**, así como funciones y métodos auxiliares para este cálculo:

[4]Grafos callejeros a través de la plataforma *OpenStreetMap*: `https://www.openstreetmap.org/`
[5]*Metro extracts City-sized portions of OpenStreetMap* `https://mapzen.com/data/metro-extracts`

Algorithm 1 Algoritmo A*

```
1: procedure aStar(origen : Node, destino : Node)
2:     nodoActual = null
3:     listaAbierta = origen
4:     repeat
5:         nodoActual = nodoConMenorF(listaAbierta)
6:         listaCerrada->añadir(nodoActual)
7:         if nodoActual == destino then
8:             return nodoActual
9:         else
10:            listaAdyacentes = getAdyacentes(nodoActual)
11:            repeat
12:                adyacente = listaAdyacentes->sacar()
13:                if adyacente NOT IN listaAbierta && adyacente NOT IN listaCerrada
   then
14:                    setCostesNodo(nodoActual, adyacente, destino)
15:                    adyacente -> setPadre(nodoActual)
16:                    listaAbierta->añadir(adyacente)
17:                else if adyacente IN listaAbierta then
18:                    if adyacente -> getG() < nodoActual -> getG() then
19:                        setCostesNodo(nodoActual, adyacente, destino)
20:                        adyacente -> setPadre(nodoActual)
21:                    end if
22:                end if
23:            until listaAdyacentes == ∅
24:        end if
25:    until listaAbierta == ∅
26: end procedure
```

Algorithm 2 Función para obtener nodo con menor valor *f(n)*

```
1: function nodoConMenorF(listaDeNodos : List of Node)
2:     resultado = listaDeNodos[0]
3:     j = 0
4:     for i = 1 UNTIL i < tamaño(listaDeNodos) do
5:         if listaDeNodos[i] -> getF() < resultado -> getF() then
6:             resultado = listaDeNodos[i]
7:             j = i
8:         end if
9:     end for
10:    listaDeNodos->sacar(j)
11:    return resultado
12: end function
```

Algorithm 3 Procedimiento para establecer los costes de *f(n)*, *g(n)* y *h(n)*

```
1: procedure setCostesNodo(padre : Node, hijo : Node, destino : Node)
2:     costeH = getCosteHeuristica(hijo, destino)
3:     costeG = getCosteG(padre, hijo)
4:     hijo -> setG(padre -> getG() + costeG)
5:     hijo -> setH(padre -> getG() + costeH)
6:     hijo -> setF(hijo -> getG() + hijo -> getH())
7: end procedure
```

El algoritmo $A*$ (1) añade al comienzo el nodo origen en la ***lista abierta***, la cual es una lista que contiene todos los nodos que quedan por evaluar. Seguirá repitiendo lo mismo mientras la lista no esté vacía o el nodo actual sea el nodo destino. De toda esta lista se saca aquel nodo con menor valor f para establecerlo después como el nodo actual. Seguidamente sacamos el nodo de la lista abierta para meterlo a la ***lista cerrada***, que es una lista de nodos que ya han sido tratados y no necesitan ser revisados.

Una vez comprobado que el nodo actual es distinto del nodo destino, tomamos todos los nodos adyacentes al actual. Por cada uno de ellos se comprueba que no se encuentre ni en la lista abierta, ni en la cerrada o que por el contrario se encuentre únicamente en la lista abierta.

Si este nodo adyacente no está estas listas, se le asigna los costes para *f(n)*, *g(n)* y *h(n)*, siendo n el nodo que estamos evaluando. Hacemos que el nodo adyacente apunte al nodo actual, el cual es su nodo padre, y añadimos este nodo adyacente a la lista abierta.

Si por el contrario, este nodo adyacente ya se encuentra en la lista abierta, hemos de revisar si el camino para este nodo es mejor usando el coste g. Si este coste es menor significa que será un camino mejor, de ser así, cambiamos el padre del nodo adyacente (por estar en la lista abierta se supone que ya tenía asignado un padre) tomando como padre el nodo actual y recalculando los costes de *f(n)*, *g(n)* y *h(n)* para este nodo.

Este proceso, como dijimos, acabará cuando se tome para explorarlo de la lista abierta el nodo destino por lo que habremos dado con el camino, y por lo tanto, con la solución. En caso contrario, si la lista está vacía, significará que no se ha encontrado el camino, por lo tanto, no se ha encontrado la solución.

La ruta se construirá siguiendo los nodos padres a los que apuntan desde el nodo destino al nodo origen.

La función *nodoConMenorF* (2) se encarga de devolver el nodo con menor valor *f(n)*. Su implementación consiste en recorrer la lista de nodos y devolver aquel con el menor valor de *f(n)*. Antes de devolver el nodo, éste es eliminado de la lista abierta.

El procedimiento *setCostesNodo*(3) se encarga de establecer los costes para *f(n)*, *g(n)* y *h(n)* necesarios para calcular la ruta con mínimo coste. En dicha función de evaluación, *g(n)* representa el coste real del camino recorrido para llegar al nodo destino y *h(n)* representa el valor heurístico del nodo a evaluar desde el actual hasta el nodo de destino.

Por lo tanto y teniendo en cuenta lo anterior, nuestro valor *g(n)* será el tiempo en segundos desde el nodo padre hasta el nodo hijo (n). Es decir, dejamos constancia del camino real que nos llevará al nodo hijo. Asimismo, el valor *h(n)* ha de ser un valor de estimación del tiempo restante para llegar al nodo destino.

De esta forma, solo nos queda asignar al nodo hijo el coste g que ya tuviese su nodo padre más el coste g aquí calculado. De igual forma para asignar el coste h al nodo hijo hemos de tener en cuenta la acumulación que tiene el coste g del nodo padre al que le hemos de sumar el nuevo valor h calculado del hijo. Luego al nodo hijo se le asigna el coste *f*, el cual es la suma de g y h.

Hay que tener en cuenta que los algoritmos citados anteriormente describen de una forma general como se comporta el algoritmo *A**. Sin embargo, la plataforma *OTP* utiliza un *Heap* para que al sacar aquellos nodos con coste mínimo su complejidad sea *O(1)* en vez de *O(n)* como podemos observar en la función *nodoConMenorF*, por lo que hay que tener en cuenta que estas funciones auxiliares al algoritmo *A** no serían usadas en esta plataforma.

3.3. La plataforma OpenTripPlanner para el cálculo de la ruta

Una vez introducidos los datos de entrada de alturas, los conceptos básicos del algoritmo de búsqueda y el funcionamiento del *algoritmo A** podremos introducir la plataforma que se encargará de toda la lógica del cálculo de ruta.

En un principio se pensó en la idea de partir de cero en la implementación del algoritmo de búsqueda así como el manejo de grafos para calcular la ruta. Pero debido a la com-

plejidad que supondría, sobretodo esto último y el manejo de los datos de altura del área que queremos cubrir (que suponen del orden de unos *500MB*), lo hacen bastante inviable trasladarlo a una aplicación móvil. Por todo esto y por tal de no alejarnos del verdadero propósito que supone el proyecto, hemos decidido usar una plataforma que hace uso de los grafos *OSM* e implementa el *algoritmo A** con la heurística de la distancia euclídea ya citados en capítulos anteriores. Sólo será necesario modificar la funcionalidad del algoritmo e introducir los módulos necesarios para este cálculo de ruta en concreto.

Por lo tanto, se ha decidido a la creación de una arquitectura de *cliente/servidor* [1] que soporte las peticiones de cálculo de rutas a través de la aplicación móvil para mostrarlo en pantalla. Dicha plataforma es ***OpenTripPlanner***.

OpenTripPlanner (*OTP*) [6] es una plataforma *Open Source multi-modal* y *multi-agencia* para planificar viajes. Lanzado en 2009, el proyecto ha suscitado una próspera comunidad de usuarios y desarrolladores recibiendo soporte incluso de agencias públicas, *startups* y empresas de transporte. *OTP* ha sido implementado en muchas partes del mundo e incluso es el motor para la creación de rutas detrás de muchas aplicaciones populares en *smartphones*.

Esta plataforma está basada en una arquitectura *cliente-servidor* que suministra una interfaz web para calcular rutas mediante un mapa y también una *API* (basada en servicios *REST* [10] para aplicaciones de terceros). *OTP* se constituye de datos abiertos de tipo ***General Transit Feed Specification*** *(GTFS)*[7] para el tránsito en las distintas moda-

[6] *OpenTripPlanner*: `http://www.opentripplanner.org/`
[7] *General Transit Feed Specification*: `https://en.wikipedia.org/wiki/General_Transit_Feed_`

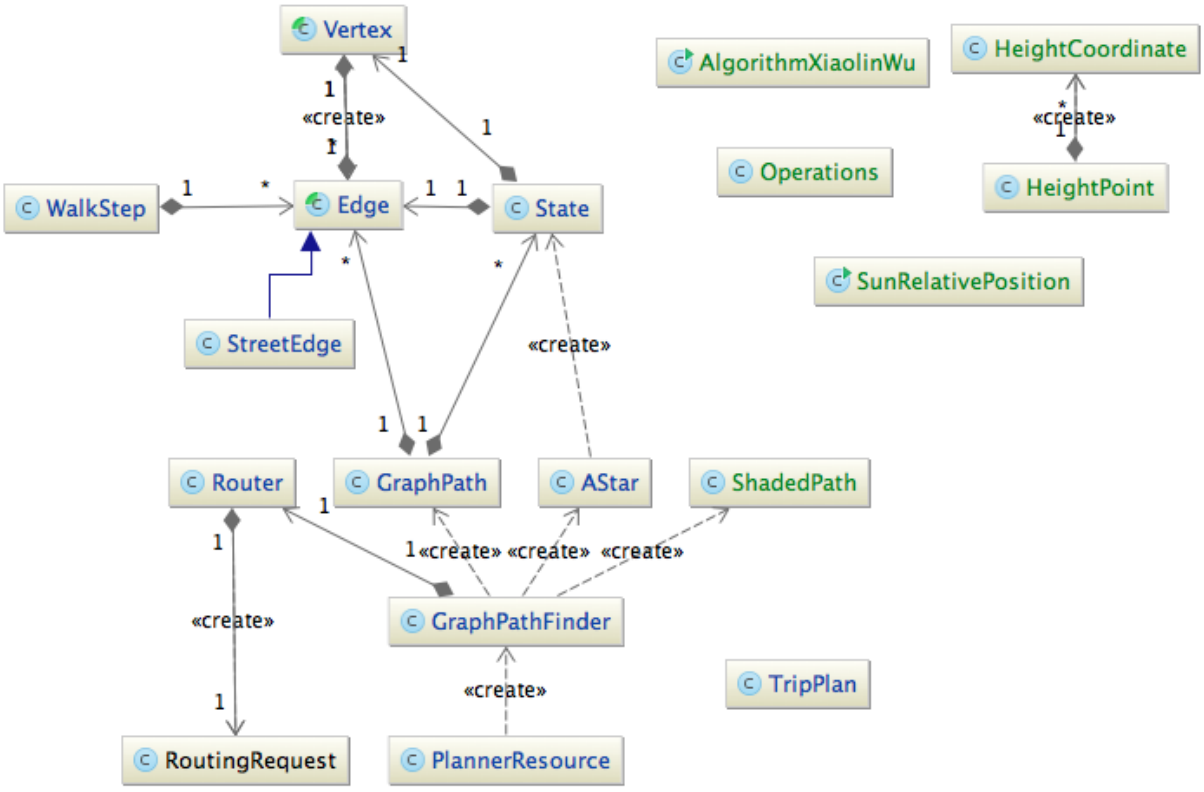

Figura 3.8: Diagrama de clases que usaremos en la plataforma *OTP*.

lidades existentes y *OpenStreetMap* para las redes callejeras.

OTP cuenta con una serie de paquetes para el cálculo de rutas. Nos fijaremos, en especial, en el paquete ***routing*** que contiene toda la lógica para el cálculo de rutas.

A continuación estableceremos el diagrama de las clases que intervendrán en el cálculo de la ruta:

`Specification`

Cuando el usuario realiza una petición a través de la aplicación móvil, introduciendo sus preferencias deseadas, *OTP* la procesará mediante la clase ***PlannerResource***. Una vez ejecutada la petición, la respuesta es devuelta en formato *XML*, *JSON* o texto plano dependiendo de la configuración del cliente.

En esta clase se establece una instancia de la clase ***Router***, la cual define la configuración de la ruta para el grafo de un área específica que hayamos elegido. A esta instancia, en su creación, se le pasará otra instancia de la clase ***RoutingRequest*** que contendrá los parámetros establecidos por el usuario para el cálculo de la ruta. Es en esta clase donde añadiremos la variable *shadedWalk* para el peso o relevancia que tendrá la sombra en la ruta que será asignada entre *0* y *10*. Con esto ya podremos introducir el parámetro *shadedWalk* en la petición para poder capturarlo y poder procesarlo.

Con el objeto de la clase ***Router*** se procede a instanciar un objeto de la clase ***GraphPathFinder***. Esta clase contiene la definición para construir el árbol de búsqueda con el camino más corto a través de la clase ***AStar*** para nuestro grafo asignado.

AStar calcula la ruta haciéndose servir de un conjunto de estados de la clase ***State***, que representa un estado del grafo de estados del *algoritmo A**. Por cada estado se asocia un ***Edge***. Cada ***Edge*** posee un par de nodos de la clase ***Vertex*** con el origen y el destino de cada arista que es la que representa la calle por donde se transita. Es en la clase ***StreetEdge*** (que hereda de ***Edge***) donde se establecerá el coste de ir de un nodo a otro.

Cada nodo de la clase ***Vertex*** posee un conjunto de datos de ***StreetEdge*** que parten y llegan a dicho nodo. Esta clase representa los segmentos de la calle y donde se realiza el coste de atravesar cada una de ellas. Aquí realizaremos las modificaciones pertinentes para asignar la ponderación resultado de la relevancia que haya querido dar el usuario a la sombra en su ruta. El cálculo de la ponderación vendrá dado por la siguiente fórmula:

$$coste = peso_{sombra} * tiempo_{exposiciónsolar} + (10 - peso_{sombra}) * tiempo_{duración} \tag{3.6}$$

Si el usuario asigna el máximo de peso de la sombra, en este caso 10, no se tendrá en cuenta el tiempo de la duración hasta el destino. Si por el contrario se asigna 0, no se tendrá en cuenta el tiempo de exposición solar para calcular la ruta. Este cálculo se hará

mediante la clase ***ShadedPath*** con el método *formulatePonderate* que recibe el peso de la sombra, el tiempo de exposición solar y el tiempo que se tarda en atravesar la calle.

Una vez establecido el coste, la clase ***AStar*** mediante una estructura de datos de tipo *BinHeap*, guardará los costes para cada estado en dicha estructura. Esta se comportará como una lista abierta que realizará el comportamiento que describimos en la sección 3.2.

3.3.1. Cálculo del tiempo de exposición solar

La característica del cálculo del tiempo de exposición solar (que sirve en la fórmula de ponderación señalada antes) es uno de los núcleos de la aplicación. Se procederá a explicar el procedimiento del cálculo en esta sección.

La clase ***StreetEdge*** es la que se centra en el cálculo del coste de atravesar un segmento de calle. Es aquí donde asignaremos el valor del tiempo de exposición solar. Para ello nos haremos servir de la clase ***ShadedPath*** que es la que se va a encargar de computar toda la lógica referente al cálculo de la sombra. Concretamente, nos enfocaremos en el método *calculateTimeSunLightExposureBetweenTwoVertexes*.

Este método se encargará de calcular el tiempo de exposición solar en segundos entre dos nodos. Estos poseen una latitud y longitud unidos por un segmento. Debido a que los datos de entrada están en coordenadas *UTM* (para cada par de nodos origen y destino) debemos convertir las coordenadas a este formato para poder consultar los datos de las alturas angulares máximas cargados en el arranque del servidor. Estos datos serán almacenados en una matriz bidimensional de la clase ***HeighCoordinate***, en la que cada posición de esta matriz representará una coordenada geográfica en *UTM* y para cada una de estas coordenadas, se guardará una estructura de datos de par *<clave, valor>* donde la clave es el *acimut* y el valor la altura angular máxima para dicho *acimut*.

Con el origen y el destino de nuestro segmento y con la clase ***SunRelativePosition*** le pasamos como parámetros la latitud, la longitud, la hora y la fecha en la que nos encontramos. Así obtendremos con exactitud el valor del *acimut* y elevación solar en estos puntos. Pero, ¿Cómo calculamos estos valores para los puntos intermedios del segmento?. Un segmento puede tener una distancia importante (a veces incluso de cientos de metros) y en

ese tramo pueden haber edificios y vegetación. ¿Cómo sabemos cuáles puntos intermedios tenemos que revisar para calcular si hay sol o no a través de todo este tramo?. La solución la encontramos con el ***Algoritmo de Xiaolin Wu***.

El ***Algoritmo de Xiaolin Wu*** [12] es un algoritmo usado para minimizar el *aliasing*, se utiliza para dibujar líneas con bordes suaves en pantallas digitales.

Figura 3.9: Aplicación del *Algoritmo de Xiaolin Wu*, a la izquierda vemos como la línea posee un *antialiasing* generado por el algoritmo y a la derecha una línea sin *antialiasing*

¿Y en qué nos beneficiaría el uso de este algoritmo?. Este algoritmo lo usaremos para pasarle como entrada un nodo origen y otro nodo destino para establecer el segmento formado por todas las coordenadas entre estos dos nodos. De esta manera podemos saber que coordenadas *UTM* se encuentran entre ambos nodos. En estas coordenadas intermedias calculamos la elevación del sol y el *acimut* a esa hora del día y la comparamos con la altura angular máxima para ese *acimut*. Si la altura angular máxima es mayor que la elevación, tenemos sombra, de lo contrario, tenemos sol. En el caso de tener sol tendremos que calcular el tiempo de exposición solar que nos llevaría caminar hasta la siguiente coordenada del segmento, en función de la velocidad del usuario. Por lo que al final de iterar por todos los puntos del segmento tendremos el tiempo de exposición solar total para ese segmento.

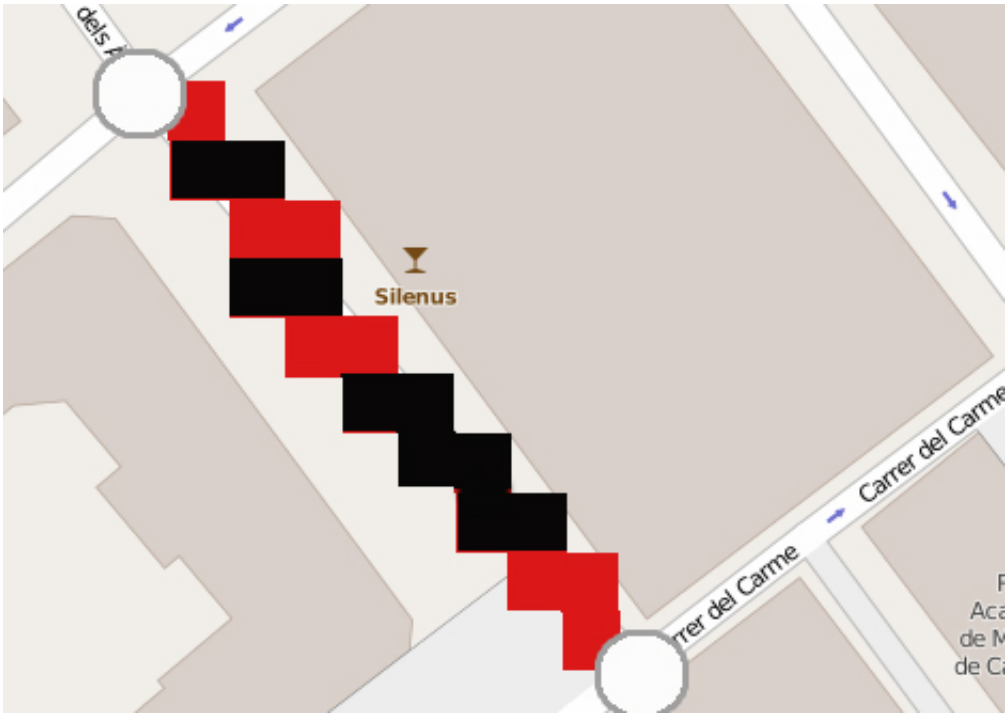

Figura 3.10: Ejemplo del funcionamiento del método ***calculateTimeSunLightExposureBetweenTwoVertexes*** con el *algoritmo de Xiaolin Wu*. El algoritmo traza una línea imaginaria entre el nodo origen y nodo destino. Para cada coordenada intermedia determinamos si hay sombra. En los recuadros negros tenemos sombra, mientras que en los rojos no. Entonces para cada recuadro rojo se calcula el tiempo que le llevaría al usuario llegar al recuadro siguiente en función de su velocidad y la distancia a dicho recuadro. Al final el tiempo de exposición solar viene dado en este caso por el tiempo de recorrer todos estos recuadros rojos a lo largo del segmento.

3.3.2. Plan de viaje

Después de que el algoritmo haya calculado la ruta de acuerdo a las preferencias del usuario, *OTP* establecerá nuestro plan de viaje mediante la clase ***TripPlan*** pasánsole como parámetro la ruta generada.

Esta clase construye el trayecto en función de una serie de pasos o etapas representados

por la clase ***WalkStep***. Cada objeto de esta clase puede estar compuesto de uno o más ***StreetEdge***. A la hora de generar el peso de la distancia total, se hace una suma de todas las distancias de estos ***StreetEdge*** asociados a un ***WalkStep***. Antes de reproducir los resultados en pantalla será también necesario sumar los tiempos de exposición al igual que la distancia para poder mostrarlo más adelante en la aplicación móvil.

Ya realizado el plan de viaje, este es devuelto como una respuesta *serializada* para ser usada y manipulada por cualquier aplicación de terceros o por el mismo servidor web desplegado. A continuación vamos a mostrar una serie de capturas de pantalla de los resultados del cálculo de la ruta en función del peso de la sombra que establezca el usuario. Estas capturas han sido tomadas por la misma aplicación web. Se han añadido los *widgets* necesarios, como es el selector del peso de la sombra, en la aplicación para poder realizar las peticiones.

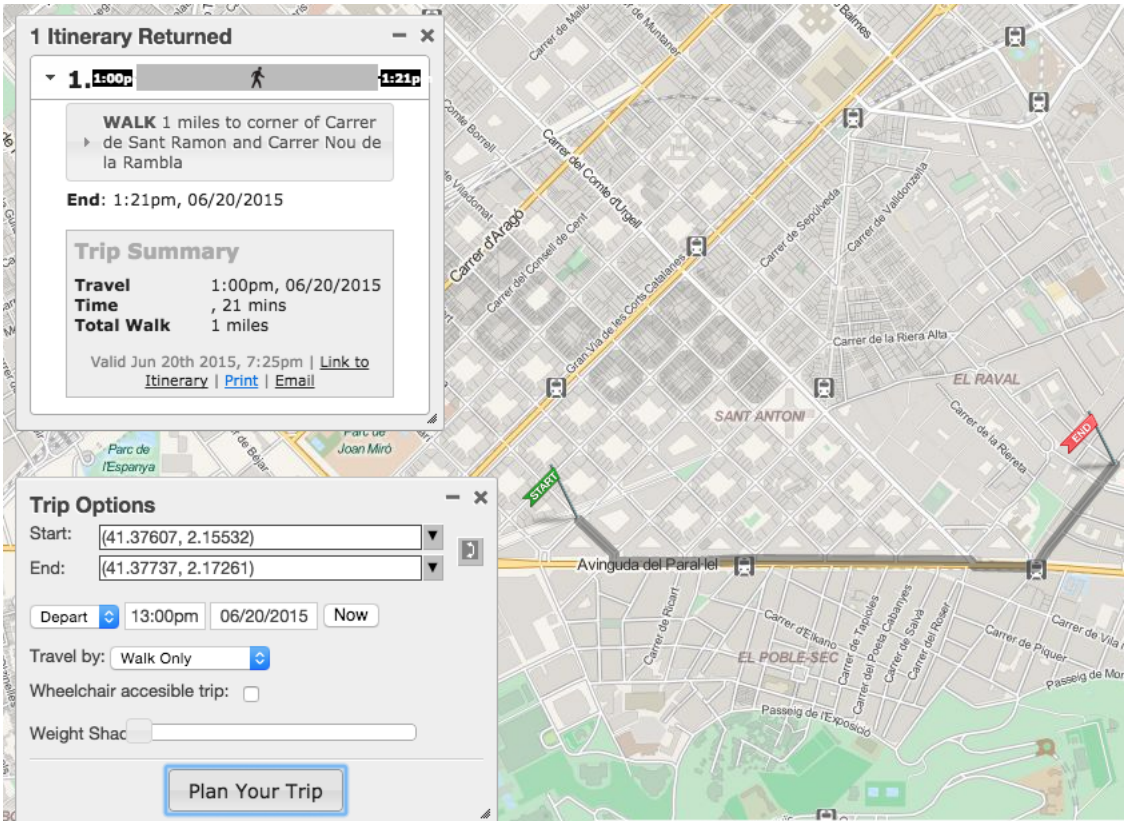

Figura 3.11: Resultado del cálculo de trayecto. En la esquina inferior izquierda tenemos el selector que determina el peso de la sombra en nuestro trayecto. En este caso tenemos asignado un peso de *0* por lo que se dará más importancia al trayecto más corto. En la imagen el tiempo total del viaje a pie es de *21* minutos. El sol se encontrará al sur (esquina inferior de la imagen).

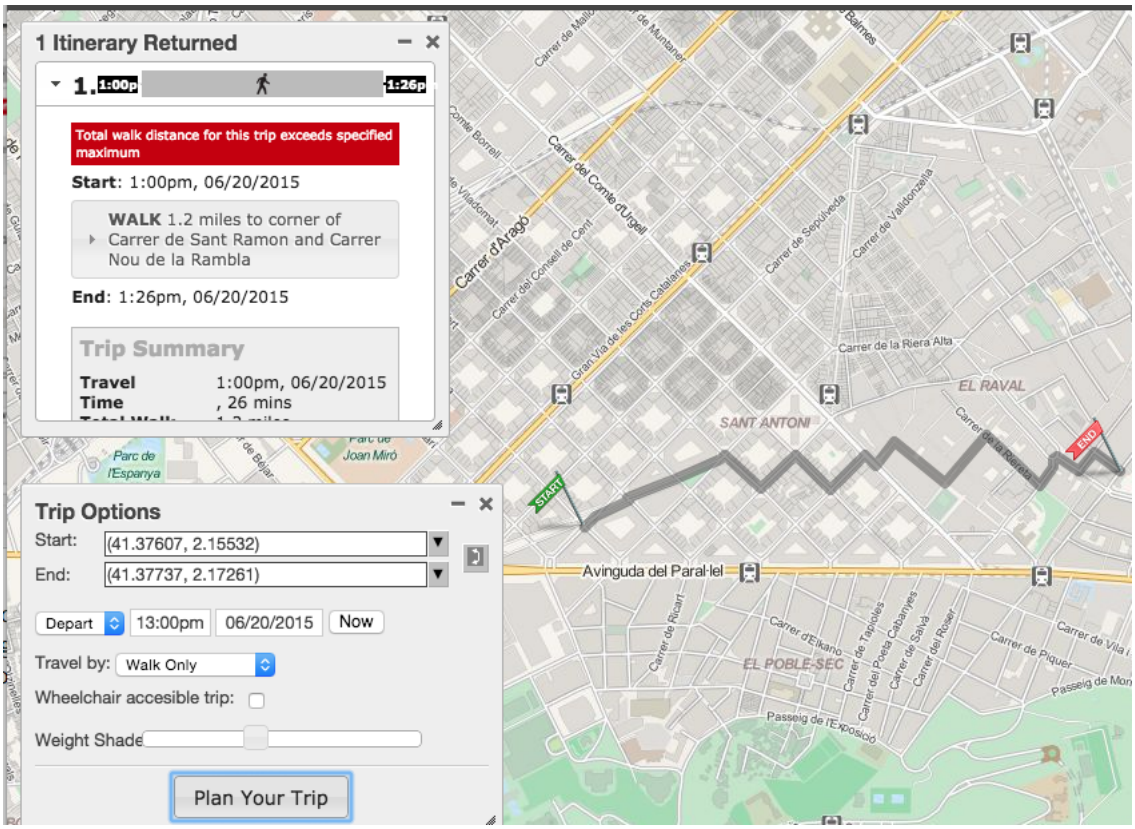

Figura 3.12: Para este caso establecemos el peso en *4* y la hora a las *13:00*, donde el sol está casi en su punto más alto. Observamos como el trayecto ahora tiene una duración de *26* minutos. El sol se encontrará al sur (parte inferior de la imagen).

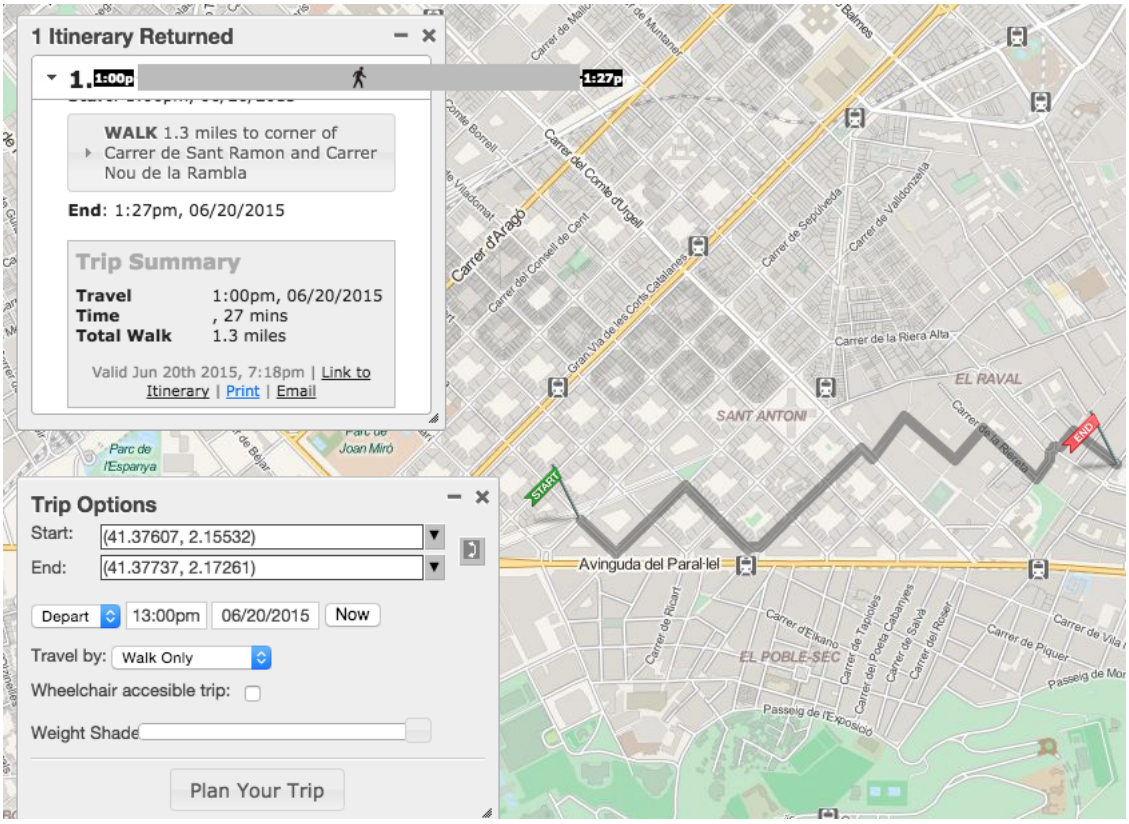

Figura 3.13: Para este caso establecemos el peso máximo en *10* y la hora a las *13:00*. Observamos ahora como el trayecto tiene una duración de *27* minutos. Esto supone un incremento de *1* minuto respecto al resultado anterior. Se puede apreciar que se intenta bordear más la ruta yendo por las calles donde hay edificios altos. Por lo que podemos determinar que está yendo por la ruta donde hay más sombra en ese instante. El sol se encontrará al sur (parte inferior de la imagen).

Capítulo 4

La aplicación móvil

Como comentamos en anteriores secciones, la arquitectura del proyecto será del tipo *cliente-servidor*. La aplicación móvil será nuestro cliente y la parte servidor será el servidor web que esté ejecutando *OTP*. Esta aplicación móvil realizará una petición al servidor para calcular la ruta mediante servicios *REST* [10]. Concretamente ejecutará una petición *GET* en la cual se le pasará como parámetros la latitud, longitud, hora, fecha, el modo de tránsito (en nuestro caso *WALK*) y el peso de la sombra. Una vez procesada la solicitud, el servidor devolverá la respuesta en *JSON* o *XML*, dependiendo de la configuración de cabecera, que para nuestro caso será *JSON*. Por lo que esta respuesta será *deserializada* y usada para realizar las operaciones y así mostrar la ruta e información relevante al usuario en la aplicación.

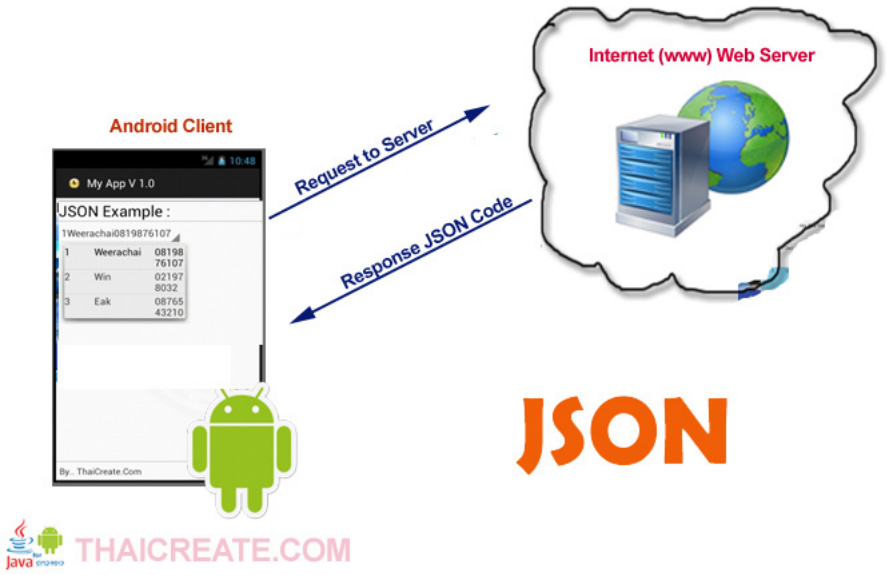

Figura 4.1: Arquitectura de la aplicación.

4.1. La estructura de la aplicación

La aplicación *Android* contará con una interfaz de ***Google Maps***. El usuario en todo momento podrá elegir el origen y el destino de su ruta manteniendo pulsado el punto del mapa. Se seleccionará a través de un diálogo informativo que se abrirá al realizar dicha acción. Una vez hecho esto, automáticamente la aplicación realizará una petición al servidor donde está ejecutándose *OTP*. Este iniciará el cálculo para más tarde devolver la respuesta en formato *JSON*. La aplicación móvil analiza la respuesta y la *deserializa* mediante la biblioteca *GSON*, instanciándola en un objeto de la clase *OTPRouteShaded*. Esta clase ha sido generada siguiendo el código de prácticas denominado *Plain Old Java Object* (*POJO*)[1], el cual nos será de utilidad a la hora de *deserializar* la respuesta que nos envíe el servidor y podamos trabajar con ella más cómodamente para mostrar la ruta.

Para construir la ruta utilizaremos la clase ***Step*** que sería la clase análoga a ***WalkStep*** en el lado del servidor. Esta clase nos define cada etapa de la ruta construida y está formada por los siguientes atributos: la distancia, tiempo de exposición solar, nombre de la calle,

[1]Plain Old Java Object: `https://en.wikipedia.org/wiki/Plain_Old_Java_Object`

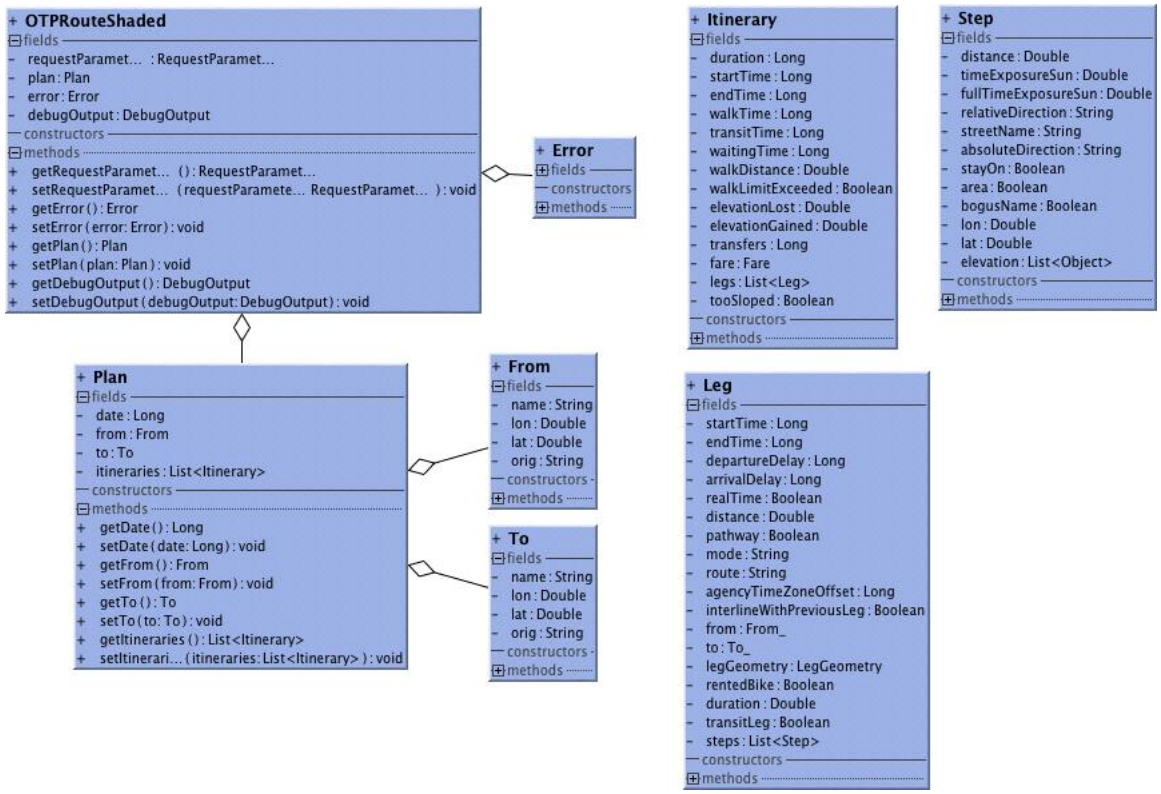

Figura 4.2: Diagrama de clases en las que se *deserializa* la respuesta en formato *JSON*.

latitud y longitud en la que empezaría la etapa. Cada una representa un segmento de la ruta. Se darán indicaciones de como el usuario debe moverse por cada una de ellas hasta llegar al destino.

La información de las mismas también será necesaria para poder crear y posicionar los marcadores en el mapa para visualizar la ruta. Cada marcador (que simboliza el comienzo de cada etapa) estará oculto. Cuando el usuario necesite saber información sobre esa etapa seleccionará el *item* que le corresponda. Seguidamente se mostrará el marcador. Si lo seleccionamos veremos información del tiempo y del porcentaje de exposición solar.

A la hora de trazar las líneas entre marcadores se usarán ***Polylines*** que trazan una línea entre dos puntos con su latitud y longitud respectivas.

Figura 4.3: Ruta representada con el uso de *polylines*. Los marcadores: origen y destino salen visualizados mientras que los intermedios están ocultos por tal de no sobrecargar la interfaz.

4.1.1. La *Action Bar*

Uno de los aspectos más importantes de la aplicación es la barra de acciones o *ActionBar*, introducida en la versión *3.0* de *Android*. En ella incluimos las diversas opciones relacionadas con las preferencias para calcular la ruta.

Las opciones presentes en la barra serán:

- Establecer la hora a través de un selector de hora.
- Establecer la fecha a través de un selector de fecha.
- Recalcular la ruta que ya fue calculada con anterioridad.
- Establecer el peso o relevancia que tendrá la sombra en la ruta a través de un deslizador en el que el usuario podrá elegir desde *0* hasta *10*.
- Mostrar información general de la ruta, así como cada una de las etapas de la misma.

- Establecer la dirección al servidor *OTP* al que se le hará las peticiones para cálculo de rutas.

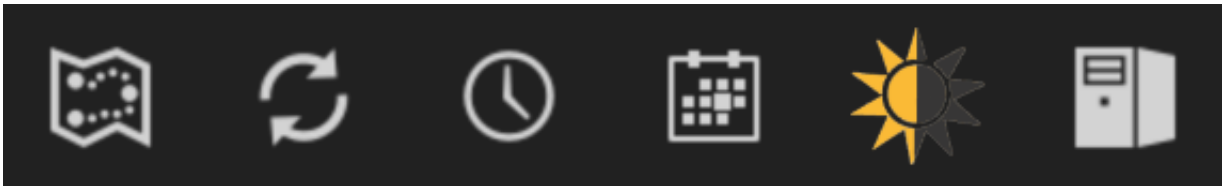

Figura 4.4: Opciones en el menú *ActionBar* de izquierda a derecha: la primera opción nos permite mostrar una lista con todas las etapas de la ruta e información general, la segunda para recalcularla, la tercera para establecer la hora de inicio, la cuarta para establecer la fecha de inicio, la quinta para asignar la relevancia de la sombra y la última opción para establecer la dirección del servidor donde se encuentre ejecutando *OTP*.

4.1.2. Información de la ruta y las distintas etapas

Para cada ruta necesitamos mostrar al usuario la siguiente información: tiempo de exposición y porcentaje solar, distancia y dirección a tomar para cada etapa. Así también como información general de la ruta: el tiempo de distancia, el tiempo de exposición solar total y la distancia total. Para ello se ha optado por usar una ventana deslizante que se activará al pulsar la primera opción de la barra de acción.

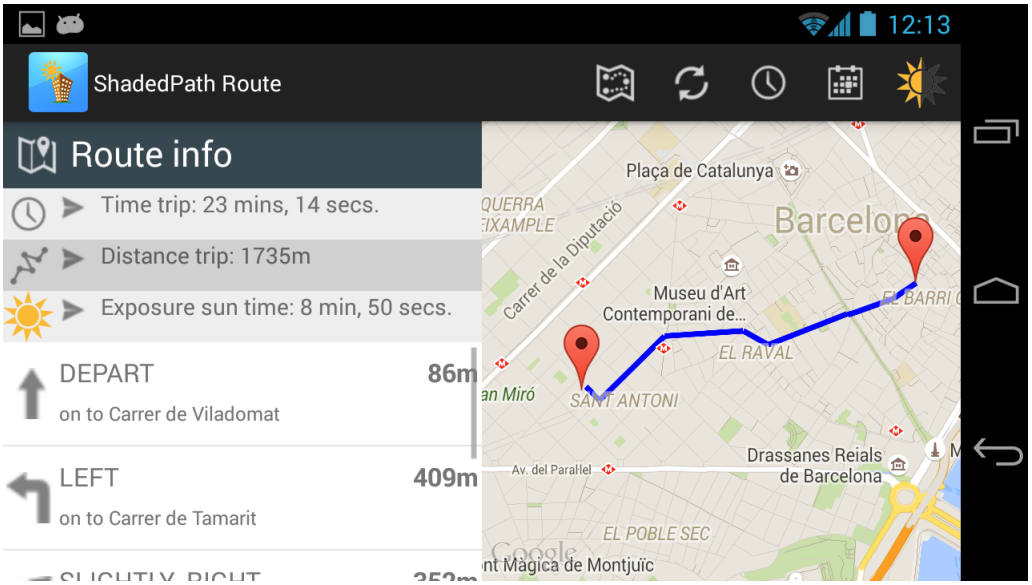

Figura 4.5: Captura de la aplicación donde puede verse la información de la ruta. Arriba tenemos el tiempo del trayecto así como el tiempo de exposición solar. Seguidamente tenemos la distancia en metros y una lista con todas las etapas de nuestra ruta y dirección que debemos tomar en cada una.

Al seleccionar una etapa se hará visible el marcador asignado en el mapa. Se nos mostrará un diálogo de información para ese marcador con el tiempo de exposición solar y porcentaje del mismo. Para calcular dicho porcentaje hemos realizado la división entre el tiempo durante el que hay exposición solar dividido entre el tiempo total de la etapa y multiplicado por 100.

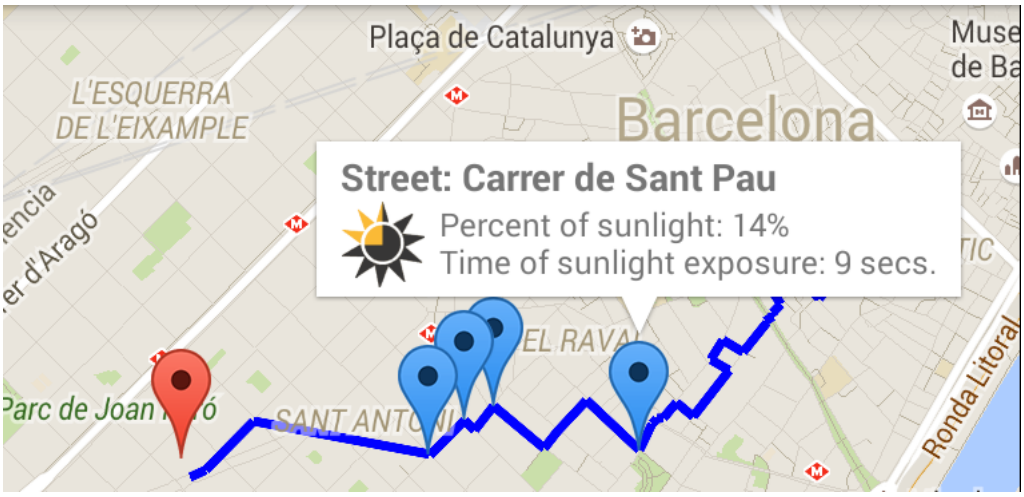

Figura 4.6: En la etapa *Carrer de Sant Pau* tenemos un tiempo de exposición solar de *9* segundos que representa un *14 %* de la exposición solar en esa etapa.

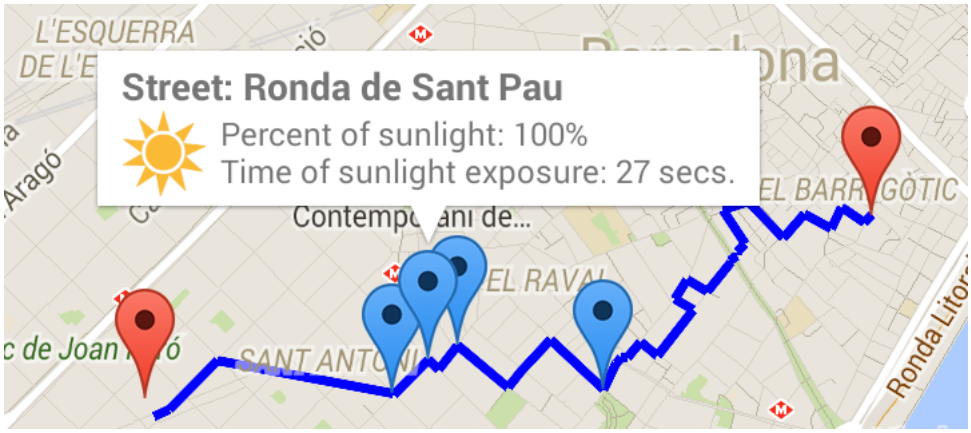

Figura 4.7: En este otro ejemplo: *Ronda de Sant Pau* tenemos un tiempo de exposición solar de *27* segundos que representa un *100 %* de la exposición solar en esa etapa. En este caso la exposición solar es máxima.

4.1.3. Notificación de errores y excepciones

Como en toda aplicación, es necesario tratar los errores para evitar bloqueos innecesarios. Estos errores serán debidos a imprevistos que hayan ocurrido en el servidor durante el proceso de cálculo de la ruta. Otro caso sería si no obtuviéramos una respuesta en un tiempo por defecto y sea necesario notificárselo al usuario.

Los diferentes avisos de errores serán los siguientes:

- Notificación al usuario de que se ha producido un error al establecer el origen o el destino de la ruta fuera del área que soporta la aplicación.
- Notificación al usuario que intenta calcular la ruta sin haber establecido el origen y el destino.
- Notificación al usuario de que el servidor está tardando demasiado en responder o que no ha podido establecerse la conexión con el servidor.

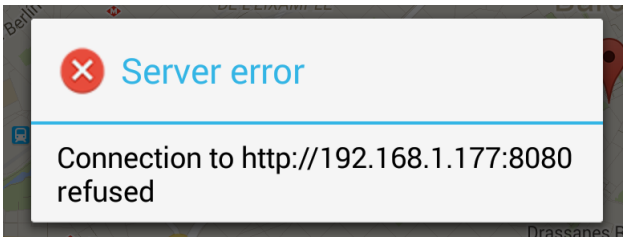

Figura 4.8: Error mostrado si el servidor no se encuentra disponible al calcular la ruta.

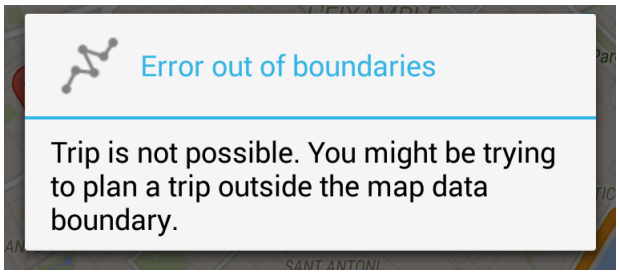

Figura 4.9: Error mostrado si se intenta calcular la ruta fuera del área que soporta la aplicación servidor.

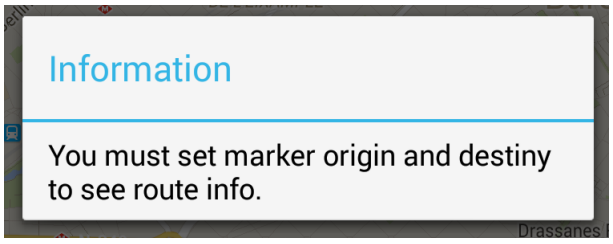

Figura 4.10: Error mostrado si se intenta calcular la ruta o mostrar la ruta si aún no se ha establecido el origen y el destino.

Capítulo 5

Conclusiones

5.1. Valoraciones finales

El campo relacionado con la búsqueda de rutas óptimas se encuentra en la actualidad en constante evolución. Cada día se incorporan nuevas opciones a tener en cuenta en el cálculo de rutas que satisfagan las necesidades del usuario. La introducción de una nueva opción, como es el cálculo de la ruta con más sombra, la convierten en una posibilidad a tener en cuenta. Sobretodo en un escenario como son las ciudades inteligentes. En ellas se buscan continuamente facilitar la movilidad a los ciudadanos.

La realización de este proyecto ha implicado documentarse en el área de astronomía en todo lo relativo al sol, principalmente el *acimut* y la elevación, como ya introdujimos en la sección de ***Preliminares*** (1.3). Una gran parte del proyecto se ha centrado en el estudio de los modelos digitales de terreno y superficie. La interpretación de estos datos son primordiales para poder calcular las rutas. Además no hay que olvidarse del entendimiento del formato *UTM* y el sistema geodésico *WSG84* que han sido claves para poder realizar esta parte.

También se ha destinado un tiempo en asimilar el funcionamiento del grafo y el algoritmo de búsqueda. Al igual que todos los mecanismos intermedios que usa la plataforma *OTP* desde que se realiza la petición de cálculo de ruta hasta que es devuelta por la aplicación. Todo esto ha ahorrado trabajo en una parte del proyecto al no tener que programar el

cálculo de ruta desde *0*, pero también se ha tomado un tiempo en entender la lógica de esta plataforma para poder realizar las extensiones y modificaciones necesarias de acuerdo a las necesidades del proyecto.

Este *TFG* me ha ayudado a comprender las bases de la realización de un proyecto de gran envergadura. Así como a entender en profundidad las distintas fases de las que se compone un proyecto. Fases que parten desde la captura de los requisitos y el estudio de los mismos hasta su desarrollo e implementación.

Como valoración final considero que este proyecto podría tener una utilidad importante para la comunidad. Actualmente no existen muchos cálculos de ruta que tengan variables para su confort. En general son enfocados a minimizar el coste a nivel de tiempo o dinero. Esto último es más presente en rutas en vehículo. Por lo que siempre que se disponga de los modelos de superficie y terreno de un área que queramos cubrir se podrá llevar a cabo su implementación sin problemas. En términos de escalabilidad será bastante importante crear una arquitectura sólida para soportar la gran cantidad de peticiones que se puedan realizar. Al igual como albergar los datos que nutre a la aplicación. Estos pueden llegar a suponer del orden de casi *1GB* para un núcleo urbano por lo que será necesario usar otro tipo de tecnologías para albergar esta cantidad de datos.

5.2. Trabajo futuro

Actualmente la aplicación sólo funciona para un área delimitada del centro de *Barcelona* de $2km \times 2km$, ya que como dijimos al comienzo, no era posible implementarlo para el área de *Málaga* por la ausencia de modelos digitales de superficie. No se descarta en un futuro implementarlo para esta ciudad cuando estén disponibles dichos modelos.

El proyecto está delimitado a un área pequeña y los datos han de ser cargados mediante ficheros. Esto podría llegar a ser contraproducente hablando en términos de escalabilidad, ya que a medida que implementemos más áreas esto se traduce en más datos. Por tanto, se hace inviable mantener demasiadas estructuras de datos con la sobrecarga que esto conllevaría en términos de memoria y rendimiento del servidor. Una de las ideas a corto plazo sería externalizar un servidor que ejecute una base de datos *Big Data*, ya que los datos a

tratar llegarían a ser de millones de registros para una área como puede ser una ciudad de $50km^2$. En vez de ir accediendo a las estructuras de datos, se haría las consultas contra esta base de datos para obtener las alturas angulares. Habría que implementar también una arquitectura *cliente-servidor* con un buen rendimiento a nivel de ancho de banda para poder soportar las peticiones que hagan los clientes.

Otro aspecto importante a nivel de funcionalidad sería idear una heurística admisible para la estimación de tiempo de exposición solar desde un punto inicial o intermedio hasta el destino. Esto mejoraría drásticamente los resultados de la búsqueda que serían más cercanos a la realidad al estimar en todo momento el tiempo de exposición solar en vez de una ponderación de la distancia.

No se descarta en un futuro realizar la misma aplicación para otras plataformas como *iOS* o *Windows Phone* pero se haría a muy largo plazo, debido a que la mayor cuota de mercado actualmente la tiene *Android* en ese aspecto como citamos en el capítulo de *Introducción* 1.

Apéndices

Apéndice A

Manual de instalación

A.1. Puesta en marcha

El servidor se puede arrancar de tres maneras distintas: desde un servidor de aplicaciones web (como puede ser *Tomcat*) o desde línea de comandos. desde un *IDE* mediante una configuración de arranque. En este documento nos vamos a centrar en las dos últimas opciones: línea de comandos para una rápida ejecución o en un *IDE* por si se desea añadir o modificar módulos y hacer pruebas.

A.1.1. Arrancar desde línea de comandos

Arrancar desde línea de comandos es muy simple. Hay que ubicarse en el directorio donde tenemos alojado el proyecto e introducir el siguiente comando:

```
java -Xmx2G -jar /<path_to_project>/target/otp-0.18.0-SNAPSHOT
    .jar --build /<path_to_project>/OpenTripPlanner --inMemory
```

Al comando se le pasa los siguientes parámetros:

- *-Xmx* asigna la cantidad de memoria para la aplicación. En este caso le asignamos $2GB$ de memoria. Desde la página de *OTP* se recomienda asignar una cantidad considerable de memoria para evitar bloqueos innecesarios. Para más información

sobre el arranque de la aplicación nos podemos dirigir aquí[1].

- El parámetro *-jar* ejecuta la clase principal del proyecto. *OTP* es un proyecto basado en *Maven*. Cualquier modificación realizada requiera la ejecución del comando *mvn clean package* para construir el fichero binario. Para más información sobre como construir el proyecto desde las fuentes nos podemos dirigir aquí[2].

- Con la opción *-build* e *-inMemory* le indicamos la ruta donde se halla el grafo y de qué manera queremos cargarlo, en este caso, en memoria.

A.1.2. Establecer el entorno a través de un IDE

Toda la información relacionada con el arranque de la aplicación desde un *IDE* se puede encontrar en este enlace[3]. En primer lugar se recomienda para el proceso usar uno de los *IDE* siguientes: *Netbeans*, *Eclipse* o *Intellij IDEA*. Abrimos *OpenTripPlanner* que viene adjunto con la memoria y procedemos a crear una nueva configuración de arranque. Tenemos que identificar la clase principal que se encuentra en el paquete ***org.opentripplanner.standalone*** y cuyo nombre es ***OTPMain***. Esta es la clase encargada de ejecutar el programa así como de arrancar el servidor web e iniciar los servicios web. Es necesario decirle al servidor donde se encuentran los grafos *OSM* a cargar en memoria para poder usarlos. Esta información la introducimos como argumento del programa en *Program arguments* y dichos parámetros serán: ***–build /pathtoproject –inMemory***, que en este caso el grafo se ubica en el directorio raíz del proyecto. Debemos tener en cuenta que siempre que se quiera mover el archivo de directorio deberemos volver a reescribir la ruta en el argumento del programa. Por último queda asignarle la memoria necesaria para ejecutar el proyecto. Se recomienda poner un mínimo de *2GB*, ya que por debajo de esa cantidad puede experimentar problemas de rendimiento y bloqueos. El argumento de la memoria lo asignaremos en el campo *VM options* mediante el parámetro ***-XmxNG*** donde *N* será el número de gigas de memoria que queremos asignar.

[1] *Basic Usage of OpenTripPlanner:* `http://docs.opentripplanner.org/en/latest/Basic-Usage/#basic-usage-of-opentripplanner`

[2] *Building OTP from Source:* `http://docs.opentripplanner.org/en/latest/Getting-OTP/#building-from-source`

[3] *Working on OTP in an IDE*: `http://opentripplanner.readthedocs.org/en/latest/Developers-Guide/index.html?highlight=IDE`

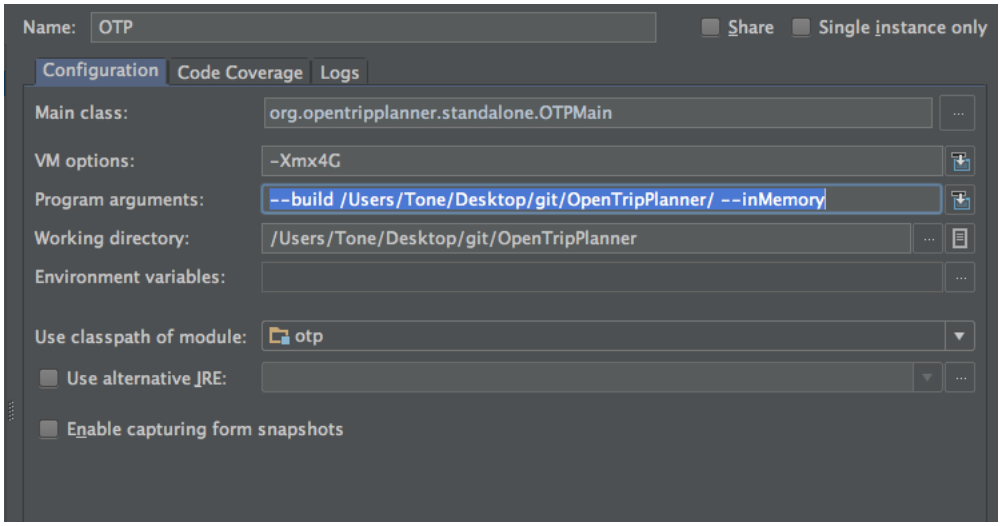

Figura A.1: Configuración de arranque en el *IDE Intellij IDEA*.

A.1.3. Interfaz web para probar el cálculo de rutas

Para la realización de pruebas en el servidor se ha optado por la interfaz web que proporciona la misma aplicación. Una vez arrancado el servidor podemos acceder introduciendo la dirección: ***http://localhost:8080***. Se nos iniciará la interfaz web con el mapa apuntando a la zona del grafo que hemos cargado, en nuestro caso, *Barcelona*.

Una vez cargada la interfaz, en la parte inferior izquierda, observaremos un recuadro de opciones para nuestro viaje con el nombre ***Trip Options***. En nuestro caso debemos seleccionar la modalidad ***Walk Only*** y aparecerá un recuadro con todas las opciones para esa modalidad: la hora, la fecha de salida y la relevancia de la sombra mediante un selector que va del *0* al *10*. Sólo queda asignar el origen y destino de la ruta en el mapa y hacer *click* en ***Plan Your Trip*** para cargar la ruta.

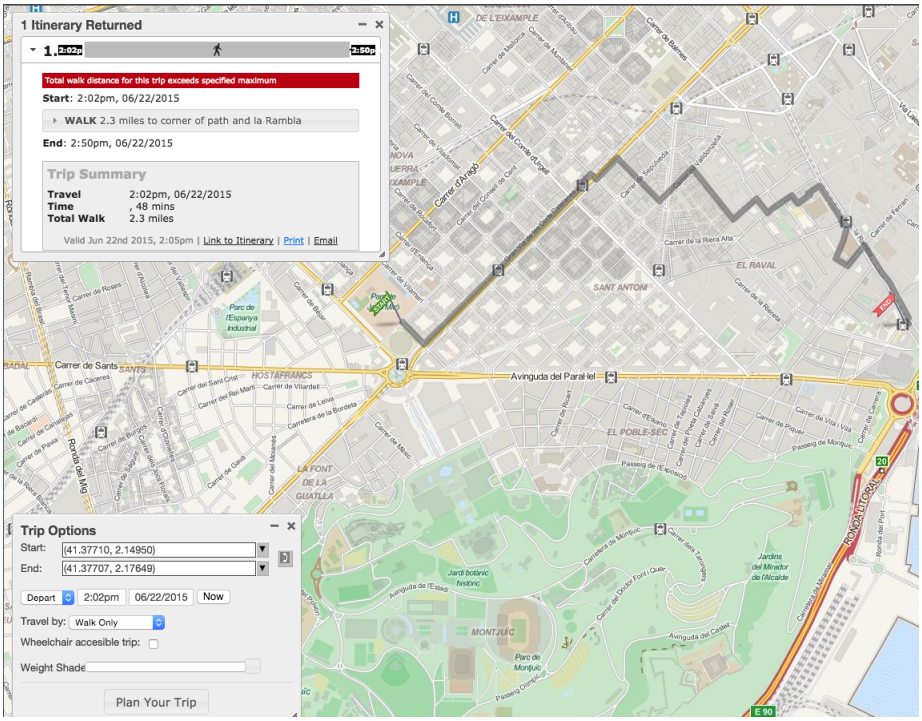

Figura A.2: Ejemplo de cálculo de ruta de la interfaz web.

Apéndice B

Aplicaciones adjuntas

Hemos tenido que crear dos aplicaciones anexas al proyecto principal para convertir los datos en bruto a una estructura apropiada para la finalidad del proyecto. Dichas aplicaciones son: ***OrderAndNormalizeMDSFile*** y ***AngleElevationFile***

B.1. OrderAndNormalizeMDSFile

Esta aplicación tiene el propósito de ordenar los puntos de elevación de un archivo en formato *XYZ* asociados a un *MDS*. El archivo o archivos suministrados por el *IGN* tiene una resolución de *0,5 puntos/*m^2 para el *MDS* y de $5m \times 5m$ para el *MDT*.

Para poder obtener las alturas angulares, por cuestiones de eficiencia y rendimiento en el cálculo, que tanto el *MDS* como el *MDT* tengan la misma resolución. El proyecto toma un archivo en formato *XYZ*. Cada coordenada X e Y es redondeada a un múltiplo de *5* y es añadida a una estructura de tipo *Map* siendo la clave la coordenada $<X, Y>$ y el valor una lista de elevaciones para esa coordenada.

Una vez cargados todos los puntos se procede a realizar una media ponderada de todos ellos con las elevaciones asociadas a cada coordenada. Esta ponderación va en función a la distancia al centro de esa coordenada. De modo que a menor distancia al centro, mayor ponderación.

Esta explicación se puede ver con más detenimiento en la sección 3.1.2.

Si deseamos añadir más de un *MDS* para normalizarlo y ordenarlo hay que tener en cuenta que las coordenadas deben ser contiguas entre sí. Esto quiere decir que si para un archivo que comienza en una coordenada X en *428.000* y acaba en *432.000*, el siguiente no debería comenzar en *440.000* si no en *432.000* al igual que las coordenadas Y que también deben estar en el mismo rango. Esto es para respetar la integridad de los datos y que no hayan 'vacíos' ya que podría provocar que haya posiciones nulas en mitad de la matriz bidimensional.

Al final de la ejecución se creará un archivo presentando la misma resolución que en el archivo *MDT*.

B.2. AngleElevationFile

Esta aplicación recoge el archivo creado anteriormente en formato *XYZ* del *MDS* y el archivo con el mismo formato para el *MDT* con la finalidad de calcular la máxima altura angular para cada coordenada y *acimut*.

El proyecto establece un incremento del *acimut* establecido por el usuario y un inicio y final del valor del *acimut* que se quieren calcular para cada punto de la cuadrícula. El programa cargará los datos en una matriz bidimensional con los datos del *MDT* y *MDS*. La matriz se recorrerá por filas y columnas para calcular las alturas angulares para cada *acimut*. Este proceso puede ser bastante largo en función del tamaño de los modelos digitales. Al final de la ejecución se obtendrá un archivo con las alturas máximas angulares asociadas a cada *acimut* para cada coordenada geográfica. A continuación mostramos como quedarían los datos representados en el archivo:

...

428235 4580000 70 20.091

428235 4580000 75 20.382

428235 4580000 80 20.473

...

El tercer y cuarto valor de cada línea corresponden al *acimut* y altura angular respectivamente. Este archivo será el que cargue la aplicación *OTP* para realizar los cálculos de la sombra en la ruta.

Para comprender mejor el funcionamiento de este proyecto nos podemos remitir a la aplicación ***TestDiagonal*** que pinta en pantalla todas las líneas. Cada una de ellas representa un *acimut* partiendo de la posición relativa del observador. Se recorrerían todas las coordenadas que las forman y se calcularía el máximo valor de altura angular de los edificios, vegetación o accidentes geográficos que se encuentren. Una ilustración de este ejemplo se mostró en la figura[3.7] de la sección 3.1.2.

El cálculo de alturas angulares puede durar bastante tiempo por lo que se aconseja usar un ordenador con buenas prestaciones. También podemos compilar el proyecto y calcular las alturas angulares por intervalos dando como resultado varios ficheros de entradas que se le pasarán a la aplicación. Una muestra de estos ficheros lo podemos encontrar en la carpeta *files* de *OpenTripPlanner* que son los mismos que se cargan al inicio de la ejecución del servidor para calcular las rutas como comentamos en capítulos anteriores.

Bibliografía

[1] Alex Berson. *Client/server architecture*. McGraw-Hill series on computer communications. McGraw-Hill, New York, Auckland, Bogota, 1996.

[2] Rachel S. Gordon. Schildt, herbert. Java 2: the complete reference. 5th ed, 2003.

[3] Muneendra Kumar. World geodetic system 1984: A modern and accurate global reference frame. *Marine Geodesy*, 12(2):117–126, 1988.

[4] John A O'Keefe. The Universal Transverse Mercator grid and projection, the professional geographer. *Association of American Geographers*, 4:19–24, 1952.

[5] Robert Joseph Peckham. *Range-Resolved Optical Remote Sensing of the Atmosphere*. Springer-Verlag, New York, 2005.

[6] Robert Joseph Peckham. *Digital Terrain Modelling*. Springer-Verlag Berlin Heidelberg, 2007.

[7] Sebastien Perochon and Javier Piqueres Juan. *Android: guia de desarrollo de aplicaciones para smartphones y tabletas*. ENI, Cornella de Llobregat, Barcelona, 2012.

[8] Ibrahim Reda and Afshin Andreas. Solar position algorithm for solar radiation applications. *Solar Energy*, 76(5):577–589, 2004.

[9] Elbridge P. Vance, Alberto Saenger, and Armando A. Armendariz. *Modern algebra and trigonometry*. Addison-Wesley, Reading (Massachusetts) [etc.], bilingua, 1st printing edition, 1964.

[10] Ruben Verborgh, Seth v. Hooland, Aaron S. Cope, Sebastian Chan, Erik Mannens, and Rik Van de Walle. The fallacy of the multi-api culture: Conceptual and practical benefits of representational state transfer (REST). *Journal of Documentation*, 71(2):233–252, 2015.

[11] Chengliang Wang, Qian Zheng, Yayun Peng, Debraj De, and Wen-Zhan Song. Distributed abnormal activity detection in smart environments. *Special issue on "Smart Learning with Sensor Network Technologies", International Journal of Distributed Sensor Networks, Hindawi Publishing Corporation*, 2014.

[12] Xiaolin Wu. An efficient antialiasing technique. *ACM SIGGRAPH Computer Graphics*, 25:143–152, 1991.

[13] R. L. Zeng, W.; Church. Finding shortest paths on real road networks: the case for A*. *International Journal of Geographical Information Science 23*, 25:531–543, 2009.