UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Generador automático de Metrominuto Documentación Técnica

Presentado por Guillermo Paredes Muga
en Universidad de Burgos — 6 de junio de 2020
Tutor: Dr. Álvar Arnaiz González y Dr. César Ignacio García Osorio

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apéndice se va a analizar todo aquello necesario para que un proyecto se desarrolle con el menor número de imprevistos posible. Para conseguir esto, es necesaria una fase de planificación donde se estimen los tiempos, la cantidad de trabajo y el dinero que es necesario invertir para sacar adelante el proyecto. Dicha planificación se divide en dos partes:

- **Planificación temporal**: fase en la que se analiza y planifica el trabajo necesario para desarrollar cada parte del proyecto, marcando fechas de inicio y final para cada una de ellas.
- **Estudio de viabilidad**: fase en la que se analizan las repercusiones legales y económicas del proyecto:
  - Económica: Análisis de los posibles costes y beneficios del proyecto.
  - Legal: Análisis de las repercusiones a efectos legales como la *Ley de Protección de Datos* o las licencias del proyecto.

A.2. Planificación temporal

Esta parte del proyecto es aquella en la que se planifica cómo va a ir avanzando el proyecto en función del trabajo requerido para cada una de las tareas de las que consta el mismo. Concretamente se estima el tiempo, es decir, cuáles van a ser los plazos para desarrollar determinadas tareas. Para

esta planificación o estimación se han empleado los conceptos generales de la metodología ágil Scrum, ya que en este caso en el proyecto sólo hay un único desarrollador aparte de los tutores. Las líneas generales que se han aplicado de esta metodología de gestión han sido:

- Desarrollo marcado por sucesivos *sprints* delimitados por dos reuniones: una al principio de cada uno y otra al final. Normalmente en la reunión de finalización de un *sprint* se marcaban los objetivos y la planificación del siguiente (siendo la primera reunión).
- La duración de los *sprints* fue de dos semanas al inicio y de una semana en la segunda mitad del mismo.
- Cada *sprint* produce un resultado o incremento del proyecto final.
- En cada *sprint* se divide el objetivo final en distintas tareas mas pequeñas.
- Las tareas se planifican y estiman en un tablero.

Sprint 0. (22/10/2019 – 28/10/2019)

En esta reunión el objetivo fundamental fue la presentación, a grandes rasgos, de en qué iba a consistir el proyecto por parte de los dos tutores: Álvar Arnaiz González y César Ignacio García Osorio. No se definió ninguna tarea, ya que simplemente se trataba de acordar si se había entendido bien el objetivo del proyecto.

Sprint 1. (29/10/2019 – 12/11/2019)

Esta reunión fue la primera en la que se comenzó a hablar de los requisitos del proyecto y de sus detalles para la planificación. Los objetivos de este sprint fueron:

- Crear correctamente el repositorio en GitHub.
- Elegir el entorno de desarrollo que se iba a utilizar y su posterior configuración para ejecutar una aplicación web con Flask.
- Hacer un primer proyecto «Hola Mundo» como primera toma de contacto con este framework.
- Incorporar en el proyecto un mapa proporcionado por el API de Google en el que fuésemos capaces de seleccionar diferentes puntos (marcadores).

Sprint 2. (13/11/2019 – 26/11/2019)

Los objetivos principales de este Sprint fueron:

- Guardar e imprimir (tanto en el cliente como en el servidor) los distintos marcadores seleccionados en el mapa.
- Dibujar la ruta entre los distintos puntos seleccionados.
- Seguir los estándares de programación
- Completar la documentación del proyecto.
- Cambiar de Visual Studio Code a Pycharm (licencia profesional).

Además de estos objetivos, debido a la posibilidad de añadir al proyecto nuevas funcionalidades y metodologías de Docker, se decidió cambiar de Sistema Operativo a Linux. Durante este sprint se siguió un curso «tutorial sobre Flask de Miguel Grinberg» //mejor cita?, en el cual, a medida que iba avanzando encontré varias mejoras para aplicar a mi proyecto y que fui incorporando. Al final de este sprint, dos de los objetivos no se consiguieron por completo, ya que daban algunos errores y se optó por una funcionalidad menor: en vez de dibujar la ruta entre todos los puntos seleccionados, sólo se dibujaba entre el primero y el ultimo; y al pasar los distintos marcadores, mediante un POST al servidor no podía pasar un objeto. Estas dos funcionalidades quedaron pendientes para el siguiente Sprint.

Sprint 3. (27/11/2019 – 10/12/2019)

El principal objetivo de este sprint fue poner al día la documentación al mismo tiempo que se seguía el tutorial de Flask mencionado en el A.2. Se encontraron distintas mejoras para realizar, así como la posibilidad de añadir el fichero `requirements.txt`, que después serviría para instalar las diferentes librerías utilizadas en el proyecto. También se acabaron las tareas que quedaron pendientes el sprint anterior.

Sprint 4. (10/12/2019 – 18/12/2019)

Los objetivos de este sprint fueron: mejorar el envío de los datos al servidor, buscar documentación y evaluar los resultados de las distintas funciones que proporciona el API de Google para Python, y ser capaces de diferenciar los datos «útiles» de dichas funciones.

Sprint 5. (19/12/2019 – 09/01/2020)

En este Sprint, los objetivos fueron:

- Mostrar la información de los diferentes puntos seleccionados.
- Obtener la matriz de distancias de todos con todos. Se obtiene como resultado una matriz en la que tenemos las distancias de todos los nodos entre sí.
- Dibujar el grafo en la parte del servidor, es decir, en Python.

Tras concluir el sprint, se alcanzaron todos los objetivos, no quedando nada pendiente para el siguiente.

Sprint 6. (10/01/2020 – 23/01/2020)

Los objetivos de sprint fueron:

- Actualizar la documentación.
- Evaluar el algoritmo *minimum spanning tree*.
- Valorar diferentes bibliotecas para dibujar el grafo en la web.

Tras concluir el sprint, quedó pendiente algunas preguntas sobre la parte de documentación, además de la implementación del grafo en el cliente.

Sprint 7. (24/01/2020 – 10/02/2020)

Durante este Sprint, los objetivos que se marcaron fueron los siguientes:

- La correcta implementación de la librería encontrada en el Sprint anterior.
- Creación de una estructura en la que a partir del árbol generado inicialmente y con la ayuda de la función minimum spanning tree de networkx, evaluar este camino mínimo en un conjunto aleatorio de arcos del árbol inicial.
- Crear el archivo JSON necesario para que la librería implementada anteriormente dibuje el grafo que necesitamos.

- Actualizar y corregir la documentación.

En este Sprint no se logró alcanzar por completo los objetivos marcados, quedando pendiente para el siguiente Sprint terminar la generación de la estructura del archivo JSON para dibujar el grafo.

Sprint 8. (10/02/2020 – 19/02/2020)

Los objetivos de este Sprint, que fue el primero en el que se pasó de realizar reuniones cada dos semanas a realizarlas cada semana, fueron:

- Terminar las tareas del sprint anterior.
- Actualizar la documentación del proyecto.
- Mejorar el sistema de generar un grafo por votos para que todos sus nodos estén siempre conectados.

Sprint 9. (20/02/2020 – 26/02/2020)

En la reunión de planificación de este sprint, que corresponde también con la de revisión del Sprint 8, se planteó la dificultad que presentaba emplear la biblioteca encontrada en los sprints anteriores para dibujar el mapa sinóptico. Esto se debe fundamentalmente a que la estructura de datos necesaria para ello era prácticamente imposible de generar de forma automática. Por esta razón, se decidió emplear otra forma para su dibujado, lo que lleva a los objetivos de este sprint:

- Eliminar referencias y usos de la biblioteca Tube Map.
- Generar el grafo en el servidor y exportarlo como svg para poder tratarlo a nuestro antojo en el cliente.
- Actualizar la documentación, junto con las correcciones del sprint anterior.

Sprint 10. (27/02/2020 – 04/03/2020)

Los objetivos de este sprint fueron:

- Generar el archivo SVG con el grafo generado. Se encontraron varias dificultades a la hora de escalar los puntos para dibujarlos en el SVG. Por ello, esta tarea quedó pendiente para el siguiente Sprint.

- Generar la documentación.
- Investigar sobre los *layouts* para grafos.
- Dar al usuario la posibilidad de elegir el número de arcos que aparecen en el grafo.

Sprint 11. (05/03/2020 – 12/03/2020)

En este Sprint, los objetivos fueron:

- Generar toda la documentación pendiente del Sprint anterior junto con la nueva generada en este.
- Resolver el escalado del grafo en SVG.
- Añadir estilos al grafo en el cliente. Para ello fijarse en el JavasScript de la librería utilizada en Sprints anteriores.
- Añadir texto al grafo SVG.
- Dar al usuario la posibilidad de elegir el número de veces que se repite el bucle que genera el grafo de votos.

Durante este sprint, al igual que en el anterior, se encontró una gran dificultad en la transformación de las coordenadas geográficas de los marcadores a pixeles para su dibujado en SVG. En este Sprint se alcanzaron todos los objetivos.

Sprint 12. (13/03/2020 – 26/03/2020)

Durante este Sprint los principales objetivos fueron:

- Eliminar un marcador cualquiera del mapa.
- Generar la documentación correspondiente.
- Resolver el problema de normalización para el correcto dibujado de los nodos en la imagen SVG.
- Incluir en el proyecto las variables de sesión.
- Dar al usuario la posibilidad de elegir el cómo quiere recorrer los puntos: en bici o a pie.

- Reestructurar la aplicación.

En este Sprint quedó pendiente la eliminación de un marcador cualquiera, ya que en la reunión de retrospectiva de este sprint se planteó la idea de mostrar en la pantalla un listado con los puntos seleccionados, con el objetivo de eliminarlos desde ahí. También quedó pendiente la documentación.

Sprint 13. (27/03/2020 – 02/04/2020)

Durante ente sprint, los objetivos fueron:

- Terminar las tareas pendientes del *sprint 12*.
- Cambiar la obtención de los votos de los arcos del grafo.
- Generar la lista de marcadores a medida que se añaden puntos al mapa.
- Buscar información sobre *Vue.js* e implementarlo en el proyecto.
- Incluir en la documentación la herramienta *lucidCharts* para generar diagramas.

Durante este sprint se encontraron dificultades con la implementación de Vue, ya que intenté añadir el cliente mediante *node.js* y añadiendo el cdn al template era suficiente. Por esta razón, quedó pendiente la modificación en la obtención de votos, aunque se consiguió añadir la lista con los marcadores (tanto añadir como borrar).

Sprint 14. (03/04/2020 – 16/04/2020)

En este sprint, los objetivos fueron:

- Cambiar el color de los puntos que sean seleccionados como centrales.
- Cambiar la *lista* de marcadores por *table-row*.
- Añadir un *checklist* a la tabla para seleccionar los nodos centrales.
- Guardar los nodos y la matriz de distancias para hacer las distintas pruebas con el fin de evitar hacer demasiadas llamadas a la *API de Google*.

- Incluir validaciones en el cliente.
- Modificar el SVG en el cliente, de tal manera que el usuario pueda mover los puntos.

Al finalizar este sprint, quedó pendiente la modificación del SVG en el cliente.

Sprint 15. (17/04/2020 – 23/04/2020)

Durante este Sprint, las tareas u objetivos fueron:

- Resolver el posicionamiento de los labels para que no se superpongan.
- Actualizar la versión de Python en el entorno virtual.
- Mejorar el código para que funcione el redirect.
- Precalcular los SVGs posibles, determinados por el número de votos de los arcos, antes de mandarlos al cliente. De esto modo, se mandan todos en una lista y es el cliente quien maneja cuál mostrar y cuál no.

Durante este Sprint se encontraron varias dificultades en calcular la superposición, por lo que quedaron pendientes esta tarea, junto con la del cálculo de los SVG.

Sprint 16. (24/04/2020 – 30/04/2020)

Las tareas principales de este Sprint fueron:

- Acabar tareas pendientes del *Sprint 15*.
- Modificar ficheros css.
- Actualizar el template base.
- Actualizar la documentación pendiente.
- Proponer un enrejillado de los nodos para que queden lo más alineados posible.

Durante este sprint, se encontraron dificultades a la hora de plantear el método que solucionaba el problema de solapamientos con las etiquetas. Finalmente, en este sprint quedó pendiente valorar otra opción para el cálculo de estos solapamientos y plantear el enrejillado de los puntos.

Sprint 17. (01/05/2020 – 07/05/2020)

En este Sprint, las tareas fueron:

- Mejorar o cambiar el método de detección de solapamientos.
- Cambiar la forma de la alerta al seleccionar un número máximo de marcadores.
- Incluir el aviso del uso de cookies.
- Ajustar la maquetación de la aplicación.
- Acabar temas pendientes del anterior Sprint.
- Actualizar la documentación.

Sprint 18. (08/05/2020 – 14/05/2020)

Durante este Sprint, los objetivos que se marcaron fueron:

- Documentación del código.
- Maquetar la aplicación.
- Concluir la discretización de los arcos para el dibujado de los labels.
- Modificar el cálculo de las posiciones para utilizar un escalado logarítmico.
- Construir una rejilla para agrupar los puntos tanto horizontal como verticalmente.

Durante este Sprint, también se empleó tiempo en modificar parte de las estructuras hechas anteriormente, ya que que a la hora de seguir avanzando me di cuenta que ya no servían y había que rehacerlas.

Sprint 19. (15/05/2020 – 21/05/2020)

Durante este Sprint, los objetivos fueron:

- Publicación de la primera release [1].

[1] https://github.com/gpm0009/TFG_MetrominutoWeb/releases/tag/v0.1

- Guardar los marcadores en el cliente, de tal modo que cuando el usuario vuelve a mapa no se pierdan los puntos seleccionados previamente.
- Ajustar los colores de las líneas al tiempo existente entre los puntos que unen.
- Cambiar el orden en la detección de los rectángulos para calcular primero las posiciones más importantes de las labels.
- Detectar el solapamiento entre líneas y ajustar cada una de ellas para que no se solapen.
- Maquetar la aplicación.

Al finalizar, quedó pendiente la documentación.

Sprint 20. (22/05/2020 – 28/05/2020)

En este Sprint, los objetivos fueron:

- Crear una nueva página para editar el SVG elegido por el usuario.
- Permitir editar el SVG anterior:
  - Arrastrar los puntos para modificar su posición.
  - Editar el texto que indica el nombre de cada punto.
- Modificar la página de ayuda.

Al finalizar este Sprint, quedó pendiente la gran parte de él, ya que me coincidió con un examen.

Sprint 21. (29/05/2020 – 04/06/2020)

Durante este Sprint, el objetivo fue acabar con las tareas pendientes del Sprint anterior. Debido a las nuevas funcionalidades de la aplicación, tuve que hacer varios cambios en el resto del proyecto.

Sprint 22. (05/06/2020 – 11/06/2020)

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apéndice se explican y especifican tanto los requisitos funcionales como los no funcionales del proyecto, así como los objetivos del proyecto.

B.2. Objetivos generales

- Crear una aplicación cliente – servidor que permita la creación automática de Metrominutos.
- Ofrecer control de usuarios (posible idea futura junto con lo comentado en la reunión de las API KEYs?)
- Permitir a los usuarios control sobre el mapa, de manera que puedan mover o eliminar los puntos seleccionados.
- Ofrecer al usuario un mapa final claro y sencillo.
- Que la aplicación final sea útil para el fomento de esta actividad.

B.3. Catálogo de requisitos

Requisitos funcionales

- **RF-1 Control de usuarios**: la aplicación debe permitir el control de usuarios.

    - **RF-1.1 Integración con Google Auth:** La aplicación debe poder hacer uso de cuentas de google para el control de usuarios.

- **RF-2 Control sobre el mapa:** La aplicación debe poder ofrecer la selección de distintos puntos sobre el mapa.
    - **RF-2.1 Integración con el API de Google:** La aplicación debe poder hacer uso de las operaciones, así como acceder a los servicios de *Google Maps* proporcionados por el API.
    - **RF-2.2 Creación:** Debe poder añadir tantos puntos como desee.
    - **RF-2.3 Modificación:** Una vez creado un marcador, futuro nodo del grafo, el usuario debe poder moverlo.
    - **RF-2.4 Eliminación:** El usuario debe poder eliminar los puntos.
- **RF-3 Generación de mapas sinópticos:** Generación de un mapa sinóptico a través de los puntos seleccionados en el *RF- 2*.
    - **RF-3.1 Modificación:** El usuario debe poder mover los puntos del mapa.
    - **RF-3.2 Apariencia:** Posibilidad de cambio de color de los puntos, así como de las líneas que los unen.
    - **RF-3.3 Eliminación:** El usuario debe poder eliminar dichos puntos.
    - **RF-3.4 Elementos adicionales:** Dibujado de elementos adicionales como pueden ser ríos, parques, o estaciones públicas.

Requisitos no funcionales

- **RNF-1 Usabilidad:** La aplicación tiene que poder usarse de forma sencilla y debe ser intuitiva.
- **RNF-2 Compatibilidad:** La aplicación tiene que poder ser compatible con los diferentes navegadores.
- **RNF-3 Responsividad:** La aplicación debe poder adaptarse al tamaño de la pantalla.
- **RNF-4 Facilidad en el despliegue:** La aplicación debe poder ser desplegada con facilidad en un servidor.
- **RFN-5 Mantenibilidad:** Debe ser sencillo añadir nuevas funcionalidades.

B.4. Especificación de requisitos

En esta sección, se presentan los casos de uso de la aplicación.

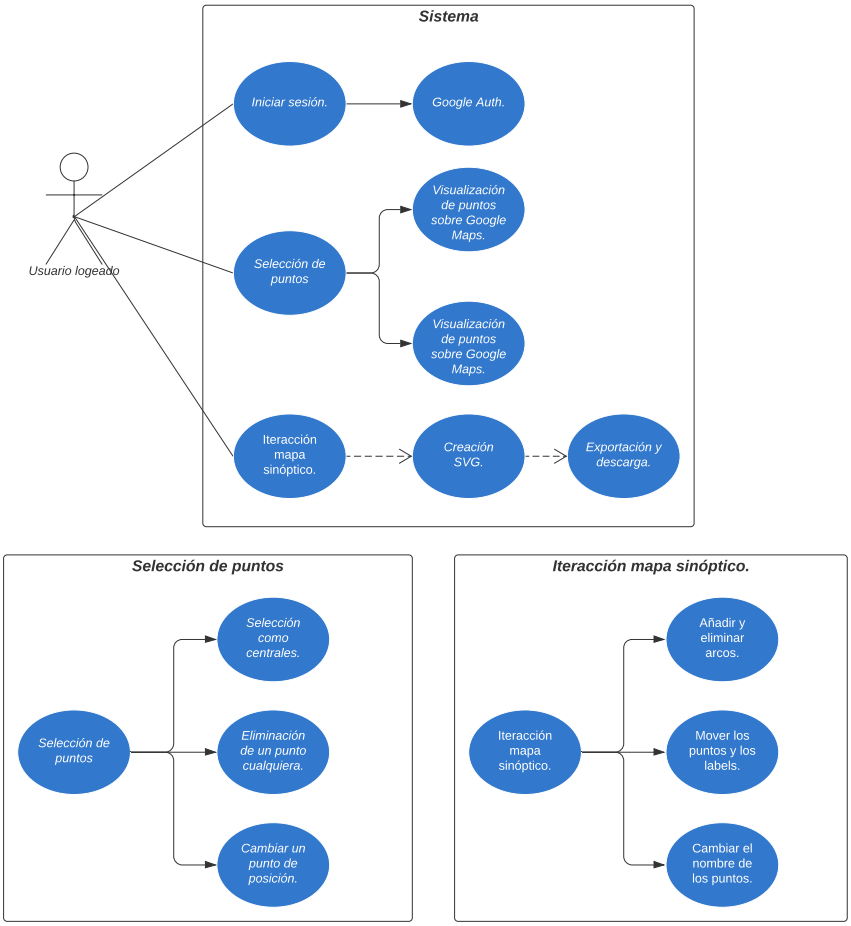

Figura B.1: Diagrama de casos de uso.

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado de la documentación se expone el diseño que ha dado lugar a la aplicación, el cual incluye el diseño de las distintas estructuras de datos, el diseño procedimental y diseño arquitectónico.

C.2. Diseño de datos

En esta sección se explican las entidades usadas por la aplicación:

- Grafos: Conjunto de datos necesarios para tratar los puntos y arcos que representan los caminos entre los distintos marcadores seleccionados.
- Datos de Google: Datos que proporciona la API de Google, como distancias y tiempos entre puntos, así como datos acerca de la ubicación del marcador.
- SVG: entidad que se usa para representar el grafo obtenido tras realizar distintas comprobaciones sobre los datos obtenidos anteriormente.

NetworkX

En este proyecto se usa networkX, D.6, para generar, modificar y visualizar grafos, en los cuales el elemento «*nodes*» representa los diferentes puntos seleccionados por el usuario, y el elemento «*edges*» representa las conexiones entre ellos.

Google API

Google se ha usado para la obtención de todos los datos necesarios para el cálculo de distancias y tiempos, así como para la selección de los diferentes puntos en el mapa. Para ello, Google proporciona un API para *Python* y otro para *JavaScript*. Las funciones que se han usado han sido:

- `distance_matrix(orígenes, destinos)`: devuelve las distancias de cada origen con cada destino.
- `directions()`: devuelve las direcciones que hay que seguir para llegar de un punto a otro.

SVG

Mediante la biblioteca *svgwrite* D.6 obtenemos el grafo mediante el dibujado de:

- Círculos: corresponden con los marcadores seleccionados previamente en el mapa.
- Líneas: corresponden a la distancia entre los puntos de sus extremos.
- Labels: texto con información acerca de los puntos y de la distancia entre ellos

C.3. Diseño procedimental

En esta sección se explican las interacciones más destacadas de la aplicación. La principal de ellas es, dentro de la visualización de los puntos, cuando se pasa a la vista del mapa sinóptico, ya que aquí entran en juego todas las funcionalidades más destacadas de la aplicación. // Diagrama de secuencia.

Cuando el usuario selecciona y guarda una serie de puntos sobre el mapa, estos pasan al servidor de tal manera que, con el API proporcionado por Google se evalúan las distancias de unos nodos a otros y se genera un grafo. A continuación, se realizan diferentes operaciones con dicho grafo para evaluar los diferentes caminos más cortos para recorrer todos los nodos.
Como resultado final, obtenemos un mapa sinóptico con la representación de estos puntos y de los caminos existentes entre ellos.

C.4. Diseño arquitectónico

La estructura del proyecto esta condicionada por el tipo de proyecto que es. Se trata de una aplicación web y por ello se ha seguido el patrón MVC (Modelo - Vista - Controlador), el cual permite separar en 3 componentes diferentes los datos, el interfaz y la lógica de la aplicación.

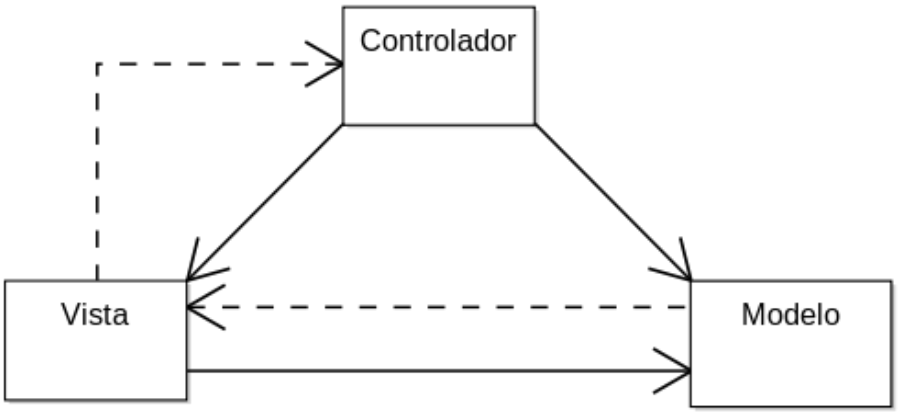

Figura C.1: Diagrama Modelo-Vista-Controlador

C.5. Maquetación

Al tratarse de una aplicación web es muy importante que la apariencia de la misma se haya cuidado y sea adaptable a cualquier tipo de dispositivo.

Para la construcción de las distintas páginas de este proyecto, se ha heredado una estructura común definida en el fichero *base.html*. Este fichero importa diferentes librerías, como *Bootstrap* y *Vue*. También añade la barra de navegación y otros elementos comunes a todas las páginas. Es por ello que desde el comienzo del proyecto y con la finalidad de evitar futuros problemas a la hora de ordenar y alinear los elementos *HTML* conviene tenerlo bien estructurado y ordenado.

A la hora de situar los diferentes elementos HTML en la página, como por ejemplo los botones de *«Aceptar»* y *«Cancelar»*, es importante tener en cuenta la usabilidad de los mismos.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apéndice se va a definir todo aquello que es necesario conocer para que se pueda continuar con el desarrollo del proyecto, desde su estructura hasta una breve descripción de como instalar la aplicación y configurar nuestro entorno de trabajo para llevar a cabo el desarrollo.

D.2. Estructura de directorios

En este proyecto, aunque al principio se siguieron distintos tutoriales sobre cómo crear la estructura de directorios y ficheros, al final se optó por la opción que propone Azure. De este modo, a la hora de realizar el despliegue no tendríamos ningún tipo de problema, ya que esta plataforma requiere determinados ficheros de configuración que se explicarán a continuación. La estructura del proyecto se divide en:

D.3. Manual del programador

En este apartado se explican los puntos a tener en cuenta por futuros desarrolladores que tengan la intención de mantener o mejorar el proyecto.

```
/ Directorio raíz
├── Documentación/ - Documentación del proyecto
│   ├── img/ - Imágenes de la documentación
│   ├── tex/ - Secciones de la documentación
│   ├── anexos.pdf - Anexos del proyecto
│   └── memoria.pdf - Memoria del proyecto
├── HolaMundo/ - App web básica
└── Metrominuto/ - Directorio del proyecto web
    ├── metrominuto_app/ - Aplicación web
    │   ├── main/ - Directorio que inicializa el blueprint de
    │   │   la aplicación
    │   │   ├── init.py - Inicialización
    │   │   ├── forms.py - Contiene los formularios de la
    │   │   │   aplicación
    │   │   └── routes.py - Contiene las rutas de la aplicación
    │   ├── static/ - Ficheros JavaScript
    │   │   ├── css/ - Estilos
    │   │   ├── img/ - Imágenes
    │   │   └── templates/ - Ficheros HTML
    │   ├── init.py - Inicializa la aplicación
    │   ├── models.py - Clases definidas en la aplicación
    │   ├── calculateRoute.py -O peraciones con el API de
    │   │   Google
    │   ├── globals.py - Variables globales del proyecto
    │   ├── graphs.py - Operaciones con grafos
    │   ├── svgfunctions.py - Operaciones para dibujar
    │   └── webapp.py - Punto de entrada para la aplicación en
    │       Azure
    ├── config.py - Fichero de configuración
    ├── metrominuto.py - Punto de entrada para iniciar la
    │   aplicación
    ├── metrominuto.txt - Fichero necesario para desplegar en
    │   Azure
    └── requirements.txt - Dependencias necesarias para
        ejecutar el proyecto
```

Dibujado de grafos

El principal objetivo de este proyecto se basa en obtener un grafo final de manera que éste sea fácilmente entendible por todos. Es por ello que a la hora de dibujar dicho grafo se plantean cuestiones y problemas como dónde colocar los textos, a que distancias, cómo orientar esos textos...

Estos problemas han resultado de gran complejidad en el desarrollo final del proyecto, ya que como mencionaba antes, es el resultado final de todo el proyecto. Para solucionar el problema de que los textos no se superpongan, tanto en las líneas o *caminos* como en los puntos o *paradas*, he tenido en cuenta dos opciones:

- **Discretización de las líneas:** este método consiste en dividir los arcos del grafo en varios puntos separados una distancia δ y comprobar si alguno de esos puntos está en el interior del rectángulo que contiene el texto que queremos colocar. Ese δ se calcula proyectando sobre cada recta una distancia en el eje x mínimamente inferior a la altura del texto, y para lo cuál nos hace falta calcular el ángulo entre dicha recta y la horizontal. Para cada texto comprendido en el punto medio de cada arco, se evalúan 8 distintas posiciones a cada lado del arco[1]. Para el texto relativo a la información de cada nodo se evalúan estas posiciones para la posición inferior, superior, izquierda y derecha del nodo, y si no se encontrase ninguna se pasaría a evaluar las esquinas del cuadrado que contiene al círculo. En la figura D.1 vemos un ejemplo de las 8 anclas en el rectángulo A. ¿Explicar el proceso de los dos puntos sobre los que se evalúan las anclas? o quedaría demasiado extenso?

[1] https://github.com/gpm0009/TFG_MetrominutoWeb/issues/91

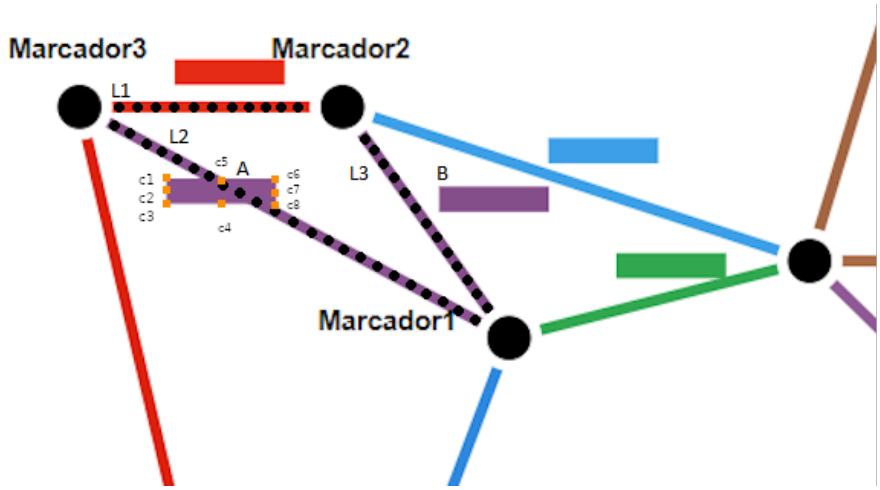

Figura D.1: Discretización.

- **Superposición de cuadrados:** Este método consiste en calcular primeramente el cuadrado que contiene el texto. Una vez que sabemos esto, podemos construir varios rectángulos en torno al punto donde queremos colocar el texto, tanto a un lado de la línea como al otro. Posteriormente, dividiremos la línea que une los dos puntos para los que queremos colocar el texto en cuatro cuadrados partiendo del punto medio. De este modo, en dos de ellos la diagonal sería parte de la línea que une los puntos, mientras que en los otros dos no habría nada. De este modo, y conociendo la dirección de la línea podemos calcular si, sobre los rectángulos que forman parte de la línea, existe una superposición con alguno de los posibles rectángulos del texto. En el caso del texto que referencia el punto, el rectángulo, o en este caso cuadrado, que se construye sería el que contiene al círculo y se evaluaría el rectángulo correspondiente del que la línea forma parte. En la figura D.2 vemos un ejemplo: los rectángulos que contienen a la línea serían B, D, H y F, mientras que los rectángulos A, C, E y G no. Por ello, siempre que el rectángulo se encuentre dentro del segundo grupo no tendremos problemas de superposición con las líneas.

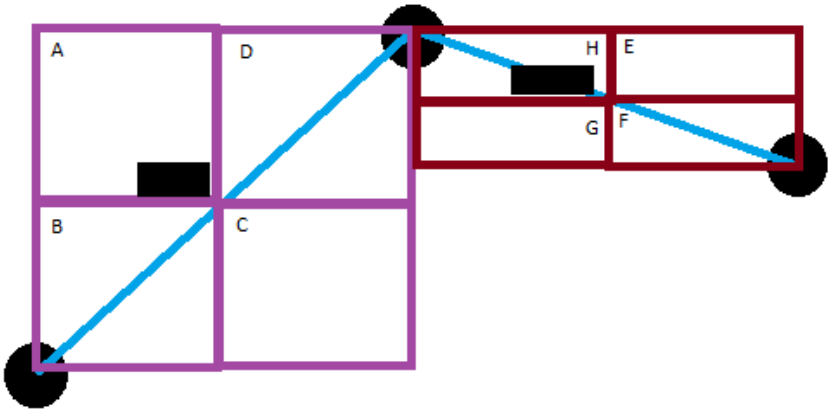

Figura D.2: Superposición de rectángulos.

Finalmente, se optó por la primera opción, ya que después de comprobar que los textos no se superponían con las líneas, había que tener en cuenta que no se superpusiesen entre ellos. Para ello, sobre la lista de puntos discretizados, se añaden las 4 esquinas de los rectángulos que contienen al texto. Con esta operación lo que hacemos es que si al colocar el siguiente texto en una de las posiciones establecidas nos encontramos con que uno de estos puntos esta en el interior de este último rectángulo, significará que ya hay un texto en esa posición y debemos probar con la siguiente.

Un aspecto clave, y a tener muy en cuenta en la evaluación de todo lo comentado anteriormente, es que debemos recordar en todo momento que estos cálculos se hacen para dibujar en SVG, por lo que el punto mas pequeño de la «coordenada x» y de la «coordenada y» se encuentra en la esquina superior izquierda. Esto nos lleva a que cada posible ancla (punto inferior izquierdo) del rectángulo del texto debemos restarle la altura de dicho texto para que la evaluación de los puntos coincida, ya que al dibujarlo en SVG el texto siempre se dibuja hace arriba desde el punto que indicamos. Para entender mejor esto, los puntos A y C de la imagen D.3 corresponderían con el punto inferior izquierdo del rectángulo que queremos evaluar, mientras que los puntos B y D serían los puntos que debemos indicar al dibujar el SVG.

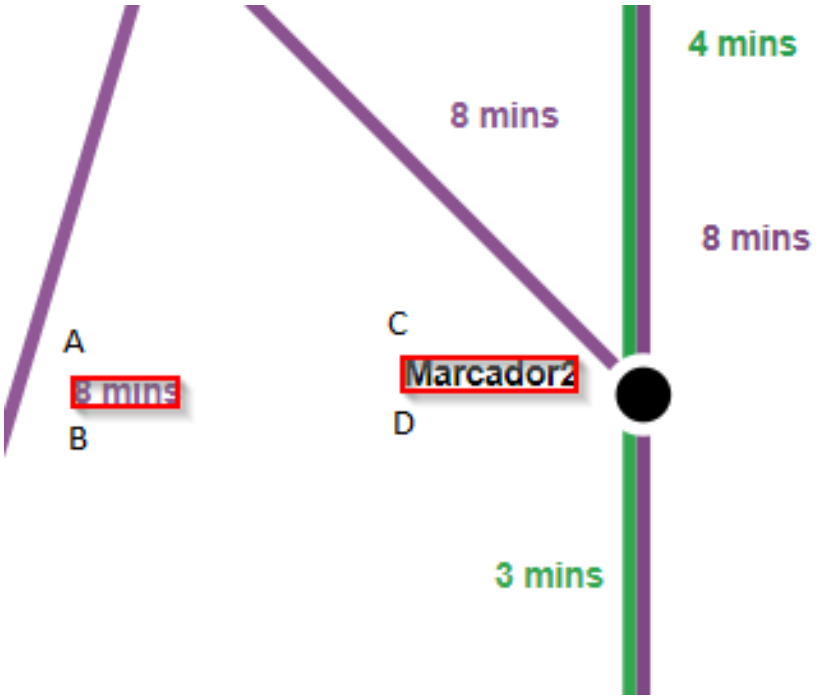

Figura D.3: Puntos del rectángulo.

D.4. Aplicaciones utilizadas

Para el desarrollo de este proyecto se tuvieron en cuenta principalmente dos editores de texto y dos herramientas para mantener el control de versiones.

- **Visual Studio Code**
- **PyCharm**
- **GitHub**
- **GitCraken**

Después de analizar y configurar ambos editores de texto, se llegó a la conclusión de que era mucho mas cómodo y útil utilizar PyCharm, ya que ofrece una configuración mas sencilla, además de permitir importar diversas librerías de una manera mas amigable. También ofrece la posibilidad de seguir los estándares de programación *PEP8* y *ECMAScript*. Para el control de versiones se ha utilizado GitCraken al comienzo del proyecto. Después, se utilizó el propio editor de texto.

D.5. Instalación y configuración

Para la instalación del proyecto se explicarán los pasos a seguir en un sistema operativo de Linux, que en este caso se trata de la versión Linux Mint 18.3 Sylvia. Tras varias pruebas, la instalación y ejecución del proyecto es idéntica salvo la propia instalación de Pycharm y Python.

Python

Este proyecto está desarrollado con la versión 3.6.3. Python se puede descargar desde el siguiente enlace: https://www.python.org/downloads/

Instalación y configuración de PyCharm

Este *IDE* tiene distribución para Linux, además de permitir a estudiantes usar la versión profesional. Para su instalación podemos usar la *snap store* de Linux, que en caso de no tenerla instalada tenemos que ejecutar el siguiente comando:

```
1 $ sudo apt update
2 $ sudo apt install snapd
```

Instalar PyCharm D.1: Instalar snapd

Después de tener instalado esto, ejecutaríamos:

```
1 $ sudo snap install
2   pycharm-community|professional --classic
```

Instalar PyCharm D.2: Instalar PyCharm

Una vez instalado *PyCharm*, la forma más cómoda de obtener el código del proyecto es mediante *git*, usando para ello el comando:

```
1 $ git clone <url_del_repositorio>
```

Configurar PyCharm D.3: Descargar el repositorio

Siendo https://github.com/gpm0009/TFG_MetrominutoWeb.git la URL del repositorio.

Para instalar las dependencias ejecutar el comando:

```
1 $ pip install -r requirements.txt
```

Configurar PyCharm D.4: Instalar requirements.txt

Claves de Google

Para la obtención de una *Google API KEY* es necesario obtener los credenciales en https://console.developers.google.com/apis/credentials. Una vez registrados en Google, en el caso de que no lo estuviésemos ya, debemos acceder a la *consola*[2] y desplegar el panel de la parte izquierda.

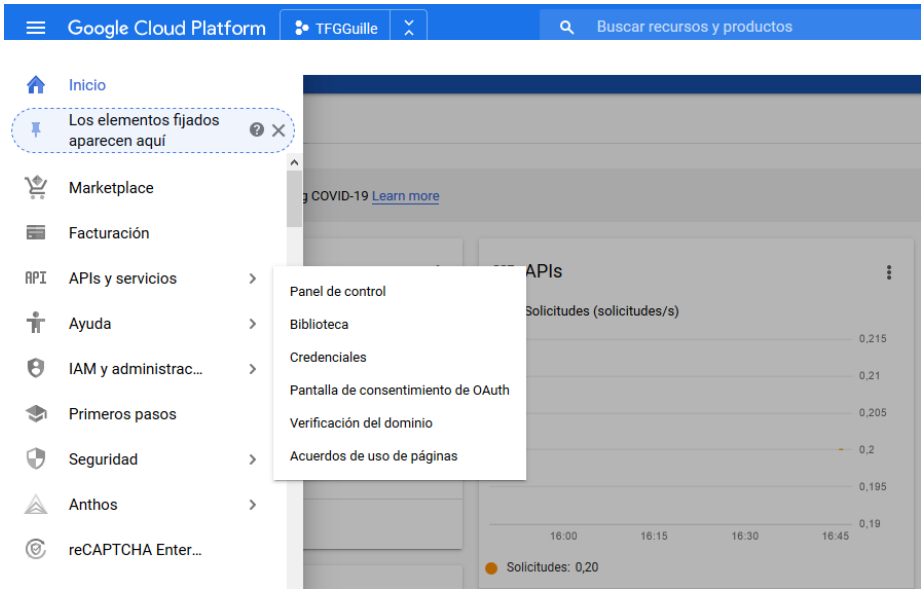

Figura D.4: Panel Google Console

Hacemos click sobre «APIs y servicios» - «Credenciales» y después sobre la clave que se nos ha generado anteriormente. Desde el apartado «Restricciones de API» podremos seleccionar que servicios queremos que proporcione nuestra clave y restringir desde dónde se va a usar.
En concreto, para este proyecto debemos tener activados:

- Directions.
- Distance Matrix.
- Geocoding.
- Maps JavaScript.
- Places.

[2] https://console.cloud.google.com

Una vez que la tenemos, debemos incluirla en el proyecto. Para ello:

```
google_maps=googlemaps.Client(key='GOOGLE_API_KEY')
```

Google Key D.5: Añadir `API_KEY`

Además, no hay que olvidar incluirla en los templates:

```
<script
 src="https://maps.googleapis.com/maps/api/
   js?key=API_KEY&libraries=places"
   type="text/javascript"</script>
```

Google Key D.6: Añadir `API_KEY` a los templates

También podemos incluirla como variable de entorno en nuestro editor. De esta manera nos aseguramos de no compartirla al realizar los commints en el control de versiones.
En este caso, en PyCharm se configura de la siguiente manera:

1. Abrir selector Run Configuration (arriba a la derecha)
2. Edit Configurations...
3. Environmental variables
4. Add or change variables, then click OK

TEXstudio

Esta herramienta para la compilación de documentación LATEX permite la instalación de diccionarios para aplicar las reglas al texto. Para ello, debemos acceder a:

```
Options -> Configure TeXstudio
Language checking
```

Configure TEXstudio D.7: Añadir diccionario

Una vez ahí, vemos que nos ofrece dos opciones para buscar diccionarios: https://extensions.openoffice.org/de/search?f[0]=field_project_tags o https://extensions.libreoffice.org/extensions?getCategories=Dictionary&getCompatibility=any. Elegimos cualquiera de ellas y descargamos el paquete del diccionario que queramos, y después lo importamos.

D.6. Bibliotecas

NetworkX

Como ya he mencionado, esta biblioteca nos permite trabajar de una forma muy amplia y completa con grafos [1]. A lo largo del proyecto se han usado:

- `graph()`: para crear un grafo no dirigido al que se añadirán nodos y arcos.
- `add_node()`: Diferentes nodos junto con atributos como la posición obtenida del API de Google y el nombre.
- `add_edge()`: Arco que conecta dos nodos. Además, los arcos contienen atributos como la distancia real que hay de nodo a nodo o el número de votos que tendrá.
- `get_edge_attributes()`: para obtener los valores de un atributo perteneciente a los arcos. Devuelve una lista que contiene el nodo de origen, el nodo destino y el atributo deseado.
- `edges(data=True)`: devuelve los arcos junto con los atributos.
- `nodes(data=True)`: devuelve los nodos junto con los atributos.
- `minimum_spanning_edges()`: devuelve un iterador con los arcos que forman el grafo de tal manera que la suma de distancias es la mínima.
- `draw_networkx()`: para dibujar el grafo.

Para su uso, es necesario instalar el requirement correspondiente e importarla en el proyecto. Para ello:

```
1 pip install networkx
```

NetworkX D.8: Instalación mediante pip.

```
1 import networkx
```

NetworkX D.9: Importación.

SVGWRITE

Esta biblioteca permite crear en el servidor nuevos archivos SVG [3]. Permite crear, dentro del dibujo SVG distintos elementos, siendo en el caso de este proyecto círculos, rectángulos, líneas y textos.

Para el dibujado hacemos uso del método `Drawing()`, del cuál dependeran todos los elementos que queramos añadir:

```
1 dwg = svgwrite.Drawing()
2 square = dwg.add(dwg.rect((0, 0), (1, 1), fill='blue'))
```

SVGWRITE D.10: Ejemplo de uso.

Para su uso, debemos instalar el *requirement* e importarla al proyecto:

```
1 pip install svgwrite
```

SVGWRITE D.11: Instalación mediante pip.

```
1 import svgwrite
```

SVGWRITE D.12: Importación.

SNAP.SVG

Es otra librería para crear gráficos vectoriales pero esta vez en la parte del cliente gracias a que es una biblioteca JavaScript [2]. Esto la hace perfecta para manipular los elementos del SVG a nuestro antojo sin tener que hacer nuevas llamas al servidor.

```
1 var s = Snap("#svg");
2 // Lets create big circle in the middle:
3 var bigCircle = s.circle(150, 150, 100);
4 // By default its black, lets change its attributes
5 bigCircle.attr({
6 fill: "#bada55",
7 stroke: "#000",
8 strokeWidth: 5
9 });
```

Snap.svg D.13: Ejemplo

Para su uso debemos incluirla en el proyecto siguiendo una de las opciones propuestas en su documentación [3]. En este proyecto, se ha optado por descargar el directorio `\dist` he incluir los ficheros JavaScript en el proyecto.

[3] https://github.com/adobe-webplatform/Snap.svg

D.7. Despliegue de la aplicación en Azure

Antes de incluirlo voy a intertar desplegarlo desde un repositorio git (ya que he elegido pycharm para no decir ahora que uso VS Code).

D.8. Compilación, instalación y ejecución del proyecto

Para el desarrollo de este proyecto se ha usado PyCharm, por lo que esta guía esta orientada a este editor. Para Visual Studio Code puede visitarse el enlace https://code.visualstudio.com/docs/python/tutorial-flask, ya que la creación y ejecución del entorno virtual no es igual.

Python

Este proyecto está desarrollado con la versión 3.7.5, pero con para desarrollar la mínima es 3.6. Python se puede descargar desde el siguiente enlace: https://www.python.org/downloads/.

Instalación

La forma más cómoda de obtener todo el proyecto es mediante *git*, usando el comando:

```
1 $ git clone <url_del_repositiorio>
```

Instalación D.14: Descargar el repositorio.

Siendo https://github.com/gpm0009/TFG_MetrominutoWeb la URL del repositorio en *GitHub*.
Ya con el proyecto descargado, el siguiente paso es instalar las dependencias. Aunque se pueden instalar sobre la instalación global de Python, es recomendable usar un entorno virtual. Para ello, debemos abrir el directorio del proyecto desde Pycharm, y una vez que lo tengamos ir a *Files - Settings - Project Interpreter - Add*, y seleccionar la opción de *Virtual Envirorment* y a continuación creamos uno nuevo.

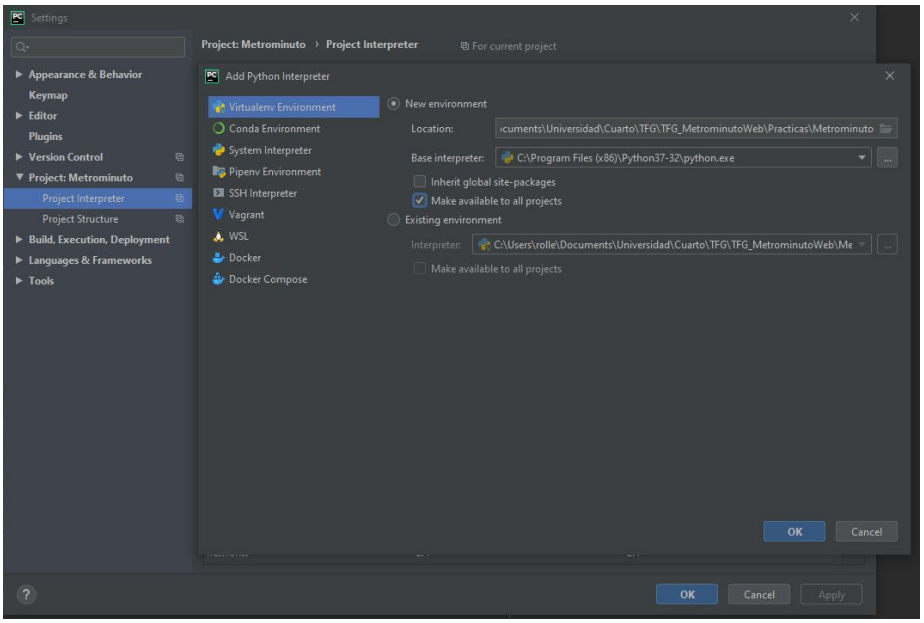

Figura D.5: Creación del entorno virtual.

Después, el editor detectará automáticamente el fichero *requirements.txt* y nos preguntará si queremos instalar las dependencias, a lo que le diremos que sí. De no ser así, hay que abrir la terminal en la parte inferior del editor y ejecutar el comando:

```
1 $ pip -r install requirements.txt
```

Instalación D.15: Instalar dependencias en el entorno virtual.

Variables de entorno

Para que la aplicación funcione correctamente hay que definir las siguientes variables de entorno:

- **`GOOGLE_API_KEY`:** clave obtenida anteriormente necesaria para obtener datos de Google.
- **`SECRET_KEY`:** clave necesaria por Flask.
- **`ENVIRORMENT`:** en el que se está actualmente: development o production.

Para su configuración, acceder desde la parte superior derecha del editor a *Edit Configurations...* y después seleccionar *Envirorment variables* como vemos en la figura D.6.

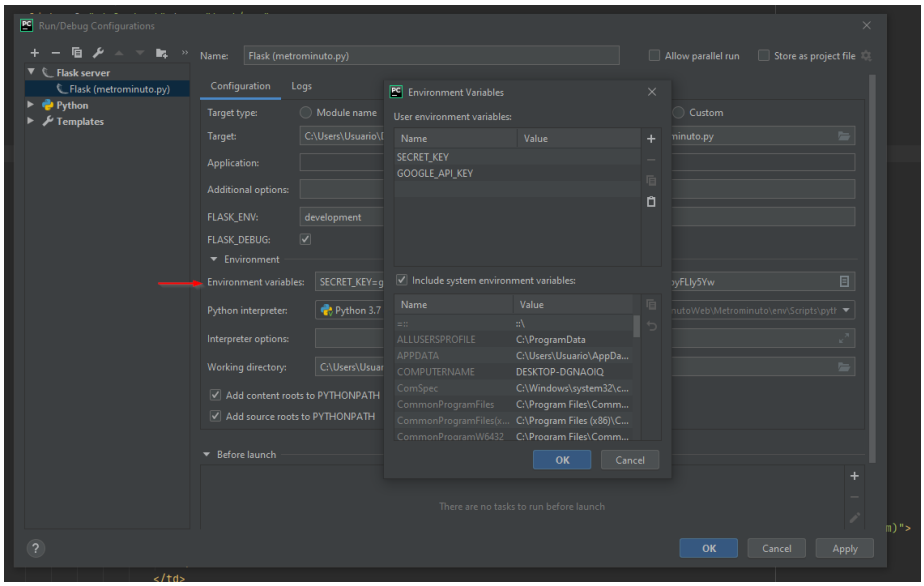

Figura D.6: Variables de entorno.

Ejecución

Para la ejecución desde el editor, es necesario que hagamos una última configuración. Debemos ir a la parte superior derecha del editor y seleccionar *Edit Configurations...* como se ven en la figura D.7.

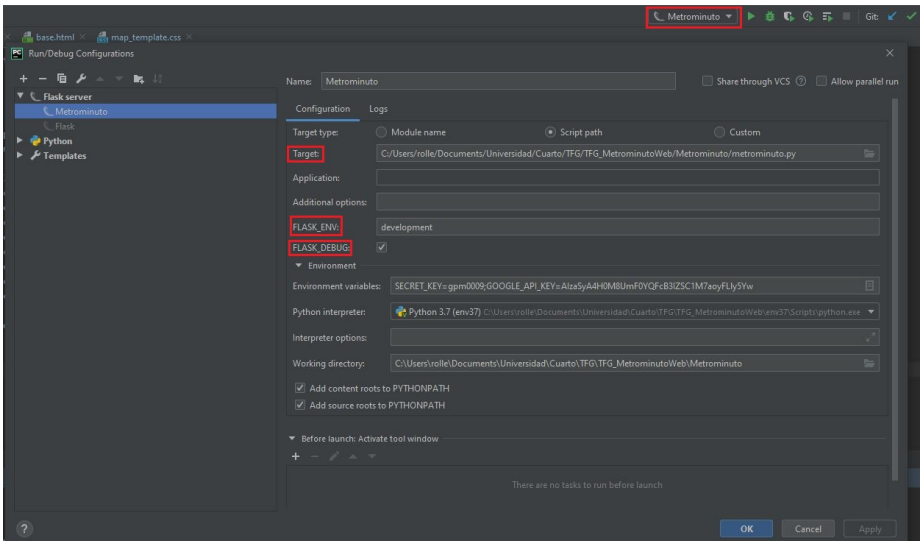

Figura D.7: Ejecución del proyecto.

En esta pantalla tenemos que elegir el punto de entrada a la aplicación, que será el fichero *metrominuto.py*, escribir el entorno en el que queremos ejecutar la aplicación y marcar la opción de Debug. Una vez hecho esto, ya podemos ejecutar la aplicación desde el panel superior derecho.

Despliegue en Azure

Para realizar el despligue de la aplicación en Azure es necesario que el proyecto tenga una estructura concreta, ya que serán los servicios de Azure los que instalen las dependencias necesarias y ejecuten el código. Dicha estructura es la definida anteriormente en el apartado *Estructura de directorios* D.2.

He hecho el despliegue usando VS Code, igual no queda bonito decir que lo he usado después de usar PyCharm. Voy a probar a desplegarlo desde git y lo incluyo.

Tras hacer el despliegue, es importante acordarse de configurar en Azure las variables de entorno que teníamos definidas D.8. Para ello debemos entrar, en Azure, a nuestro *APP Service* y añadirlas en el apartado de *Configuración* que aparece en el menú desplegable de la izquierda, véase la figura D.8.

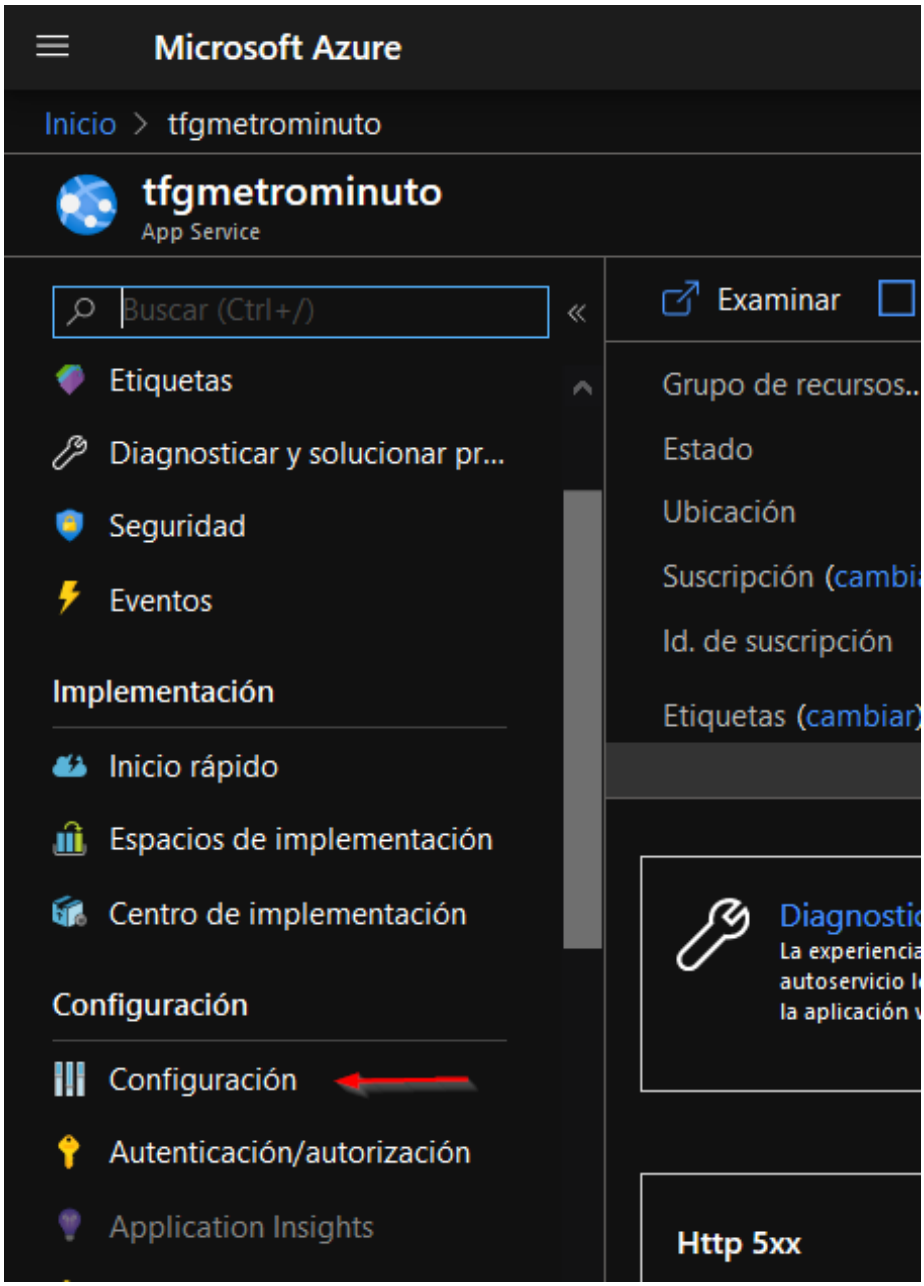

Figura D.8: Configuración Variables de entorno en Azure, paso 1.

A continuación, seleccionar *Configuración de la aplicación* y *Nueva configuración de la aplicación*, como vemos en la figura D.9.

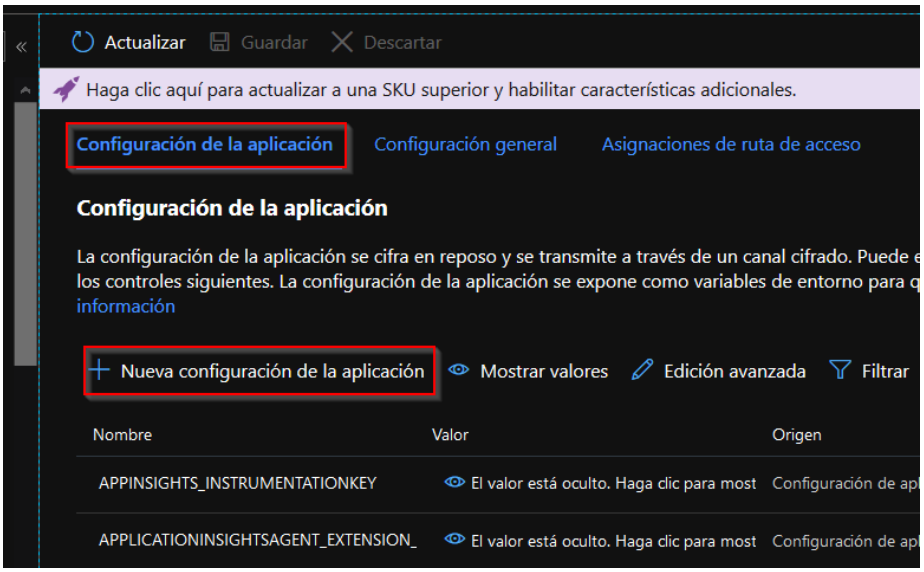

Figura D.9: Configuración Variables de entorno en Azure, paso 2.

D.9. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

En este apéndice se explica los requisitos que debe cumplir el usuario para ejecutar la aplicación, como lanzarla y como usarla.

E.2. Requisitos de usuarios

Al tratarse de una aplicación web los requisitos que debe cumplir el usuario son los siguiente:

- Navegador web instalado.
- JavaScript activo en el navegador.
- *Cookies* activas en el navegador.

La aplicación esta diseñada para que sea usable tanto en ordenadores como en dispositivos móviles, debido a que su funcionalidad esta orientada a que los principales usuarios estén en continuo movimiento.

E.3. Instalación

Debido a que se proporciona una aplicación web no es necesario instalarla para poder usarla.

E.4. Manual del usuario

Bibliografía

[1] Networkx documentation. https://networkx.github.io/documentation/stable/index.html.

[2] Snap.svg documentation.

[3] Svgwrite documentation. https://svgwrite.readthedocs.io/en/latest/overview.html.