ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MASTER EN INGENIERÍA DE TELECOMUNICACIÓN

Aplicación de Algoritmos de Cálculo de Rutas a redes de Transporte Público Urbano

Autor: Manuel García Bellido

Director: Miguel García Domínguez

Madrid

Junio 2018

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

"Aplicación de algoritmos de cálculo de rutas a redes de transporte público urbano"

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2017/18 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Manuel García Bellido Fecha: 10/ 06/ 2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Miguel García Domínguez Fecha: 10/ 06/ 2018

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º.Declaraciónde la autoría y acreditación de la misma.

El autor D. Manuel García Bellido DECLARA ser el titular de los derechos de propiedad intelectual de la obra: "Aplicación de algoritmos de cálculos de rutas a redes de transporte urbano", que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º.Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducir la en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º.Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º.Deberes del autor.

- El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de él un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y confines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 10 de Junio de 2018

ACEPTA

Fdo……Manuel García Bellido……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MASTER EN INGENIERÍA DE TELECOMUNICACIÓN

Aplicación de Algoritmos de Cálculo de Rutas a redes de Transporte Público Urbano

Autor: Manuel García Bellido

Director: Miguel García Domínguez

Madrid

Junio 2018

APLICACIÓN DE ALGORITMOS DE CÁLCULO DE RUTAS A REDES DE TRANSPORTE PÚBLICO URBANO.

Autor: García Bellido, Manuel.
Director: García Domínguez, Miguel.
Entidad Colaboradora: BAOLAU.

RESUMEN DEL PROYECTO

En este proyecto se ha implementado el cálculo de rutas en redes de transporte urbano en la página web de la empresa Baolau. A lo largo del proyecto se han realizado las tareas necesarias para integrar esta nueva funcionalidad, desde la extracción de los datos hasta la implementación y representación de los resultados en su página web. Concretamente, el objetivo principal era añadir la red ferroviaria urbana de las ciudades de Bangkok y Singapur en su sistema.

Palabras clave: Baolau, Datos, Desarrollo, Transporte Urbano, Rutas, Nodos, Algoritmo.

1. Introducción

Este proyecto se ha llevado a cabo durante unas prácticas internacionales de 5 meses de duración en la empresa Baolau, con sedes en Singapur y Vietnam. Su producto principal, Baolau.com, se trata de un motor de búsqueda de rutas multi-transporte que opera en Vietnam, Camboya, Laos, Tailandia y Myanmar. Tradicionalmente, las rutas de su sistema han estado formadas por transportes como aviones, trenes, autobuses y ferris.

Respecto al funcionamiento de la página, el usuario introduce la ciudad o estación de origen y el destino y el motor de búsqueda le proporciona los resultados óptimos para un día determinado. Se puede dar el caso que en la ruta haya una ciudad de tránsito en la que el usuario debe trasladarse de la estación de llegada a otra y coger un nuevo transporte que le lleve al destino final. En esas situaciones, el sistema solo era capaz de ofrecer dicha interconexión mediante un taxi.

Debido a que muchas ciudades de un tamaño considerable cuentan con más de un método de transporte público, el principal objetivo era aumentar el número de posibilidades en el entorno urbano. A pesar de que este tipo de rutas no son reservables, la inclusión de ellas en el sistema podría tener un gran impacto en la página porque hace que el usuario no tenga la necesidad de abandonar la plataforma en busca de otras opciones que le satisfagan más.

2. Definición del proyecto

Por tanto, el objetivo principal del proyecto es añadir al menos un nuevo tipo de transporte urbano que acompañe al taxi como método de interconexión de estaciones de una misma ciudad. Para ello, durante el inicio del proyecto se determinó que se iban a introducir en el sistema las redes ferroviarias de dos ciudades piloto, Bangkok y Singapur. A continuación se detallan los principales objetivos del proyecto.

- Obtención de la información: Como cualquier nodo y ruta del sistema, los elementos de la red urbana también deben estar definidos con ciertos atributos. Las

estaciones deben tener su nombre en distintos idiomas, su localización, su dirección y su horario. En el caso de las rutas, estas deben contar con atributos como la duración, la frecuencia de llegada o las distintas tarifas disponibles.

- Resultados precisos: Tras su implementación, tanto las rutas directas como las que posean trasbordos deben mostrar información precisa y verdadera, aunque su finalidad sea mostrarla solamente de manera informativa.

- Representación específica: Otro objetivo es que el usuario sea capaz de distinguir fácilmente que se trata de una ruta urbana al verla junto a las otras. Para ello, se diseñará una representación distinta a la mostrada tradicionalmente en Baolau y que muestre la información estrictamente necesaria.

- Creación de páginas urbanas: Por último pero no por ello menos importante, se crearán dos páginas cuyo contenido sea solamente las rutas y estaciones del transporte público elegido. De esta forma, se tendrá una página específica para Bangkok y Singapur a la que los usuarios puedan acceder durante su estancia en ellas.

3. Descripción de la arquitectura

La siguiente figura muestra las tareas que son necesarias realizar desde el comienzo del proyecto hasta la obtención de los resultados esperados ordenados cronológicamente.

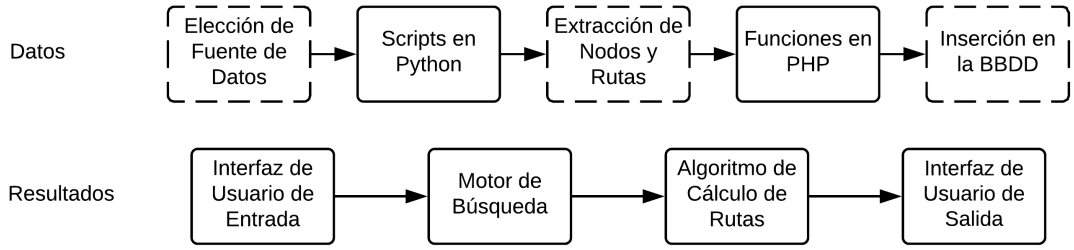

Ilustración 1 – Tareas ordenadas cronológicamente.

Respecto a los datos, el primer paso es encontrar la base de datos capaz de proporcionar la información buscada. Con el fin de extraer los atributos de los nodos y rutas se programa unos scripts en el lenguaje Python encargados de dicha tarea. Una vez se tiene esta información en local, a través de dos funciones de PHP se generan las entradas en la base de datos correspondientes a las redes urbanas.

Una vez los datos son accesibles para el sistema, el usuario realiza una búsqueda a través del interfaz de usuario diseñado específicamente para las páginas urbanas. El motor de búsqueda y el algoritmo de cálculo de rutas se encargan de proporcionar la ruta óptima directa y combinada respectivamente. Finalmente los resultados son mostrados con un aspecto propio de las rutas urbanas.

4. Resultados

En la siguiente captura de pantalla se muestra un ejemplo de un resultado devuelto por la página urbana de Singapur.

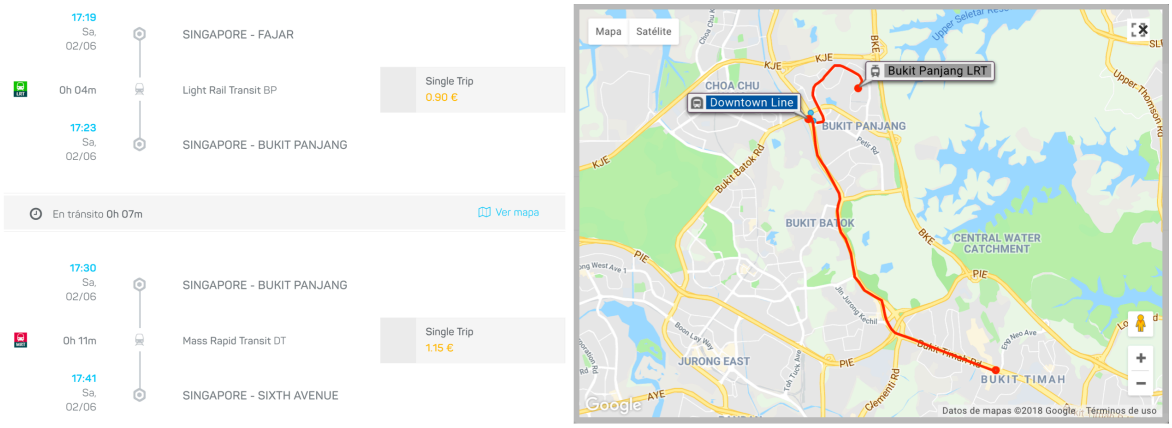

Ilustración 2 – Resultados entre dos estaciones de la red urbana de Singapur.

En este ejemplo, a la izquierda se muestra una ruta combinada en la que se tienen que coger dos tipos de transportes y a la derecha su representación en el mapa. Como se pude observar, la información mostrada en las rutas urbanas ha sido simplificada, por ejemplo, omitiéndose la dirección de los nodos. Respecto a su integración con la página principal de Baolau, las rutas se representan de una manera similar pero la diferencia es que los nodos urbanos no son accesibles desde el formulario de búsqueda.

Este es tan sólo un ejemplo de las rutas que es posible obtener tras el desarrollo de este proyecto. Como resumen, entre ambas ciudades se han introducido más de 200 estaciones y alrededor de 5000 rutas al sistema de Baolau.

5. Conclusiones

Tanto los objetivos marcados al principio del proyecto como la buena planificación de las tareas y el alcance del mismo han producido que la implementación se finalice a tiempo y correctamente. La metodología seguida durante el desarrollo del proyecto y la buena comunicación con el equipo de desarrolladores también ha favorecido esta situación.

Respecto a los resultados, estos han sido muy satisfactorios. El rendimiento del sistema con estas rutas ha sido el esperado y se han alcanzado los niveles requeridos de precisión de los resultados. Una situación similar se encuentra en el interfaz de usuario, en dónde la página principal integra de manera adecuada las nuevas rutas urbanas. Finalmente, el diseño y funcionalidades implementadas en las páginas urbanas de Bangkok y Singapur completan el proyecto otorgando un valor añadido a los nodos y rutas urbanas.

6. Bibliografía

- Página web oficial de Baolau, https://www.baolau.com

APLICATION OF ROUTE CALCULATION ALGORITHMS TO URBAN PUBLIC TRANSPORT NETWORKS.

Author: García Bellido, Manuel.
Supervisor: García Domínguez, Miguel.
Collaborating Entity: BAOLAU.

ABSTRACT

This project is based on the implementation of transportation networks route-calculation in the company Baolau. Tasks involved include data extraction, logic implementation and the display of results on the web page. Specifically, the main purpose of this project consists on adding the urban railroad network to Baolau's system.

Keywords: *Baolau, Data, Development, Urban Transportation, Routes, Nodes, Algorithm.*

1. Introduction

This project has been carried out during a five-month internship in Baolau, located both in Singapore and Vietnam. The company's main product, Baolau.com, is a multi-transport routes search machine operating in Vietnam, Cambodia, Laos, Thailand and Myanmar. Traditionally, the company's system has integrated routes by plane, train, bus and ferries.

Regarding the web page functioning, the user inputs the city or origin point and the destination, and the search machine provides the optimal results for a certain date. Sometimes a transitional route between two points inside a city is needed, for example, changing from one station to another. So far, the system has been only capable of supplying taxis as the only means of transport for these situations.

As most cities own more than just one type of public transportation, the main goal consists on providing the user more possibilities of moving around them. Even if these type of routes cannot be booked, their incorporation into the system might have a good impact on the web page as users would not have the need of looking elsewhere for more satisfactory results.

2. Project definition

This project primarily seeks to add at least one more option to the urban transportation means (apart from the taxis) to connect two points inside a city. For this purpose, Singapore and Bangkok were the two pilot cities whose railroad networks was chosen to be integrated into the system.

The main goals of this project are:

- Information gathering: Just as every node and route inside the system, those belonging to the urban network must also be defined by means of certain attributes. Stations must include their names in different languages, their location, address and opening hours. Routes are defined by their duration, frequency or the different fares assigned to them.

- Accurate results. After their implementation, direct as well as combinational routes must include real and accurate information, even though their main purpose is merely informational.

- Particular representation. Users must be able to discern urban routes from the rest. A new representation will be designed for this purpose that will only provide the required information.

- Urban sites creation. Last but not least, two different web pages will be configured which will only include the chosen public transportation routes and stations. This way, two different sites will exist (one for Singapore and another one for Bangkok), that can be accessed by users.

3. Architecture description

Figure 1 shows the tasks necessary to develop this project, chronologically sorted.

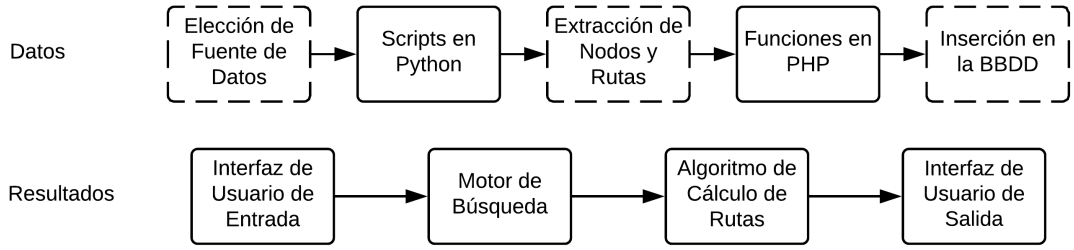

Illustration 1 – Chronologically sorted tasks.

The first step consists on selecting the database that is going to provide the needed information. A script written in Python will extract the nodes and routes attributes to store them locally. Afterwards, two PHP functions will transfer this information into Baolau's urban network database.

Once the data can be accessed from the company's system, the user can use the urban site interface to look for the connection routes between two points. Direct routes as well as combinational routes will be then calculated and displayed to the user employing the specific appearance used for urban routes

4. Results

Figure 2 shows a screenshot with Singapore urban site.

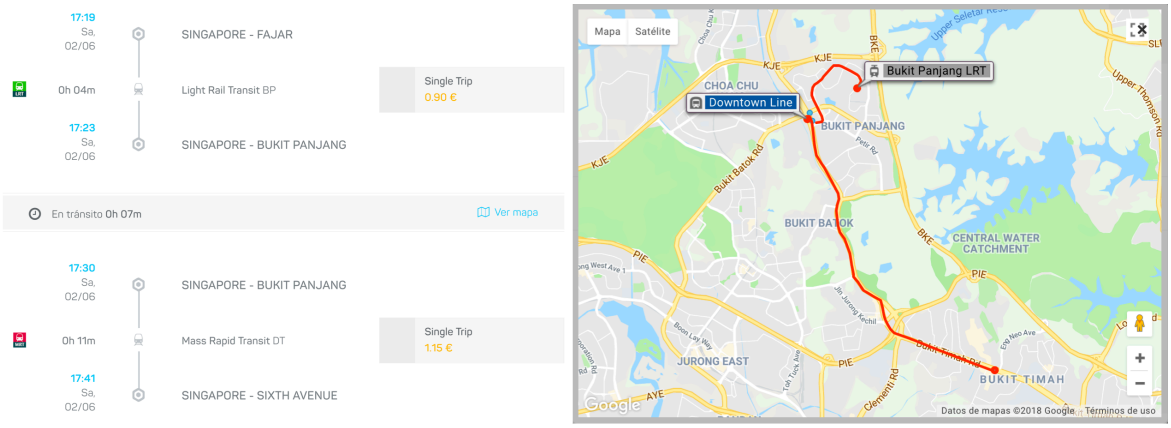

Illustration 2 – Connection between two station in Singapore urban site.

A combinational route is shown on the left of Figure 2, in which two different means of transportation need to be taken. The routes are represented on the right side of the image. As it can be seen, the information shown in urban site routes has been simplified. For instance, the node address has been omitted. Regarding the urban site integration to Baolau's principal page, routes are shown similarly. However, urban site nodes cannot be accessed from the search form.

This is just an example of the results that can be obtained after developing this project. To sum up, more than 200 stations and around 5000 routes have been added to the company's system.

5. Conclusions

The goals specification at the beginning of the project as well as a good task planning and scope definition have led to accurately finishing the project and in time. The methodology followed and extensive communication among the team members have encouraged this situation.

Results have also been very satisfying. The system's performance is as expected and the required accuracy level for results has been reached. The interface implementation can also be considered a success, thanks to the correct integration of the urban site routes. Finally, the design and features implemented in Singapore and Bangkok urban sites satisfactorily complete the project thanks to the added value their nodes and routes provide.

6. References

- Official Web Page of Baolau, http://www.baolau.com

Índice de la memoria

Capítulo 1. Introducción ***6***

1.1 Motivación del proyecto 9

Capítulo 2. Descripción de las Herramientas ***12***

2.1 APIs de Google 14

2.2 Lenguajes de Programación 15

2.3 Bitbucket y Sourcetree 16

2.4 Herramientas de Desarrollo Web Local 17

Capítulo 3. Estado de la Cuestión ***18***

3.1 Extracción de los datos 18

3.2 Algoritmo y motor de búsqueda 19

3.3 Mini-Site 20

3.4 Otro punto de vista 21

3.5 Una solución similar existente 22

Capítulo 4. Definición del Trabajo ***23***

4.1 Justificación 23

4.2 Objetivos 24

4.3 Metodología y Planificación 26

4.4 Estimación Económica 29

Capítulo 5. Sistema Desarrollado ***30***

5.1 Estudio de Viabilidad 30

5.1.1 Información Necesaria *31*

5.1.2 Posibles Fuentes de Datos *35*

5.1.3 Posible Alternativa *37*

5.1.4 Decisión Final *37*

5.2 Extracción de Datos 40

5.2.1 Uso de las Google APIs *40*

5.2.2 Nombres y Localización *50*

5.2.3 Rutas *53*

5.2.4 Localizaciones Exactas *56*

5.2.5 Rutas de Singapur *58*
5.2.6 Formateo de Ficheros *61*
5.2.7 Frecuencias y horarios *63*
5.2.8 Idiomas y Código de Líneas *67*
5.2.9 Tarifas *68*
5.3 Inserción de la Información en la Base de Datos 70
5.3.1 Inserción de Nodos *72*
5.3.2 Inserción de Rutas *74*
5.4 Implementación 77
5.4.1 Resultados Esperados *77*
5.4.2 Coexistencia de Rutas Urbanas e Interurbanas *80*
5.4.3 Motor de Búsqueda y Algoritmo de Dijkstra *84*
5.4.4 Interfaz de Usuario *91*

Capítulo 6. Análisis de Resultados 101

6.1 Datos 101
6.2 Interfaz de Usuario 103
6.3 Motor de Búsqueda y Rutas Resultado 105

Capítulo 7. Conclusiones y Trabajos Futuros 109

Capítulo 8. Bibliografía 112

Índice de figuras

Figura 1. Cronograma del proyecto. 27
Figura 2. Respuesta de Google Place API. Ejemplo JSON. 43
Figura 3. Respuesta de Google Geocoding API. Ejemplo JSON. 45
Figura 4. Respuesta de Google Directions API. Ejemplo JSON. 48
Figura 5. Respuesta de Google Geocoding API. Ejemplo JSON. 49
Figura 6. Problema en rutas largas. Google Maps Singapur. 54
Figura 7. Problema de las localizaciones exactas. Google Maps Singapur. 57
Figura 8. Nodo en formato JSON. 62
Figura 9. Ruta en formato JSON. 62
Figura 10. Ejemplo. Ruta combinada desplegada 93
Figura 11. Ejemplo. Mapa de una ruta combinada 95
Figura 12. Ejemplo. Resumen de la búsqueda y rutas resultado plegadas. 97
Figura 13. Formulario de la página urbana de Singapur. 97
Figura 14. Miniatura de la página de entrada de Singapur 99
Figura 15. Versión móvil de la página urbana de Singapur. 100
Figura 16. Mapa de Singapur. MRT y LRT. 102
Figura 17. Mapa de Bangkok. MRT, BTS y ARL. 103
Figura 18. Ejemplo. Rutas combinadas para distintas preferencias 106
Figura 19. Integración de rutas urbanas en la página principal. 108

Índice de tablas

Tabla 1. Estimación económica del desarrollo. 29
Tabla 2. APIs utilizadas.....38
Tabla 3. Otras páginas utilizadas para extraer tarifas. 39

Capítulo 1. INTRODUCCIÓN

La totalidad de este proyecto forma parte de unas prácticas internacionales llevadas a cabo en la empresa Baolau con sedes en Singapur y Vietnam. Su producto, la página Baolau.com [1], fue creado en 2013 para proporcionar un servicio que por entonces no existía dónde lo pretendían ofrecer. Consiste en un servicio de búsqueda multi-transporte cuya actividad principal se centra en los países de Vietnam, Camboya, Tailandia, Myanmar y Laos. A partir de dos ciudades o puntos de interés en alguno de estos países, la página proporciona gran variedad de rutas posibles que los conectan.

Dada la infraestructura de servicios de transporte existente en estos países asiáticos, existen pocas aplicaciones que operen en estas zonas y que ofrezcan este tipo de servicios. Adicionalmente, si el usuario que pretende hacer un viaje por estas zonas reside en otros países extranjeros, es muy posible que se encuentre con limitaciones a la hora de obtener información o realizar reservas. La dificultad a la hora de planificar el transporte en estos países era su objetivo a resolver.

En los últimos años, han logrado acumular gran cantidad de información de los medios de transporte más utilizados y colaborar con ellos, obteniendo así su principal ventaja competitiva. Sus rutas pueden estar compuestas de aviones, ferris, trenes, autobuses y taxis. Colaborando con las principales compañías de los tipos de transporte mencionados, son capaces de cubrir la mayoría de las rutas existentes en el territorio de estos países.

Actualmente su red de transportes se encuentra en constante crecimiento, tanto geográficamente introduciendo más países como en colaboraciones con nuevas empresas de transporte. Parte de los recursos de la empresa es reservado expresamente para esta tarea ya que es fundamental para lograr mantener su ventaja competitiva y seguir teniendo éxito. Está misión de aumentar su capacidad de ofrecer más servicios y en más lugares es la base sobre la que se sitúa este proyecto.

Siendo los turistas su mayor volumen de clientes, una parte esencial de su servicio es ofrecer conexiones entre los grandes núcleos urbanos de los países en los que operan ya que suelen ser los más transitados. Esta conexión interurbana suele ser llevada a cabo principalmente mediante aviones, trenes y autobuses mientras que la conexión intraurbana se realiza mediante taxis. Hasta ahora se han centrado en las interurbanas ya que son las rutas que requieren de mayor anticipación a la hora de planificar un viaje y es posible reservar con antelación.

Entre sus objetivos más próximos, se encuentra amplificar el número de opciones a nivel intraurbano. La mayoría de los países de un cierto tamaño cuentan con otros tipos de transporte público además de los taxis, como por ejemplo, la red ferroviaria urbana o metro. A pesar de que estos tipos de transporte urbano no suelen requerir de una reserva previa y por tanto no tendrían el uso corriente del resto de rutas, es conveniente que la página web las tenga a su disposición.

Entre algunas de las ventajas que podría suponer trabajar con rutas urbanas se encuentra tener la posibilidad de puntos de origen y destino más realistas. En lugar de partir desde, por ejemplo, un aeropuerto, podría comenzar desde la parada de metro más cercana al usuario y llegar a la más próxima a su estancia vacacional. Del mismo modo, con rutas urbanas es posible ofrecer más opciones de interconexión entre estaciones de tren, de autobús o aeropuertos además de los taxis ya implantados.

En este contexto, el problema principal era la limitación de un solo método de transporte para conectar las rutas interurbanas entre sí. Con el objetivo de ofrecer rutas más realistas, con más alternativas y con precios más competitivos era necesario romper con esa limitación. Por tanto, la solución de este problema y en lo que consistirá este proyecto, es la introducción de un nuevo método de transporte urbano en su motor de búsqueda.

Estudiando la viabilidad del proyecto, se tuvieron en consideración los dos métodos de transporte más usados en la mayoría de los países, la red de autobuses y el metro. Cómo añadir un nuevo tipo de transporte que además funciona de distinta forma en cada país puede llegar a ser muy complejo y costoso, se seleccionó de entre ambos el más sencillo de

implantar. Adicionalmente, se introducirá solamente en dos de los países donde operan (concretamente en dos ciudades) de forma que puedan observar si su efecto es el esperado.

Además de estas dos posibles soluciones, el metro o el autobús, se puede tener en cuenta una tercera cuyo uso se encuentra en constante crecimiento. Se trata de aplicaciones móviles como Uber [2] o Grab [3] y que en estos países han tenido una gran acogida. Sin embargo, al ser un transporte muy similar al ya utilizado, los taxis, desde un primer momento fue descartado ya que no suponía para el usuario una alternativa muy distinta y no aportaba tanto valor cómo las otras dos opciones.

De las soluciones propuestas, el metro aúna ciertas características que le convierten en el candidato ideal. En primer lugar, la red del metro es considerablemente más sencilla tanto en número de nodos como de rutas que la red de autobuses. De la misma forma, el trayecto de los autobuses urbanos puede sufrir cambios más fácilmente, lo cual implica una mayor dificultad en mantener las rutas actualizadas en el sistema. Por último, otra ventaja de elegir el metro sobre los autobuses es que la frecuencia de llegada y duración de los trayectos es prácticamente constante mientras que en los autobuses sufre grandes variaciones debido al tráfico propio de las grandes urbes.

Las dos ciudades elegidas para implantar este nuevo método de transporte son Bangkok y Singapur. Ambas ciudades forman parte de la red de Baolau pero no del conjunto de los núcleos urbanos y países en el que se encuentra la mayoría de su volumen de negocio. Por esa razón, el posible impacto que pudiera causar, tanto positivo como negativo, sería más controlable que si se implanta en otras ciudades con mayor tránsito.

Debido a estas razones y como conclusión de esta breve introducción, el objetivo principal de este proyecto es la implantación de la red de metro de las ciudades de Bangkok y Singapur en el motor de búsqueda de Baolau.com. Tras este proceso las nuevas rutas deberán ser usadas por el algoritmo preferencial de cálculo de rutas de su motor de búsqueda y coexistir con el resto de rutas del sistema de manera natural.

1.1 MOTIVACIÓN DEL PROYECTO

Conseguir que Baolau.com sea la única plataforma necesaria a la hora de planificar un viaje a través de los países en los que opera es el principal objetivo a cumplir. Por ello, es crítico que el usuario no se vea limitado por las opciones que ofrece el motor de búsqueda. Cómo se ha explicado en la introducción, esta problemática solamente se encuentra en las conexiones intraurbanas y no en las interurbanas, en las que ya es posible encontrar gran cantidad de rutas y alternativas. Aunque las rutas urbanas no vayan a ser parte del conjunto de rutas con las que Baolau obtiene beneficios, siguen siendo importantes ya que es posible que sea la causa de que el usuario abandone la página en busca de otras opciones que le satisfagan más.

Resolver este problema es la motivación principal de este proyecto. Como resultado, las nuevas rutas urbanas compuestas de las redes de metro podrán aparecer tanto al comienzo y final del viaje cómo en forma de interconexión entre dos estaciones de otros tipos de transporte. Si el resultado de este proyecto es el esperado, se estudiará integrar el resto de transportes urbanos de una forma similar para cubrir todas las posibilidades y mejorar la experiencia del usuario al usar la página web.

Técnicamente, la introducción de las rutas urbanas puede parecer una tarea sencilla ya que, aun tratándose de un contexto distinto, siguen siendo rutas. Sin embargo, elementos clave como la tarificación o los horarios de estos sistemas de transporte funcionan de manera totalmente distinta a la de una típica ruta interurbana. Aunque estas diferencias se explicarán de manera detallada en los siguientes apartados, un ejemplo podría ser la frecuencia. Mientras en una ruta típica de un tren interurbano se ha establecido previamente una hora de salida y de llegada , una línea de metro suele funcionar sin horarios fijos y las llegadas del siguiente tren suele venir determinado por una determinada frecuencia.

Este sencillo ejemplo, es una diferencia que conlleva gran cantidad de modificaciones en el código actual de la página, desde la interfaz de usuario hasta la estructura de la base de

datos, pasando por la lógica del motor de búsqueda y su algoritmo. De todos modos, los cambios suponen una pequeña parte del conjunto total de la página y es necesario y conveniente reutilizar lo ya existente. A lo largo del desarrollo del proyecto, se describirá en detalle que parte del código tuvo que ser desarrollado para este fin específico.

Al contrario que con el resto de rutas, las urbanas también tienen ciertas ventajas. Al tratarse de un sistema de transporte en el que no es necesario obtener un billete previo al viaje, no es posible realizar una reserva. Debido a este hecho, la información relacionada con el precio estimado del trayecto será sólo de uso informativo. Esto implica que a nivel técnico no será necesario realizar ciertas tareas que sí lo serían para las rutas interurbanas. Llevar a cabo la implementación de todos estos cambios de la forma más eficiente posible con el fin de introducir este nuevo tipo de rutas es la principal motivación de este proyecto.

En segundo lugar, actualmente la aplicación de Baolau es utilizada para planificar grandes viajes y es necesario poder conectar estaciones de una misma ciudad con algún método de transporte. Sin embargo, en un futuro, estas conexiones urbanas pueden hacer que el uso de la aplicación sea distinta al actual. Por ejemplo, podría ser utilizada por la población de una ciudad para planificar un trayecto entre dos puntos de ella sin necesidad de que fuera parte de un viaje más largo que envuelva viajar a otra ciudad.

Este posible segundo uso de la aplicación se podría dar con una subpágina propia para cada ciudad y su objetivo sería facilitar el transporte solamente a nivel urbano. Aunque de momento Baolau no posea la intención de hacer esto de manera oficial, la introducción del transporte público urbano en su sistema facilita posibles desarrollos de este tipo próximamente. Desde esta perspectiva, los futuros usos de las rutas urbanas también suponen una motivación para llevar a cabo este proyecto.

Uno de los procesos que hay que realizar para lograr implantar las rutas urbanas es la extracción de información. Está tarea da lugar a principios del proyecto y la forma en la que se llevará a cabo también forma parte de la motivación. Normalmente, la extracción de este tipo de datos suele ser un proceso que envuelve a terceras partes o en su defecto,

tiende a ser muy tedioso. La principal causa de este problema es que esta información es de difícil acceso o no se encuentra disponible.

En este contexto, y por varias razones que se explican más adelante, se ha decidido automatizar el proceso de extracción de datos lo máximo posible. Lo que se pretende con esto es que la recolección de información de las rutas urbanas sea lo más fácil posible en futuras ocasiones. Como se ha explicado en la introducción, a lo largo del proyecto se trabajará con dos ciudades y se dejará de lado otras en las que Baolau también quiere implantarlo pero no hasta que se comprueba el resultado de este proyecto. Pensando en esta futura implantación total conviene automatizar el proceso de extracción de datos.

En la planificación del proyecto se ha reservado gran parte del tiempo total para esta tarea ya que, dependiendo de varios factores, puede sufrir retrasos si esta se complica. Algunos de los factores de los que depende directamente son la elección de las fuentes de datos, la programación de los scripts encargados de automatizar el proceso o incluso la necesidad de licencias a nivel económico. Aun siendo de dudosa viabilidad, la automatización de la extracción de los datos de las rutas urbanas forma parte de la motivación del proyecto debido a sus grandes futuras ventajas.

Por último, otro motivo por el que este proyecto es interesante es el hecho de lograr que la implantación funcione correctamente. Especialmente debido a que la página ha sido programada en un lenguaje de programación desconocido por el alumno como es PHP [4]. También es verdaderamente útil conocer las herramientas de desarrollo web que se utilizan en un entorno laboral cómo el que encontramos en la empresa dónde se realiza el proyecto. Estos recursos, que se explican con más detalle en el siguiente apartado, unidos a un entorno de desarrollo web en equipo, forma un contexto nuevo para el alumno y supone la cuarta motivación para llevar a cabo este proyecto satisfactoriamente.

Capítulo 2. DESCRIPCIÓN DE LAS HERRAMIENTAS

En esta sección se describen cada una de las herramientas y recursos que se han usado en el desarrollo del proyecto. Como se ha explicado anteriormente, es posible diferenciar dos procesos principales en el proyecto, por un lado la extracción de datos y por otro el desarrollo web. Esta diferenciación también sucede en las herramientas utilizadas para ello.

Respecto a la recolección de datos, la fuente a utilizar son tres APIs (*Application Programming Interface* o en español, Interfaz de Programación de Aplicaciones) proporcionadas por Google [5]. Estas APIs ofrecen un cómodo acceso a información que Google utiliza en algunas de sus aplicaciones más famosas como es Google Maps [6]. El hecho de utilizar una API facilitará la automatización de la extracción de datos.

Los ficheros encargados de esta extracción serán programados mediante el lenguaje de programación Python [7]. Se ha elegido este lenguaje ya que es uno de los más populares para el tipo de tarea que se requiere. Con una sintaxis muy sencilla, el código será muy leíble sin necesidad de ser muy extenso. Además, al tratarse de un lenguaje muy usado para el *backend* de aplicaciones web, ofrece funciones que simplifican muchas tareas que se llevarán a cabo en este proceso.

La otra gran parte del proyecto se trata de un desarrollo web en equipo con los desarrolladores de Baolau. Como en cualquier desarrollo de una aplicación se ha de llevar un especial cuidado con la gestión de versiones, y con mayor razón si se trata de un desarrollo en el que participan más de una persona. Por tanto se utilizarán Bitbucket [8] y Sourcetree [9] como aplicaciones de gestión de cambios.

Para que el desarrollo en paralelo sea posible en un equipo, se programará de manera individual en local. MAMP [10] es una herramienta que facilita la configuración del servidor web y base de datos. Consiste en una solución única con la que se puede montar un entorno web de manera local y es ideal para este tipo de desarrollos. La única

herramienta adicional que será necesario utilizar es un editor de código y un depurador. La IDE (*Integrated Development Environment* o Entorno de Desarrollo Integrado) elegida es Visual Studio Code de Microsoft [11].

Por último, los lenguajes de programación utilizados para programar la página web y tener acceso a la base de datos son PHP, Javascript [12] y SQL [13]. PHP, capaz de realizar funciones tanto en el *backend* como en el *frontend* se integra perfectamente con HTML [14] facilitando el desarrollo. Para ofrecer ciertas funcionalidades exclusivas del *frontend* se usa el lenguaje más popular para ello como es javascript. Para finalizar, mediante sentencias SQL se accede a la base de datos en los momentos en los que se necesite.

Salvo las herramientas usadas para la extracción de datos, no se ha tenido que tomar una decisión entre otras posibles alternativas ya que era necesario usar las usadas en la empresa dónde se realizaban las prácticas. Por esta razón, no se ha hecho y no se refleja en este informa ningún estudio previo entre otras posibles herramientas para las mismas funciones. Adicionalmente, todas estas herramientas son de uso gratuito y no han su elección no ha supuesto ningún coste, excepto con la superación de un determinado límite de usos de las APIs de Google que se explica más adelante.

A continuación se explican con más detalle cada una de estas herramientas y recursos.

2.1 APIS DE GOOGLE

Todos los datos excepto las tarifas serán extraídos a través de tres APIs de Google. Google ofrece una funcionalidad específica en diferentes servicios con distintas APIs en lugar de una única API que englobe todas sus funcionalidades. Esto permite a los desarrolladores usar exactamente los servicios que requieren sin la complejidad de grandes documentaciones. Además, el uso de cada una viene limitada por un determinado número de peticiones a sus servidores (por lo general 2500 peticiones diarias) para su versión gratuita. A continuación se introducen las tres APIs elegidas.

En primer lugar, se hará uso de la Google Places API [15], la cual permite buscar sitios sin necesidad de una dirección a través de palabras clave. La base de datos a la que se accede a través de estas peticiones es la misma usada por Google Maps por lo que es muy probable que se encuentren las estaciones buscadas. El objetivo es obtener las coordenadas de dichas estaciones (los nodos de la red urbana) a partir de sus nombres.

La API de la cual se hará más uso es la llamada Google Directions API [16]. Con este servicio obtenemos posibles rutas entre varios puntos del mapa generalmente representados en coordenadas. Al limitar el tipo de transporte a trenes urbanos se obtiene las rutas que realiza el metro. A través de varios scripts de Python que ejecutan unas determinadas peticiones se puede obtener información como la duración, la distancia, el horario e incluso, la frecuencia de los trenes.

Por último, la Google Geocoding API [17] se utilizará para realizar una serie de comprobaciones. Concretamente, la llamada geocodificación inversa permite obtener el tipo de sitio a partir de sus coordenadas. Sin embargo, no se hará uso de la geocodificación normal ya que esta API solamente devuelve las coordenadas a partir de una dirección, y no a partir de ciertas palabras clave como ocurría con la Google Places API.

2.2 *LENGUAJES DE PROGRAMACIÓN*

Siguiendo con la recolección de datos, el lenguaje de programación utilizado para desarrollar los scripts que, ejecutando peticiones a través de las anteriores APIs obtenían los campos buscados, es Python. Python es un lenguaje *open source* multiparadigma que soporta tipado dinámico y es multiplataforma. Aunque no se ha utilizado ninguna característica que no encontremos en otros lenguajes como Java [18], su legibilidad y simplicidad ya era una ventaja. Además su intérprete incluye un modo interactivo en el que se pueden introducir expresiones una por una, lo cual facilita este tipo de desarrollos. Por último y no menos importante, la comunidad de Python es muy activa y extensa, por lo que cualquier tipo de duda durante el desarrollo no supone un gran esfuerzo.

PHP y Javascript son los lenguajes con los que la interfaz de usuario de Baolau ha sido desarrollada. Son la pareja de lenguajes más común en desarrollo web y no es necesario detenerse mucho en ellos. PHP es el acrónimo de *PHP Hypertext Preprocessor* en inglés, y fue desarrollado expresamente para el desarrollo web de contenido dinámico. Como característica principal de PHP, destaca su capacidad para poder estar escrito en el propio fichero de html. Cubriendo ciertas limitaciones de PHP, se encuentra Javascript, muy usado en desarrollo web y con infinidad de usos en el lado del cliente.

Por último, SQL, en inglés *Structured Query Language*, es un lenguaje específico que da acceso a bases de datos relacionales y el que se ha usado en este proyecto. Concretamente, la base de datos utilizada para desarrollar este proyecto es MySQL [19], que como su propio nombre indica, es accedida con este lenguaje.

2.3 BITBUCKET Y SOURCETREE

A la hora de realizar un desarrollo web compartiendo código entre varios desarrolladores, es extremadamente importante llevar un control de los cambios que se están produciendo al mismo tiempo. Dos de los aspectos más importantes que ofrecen este tipo de herramientas es facilitar la gestión de conflictos y llevar un control de las versiones del código de manera más cómoda. Para esta tarea, las herramientas usadas, tanto en la empresa como en este proyecto, son Bitbucket y Sourcetree.

Respecto a Bitbucket se trata de una solución Git [20], lo cual quiere decir que utiliza el software de control de versiones llamado Git. Expresamente desarrollado para los trabajos en equipo, ofrece una serie de funcionalidades como por ejemplo la división en ramas con permisos individuales o la inclusión de comentarios en las revisiones de código. Actualmente no existe una versión oficial de escritorio pero colabora estrechamente con Sourcetree.

Sourcetree es una herramienta de escritorio para acceder a los repositorios de Git como el utilizado con Bitbucket. Ejecutado en local y con una interfaz gráfica rediseñada, el acceso a estas aplicaciones es más cómodo. Adicionalmente, el uso de estas herramientas también se puede combinar con los IDEs de desarrollo manteniendo un código consistente en ambas plataformas de manera automáticamente.

Estas dos aplicaciones software tienen un gran papel en el desarrollo de las modificaciones diarias que sufre la página web de Baolau y sin ellas no se podría tener un desarrollo tan ágil. En la realización de este proyecto se ha hecho uso de ambas a la hora de planificar e implementar los cambios a realizar en la página.

2.4 *HERRAMIENTAS DE DESARROLLO WEB LOCAL*

Antes de hacer uso de las herramientas mencionadas anteriormente, es necesario que cada desarrollador trabaje independiente en la parte del código que le corresponde, por tanto, su equipo debe estar configurado para dicha tarea previa. Este entorno de desarrollo local suele estar compuesto de un servidor web que simule el acceso a la página mediante el protocolo http. Además es necesario que se tenga un duplicado de la base de datos (o una sección de ella) en local generalmente también en un servidor.

En el desarrollo de este proyecto se ha decidido usar una herramienta llamada MAMP. Esta aplicación es una solución única y sencilla para llegar a tener un entorno web sin necesidad de instalar y configurar cada servidor por separado. Se trata de un programa cuyo nombre es un acrónimo del software que se tiene tras su instalación.

La primera M se corresponde con el sistema operativo Mac OS X [21] en el que es capaz de funcionar (también existe una versión para Windows [22] con el nombre XAMP). La siguiente letra hace referencia a el servidor web conocido como Apache [22]. Como ya se ha explicado, para poder observar los resultados según se va desarrollando nuevo código, es necesario acceder a la página a través del protocolo que se usará en producción. Apache es uno de los servidores web más comunes para este fin.

El siguiente es el ya introducido sistema gestor de bases de datos MySQL. MySQL está considerada como la base de datos de código abierto u *open source* más popular del mundo en entornos de desarrollo web. MAMP se encarga de instalar esta herramienta junto con las últimas a las que corresponde la última letra, los lenguajes de programación Python, PHP y Perl [23]. Esta es selección de lenguajes de programación son los más usados en programación web y es por eso por lo que MAMP los incluye. Se hace uso de un editor de código que soporte los lenguajes de programación que se utilizan en el proyecto.

Capítulo 3. ESTADO DE LA CUESTIÓN

Tal y como se explica más detalladamente en el apartado de objetivos del proyecto, la implantación de un nuevo método de transporte en su motor de búsqueda conlleva dos procesos principales. En primer lugar es necesario realizar la extracción de la información de los nodos y rutas que forman la red urbana. Tras su correcto almacenamiento en la base de datos, el segundo proceso consiste en realizar los cambios necesarios en su sistema para que el algoritmo de cálculo de rutas sea capaz de usar la nueva información y proporcionar los resultados esperados. En este apartado se explica brevemente la situación previa al proyecto respecto a estos dos procesos.

3.1 EXTRACCIÓN DE LOS DATOS

En la mayoría de los casos, la extracción de datos es un largo proceso en el que elegir la fuente de dónde provienen es lo más importante. En un principio se investiga si los datos están disponibles de las fuentes oficiales, a través de la página web o poniéndose en contacto con la empresa de transporte en cuestión. Si eso no es posible, se tiende a ir a otras fuentes de terceros de las que se puede obtener la información de cierta forma automatizada y en el peor caso, manualmente. Además, este es un proceso que se debe repetir según pasa el tiempo y la información de los nodos y las rutas quedan desactualizadas. Cómo se ha explicado anteriormente, en el caso particular de este proyecto, se ha intentado automatizar al máximo este proceso.

Los pasos comentados son los que se suelen seguir a menudo para extraer datos. En la mayoría de los casos, Baolau no tiene la necesidad de ir a fuentes de terceros o extraerla de forma manual ya que colabora de primera mano con la empresa de transporte en cuestión. Al tratarse de una colaboración, las empresas les proporcionan los ficheros con la información necesaria y en dónde solamente hace falta darles el formato que se siga en la

base de datos. Sin embargo, esto suele ocurrir en las rutas interurbanas y no en el contexto urbano.

Hasta el momento, las únicas rutas urbanas almacenadas en el sistema son las realizadas por taxis. Estas fueron creadas principalmente para realizar alguna interconexión entre dos estaciones de la misma ciudad y poder formar combinaciones de más rutas. La razón por la que se tomó la decisión de usar taxis como método de transporte urbano fue, en parte, por la facilidad a la hora de extraer los datos de estas rutas. Se trata de unas rutas dinámicas, sin un horario fijo y sin puntos de partida y llegada fijos. De esta forma, introducir los datos de cada una de las rutas se podía hacer de manera automatizada a partir de los nodos origen y el destino que se quisieran conectar.

Por esta razón, inicialmente no hubo necesidad de extraer información tal y como se hace normalmente con el resto de rutas del sistema. Para la consecución de este proyecto se debe seguir un enfoque totalmente distinto debido a la naturaleza del sistema de transporte a implantar. Más parecido a las rutas tradicionales, se tienen que extraer los nodos (estaciones de metro) y rutas (líneas de metro) con información como la geolocalización, los nombres de las estaciones, nombre de la línea, la frecuencia, las tarifas, etc.

3.2 ALGORITMO Y MOTOR DE BÚSQUEDA

En esta sección se explica la situación inicial respecto al algoritmo de cálculo de rutas y al motor de búsqueda. Estos dos componentes de la aplicación son los más importantes ya que generan el resultado que el usuario está buscando cuando accede a la página. Gracias a ellos, se pasa del conjunto de todas las rutas previamente almacenadas en la base de datos a una cantidad pequeña de rutas o combinaciones de ellas que muestran el camino entre el origen y el destino que el usuario ha introducido para una fecha determinada.

Por una parte, el llamado motor de búsqueda es el encargado de preparar los datos sacados de la base de datos de tal forma que el algoritmo los pueda utilizar. Este proceso es esencial ya que dependiendo del tipo de ruta, orígenes, destinos y otros factores, existen

una serie de excepciones que hay que tener en cuenta previo a ejecutar el algoritmo. Adicionalmente, el motor también se encarga de diferenciar (y hacer distintas acciones) si los resultados serán rutas directas o combinaciones de ellas.

A la hora de implementar las rutas urbanas, el motor de búsqueda sufrirá muchos cambios ya que estas nuevas rutas requieren de muchas excepciones para que el algoritmo de búsqueda funcione. Es por esta razón por la que este componente de la aplicación se ha incluido en este apartado.

El algoritmo, llamado por el motor de búsqueda en diversas ocasiones, se encarga de buscar el camino óptimo y devolverlo al motor de búsqueda. El algoritmo del camino más corto de Dijkstra es el algoritmo usado a lo largo del proyecto. Este algoritmo es ampliamente conocido en muchos campos de la tecnología y ha sido usado para muchas aplicaciones de diversa naturaleza. Cómo su propio nombre indica, su función principal es la de sacar el camino más corto, sin embargo, con la implementación adecuada, no tiene por qué tratarse de distancias sino de varios factores. En el caso particular de Baolau, el algoritmo calcula el camino óptimo en base a dos factores que son la duración y el precio del viaje. La razón principal es porque son estos dos factores los más importantes para el usuario a la hora de diferenciar las rutas entre sí.

Respecto al algoritmo, es posible que a lo largo del proyecto sufra modificaciones e incluso se reemplace por otro que sea capaz de devolver más de una ruta como resultado. Por esta razón y debido que está estrechamente relacionado con el motor de búsqueda y el proyecto de rutas urbanas, era relevante conocer la implementación actual.

3.3 MINI-SITE

En el primer apartado de este documento se ha comentado la posibilidad de usar la aplicación resultado de este proyecto de una manera distinta. Si bien la idea inicial es integrarlo en la página actual para mostrar más opciones de interconexión urbanas, también se podría hacer uso de manera local dentro de una ciudad sin que las rutas tengan que

formar parte de un viaje. Para ello se creará una subpágina para cada ciudad con un uso exclusivamente urbano.

Respecto a esta tarea, ninguna de estas subpáginas existe a comienzos del proyecto y se deberán crear con las modificaciones pertinentes. Estas páginas tendrán un interfaz propio de cada ciudad y acceso a solamente nodos y rutas situados en su territorio y que formen parte de la red urbana. En apartados posteriores se definirá de una forma más detallada las funciones que se incluirán en ellas.

3.4 OTRO PUNTO DE VISTA

Respecto a la extracción de datos, también es necesario atender a otra solución radicalmente distinta a lo buscado inicialmente pero que proporcionaría una resultado similar para el usuario. En lugar de recolectar la información de los nodos y las rutas de la red urbana e implementarlas como se ha descrito, sería posible usar un servicio que proporcionara directamente la ruta óptima entre dos puntos determinados.

El mayor proveedor de este tipo de servicios es Google a través de sus APIs para desarrolladores. Google cuenta con varias aplicaciones que proporcionan un resultado parecido al que se quiere lograr con este proyecto. Además de implementarlo en sus propias aplicaciones ofrece mediante distintas APIs este tipo de servicios a otras empresas. Sin ir más lejos, la Google Directions API ofrece el conjunto de rutas más recomendables entre dos puntos del mapa y mediante distintos métodos de transporte.

Aunque pueda parecer a primera vista la solución más simple y fácil de implementar dado el problema a resolver, también presenta varios inconvenientes que se discutirán más adelante. De todos modos, es interesante tener en cuenta este nuevo punto de vista que, aunque se aleja desde un principio a lo que se quiere desarrollar, produce buenos resultados. Durante el estudio de viabilidad del proyecto se realizará una comparativa mostrando las ventajas y desventajas de utilizar un API externa como la mencionada.

3.5 UNA SOLUCIÓN SIMILAR EXISTENTE

La aplicación por excelencia que lleva a cabo la función que pretende integrar este proyecto es Google Maps. Sin extenderse a la actividad total de Baolau, en lo que a rutas urbanas se refiere, esta aplicación cumple con creces a la hora de interconectar dos puntos urbanos. Por ejemplo, algunas funciones diferenciadoras son la integración con el tráfico actual, uso de prácticamente todos los métodos de transporte públicos, e incluso la integración con aplicaciones de transporte de terceros como Uber. Todas estas razones, combinadas con su gran exactitud la convierten en una de las aplicaciones más utilizadas de navegación.

Aunque se tuviera la intención de igualar su exactitud de resultados y características, no sería posible ya que cuenta con información recolectada de forma exclusiva, como por ejemplo, la geolocalización de los usuarios con dispositivos Android, el cual no está al alcance de este proyecto. Sin embargo, lo que se pretende conseguir con este proyecto no es alcanzar su precisión ni cantidad de opciones a la hora de planificar un trayecto urbano sino integrar una pequeña parte de estas funciones en la aplicación actual. De este modo, en este proyecto se hará uso de Google Maps a la hora de hacer pruebas y comparar los resultados y no desde una perspectiva de competitividad.

También existen algunas aplicaciones y servicios web, mayormente para la ciudad de Singapur, que realizan funciones parecidas a las buscadas. Sin embargo, todas ellas quedan limitadas geográficamente a la ciudad de Singapur y no se encuentran correctamente actualizadas. En ese sentido, las páginas web oficiales del servicio de metro de ambas ciudades deberían poseer la información más actualizada tanto de las estaciones cómo de las rutas.

Capítulo 4. DEFINICIÓN DEL TRABAJO

En esta sección se explican algunos aspectos necesarios para la correcta consecución del proyecto, desde la justificación y objetivos del mismo, hasta la metodología utilizada y la planificación de las tareas a realizar. Finalmente se muestra una estimación económica del proyecto.

4.1 JUSTIFICACIÓN

En forma de recordatorio y como introducción para los apartados restantes de esta sección a continuación se comenta las razones principales que justifican la realización de este proyecto.

En primer lugar, llevar a cabo este proyecto nace a partir de la necesidad de resolver un problema no crítico de las rutas resultado. En concreto, el problema es la existencia de las rutas de taxis cómo única opción de ruta urbana en toda la red de transporte de Baolau. La solución consiste en incluir las redes de metro para que sus rutas aparezcan junto con la opción del taxi de manera informativa. El alcance de este proyecto cubrirá la implantación de rutas urbanas en su sistema para las ciudades de Singapur y Bangkok.

La segunda razón relevante es dar el primer paso en el contexto de rutas urbanas. Aunque no es estrictamente necesario, es conveniente para la empresa estudiar la forma de implantar rutas urbanas pensando para no quedarse en una posición de falta de competitividad en el futuro. De hecho, como resultado de este proyecto también se lanzarán dos páginas propias de cada ciudad a nivel urbano con un funcionamiento ligeramente distinto al de la página global.

Por último, también será necesario implantar dicha funcionalidad en conjunción con el algoritmo de cálculo de rutas que se utilice en ese momento. Estas tres razones principales forman la justificación del proyecto.

4.2 OBJETIVOS

Este proyecto ha sido realizado en su totalidad durante unas prácticas laborales de 5 meses de duración y sin posibilidad de prorrogarlo. Debido a estos límites en el calendario, ha sido esencial planificar adecuadamente los objetivos que se presentan en este apartado. Según el alcance del proyecto, se han planificado un total de cinco objetivos principales que se llevarán a cabo cronológicamente. A continuación se resumen brevemente cada uno de ellos.

- **Búsqueda de fuentes de información y recolección de datos**: Para que el proyecto sea viable es necesario tener acceso a fuentes de datos que proporcionen información de las estaciones, líneas, tarifas y horarios de la red ferrovial urbana de Singapur y Bangkok. Una condición crítica para que sea fuentes válidas es que la información esté lo más actualizada posible y cuyo acceso y recolección se pueda hacer de una forma automatizada y no de manera manual debido a los altos costes que supone.

  Tras elegir fuente de información más adecuado para este fin se recolectará la información requerida. En la siguiente sección se explica con más detalle cada uno de estos campos. Uno de los aspectos más importantes de esta tarea es intentar extraer esta información de la manera más automatizada posible. La razón principal de ello es el ahorro económico y temporal de este paso en futuras implementaciones para otras ciudades.

- **Inyección de la información en la base de datos SQL**: Una vez recolectados los datos y formateados de la manera correcta se introducirán en una base de datos accesible con sentencias SQL para que sea compatible con la actual base de datos de desarrollo. La forma en la que se almacenan estos datos puede ser fundamental para su posterior acceso y uso por lo que será necesario hacerlo pensando en el siguiente objetivo. Si las nuevas rutas urbanas estuvieran compuestas de campos

nuevos, el formato elegido para almacenarlos puede llegar a ser crítico para su correcta implantación.

- **Estudio e implementación de las nuevas rutas en el sistema**: Estudiar las posibilidades para implementar las nuevas rutas en actual motor de búsqueda y su algoritmo de cálculo de rutas. Llevar a cabo su implementación y comprobar que los resultados son los esperados. Es esencial que los cambios realizados tanto en el sistema cómo en el algoritmo no perjudiquen al rendimiento de ambos ni a las rutas existentes.

- **Representación de los resultados**: Representar los resultados de la manera más correcta, tanto si las nuevas rutas forman parte de otras rutas o son resultado único de la búsqueda. Se mostrará el resultado a través de la interfaz gráfica de Google Maps para mejorar la experiencia de usuario. Adicionalmente, se crearán las dos páginas urbanas de ambas ciudades y sus respectivos interfaces. Estas páginas deberán estar listas para pasar a producción al finalizar el proyecto y por tanto, el diseño y la implementación de sus funciones es una parte muy importante de este objetivo.

- **Perfeccionar la coexistencia de las rutas urbanas con las interurbanas**: Es necesario que tras la consecución de los anteriores objetivos, todas las características de la página web no se hayan alterado y funcionen de forma correcta. Para ello, habrá que resolver los conflictos que hayan producido los nuevos cambios y optimizar el código de la página para que ambos tipos de rutas coexistan correctamente. Igualmente, los cambios que han supuesto las dos páginas urbanas no deben afectar al funcionamiento de la página global y sus funciones.

4.3 METODOLOGÍA Y PLANIFICACIÓN

Respecto a la metodología del proyecto, al tratarse de un proyecto de desarrollo software, se debe seguir el avance del proyecto de forma periódica. Los objetivos descritos previamente, los cuales suceden de forma cronológica durante el proyecto, marcarán los hitos del proyecto. Se comenzará con la siguiente tarea tras finalizar la anterior y tener el visto bueno por parte del director del proyecto. De esta forma se llevará una supervisión que, sin consumir mucho tiempo, sea de gran utilidad.

Adicionalmente, durante el desarrollo se mantiene una comunicación directa con los miembros del equipo de desarrollo web para prevenir conflictos en el código si más de un desarrollador está programando al mismo tiempo. En otros casos, también es conveniente la asignación de tareas que aunque no formen parte del alcance de este proyecto si puedan estar relacionadas con él. Por ejemplo, la traducción a otros idiomas de ciertas partes del interfaz gráfico no entran dentro de este proyecto pero sin embargo es necesario para que la página esté operativa.

Cómo ya se ha explicado, el desarrollo software se llevará a cabo en local gracias a un servidor web y a una base de datos. Tras la confirmación de los cambios por parte del supervisor, el código será combinado con el ya existente de manera inteligente y resolviendo los conflictos que pudieran existir. Las herramientas utilizadas para este proceso son las mencionadas en el apartado 2.3 de este documento. Este procedimiento es muy común en el desarrollo de código distribuido en distintos miembros de un equipo y hace que el control de versiones se produzca de forma ordenada y sin problemas.

Generalmente, durante el desarrollo del proyecto, se combinará el código tras la implementación de cualquier funcionalidad por pequeña que sea. Es recomendable no subir grandes bloques de código que aúnen muchas funcionalidades para que el control de versiones y cambios sea más fácil de gestionar si surge algún problema en un futuro. De hecho, una de las ventajas de este tipo de herramientas es su capacidad de guardar el código para que sirva como back up y por seguridad, conviene hacerlo con regularidad.

A continuación se muestra un cronograma simplificado del plan de trabajo que se ha seguido en este proyecto. Este plan fue creado a principios del proyecto y se ha seguido sin cambios sustanciales, lo cual muestra un síntoma de buena organización del desarrollo y alcance del proyecto.

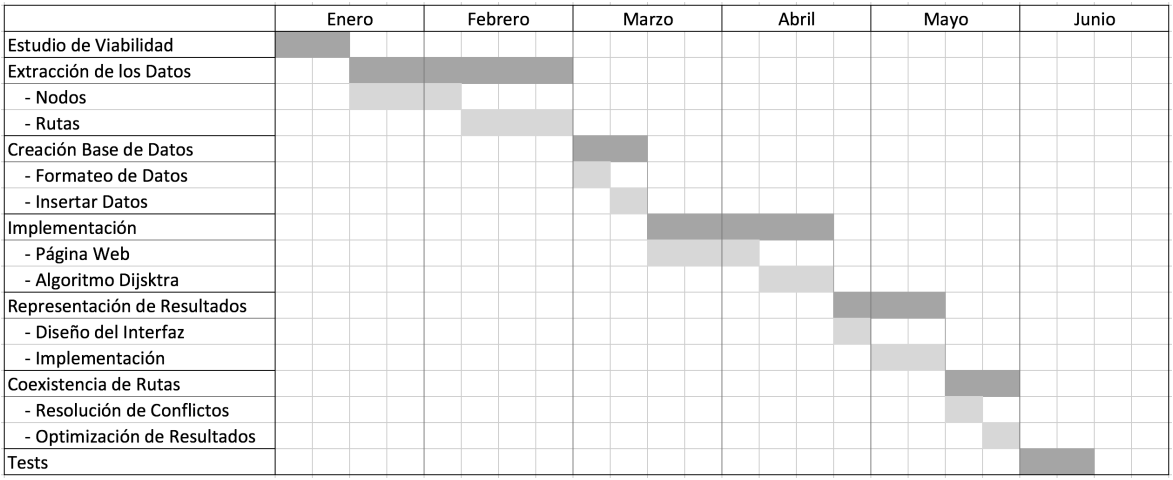

Figura 1. Cronograma del proyecto.

En primer lugar, se llevará a cabo un breve estudio en el que se determinen qué datos son los necesarios para integrar este nuevo método de transporte y se busque fuentes disponibles para obtener dichos datos. Si es necesario, también se intentará contactar con las fuentes oficiales para obtener la información de primera mano. Cómo ya se ha explicado, este proceso puede ser algo costoso en tiempo por lo que se ha planificado un total de dos semanas para esta tarea.

En la siguiente etapa del proyecto se extraerán los datos de la red ferrovial urbana de ambas ciudades, Singapur y Bangkok. Se comenzará con los datos relacionados con las estaciones como por ejemplo, los nombres de ellas en diferentes idiomas, sus coordenadas, su horario de apertura, etc. Tras los nodos, se obtendrán los datos de las líneas de metro o rutas, que contiene información como la duración, la distancia, la frecuencia de los trenes, etc. Esta tarea conlleva mucho tiempo ya que se pretende crear complejos scripts que automaticen esta tarea en un futuro.

Tras la obtención de los datos, se incluirán en la base de datos siguiendo el formato adecuado. Para ello será necesario crear otros scripts encargados de dicha tarea y que limpien, formateen y en algunos casos, compriman los datos para su correcto almacenamiento. En esta tarea se debe tener en cuenta la forma en la que en un futuro serán accedidos los datos. Se pretenden tener los datos perfectamente recolectados a mediados del mes de Marzo.

La implementación está formada por dos sub tareas. La primera se trata de las modificaciones a realizar en el código de la propia página web para que se pueda llamar al algoritmo de cálculo de rutas mientras que la segunda son los cambios del propio algoritmo. En este proceso, la implementación de esta parte no debe causar efectos negativos al resto de funciones de la página, por lo que se comprobarán todos los cambios cuidadosamente. En un principio, se va a usar el algoritmo del camino más corto de Dijkstra, pero es posible que se utilice otro que se esté desarrollando en ese momento.

A finales de Abril, se diseñará un sección de la página por cada ciudad para realizar las pruebas de visualización de resultados. Como se ha explicado en apartados anteriores, se usará el interfaz gráfico de Google Maps para mostrar las rutas desde origen y destino. Para llegar a ese punto previamente se habrá diseñado las principales funciones de esa sección de la página. La implementación de su funcionalidad no debería ser muy distinta a la realizada en la tarea anterior.

Una vez finalizados todos los objetivos principales del proyecto se llevará a cabo una serie de pruebas para comprobar la coexistencia de las nuevas rutas con las ya existentes. Se comprobará que el motor de búsquedas haga un uso correcto de estas rutas en conjunción con las rutas interurbanas. Para ello, esta tarea ha sido dividida en dos sub tareas llamadas resolución de conflictos y optimización de resultados y ambos van de la mano con la siguiente y última tarea, los *tests*. Según esta planificación, el proyecto tendrá una duración de cinco meses y medio y finalizará a mediados de Junio.

4.4 ESTIMACIÓN ECONÓMICA

En este apartado se presenta brevemente una estimación del coste total que conlleva el desarrollo de este proyecto. Cómo se verá a continuación, al hacer uso de herramientas *open source* y licencias gratuitas, esta estimación es muy sencilla y no requiere de un estudio mayor.

Respecto al equipo que se ha usado durante el proyecto, este tiene unas especificaciones de gama media. Cómo se puede comprobar, para este tipo de desarrollos web no es necesario ninguna característica fuera de lo común. El coste estimado de este equipo es de 1200 euros. Para el coste de la implementación se ha tenido en cuenta el salario de las prácticas como referencia. El tiempo consumido por el desarrollador durante los cinco meses del proyecto supone un total de 6000 euros y es la mayor parte del coste total del proyecto.

Todas las herramientas, tanto las usadas en local como en equipo, son gratuitas en sus totalidad o tienen una versión gratuita. En el caso de las licencias de las APIs de Google, estas también son gratuitas en el caso de no superar las 2500 peticiones diarias. Por esta razón y como se verá en el siguiente apartado, los scripts utilizados para extraer la información se han desarrollado para que hagan las menos peticiones posibles y no alcanzar ese límite tan rápido. A continuación se muestra una tabla con estos costes.

Concepto	**Justificación**	**Coste**
Equipo	Se ha usado un portátil con un procesador con rendimiento medio, 8GB de memoria RAM, 250GB de disco duro y Mac OS X cómo sistema operativo.	1200 €
Herramientas y fuentes de datos	Todas las herramientas y licencias de las fuentes de datos han sido gratuitas.	0 €
Coste de la implementación	Coste de un desarrollador web en prácticas en jornada completa (1200€/mes) por cinco meses.	6000 €
Total		7200 €

Tabla 1. Estimación económica del desarrollo.

Capítulo 5. SISTEMA DESARROLLADO

Este es el apartado principal del documento y en el que se explica al detalle todos los procedimientos que forman el proyecto. Cómo se ha explicado en las anteriores secciones, el proyecto cuenta con varias etapas que se han llevado a cabo de forma cronológica. A la hora de documentarlas, también se va a llevar este orden ya que facilita el entendimiento del proyecto y sus etapas.

5.1 ESTUDIO DE VIABILIDAD

En prácticamente cualquier proyecto de cierto tamaño es necesario realizar un estudio de viabilidad previo al comienzo del mismo. Con el estudio se realiza una breve investigación que ayuda a tener en cuenta los factores más importantes que determinan si el proyecto se va a poder llevar a cabo tal y como se espera. Dependiendo del proyecto, estos estudios son críticos para una buena planificación y futuro desarrollo del proyecto.

En el caso particular de este proyecto, no se ha pretendido llevar a cabo un estudio exhaustivo de todas las etapas y objetivos del mismo, sino solamente determinar si existía alguna gran dificultad que iba a impedir su desarrollo. Observando el alcance del proyecto se puede presuponer que la etapa en la que puede existir alguna dificultad externa que impida la consecución de las siguiente etapas es la extracción de datos. Una vez se tengan los datos necesarios para continuar, los siguientes pasos no suponen un desafío que no se pueda remediar.

Dada la infraestructura ya creada de la empresa, con sus bases de datos, lógica y recursos humanos, tanto la implementación del proyecto como la resolución de conflictos no supone un gran esfuerzo. Las únicas cuestiones que podrían suponer un impedimento para llevar a cabo el proyecto es la limitación temporal de 5 meses y la dificultad de la implementación por parte del alumno. Por esta razón, en la planificación se han tenido en cuenta estas posibles amenazas.

Un factor positivo a nivel de planificación, es que si en algún momento surge algún imprevisto que retrase el desarrollo de algunas de las etapas y no es posible finalizar el proyecto entero, no supone una pérdida de recursos para la empresa. Esto es debido a que cada una de las etapas se desarrollan cronológicamente. De esta forma, si se interrumpiese el desarrollo, por ejemplo, al finalizar la implementación, otro desarrollador pudiera continuar con la etapa del interfaz gráfico sin necesidad de rehacer las etapas anteriores.

Por tanto, la etapa más crítica del proyecto y de la cual depende el resto del mismo, se trata de la extracción de datos. Particularmente, la parte esencial de esta etapa es la búsqueda de fuentes de datos que puedan proporcionar todos los datos necesarios. Al ser un factor externo a la empresa y depender de terceras partes, este estudio de viabilidad se centra en su búsqueda.

5.1.1 INFORMACIÓN NECESARIA

En este apartado se define qué datos son los que se requieren para poder implementar la red urbana. Tal y como se ha explicado en secciones anteriores, el transporte público escogido ha sido el transporte ferroviario urbano conocido comúnmente cómo metro. Tanto en la implementación de la red interurbana como en el de intraurbana, es posible hacer una división de los datos en nodos y rutas. En el caso del metro, cada estación será tratada cómo un nodo y las propias líneas que las interconectas serán las rutas.

Aunque la mayor parte de la red urbana de trenes de ambas ciudades la forma el metro, existen distintos tipos de trenes y transportes y nos referiremos a ellos con nomenclaturas distintas según sus nombres específicos. En Singapur, aparte del metro o MRT (*Mass Rapid Transit*) [25], se encuentra el tren ligero con el nombre LRT (*Light Rail Transit*) [26]. Respecto a la red urbana de Bangkok, se encuentra el metro subterráneo con el mismo nombre MRT [27], un tren elevado conocido como BTS ó *Skytrain* [28] y por último un tren ligero que interconecta el centro de la ciudad con el aeropuerto llamado ARL (*Airport Rail Link*) [29].

A pesar de ser distintos medios de transporte, la información que se requiere de cada uno de ellos es la misma a no ser que no aplique. Esto es así ya que los datos deben tener el formato más parecido posible a los almacenados previamente en la base de datos de Baolau. En los dos siguientes apartados se explica con más detalle la información que debe contener cada nodo y ruta que forman la red urbana.

5.1.1.1 Nodos

La información que la fuente de datos debe ser capaz de proporcionar de cada estación es la siguiente:

- **Nombre en varios idiomas**: La base de datos debe contar con el nombre del nodo en, al menos, los seis idiomas en los que opera. Estos son el inglés, el castellano, el vietnamita, el japonés, el chino tradicional y el chino simplificado. Si el nombre de la estación en cuestión no tuviera traducción a alguno de estos idiomas se almacenaría en su idioma original.

- **Coordenadas**: Las coordenadas de la estación es una parte fundamental de la información del nodo ya que se usará más adelante para obtener las rutas con exactitud y en la implementación para ser capaces de mostrar la posición de las estaciones en un mapa.

- **Dirección**: La dirección de la ruta también debe ser obtenida y almacenada ya que las rutas que se le muestran al usuario final cuenta con esta información.

- **Horario**: Por último, y cómo única diferencia entre las redes interurbanas y las intraurbanas, se obtendrá el horario en el que opera cada estación. Más adelante se explica detalladamente por qué es necesario conocer el horario y horas de salida aproximada del primer y último tren del día. Se deberá tener en cuenta el horario tanto en días laborables cómo fines de semana.

El formato en el que se obtiene esta información puede variar según la fuente de datos que se vaya a utilizar. Esto no es un factor determinante a la hora de elegir la fuente ya que más

adelante se deberá formatear la mayoría de los datos de todos modos para que siga el mismo formato que el resto de la página y sea lo más consistente posible. Por tener una preferencia si existen varias posibilidades, el popular formato JSON [30] será el elegido debido a su simplicidad y fácil acceso con cualquier lenguaje de programación.

Así pues, la información que define unívocamente a un nodo o estación es el conjunto de estos cuatro campos, al que se le añade el nombre de la ciudad y el país en el que están. Estos dos últimos no se han enumerado junto con los anteriores ya que no es necesario que la base de datos lo proporcione porque son comunes a todos las estaciones de la red.

5.1.1.2 Rutas

A continuación se explican los distintos datos que son necesarios de cada ruta y con las que quedan definidas unívocamente.

- **Origen**: El origen del que parte el tren debe estar definido ya sea por el nombre de la estación, sus coordenadas o algún identificador si los nodos también lo tuvieran. Durante la implementación en la base de datos, cada nodo tendrá un identificador único y del que se hace uso para definir el origen y destino en una ruta. Sin embargo, durante la extracción de los datos, no es común que tengan tal identificador y por tanto se ha decidido almacenar toda la información posible que tenga en común con el nodo que pretende identificar.

- **Destino**: La información del nodo destino está definida en una ruta de la misma forma que el nodo origen.

- **Duración**: Respecto a la duración, esta puede estar definida una resolución de segundos o minutos para que sea viable.

- **Distancia**: La distancia entre el origen y el destino no es estrictamente necesaria ya que no se almacena en la base de datos. Sin embargo, en ciertas ocasiones, tener la distancia puede ser crítico para definir otro parámetro que sí que es totalmente necesario. Por ejemplo, en el caso concreto del metro de Singapur, el precio del

trayecto viene determinado por la distancia exacta entre la estación origen y destino. En ese caso en el que no es posible obtener la tarifa de otro modo sí que es crítico tener la distancia.

- **Tarifas**: El precio del trayecto o la tarifa es una parte fundamental de la ruta y la fuente de datos debe ser capaz de proporcionarla. Además, en muchas ocasiones hay más de una tarifa dependiendo de varios factores como la edad del usuario o forma parte de un viaje único. Como la página de Baolau permite seleccionar el tipo de divisa, en la base de datos debe estar tanto la tarifa principal como el resto almacenadas con una determinada divisa.

- **Frecuencia**: Otra diferencia relevante con respecto a las rutas no urbanas es la existencia del campo de frecuencia. Si bien no es necesario saber con exactitud este dato, si lo es tener un tiempo de espera aproximado entre cada tren ya que en ocasiones se puede alargar considerablemente. En casi todos los tipos de transporte que se tratan en este proyecto se encuentra una frecuencia variable dependiendo de si es hora punta o no, o sí se trata de un fin de semana. Esta frecuencia variable se almacenará con un determinado formato en el campo de frecuencia.

- **Código del tipo de transporte**: Generalmente, este código es un acrónimo del nombre del tipo de transporte como por ejemplo el *Mass Rapid Transit* o MRT de Singapur. La página web hace uso constantemente de este campo para diferenciar el comportamiento del motor de búsqueda en función del tipo de transporte o incluso para mostrar o no elementos gráficos al usuario en su interfaz.

- **Código de la línea**: Por último, el código de la línea es el nombre de la línea en cuestión. En muchas ocasiones, no se trata de un nombre completo sino el acrónimo de este (por ejemplo, la línea circular de Singapur tiene el código CC) y por esa razón nos referimos a él cómo código de la línea.

Estos son cada uno de los datos o campos del cual se debe componer cada ruta. Lógicamente, no se tiene porque usar solamente una única base de datos para obtener toda

la información. Como se explica en los apartados siguientes, todos estos datos provienen de varias bases de datos y se juntan en un único fichero y formato a medida que se van obteniendo.

5.1.2 POSIBLES FUENTES DE DATOS

Cómo se ha visto en el apartado anterior, el objetivo en los primeros momentos del proyecto era encontrar una o varias bases de datos que llegasen a dar la información mencionada. El primer paso fue buscar en las páginas oficiales, que al tratarse de transporte público, son páginas del gobierno. En ciertos casos, suelen existir APIs o cierto acceso a una base de datos de servicios del gobierno. El mejor ejemplo es el caso de Singapur, en el que a través de la página mytransport.org [31], creada por las autoridades oficiales del transporte de tierra (LTA) [32], te dan acceso a APIs de varios servicios. Entre ellos se encuentra acceso a la red de autobuses y sus posiciones GPS o a la monitorización del tráfico en tiempo real de la ciudad.

Desgraciadamente, no ofrecían ninguna API que diera acceso a la base de datos de la red metro, tanto para la ciudad de Singapur como para Bangkok. Otra alternativa oficial podría ser la descarga de parte de la base de datos de otras páginas del gobierno. Tras investigar en todas las páginas oficiales que tuvieran que ver con servicios públicos se logró obtener el nombre de los nodos actualizados de toda la red de Singapur. Respecto a Bangkok, no se encontró una base de datos que proporcionase información valiosa para el proyecto.

La última alternativa oficial fue ponerse en contacto con las autoridades con el fin de tener dicho acceso, pero de igual forma no fue posible. La idea inicial era obtener una API o base de datos estructurada para poder manejar y obtener todos los datos con facilidad y de manera automatizada. Sin embargo, otra opción factible es obtener los datos de una página que muestre los datos de manera informativa sin necesidad de tener acceso a ellos de manera directa. Un ejemplo de ello vuelve a ser la página mytransport.org. En una sección de la misma, te da la opción de seleccionar la estación origen y destino y te devuelve información detallada de dicha ruta.

Sin embargo, como se ha explicado, automatizar un proceso que envuelve peticiones de este tipo y tener que leer datos del código fuente de la página hacía el proceso muy complejo. Esta situación era similar para ambas ciudades y no existían vías oficiales disponibles. Por tanto, el objetivo pasaba a ser la obtención de la información a través de páginas de terceros y cerciorándose de la veracidad de los datos. Al igual que en el caso anterior, estas páginas seguían siendo una limitación al proporcionar acceso a su base de datos.

Algunas páginas que muestren información de la red urbana de Singapur son data.gov.sg [33] y *mytransport.sg*. Por parte de Bangkok algunas de ellas son *bangkokmrt.com* [34], *bangkokbts.com* [35], *transitbangkok.com* [36] o *railway.co.th* [37].

Por último y con el fin de desechar la alternativa manual, se hizo una búsqueda de las aplicaciones que ofrecían servicios similares a los que se pretende cubrir con este proyecto y averiguar su fuente de datos. La aplicación web por excelencia en geolocalización y servicios de guiado es Google Maps. Tras analizar la posibilidad de usar los servicios de Google como fuente de datos se ha llegado a lo siguiente.

Google cuenta con una serie de APIs con el que proporcionan a los desarrolladores acceso a su gran base de datos. Estas APIs están muy acotadas para que cada una de ellas proporciona tan sólo un servicio. De esta forma, al tener que obtener bastante información distinta es obligatorio hacer uso de varias APIs con sus posibles inconvenientes. El uso específico de cada una de ellas se describe con detalle en el siguiente apartado.

Respecto a la veracidad de los datos que se pueden obtener de las APIs de Google en general y a la red de metro de Singapur y Bangkok en particular, en la mayoría de los casos provienen de fuentes oficiales. Esto se puede observar cerca de la información de la rutas o las estaciones en el propio buscador de Google.

De esta forma, es posible acceder a las fuentes oficiales de las cuales se pretendía obtener información inicialmente. Adicionalmente, se simplifica el acceso a los datos a través de una sola fuente de información porque, aunque son varias APIs, todas tienen un formato

muy similar. Por último, al tratarse de APIs, se puede llegar a automatizar el proceso de extracción de datos a través de scripts que realicen dichas tareas.

5.1.3 POSIBLE ALTERNATIVA

Respecto a la solución alternativa introducida en apartados anteriores de utilizar directamente en tiempo real el servicio de Google Directions existen las siguientes desventajas:

- Dependencia de terceros: En prácticamente cualquier situación de este tipo es conveniente no tener que depender de otras páginas. Aunque la fuente sea una de las más viables que puedan existir tener que realizar una petición adicional implica un cierto riesgo por mínimo que sea.
- Falta de información: La información que se podría obtener de Google Directions sería insuficiente y se debería hacer una segunda petición a otra fuente de datos o tener parte de los datos almacenados en local. Por ejemplo, un atributo que no está disponible a través de esta API es la tarifa del trayecto.
- Situaciones imprevistas: En cualquier momento es posible que cambien la respuesta de las peticiones y sea necesario realizar varias modificaciones. Tener los nodos y las rutas almacenadas en la base de datos proporciona más control en ese aspecto.
- Rendimiento: A pesar de tener que realizar el cálculo de la ruta óptima, el rendimiento no tiene por qué ser mejor ya que depende de la conexión a internet.
- Coste: Una vez pasados el límite de 2500 peticiones diarias, cada petición supone un cierto coste para Baolau. Del modo alternativo, solamente se tendría esta petición durante la extracción de datos, que sucede cada menos tiempo.

Por estas desventajas se ha decidido descartar esta aproximación.

5.1.4 DECISIÓN FINAL

Tras conocer todas las alternativas que podrían proporcionarnos la información buscada es necesario decidir cuál de ellas se va a utilizar. La decisión no es nada compleja ya que la única fuente de datos accesible a través de una API es la proporcionada por Google.

Debido al gran número de nodos y rutas que forman la red urbana de estas dos ciudades, un requisito muy recomendable es que su recolección se pueda automatizar.

También, con vistas a un futuro no muy alejado, es conveniente que la mayor parte de la extracción de datos esté automatizado por el ahorro de recursos en implementaciones de redes urbanas de otras ciudades. Debido a que Google Maps opera prácticamente en todo el mundo, si no se encontrase una base de datos oficial en otras ciudades se podría usar el mismo procedimiento usado en este proyecto.

A continuación se muestra una tabla en la que se puede ver qué API se utilizará para extraer cada uno de los campos requeridos de los nodos y las rutas tanto para Singapur como para Bangkok.

Campo	**API**
Nombre de los nodos	Google Places API
Coordenadas los nodos	Google Geocoding API
Dirección los nodos	Google Directions API
Horario de los nodos	Google Directions API
Duración de las rutas	Google Directions API
Distancia de las rutas	Google Directions API
Frecuencias de las rutas	Google Directions API

Tabla 2. APIs utilizadas.

Como se puede comprobar toda la información de los nodos proviene de tres APIs mientras que parte de la información de las rutas proviene de solamente de la Google Directions API. El código del tipo de transporte y el código de la línea se extrae en muchos casos del propio nombre de los nodos. En el siguiente apartado se detalla el procedimiento que se ha seguido para obtener cada uno de los campos a través de las APIs mencionadas y su funcionamiento de peticiones y respuestas.

Las páginas web que se muestran a continuación son necesarias para completar toda la información ya que proporcionan información de las tarifas de cada tipo de transporte. Con las tarifas si es necesario realizar una distinción entre las ciudades y los tipos de transporte ya que los precios varían entre ellos y en el tipo de divisa.

Tarifas de	**Extraído de**
MRT y LRT Singapur	data.gov.sg
MRT Bangkok	www.transitbangkok.com/mrt.html
BTS Bangkok	www.bangkokbts.com/bts-fare.html
ARL Bangkok	www.bangkokairporttrain.com

Tabla 3. Otras páginas utilizadas para extraer tarifas.

Se han usado páginas locales, del gobierno y de terceros para lograr tener todas las tarifas. Si bien su acceso no es igual que en una base de datos se automatizará la extracción de tarifas lo máximo posible. En la mayoría de los casos será necesario extraer y dar un formato adecuado a los datos para poder manejarlos con facilidad y relacionarlos con cada ruta.

Como conclusión, las principales ventajas que encontramos con la anterior selección de fuentes de datos son varias. Por un lado, acceder a través de los servicios de Google nos proporciona información en muchos casos de fuentes oficiales que no sería posible obtener de otro modo. Las APIs de Google permite automatizar el proceso de extracción de datos a través de los scripts que se explicarán en la sección siguiente de este documento. Adicionalmente, solamente será necesario acceder a otras fuentes de datos para obtener las tarifas y el resto están cubiertos por Google.

Sin embargo existe una limitación que se deberá tener muy en cuenta durante el desarrollo y el proceso de extracción de datos. El uso de estas APIs están limitadas a un total de 2500 peticiones diarias con la licencia gratuita y siendo las siguientes peticiones de pago. Tras evaluar las opciones económicas se ha decidido trabajar dentro de este límite debido al

volumen de peticiones necesarias totales y la planificación temporal. En la sección de extracción de datos, se explica de qué forma esta limitación se ha tenido en cuenta en el desarrollo de los scripts.

5.2 EXTRACCIÓN DE DATOS

En esta segunda etapa del desarrollo del proyecto se desarrollaron los scripts encargados de consultar a la base de datos de Google para extraer la información necesaria. En primer lugar se comienza con unos ejemplos del uso de las APIs para entender el funcionamiento de las mismas y observar el formato y contenido de las peticiones y respuestas. Tras esta breve introducción, se explican cada uno de los scripts encargados de extraer todos los campos mencionados en el apartado anterior. Para ello se hará uso de pseudocódigo para no aumentar la complejidad de la sintaxis de manera innecesaria. Primero se tratarán los nodos seguidos de los campos de las rutas.

5.2.1 USO DE LAS GOOGLE APIS

En el apartado anterior se mencionaron las tres APIs que se han utilizado para extraer los datos. A continuación se muestra cómo es posible hacer consultas a ellas con una serie de ejemplos. Para más información acerca de la sintaxis utilizada y las opciones disponibles para realizar peticiones es conveniente leer la documentación de la página oficial de desarrolladores de Google de cada una de estas APIs.

5.2.1.1 Google Places API

Las tres principales funciones de esta API son la detección de la ubicación actual del dispositivo móvil que hace la consulta, la búsqueda de negocios y puntos de interés y la adición de la función de autocompletado a aplicaciones. Lógicamente, ni la primera ni la tercera son de ayuda para extraer información de las estaciones de la red de metro. La principal razón de uso de esta API es la de obtener los nombres de las estaciones en los seis idiomas en lo que opera Baolau.

La sintaxis que siguen las peticiones de esta API es la siguiente:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/outputFormat?
parameters
```

En dónde *output* es el formato de salida y *parameters* son los parámetros que definen la petición. El formato puede ser elegido entre JSON y XML. Además del formato, los siguientes tres parámetros son obligatorios:

- **Key**: Se trata de un clave que Google proporciona a sus desarrolladores correspondiente con su licencia. Entre sus funciones, se encuentra la identificación del usuario y la de llevar la cuenta de peticiones que ha hecho durante ese día.

- **Location**: Las coordenadas (latitud y longitud) alrededor de las cuales se debe obtener información de los distintos lugares que es capaz de encontrar.

- **Radius**: Es le último parámetro obligatorio y determina el radio de búsqueda de lugares desde el punto definido en *location*.

Siendo los parámetros anteriores totalmente obligatorios para poder realizar la petición también existe gran cantidad de ellos que son opcionales. Para no alargar esta introducción de manera innecesaria, a continuación solamente se explican aquellos que se han utilizado en el desarrollo:

- **Type**: Este parámetro también se utilizará en las otras APIs y ha sido fundamental durante todo el proceso de extracción de datos. Este parámetro restringe los lugares o sitios en la búsqueda según un tipo o lista de tipos admitidos. Algunos de los valores que puede tomar este parámetro son, por ejemplo, *store*, *hospital* y el más importante para nuestra aplicación, *transit_station*.

- **Lenguage**: Como su propio nombre indica, este parámetro determina el lenguaje en el que estará escrita la respuesta. Este parámetro es clave para la función que buscamos con esta API que es obtener los nombres de las estaciones en distintos idiomas. La lista de los códigos correspondientes a los idiomas admitidos se puede encontrar en la documentación.

Para finalizar con esta API, se muestra un ejemplo de petición y respuesta similar al realizado posteriormente en la extracción de datos. La petición realizada es la siguiente:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json?location
=1.2825422,103.7818105&radius=50&type=transit_station&key=AIzaSfikfky
&language=en
```

Los parámetros introducidos quiere decir que se está buscando un sitio de tipo *transit_station*, en el área formada por la circunferencia de 50 metros de radio y centro las coordenadas 1.2825 de latitud y 103.7818 de longitud. Para este ejemplo estás coordenadas se han buscado de forma visual en Google Maps. El último parámetro determina que la respuesta será entregada en inglés. En el momento en el que se realizó está petición se obtuvo la siguiente respuesta:


```
{
   "html_attributions" : [],
   "results" : [
      {
         "geometry" : {
            "location" : {
               "lat" : 1.2827808,
               "lng" : 103.7821761
            }
         },
         "icon" : "https://maps.gstatic.com/mapfiles/place_api/icons/bus-71.png",
         "id" : "0facbbf66fee02000f02b451ef7ee5360e04399a",
         "name" : "Haw Par Villa Stn",
         "place_id" : "ChIJnRlz3K4b2jER5KQJaIvC37I",
         "rating" : 5,
         "scope" : "GOOGLE",
         "types" : [
            "bus_station",
            "transit_station",
            "point_of_interest",
            "establishment"
         ],
         "vicinity" : "Singapore"
      }
   ],
```

```
   "status" : "OK"
}
```

La respuesta ha sido simplificada ya que era muy extensa pero es posible ver la información más relevante y la estructura y formato que presenta. Nótese que el único resultado que se ha mostrado en esta representación se encuentra entre corchetes, lo cual quiere decir que este tipo de peticiones devuelve más de un sitio. El conjunto de sitios que devuelve son aquellos que se encuentran en el área determinada y que cumplen con los requisitos que se han puesto con los parámetros opcionales. Para facilitar la lectura de los JSON, a lo largo del documento se mostrarán cómo figuras gracias a un intérprete de este formato. La anterior representación queda de la siguiente forma:

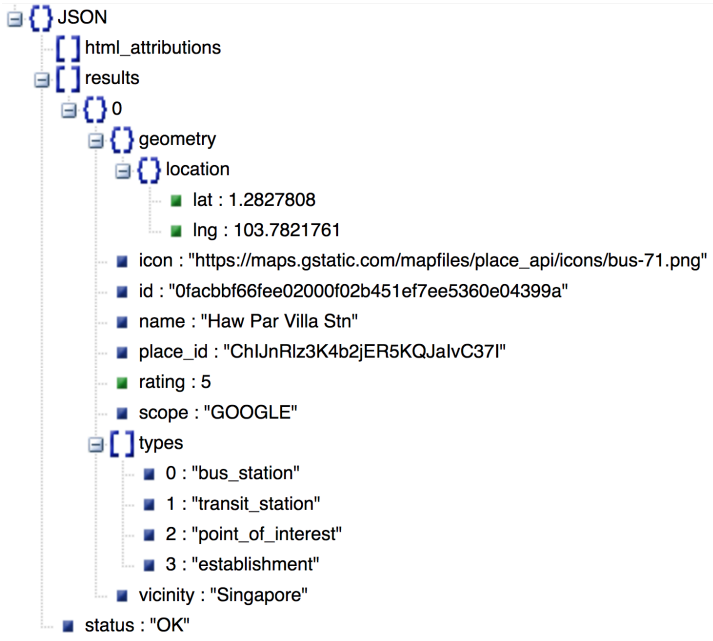

Figura 2. Respuesta de Google Place API. Ejemplo JSON.

Entre toda la información que nos proporciona esta API, se encuentra la posición exacta de dónde se ha encontrado la estación de metro, un identificador único del sitio, el nombre en el idioma especificado, la valoración que le han dado los usuarios de Google Maps y una lista de tipos que se corresponden con la estación buscada. Gracias a esta API y con

peticiones similares a la mostrada en el ejemplo anterior, será posible obtener el nombre de todas las estaciones de las redes urbanas en distintos idiomas.

5.2.1.2 Google Geocoding API

Mientras que la API de Google Places sirve para obtener información a partir de unas determinadas coordenadas, la Google Geocoding API se encarga del proceso inverso. A partir de la dirección o palabras clave es capaz de obtener las coordenadas del sitio al que se refiere. Se hace uso de esta API para obtener las coordenadas de los nodos. Su funcionamiento, basado en peticiones y respuestas, es similar a la anterior.

Para realizar una petición la sintaxis que debe cumplir esta es la siguiente:

```
https://maps.googleapis.com/maps/api/geocode/outputFormat?parameters
```

El significado de las opciones que se deben introducir es el mismo pero los parámetros son distintos. En este caso los parámetros obligatorios son la clave y la dirección. Este último puede contener palabras clave que no formen parte de la dirección. Tal y como se explica en la implementación, el único parámetro opcional que se utiliza es *region*. En este se puede especificar el país dónde va a estar acotada la búsqueda y se encuentra determinada con un identificador de dos letras. A continuación se muestra la petición de ejemplo de esta API.

```
https://maps.google.com/maps/api/geocode/json?region=sg&address=Haw%2
0Par%20Villa%20MRT%20Station&key=AIQp83v
```

De tal forma que está petición pretende obtener información a partir de las palabras clave “Haw Par Villa MRT Station” en la región de Singapur (con el identificador sg). En el ejemplo se ve claramente que el proceso de geocodificación no solamente tiene que partir de una dirección sino de las palabras clave que uno quiera. La respuesta a esta petición ha sido la siguiente:

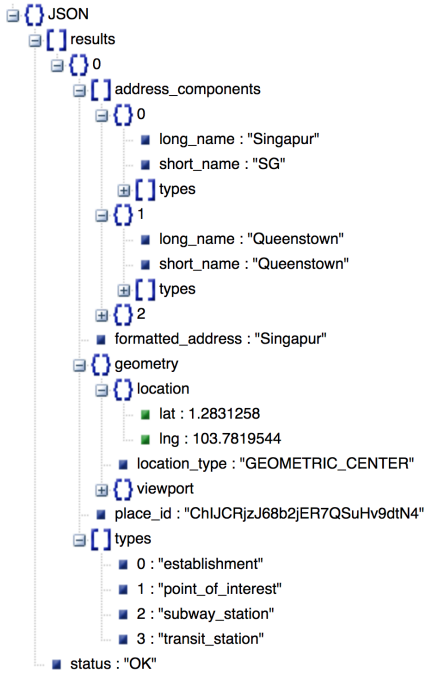

Figura 3. Respuesta de Google Geocoding API. Ejemplo JSON.

En este caso la respuesta de esta petición es más breve ya que solamente se ha obtenido un resultado (en el anterior ejemplo se suprimieron para no complicar la vista del resultado). Entre la información que se puede encontrar está la dirección dividida en componentes. En la imagen se puede por ejemplo la ciudad, Singapur, y el barrio, Queenstown. Más abajo tenemos la información buscada, las coordenadas, divididos en latitud y longitud. Para finalizar, el identificador del sitio y la lista de los tipos que puede tener el resultado.

5.2.1.3 Google Directions API

En este último subapartado se explica la API de la cual se va a hacer uso prácticamente durante todo el proceso de extracción de datos. Como se ha visto en el apartado anterior, gracias a esta API se puede obtener información de las rutas como la duración, la distancia, el horario de funcionamiento y la frecuencia de los trenes a distintas horas del día. La posibilidad que ofrece esta API es totalmente esencial para la aplicación y sin ella la implementación no pudiera haber sido posible, o al menos, con la extracción de datos automatizada.

La Google Directions API es la encargada de proporcionar indicaciones entre dos o más ubicaciones. Adicionalmente, también ofrece la posibilidad de elegir distintos medios de transporte y la hora de salida. La posibilidad de indicar esto con ayuda de parámetros ha sido crucial para obtener la todos los datos. Por último, no sólo permite saber el camino hasta que se va a seguir con el transporte determinado sino también la distancia que recorre y la duración de toda la ruta y sus segmentos.

La sintaxis de la petición, cómo en todos los casos, es parecida a las ya explicadas.

```
https://maps.googleapis.com/maps/api/directions/outputFormat?parameters
```

Respecto al formato de salida, se soportan los formatos XML y JSON. Esta vez, los parámetros obligatorios son:

- **Key**: La clave correspondiente a la licencia de uso de la API.
- **Origin**: El origen de la ruta descrita en coordenadas (latitud y longitud), en palabras clave (de una forma similar a las mostradas en la API anterior) y el identificación del sitio (recomendable si se ha obtenido anteriormente de alguna forma).
- **Destination**: El destino de la ruta escrita de cualquiera de las tres formas mencionadas en el parámetro anterior. Un dato interesante es que si se usan palabras clave para definir el destino o el origen estos no tienen por qué ser los mismos que los que proporciona la API de geocodificación.

Las peticiones de esta API cuentan con gran cantidad de parámetros opcionales pero muchos de ellos son a su vez dependientes del método de transporte. Los parámetros de este tipo que se usarán a lo largo de los scripts que componen la extracción de datos son los siguientes:

- **Mode**: Especifica el medio de transporte que se debe usar para calcular las indicaciones del resultado. El valor por defecto de este parámetro es *driving* por lo

que si se deja sin especificar en la petición se obtendría la ruta en coche. La lista de los distintos modos se encuentra descrita en la documentación de esta API. Para este proyecto siempre se utilizará este parámetro con el modo *transit* que corresponde al servicio público de transporte.

- **Arrival_time**: Determina la hora deseada de llegada al destino. Este parámetro es un valor entero correspondiente al número de segundos desde la medianoche, UTC del 1 de Enero de 1970.

- **Departure_time**: Especifica la hora de salida desde el origen. Al introducir este parámetro no es posible usar también el anterior y viceversa. Es decir, el uso de ambos en una petición no está permitido.

- **Transit_mode**: Mediante este parámetro es posible seleccionar el medio o medios de transporte público preferente de la ruta. Entre los valores disponibles, existen varios que se refieren a distintos tipos de trenes como el tren (*train*), el tren subterráneo o metro (*subway*) o el tranvía o tren ligero (*tram*). Debido a las peticiones concretas y la estructura de la red urbana de transporte de ambas ciudades se escogerá la opción *rail* que aúna los tres tipos de trenes anteriores. De esta forma este parámetro no debe ir cambiando en la petición según el tipo de tren que se trate.

- **Transit_routing_preference**: Este parámetros especifica preferencias en las indicaciones de la ruta resultado cuando se ha establecido el parámetro *mode* en *transit*. De esta forma, en lugar de que la API devuelva la ruta que ella considera mejor, se restringen las opciones y te proporciona una que encaje mejor con estas preferencias. Los valores que pueden tomar este parámetro son *less_walking* y *fewer_transfers*. El primero otorga preferencia a las rutas que tienen las distancias andando más cortas mientras que el otro prioriza las rutas con menos número de trasbordos entre líneas de metro o autobuses.

Si no se hace uso de algunos parámetros no es necesario proveer la clave en la petición y por tanto, no existe la limitación de las 2500 peticiones diarias. Como ya se ha dicho, haciendo un buen uso de estos parámetros opcionales, es posible sacar los datos necesarios para seguir con la siguiente etapa del proyecto. A continuación se muestra un ejemplo de una petición que hace uso de todos estos parámetros.

```
https://maps.googleapis.com/maps/api/directions/json?&origin=1.2825422,103.7818105&destination=1.3149545,103.7652989&mode=transit&transit_mode=rail&key=AIzaSyDdeUTvWGgvhw4DDblEssf8E&departure_time=1526467248&transit_routing_preference=less_walking
```

Esta petición hace una consulta muy concreta a la base de datos ya que usa muchos de los parámetros mencionados. Entre ellos, se encuentran los tres obligatorios que son el origen (las coordenadas correspondientes a la estación de Haw Par Villa), el destino (las coordenadas de la estación de Clementi) y la clave del usuario. Además, se especifica que la ruta se debe hacer en tranvía, tren o metro, saliendo el 16 de Mayo de 2018 a las 17:40 y haciendo preferencia a las rutas en las que se anda menos. Tras lanzar la petición se recibe la siguiente respuesta. Para mostrar todo su contenido principal, se muestra en dos figuras.

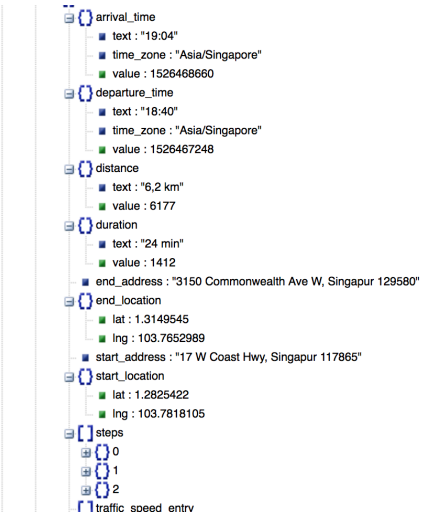

Figura 4. Respuesta de Google Directions API. Ejemplo JSON.

En esta figura se han desplegado la información global de la ruta resultado. Además de la información que se especificó al hacer la petición cómo el origen y el destino o la hora de

salida, aparecen otros datos de gran valor cómo la hora de llegada al destino final y la duración y distancia total del trayecto. También se puede ver en el atributo de *steps* que la ruta cuenta con tres indicaciones.

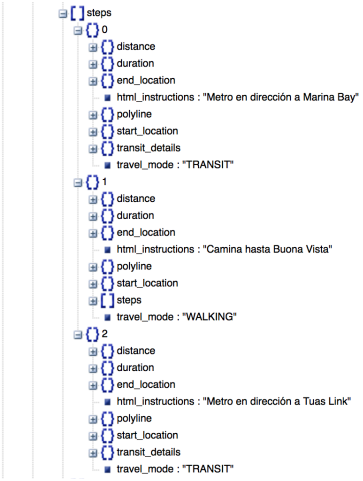

Figura 5. Respuesta de Google Geocoding API. Ejemplo JSON.

Al desplegar cada una de las indicaciones se puede observar que estas a su vez cuentan con información muy detallada acerca de la ruta. En primer lugar se puede observar que cada segmento contiene un método de transporte determinado. En este caso se empieza desde una estación por lo que el método de transporte es público y se identifica con *transit*, más tarde se anda y se vuelve a coger otro metro hasta el destino final. Por tanto, este es un claro ejemplo de una ruta que conlleva un trasbordo.

Como si se tratasen de rutas independientes, la información que se obtiene define perfectamente cada segmento con su origen, destino, duración y distancia. De hecho, el segundo segmento a su vez contiene 2 pasos para más detalle. Además encontramos otro atributo llamado *polyline* que son un conjunto de coordenadas para poder definir el trayecto en un mapa. Por último, también está disponible unas instrucciones en un determinado idioma que es posible definir con el parámetro lenguaje para ayudar al usuario final a través de la aplicación de Google Maps.

Aunque los últimos atributos mencionados no se van a recolectar ya que no son de utilidad para cumplir los objetivos, era necesario mostrar la complejidad de las respuestas de esta última API. De esta forma, es posible observar que está última API puede proporcionar más información que las anteriores, y que, con un uso adecuado, se podrán obtener el resto de datos que las otras no consiguen aportar.

5.2.2 NOMBRES Y LOCALIZACIÓN

En este y los siguientes apartados de la sección 5.2 se explica todo el proceso de extracción de datos. Para ello, se han programado varios scripts en lenguaje Python con el que se han ido obteniendo progresivamente todos los campos. La extracción de datos comienza con los nombres y las localizaciones de las estaciones y en este apartado se explica cómo se han logrado obtener mediante la Google Geocoding API.

Antes de explicar con detalle el script de este apartado, es necesario aclarar algunos conceptos que afectan a todos ellos. En primer lugar, la licencia gratuita de estas APIs solamente permiten realizar 2500 peticiones diarias. Este límite supuso programar los scripts con las menos peticiones posibles aunque en pocos casos se superó ese límite. Respecto a la clave de la licencia que se ha mostrado en la sintaxis de cada petición, esta se encuentra omitida en el script que se muestra a continuación y los siguientes.

Respecto a la explicación y representación de ellos, se ha optado por describirlos con pseudocódigo extraído a partir del código general. Se han omitido las partes del código que aunque aporten cierta funcionalidad no ayudan a comprender el objetivo del script. Del mismo modo, las peticiones se muestran de manera simplificada pero usando el nombre de los parámetros reales y sus valores. Para ahorrar líneas de código, se ha dado por hecho que la variable *i* obtiene un valor incremental en cada iteración dónde se encuentre.

La mayoría de los nombres de las variables se han mantenido respecto al código original y por tanto, se encuentran en inglés. En algunos casos se ha tendido a simplificar una situación con una simple palabra para ahorrar espacio y ser más claro. Por ejemplo, para indicar que la ejecución muestre un error y su respectivo mensaje en el pseudocódigo se ha

utilizado la palabra *error* sin añadir nada más. Respecto al código de colores se ha destacado en color azul el texto y alguna palabra clave y en color naranja los bucles y condiciones. Por último, se han añadido comentarios aclarativos en las partes significativas del código para facilitar su compresión.

El pseudocódigo de este apartado se encarga de obtener las localizaciones de las estaciones a partir del nombre de ellas. Para obtener dichos nombres en el caso de Singapur se ha accedido a su descarga desde una base de datos oficial que se puede encontrar en data.gov.sg. En este fichero se obtiene el nombre de todas las estaciones divididas por líneas y ordenadas de manera geográfica. Junto a cada nombre se encuentra un identificador de la línea y el número de la estación respecto a esta línea. Por ejemplo la primera estación de la línea circular con el nombre Dhoby Ghaut aparece en una fila de este fichero como "CC1 Dhoby Ghaut". Este identificador será de utilidad más adelante para extraer el código de cada línea.

Respecto a todas las estaciones de la ciudad de Bangkok, se ha usado la propia aplicación de Google Maps, aunque se podrían haber extraído de otras páginas. Para ello se simula una búsqueda entre la primera y última estación de cada línea seleccionando el transporte público. Entonces, en la descripción de la ruta se muestra una lista de todas las estaciones de la línea ordenadas. La mayor ventaja de este procedimiento es su facilidad para extraer cada línea y la actualización de estas estaciones (ya que google siempre tiende a tener actualizados sus mapas).

A continuación se muestra el pseudocódigo del primer *script*, encargado de extraer las coordenadas de cada estación a partir de sus nombres

```
// Variables
mods = [" MRT", " station"]
geo_req = "http://maps.google.com/maps/api/geocode/json?
          region = sg
          &address = "
inv_req = "https://maps.google.com/maps/api/geocode/json?
          result_type = transit_station
```

```
          &latlng = "

for name in stations_names: // Para cada nodo
    while True:
        trie = geo_req + name
        first_resp = http.get(trie) // Primer intento - solo con nombre
        if resp['status']=="OK":
            req = inv_req + resp['lat'],resp['lng'] // Geocodificación inversa
            resp = http.get(last_req)
            if resp['status']=="OK":
                if 'transit_station' in resp['types']: // Si es de tipo estación de tren
                    print(num, name, resp['lat'], resp['lng']) // Muestra el resultado
                    break
            name = name + mods[i] // Se añade modificacion
            if more_than_2_tries: print(error) break // Error si no encuentra de tipo estación
        elif json_resp['status']=="ZERO_RESULTS":
            name = name + mods[i]
            if more_than_2_tries: print(error) break // Error si no hay resultados
print("Finished!")
```

Psudocódigo 1. Extracción de las localizaciones de las estaciones.

En el apartado de variables se han definido las tres siguientes. La primera llamada *mods* cuenta con varias palabras clave que se deben añadir al nombre de la estación en caso de que la identificación de la estación no se produzca correctamente. Para extraer las coordenadas del nodo se realiza una geocodificación con la petición mostrada en la segunda variable. La tercera variable se trata de una petición para invertir este procedimiento.

Para cada nodo de la línea se realiza una petición de geocodificación con su nombre para saber sus coordenadas. Para asegurarse de que el resultado es el correcto, se realiza unas segunda petición para obtener dicha estación de nuevo a partir de las coordenadas obtenidas. Para comprobar que efectivamente es correcto se observa si el tipo de sitio se trata de una estación de tránsito (*transit_station*). Si todas estas condiciones se cumplen se imprimen por pantalla un número creciente junto con el nombre de la estación y sus coordenadas (latitud y longitud).

En caso de que no se encontrará en primer lugar las coordenadas de la estación o que en la segunda petición no se obtuviera un sitio de ese tipo, se vuelve a repetir el proceso añadiendo una palabra clave del vector de modificaciones *mods*. Si se excediera estas modificaciones se lanza un mensaje de error en la ejecución. De esta forma es posible redefinir las modificaciones para que la ejecución se produzca correctamente. Tras este proceso, se obtuvo un fichero con la información requerida.

5.2.3 RUTAS

Tras la localización de las estaciones, el script mostrado en este apartado es el encargado de obtener las rutas que puedan existir entre ellos. A la hora de almacenar en la base de datos la ruta de una sola línea deben existir rutas independientes para todas las combinaciones posibles de las rutas de la línea. Por ejemplo, si una línea tiene 3 paradas, se deberían obtener las rutas desde cada parada hasta las otras dos, es decir, 1-2, 1-3, 2-1, 2-3, 3-1, 3-2. Una posible alternativa en la implementación pudiese haber sido tener solamente conexión con la estaciones adyacentes. Sin embargo, se ha optado por la primera aproximación para parecerse en ese aspecto con las rutas ya introducidas en la base de datos original de Baolau.

Como se ha visto, la forma en la que se va a obtener estas rutas es mediante la API de Directions de Google. Gracias a ella se obtiene el camino que el algoritmo de búsqueda de Google considera óptimo teniendo en cuenta las condiciones descritas en los parámetros de la petición. Por tanto, debería funcionar correctamente si el *script* es capaz de hacer ejecutar tantas peticiones como combinaciones haya. Sin embargo encontramos un problema en esta solución.

Dependiendo de la forma que tenga la línea y de la distribución de otros métodos de transporte, es posible que el camino óptimo que google muestre como resultado no sea a través de la línea que se pretende recorrer. La siguiente figura ilustra dicha problemática.

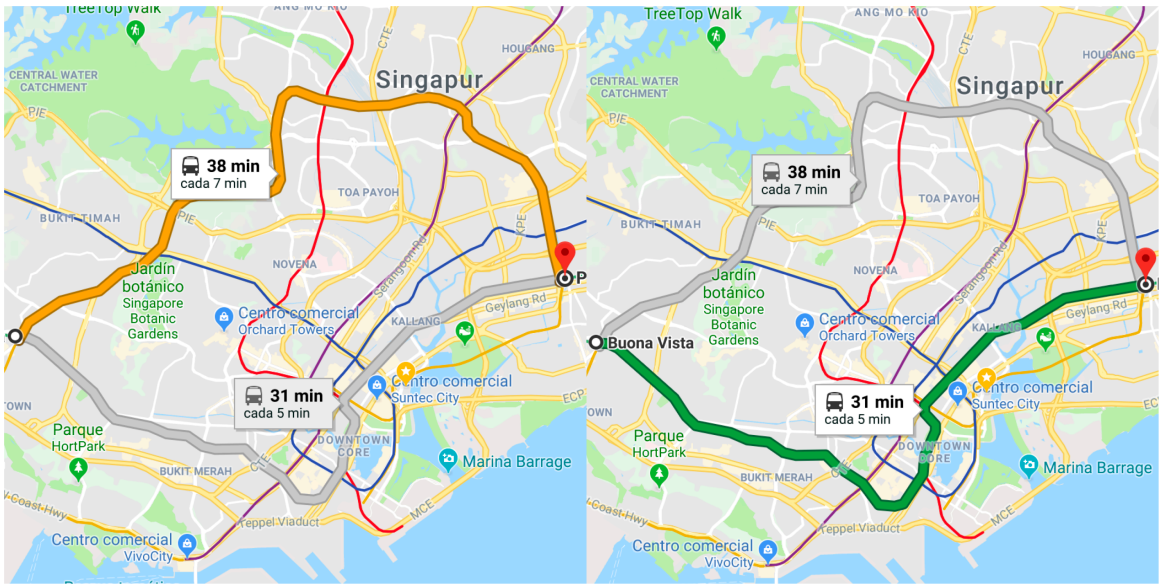

Figura 6. Problema en rutas largas. Google Maps Singapur.

Supongamos que se está queriendo obtener la ruta que recorre la línea amarilla entre los dos puntos que están marcados en la imagen. El problema surge cuando, al ejecutar la petición, el camino más corto es el recorrido de la línea verde ya que es más directa que ir por la amarilla. Esta situación, en una red urbana como la de Singapur ocurre constantemente y además, a través de esta API hay que tener en cuenta también las interconexiones que se realicen en otros medios de transporte como el autobús.

Debido a esta situación y dado que en Singapur en muchos casos no es posible recorrer más de un par de paradas sin que sufrir este problema se van a obtener solamente las rutas con las estaciones adyacentes. Aún sin ser la situación ideal, se ha procedido inicialmente de esta forma con dicha ciudad y más tarde se estudia la posibilidad de combinar estas rutas individuales.

Debido a la forma que tienen las líneas de transporte ferroviario de Bangkok y al resto de transporte públicos en ningún momento se ha detectado que ocurra este problema. Por tanto, cuando se trate de Bangkok, se ejecuta una petición por cada combinación dentro de una línea. Es necesario aclarar, que las combinaciones de paradas no tienen en cuenta el

resto de líneas ya que el encargado de realizar esta tarea en Baolau será el algoritmo de cálculo de rutas combinadas.

A continuación se muestra el pseudocódigo del *script* encargado de esta tarea.

```
// Variables
input_file = path/nodos
json_data = text2json(input_file)
comb = permutations(len(json_data)) // Generación de todas las combinaciones
req = "https://maps.googleapis.com/maps/api/directions/json?
        &origin =
        &destination =
        &mode = transit
        &transit_mode = rail"

for tuple in comb: // Para cada estación
    orig_location = json_data[tuple[0]]
    dest_location = json_data[tuple[1]]
    req = req + orig_location + dest_location // Peticion con origen y destino
    resp = http.get(req)
    // Se incluye la información anterior en la ruta respuesta
    resp['orig_num'] = orig['number']
    resp['orig_name'] = orig['station_name']
    resp['orig_lat'] = orig_location
    resp['dest_num'] = dest['number']
    resp['dest_name'] = dest['station_name']
    resp['dest_location'] = dest_location
    save resp in output_file // Guardar en fichero de salida
```

Pseudocódigo 2. Extracción de Rutas.

Las primeras líneas de código serán comunes en muchos de los *scripts* y consiste en transformar la información del fichero de entrada a formato JSON para que se pueda acceder con más facilidad posteriormente. La tercera línea muestra la operación matemática necesaria para generar tuplas que contengan todas las combinaciones. Aunque se haya omitido, en el caso de ejecutar el *script* para alguna línea de Singapur esta línea genera solamente las combinaciones secuenciales. En la definición de la petición (*req*) se puede observar cómo se va a hacer uso de la Google Directions API.

El pseudocódigo muestra un funcionamiento más sencillo que el anterior *script* ya que, para cada nodo, ejecuta la petición que interconecta origen y destino de cada tupla de *comb* y guarda la respuesta en la variable *resp*. Además, al resultado se añade la información de ambos nodos y posteriormente se almacena en el fichero de salida. Más tarde se soluciona con otro script el problema que ha quedado pendiente en las rutas de Singapur.

5.2.4 LOCALIZACIONES EXACTAS

Antes de continuar con el resto de campos, la localización que se ha obtenido usando la geocodificación y geocodificación inversa en el primer script podría generar problemas en un futuro. Tras investigar la situación, Google almacena más de una sola pareja de coordenadas en este tipo de sitios. Aunque pueda parecer que la localización que se requiere es la que muestra Google cuando se busca por el nombre de la estación, en este caso no es así. Para que la Google Directions API funcione de la manera esperada y sea de utilidad las coordenadas de cada estación debe ser aquel punto desde dónde parta o lleguen los trenes a dicha estación. Al no ser ese punto el que se obtuvo anteriormente, las indicaciones del resultado siempre suelen mostrar varios segmentos en el que el usuario debe andar y no solamente el trayecto en tren. La siguiente figura muestra un ejemplo gráfico de la problemática.

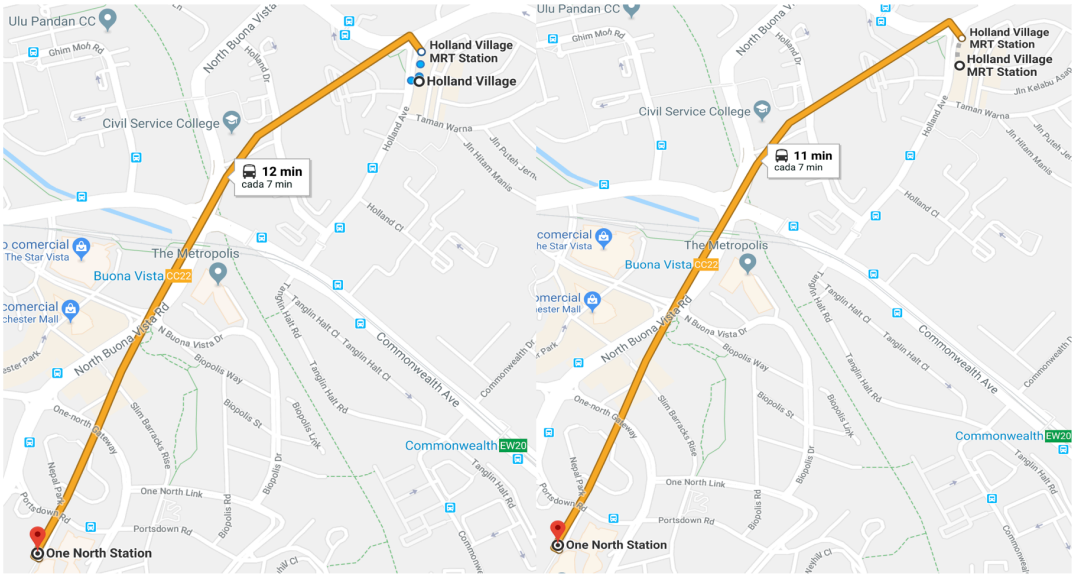

Figura 7. Problema de las localizaciones exactas. Google Maps Singapur.

La figura de la izquierda muestra la búsqueda realizada desde las coordenadas que Google tiene de ambas estaciones y en la que se observa que el usuario debe andar unos metros hasta coger el tren. Esto no quiere decir que haya un defecto en la base de datos sino que hacen distinción entre el punto del tren y la propia estación (lo cual tiene mucho sentido si la estación es grande y es necesario andar dentro distancias considerables). En la figura de la derecha se observa la ruta equivalente que se está buscando y en la que al no tener que andar, la duración total de ella se ha reducido un minuto.

Desde este momento, estas coordenadas serán referidas a lo largo de este documento cómo coordenadas exactas. En el ejemplo de la figura, la primera petición daría lugar a una respuesta en la que existen más de un elemento en el vector de *steps* de la respuesta (véase el ejemplo la figura 4). Por tanto, una forma de averiguar si las coordenadas iniciales son efectivamente las coordenadas correctas es detectando el número de estos pasos. En el caso de que solamente haya un paso y el modo de viaje de este sea *transit* significará que las coordenadas origen y destino de la petición eran coordenadas exactas. El siguiente *script* muestra este funcionamiento.

```
// Variables
input_file = path/routes

for row in input_file: // Para cada ruta
    transfer_count = 0
    route = json.loads(row)
    for step in route['steps']: // Para cada step
        if(step['travel_mode']) == "TRANSIT": // Si se coge el tren
            entry = route['orig_num']+ route['orig_name'] + step['start_location']['location']
            if entry not in output: // Se almacena el origen de la ruta en una variable
                output.append(entry)
            transfer_count + 1
        else:
            print(error + Try with step['locations]) // Posible alternativa
    if transfer_count > 1: // Si la ruta tiene más de un tren
        print(error + more than 1 transfer]
    if transfer_count < 1: // Si no se ha cogido ningún tren
```

```
        print(error + no transfers)


save output in output_file // Guardar en fichero de salida
```

Pseudocódigo 3. Comprobación de localizaciones.

En primer lugar se cuenta el número de pasos o *steps* de cada ruta obtenida en el *script* anterior. Si alguna de las rutas tiene más o menos de un paso y este no tiene el atributo *travel_mode* igual a *transit*, el *script* lanzará un error con la ruta que se debe revisar. Los casos de error están diferenciados gracias a la variable contadora de trasbordos llamada *transfer_count*. Tras la ejecución de este fichero las pocas coordenadas que no eran exactas se corrigen manualmente en el fichero de salida del primer *script* y se repite este proceso de análisis. El objetivo de esta etapa es ejecutar este *script* sin que lance ningún error.

5.2.5 RUTAS DE SINGAPUR

En este apartado se retoma la problemática presentada en el apartado 5.2.3. Al descartar la opción de extraer directa todas las combinaciones de las líneas de Singapur se han de obtener de otra manera. La primera opción es generar todas las combinaciones juntando cada uno de los segmentos y restando un posible tiempo de espera que podría haber sido añadido. A continuación se estudia la viabilidad de esta primera aproximación.

Para ello se ha extraído partes de las líneas en las que, por su localización geográfica y el del resto de transporte públicos, es posible obtener todas las combinaciones tal y cómo se hizo con las rutas de Bangkok. La idea, es poder comparar los conjuntos extraídos de esta forma con los formados a partir de los segmentos individuales. De esta forma, será más fácil distinguir el tiempo de espera que Google introduce normalmente y restárselo más adelante. El *script* encargado de esta comparación se muestra a continuación.

```
// Variables
waiting_time = 0 // Tiempo de espera a añadir entre paradas
input_file_comb = 'path/routes_comb'
input_file_sec = 'path/routes_secuencial'
```

```
with open(input_file_sec) as f: // Fichero de rutas secuenciales
    for row in f: // Para cada ruta
        json_object = json.loads(row)
        for step in json_object['steps']: // Para cada step
            // Se añade el tiempo de espera y se almacena
            distance = distance + json_object['step']['distance']
            duration = duration + json_object['step']['duration']
            if not_first_train: duration = duration + waiting_time
        results_sec.append(duration)

with open(input_file_com) as f: // Fichero de rutas combinacionales
    for row in f: // Para cada ruta
        // Se almacena con el mismo formato anterior para su comparación
        json_object = json.loads(row)
        results_comb.append(json_object['step']['duration'])

// Resultados - Comparación de secuencial contra combinacional
for index in results_sec // Recorremos todas las rutas
    print(Absolute results = results_sec - results_comb) // Absolutos en minutos
    print(Relative results = results_sec/results_comb) // Relativos
```

Pseudocódigo 4. Comparación entre rutas.

El *script* parte de los dos ficheros mencionados, por un lado las rutas de todas las combinaciones de un tramo de una línea y en otro fichero las rutas individuales entre estaciones adyacentes de ese mismo tramo de la misma línea. Todo el proceso se repitió con varias líneas para tener más muestras. Más tarde se explica el porqué de la variable *waiting_time*.

En primer lugar, con el fichero de rutas secuenciales se juntan los segmentos para crear unas rutas comparables. Para ello se agregan las duraciones que aparecen en único *step* que tiene cada ruta y no la duración total de la ruta que está fuera de ese campo *step*. Esto se ha hecho así para evitar los tiempo de espera que google introduce en el resultado total de la búsqueda. Aunque no se muestra en el pseudocódigo, también se ha confirmaba que se estaban sumando los segmentos correctamente gracias a la distancia de ellos. El resultado de esta primera parte se guarda en la variable *results_sec*.

Para facilitar su entendimiento, a continuación se muestran algunos valores extraídos de su ejecución.

```
1-2,889,60
1-3,3053,240
1-4,7437,480
…
```

Lo cual quiere decir que de la primera a la segunda estación, la distancia es de 889 metros y se duración es de 60 segundos. Y así sucesivamente según se van sumando segmentos. En la segunda parte del código se extrae esta información para las combinaciones originales y también del propio *step* para no tener en cuenta el tiempo de espera. La duración de los segmentos se guardan en la variable *results_comb*. Por último, la ejecución muestra la comparación entre ambos resultados de forma absoluta y relativa comparando las duraciones de los segmentos.

Para comprobar el comportamiento de esta comparación en ambas ciudades, aun no siendo necesario se llevó a cabo también para las líneas de Bangkok. A continuación se muestra los resultados para esta ciudad.

```
// Línea ARL
Absoluto (segundos): [0, -5, -10, -15, -20, -25, -30]
Relativo: ['0.00', '-0.02', '-0.02', '-0.02', '-0.02', '-0.02', '-0.02']

// Línea MRT Azul
Absoluto (segundos): [0, -25, -50, -75, -100, -125, -150, -175, -200, -225, -250, -275, -300, -
325, -350, -375, -400, -400]
Relativo: ['0.00', '-0.11', '-0.14', '-0.16', '-0.17', '-0.17', '-0.18', '-0.18', '-0.18', '-
0.18', '-0.19', '-0.19', '-0.19', '-0.19', '-0.19', '-0.19', '-0.19', '-0.18']
```

Los resultados muestran que cuando se han extraído la ruta completa (no habiendo juntado los segmentos) la API le añade unos segundos entre parada y parada. Probablemente esta pausa pueda ser debida al tiempo que están las puertas abiertas del tren esperando que entren los pasajeros y como se puede ver, depende del tipo de transporte. En el caso del

ARL el tiempo de espera es de 5 segundos entre paradas y respecto a la duración total no le afecta mucho. Sin embargo para la línea de MRT Azul son 25 segundos y respecto al total pasadas varias paradas la diferencia si es muy notable.

Por tanto, en el caso de Bangkok, si debido a la posición geográfica de las líneas se hubiera tenido que extraer las rutas como segmentos se tendría que tener en cuenta estos tiempos y añadirlos posteriormente. Gracias a la variable *waiting_time* de este script se puede ver el efecto que tiene añadir un determinado número de segundos. Por ejemplo si se pusieran 25 segundos (antes estaba en cero) el resultado para la línea MRT Azul sería:

```
// Línea MRT Azul
Absoluto (segundos): [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25]
Relativo: ['0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00',
'0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.01']
```

Lo cual arreglaría el problema. En el caso de las líneas probadas en Singapur, con el tiempo de espera ajustado a cero segundos, todos los vectores resultados tenían el valor cero tanto en valor absoluto cómo relativo. Por tanto quiere decir que, para el tipo de transporte de Singapur, no es necesario añadir ningún segundo entre segmentos para lograr el mismo resultado que el ideal. Esta demostración es la prueba por la que, en la posterior generación de las rutas completas de Singapur, no se añade ningún segundo en cada parada de una ruta.

5.2.6 FORMATEO DE FICHEROS

El siguiente paso en la extracción de datos es la limpieza de los JSON respuesta que ha devuelto las peticiones a la API previas. Se aprovecha este apartado para mostrar también la generación de los nodos, ya que hasta ahora sólo se tenía el nombre y la localización en un fichero de texto plano sin formato JSON. Aunque estas tareas se llevaron a cabo con dos scripts se van a omitir debido a su simplicidad. En su defecto a continuación se muestra un ejemplo del resultado de una ruta y un nodo en formato JSON.

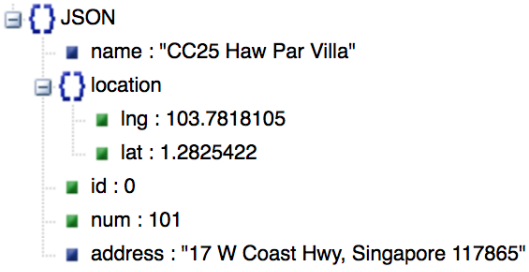

Figura 8. Nodo en formato JSON.

En la figura muestra el ejemplo de la información que contendría la estación número 25 de la línea circular de Singapur CC. Hasta el momento los datos que contiene son el nombre junto con el código identificador de la línea, seguido de la localización cómo coordenadas. Tanto el campo identificador id como el del número (número de la estación en el fichero original de nombres) se almacenaron por si fuese necesario en un futuro y por último se añadió la dirección de la estación.

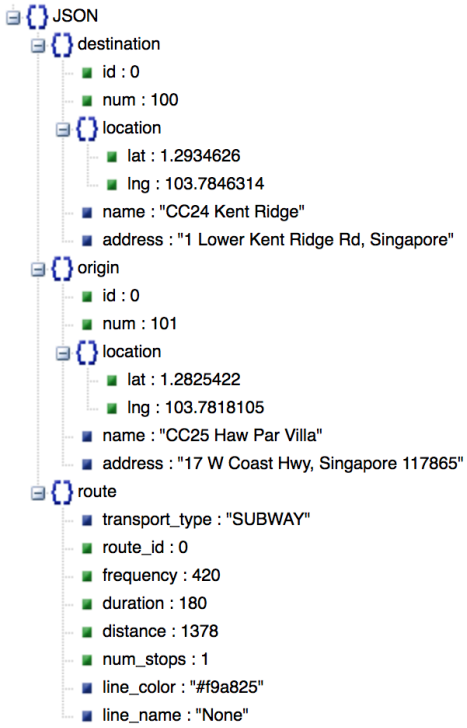

Figura 9. Ruta en formato JSON.

Respecto a la ruta, se encuentra la misma información para el nodo origen y destino. Más tarde cuando los nodos tengan un identificador único en la base de datos solamente estarán identificados en las rutas por él. Además de la duración explicada en el apartado anterior se incluyó información cómo el color de la ruta, el número de paradas, el tipo de transporte determinado por google, el identificador, etc. En principio estos valores no tienen un uso en particular pero parece conveniente almacenarlos y no descartarlos de momento. Esto se llevó a cabo tanto para la rutas y nodos de la ciudad de Singapur cómo para Bangkok.

5.2.7 FRECUENCIAS Y HORARIOS

Uno de los parámetros que no se ha visto en los ejemplos de las rutas del apartado anterior es la frecuencia que siguen los trenes en dicha ruta. Esto es una de las principales diferencias respecto a las rutas interurbanas y es fundamental extraerlo correctamente. También relacionado con el tiempo, en un principio los nodos debían tener el horario de apertura o del primer y último tren. En el siguiente script se muestra cómo es posible obtener la frecuencia accediendo a la base de datos con la Google Directions API con unas determinadas peticiones.

```
// Variables
input_file = 'path/routes'
start_time_approx = ... // Hora de apertura aproximada - 5 AM
end_time_approx = ... // Hora de cierre aproximada - 1 AM
req = "https://maps.googleapis.com/maps/api/directions/json?
        &origin=
        &destination=
        &departure_time=
        &mode=transit
        &transit_mode=rail
        &transit_routing_preference=less_walking"
resolution = 900 // 15 minutos de resolución
each_direction = ['forward','backward']
each_day = ['weekdays','no_weekdays']

with open(input_file) as f:
    for direction in each_direction: // Dos casos: forward o backward
        // Se actualiza el origen y el destino
```

```
        orig_location = route[direction]['origin']['location']
        dest_location = route[direction]['destination']['location']
        for day in each_day: // Dos casos: weekdays o no_weekdays
            extra_time = start_time_approx
            while extra_time < end_time_approx: // Bucle de peticiones
                request_time = start_time_approx + extra_time
                first_req = req + orig_location + dest_location + request_time
                json_resp = http.get(req)
                // Extraemos la hora y la frequencia
                departure_time_text = ['departure_time']['text']
                departure_time_value = ['departure_time']['value']
                total_duration = json_resp['duration']['value']
                transit_duration = json_resp['steps'][0]['duration']['value']
                frequency = total_duration - transit_duration
                extra_time = + resolution // Se añaden 15 minutos

                // Detección del último tren
                if transit_duration = usual_transit_duration:
                    frequency_tuples.append(
                        [departure_time_text,departure_time_value,frequency])
                else:
                    end_time_value = departure_time_value
                    end_time_text = departure_time_text

            // Se genera el vector con los distintos intervalos
            for tuple_iter in frequency_tuples:
                    schedule.append(tuple_iter)
            schedule.removeDuplicatesByFrequency() // Se eliminan duplicados
            schedule.append([end_time_text,end_time_value,0]) // Final
            json_frequency = json.loads(json.dumps(schedule))
            // Se guarda la frecuency en formato JSON
            route['route']['schedule'][day]['frequency'] = json_frequency

for row in f: // Se añaden las frecuencias a las rutas del fichero
    final_route = json.loads(row)['schedule'] = route['route']['schedule']
    output.append(final_route)
save output in output_file // Guardar fichero de salida
print("Finished!")
```

Pseudocódigo 5. Extracción de frecuencias.

Este *script* se trata del más complejo ya que requiere de muchas iteraciones y comprobaciones. Entre las variables más importante se encuentran la hora de aproximada de apertura y cierre de la estación. Estas variables contienen una rango de horas en las que es muy probable que exista actividad en la estación y su función principal es el ahorro de peticiones innecesarias a la API.

Hasta ahora no se ha tratado el tema del límite del número peticiones a las APIs. Este número no es un problema ya que cada uno de los scripts se han ido ejecutando de manera progresiva según se iban desarrollando. Al haber hecho uso de las peticiones de forma eficiente, no se han producido casos en los que el límite tuviera un papel importante y se tuviera que rehacer los *scripts*.

Respecto a la petición que se realiza en cada iteración, los parámetros que varían son el origen, el destino y la hora de salida. También se ha definido la variable *resolution* que especifica cada cuanto tiempo se realiza una petición y que por tanto, determina la precisión con la que se han obtenido estas frecuencias. Para finalizar con la variables, por si las frecuencias son distintas dependiendo del día y de la dirección de la ruta se han creado las variables *each_day* y *each_direction* respectivamente.

Básicamente el funcionamiento del *script* es el siguiente. El código se repite para cada dirección y cada día (fin de semana y día de diario). En resumen, lo que hace es una petición cada 15 minutos (establecido con la variable *resolution*) entre la primera y segunda parada de la línea. Realizando la diferencia entre la duración total del trayecto (en el que google ha añadido el tiempo de espera) y la duración del propio recorrido en tren (dentro del propio *step*) es posible obtener el tiempo de espera, que se puede traducir en la frecuencia del propio tren.

En el *script* se va comprobando que los valores no se salgan de un determinado rango de valores para prevenir errores. Una vez se tiene las frecuencias en ambas direcciones y para ambos días en todas las horas del día, se almacena en una variable. Más tarde se remueven los duplicados en base al tiempo de espera para que solamente queden aquellas entradas que determinan un cambio en la frecuencia y a la hora a la que sucede. Por último, se

introduce una última entrada con un valor de frecuencia igual a 0 que especifica el último tren registrado. A continuación se muestra el contenido de la variable.

```
// Para cada día de la semana y dirección se obtiene lo siguiente:
[['6:05am', 720], ['8:20am', 300], ['11:05am', 600], ['12:00am', 0]]
```

En el ejemplo se puede observar la hora de salida del primer tren junto con la frecuencia del tren, a las 6:05 de la mañana y 12 minutos o 720 segundos. Las siguientes dos entradas suponen la hora a la que sucede un cambio en dicha frecuencia (a las 8:30 baja el tiempo de espera a 5 minutos). Y el último determina a la hora a la que sale el último tren (a las 12:00 de la mañana).

Respecto a la veracidad de la frecuencia obtenida, en este caso se han comprobado los valores con la página oficial del metro de Singapur. Respecto a la duración de estos, Google ha supuesto en la mayoría de los casos un aumento de 2 o 3 minutos respecto a los expuestos en la página. Esto quiere decir que Google no ha obtenido esta información de la base de datos oficial sino que probablemente se haya basado en la información de geolocalización que recoge de los dispositivos Android de los usuarios.

Comparando el valor de espera oficial con el experimentado en la vida real, se podría defender que no siempre tarda lo estipulado y por tanto los valores reales están más cerca de los registrados por Google y sus usuarios. Finalmente en el script se ha propagado la frecuencia obtenida en la primera estación de la línea al resto de estaciones debido al funcionamiento de este medio de transporte.

Por último, este script se tiene que ejecutar una vez por línea, ya que cada línea tiene unas horas pico y frecuencias determinadas y tanto para la ciudad de Singapur como Bangkok. No se muestra el script usado para obtener el horario ya que en la implementación final de las rutas no fue utilizado. Más adelante se explica cómo fue usado el campo de frecuencias recién obtenido para tal fin.

5.2.8 IDIOMAS Y CÓDIGO DE LÍNEAS

En el *script* de este apartado, y para finalizar por completo con la información que necesitan los nodos, se extrae el nombre de la estación en diferentes idiomas y el código de la línea. Para la extracción del nombre en distintos idiomas se ha usado la Google Places API. A continuación se muestra el pseudocódigo del *script*.

```
// Variables
input_file = 'path/nodes'
req = "https://maps.googleapis.com/maps/api/place/nearbysearch/json?
        location=
        &radius=
        &type = transit_station
        &language= "
radius = 100
country = "Singapore"
languages_code_baolau=["vi","ja","cn","tw","es"]

with open(input_file) as f:
    for row in f: // Para cada nodo
        node = json.loads(row)
        location = node['location'] // Información del nodo
        name = node['name']
        address = node['address']
        if country == "Singapore":
            vessel = name.split(' ', 1)[0][:2] // Generación de Vessel
            name = name.split(' ', 1)[1]
        else:
            vessel = vessel_forced
        json_languages['en'] = name
        for language in languages_code_baolau: // Para cada lenguage
            // Se realiza la petición y se obtiene el resultado
            req = req + location + radius + lenguage
            resp = http.get(req)
            if 'transit_station' in resp['types']:
                json_languages[language_code_baolau] = json_resp['name']
            else:
                print(error + req)
        // Se almacena el resultado en formatos JSON
        node['name'] = json_languages
```

```
        node['vessel'] = vessel
        save node in output_file // Guardar fichero de salida
print("Finished!")
```

Pseudocódigo 6. Extracción de Idiomas y Códigos de Línea.

En primer lugar, la petición a utilizar es ligeramente distinta a la usada con las APIs anteriores y los dos parámetros opcionales que se ven a modificar a lo largo del script son la localización y el lenguaje. Se ha definido un radio de búsqueda de 50 metros para la búsqueda de la estación. Por último se ha creado un vector con los códigos correspondientes a los seis idiomas utilizados en la página de Baolau.

Respecto a la ejecución, la forma de generar la código de la línea (*vessel* en el código) es distinto para las estaciones encontradas en la red de Singapur que las de Bangkok. La razón de este hecho es debido a que el nombre de las estaciones de Singapur encontradas en una base de datos oficial ya incluían este código. De esta forma, en el caso de Singapur simplemente se separa del nombre y en el caso de Bangkok se introduce fijo con la variable *vessel_forced*.

Respecto a los idiomas, se realizan tantas peticiones cómo veces haya cambiado el parámetro *lenguage*. Se comprueba que efectivamente se trata de la estación en cuestión con un radio relativamente pequeño y comprobando que el sitio es del tipo *transit_station*. Si por alguna razón no se encuentre la estación en ese radio es recomendable ajustarlo dependiendo de la zona geográfica en la que se encuentre. Tras todas las iteraciones todos los nodos correspondientes a una línea obtienen el código de la línea y el nombre en diferentes idiomas.

5.2.9 TARIFAS

Por último y para finalizar con la extracción de datos en su totalidad, se ha dejado la extracción de las tarifas. La principal razón por la que se ha seguido este orden es debido a que el resto de atributos se sacaban de forma automatizada para todas las líneas. En Bangkok el precio de los distintos medios de transporte depende del número de paradas

recorridas mientras que en Singapur depende de la distancia entre el origen y el destino recorrido por el tren.

Las tarifas de los transportes de Singapur se han obtenido de la fuente oficial *data.gov.sg*. Existen cinco tipos de tarifas dependiendo del tipo de usuario y de factores como la edad, si es estudiante, si posee la tarjeta de transporte etc. Se decidió obtener los cinco aunque se almacenaran en un primer momento solamente dos de ellas.

Respecto a las tarifas se han obtenido de las fuentes descritas en el apartado 5.1.3. Dependiendo del transporte la información era dada de distinta forma. En el caso del BTS la información provenía de una tabla con todas las combinaciones entre las estaciones de ambas líneas. Respecto al metro subterráneo o MRT y el tren que interconecta el aeropuerto con la ciudad (ARL), las tarifas dependían estrictamente del número de paradas recorridas. Al contrario que en Singapur estas fuentes solamente tenían registradas un solo tipo de tarifa.

Tras lograr extraer todos los datos y formatearlos para tener acceso a ellos, es necesario ejecutar el siguiente *script* (para Singapur, el caso más complejo).

```
// Variables
input_file = 'path/routes'
country = "Singapore"
fares_file = 'path/fares'

with open(fares_file) as f: // Se cargan las tarifas
    for row in f: fares.append(json.loads(row))

with open(input_file) as f:
    for row in f: // Para cada ruta
        route = json.loads(row)
        distance = route['route']['distance']
        for fare in fares: // Para cada tarifa
            // Se obtiene las fares en JSON según la distancia de la ruta
            if distance >= fare['from_distance']
            and distance <= (fare['to_distance'] + 99)
            and fare['applicable_time'] == "All other timings":
```

```
                json_fares[fare['fare_type']] = fare['fare_per_ride']
        route['route']['fares'] = json_fares
        save route in output_file // Guardar fichero de salida
print("Finished!")
```

Pseudocódigo 7. Extracción de Tarifas.

El funcionamiento es muy sencillo una vez se tienen las tarifas introducidas en una variable y formateadas tipo JSON. Para cada ruta se obtiene la distancia y se busca su tarifa correspondiente para cada tipo de tarifas mediante un bucle. Estas trifas se almacenan en el propio JSON de la ruta y guardan a su vez en un fichero. El *script* para el resto de transportes es ligeramente distinto y no es necesario mostrarlos.

Con la ejecución de este último *script* se completa la extracción total de la información de nodos y rutas. La información se encuentra almacenada en dos ficheros correspondientes a los nodos y a las rutas para cada una de las líneas. Tal y como muestra la planificación del proyecto, se ha invertido una gran parte del tiempo total debido a la importancia de la precisión en los datos recogidos y la deseable capacidad de automatizar este proceso para futuras implementaciones.

5.3 INSERCIÓN DE LA INFORMACIÓN EN LA BASE DE DATOS

En esta sección del desarrollo se introducen los datos previamente comentados a la base de datos de Baolau. De una forma similar al desarrollo web, también se descarga en el equipo local un pequeña porción de la base de datos actual para tener una muestra de los datos y sobre todo, la estructura de la misma. Como se ha explicado en el apartado de los recursos, se ha hecho uso un servidor de mySQL generado mediante la aplicación MAMP. Al contrario que en el caso del desarrollo web, el uso de las herramientas de Sourcetree no sirven para llevar a cabo la gestión de la base de datos. Al finalizar la inserción de datos será necesario generar un fichero con extensión .sql con las entradas que se hayan introducido.

La base de datos está estructurada con una gran cantidad de tablas que realizan una división de las funciones principales de la página. La explicación de cada una de ellas se encuentra fuera del alcance del proyecto por lo que solamente se describe la función de aquella tablas de las que exclusivamente depende el proyecto. Del mismo modo, en las propias tablas existen algunos campos que no aplican al entorno urbano y por tanto no se detallan.

En primer lugar, la tabla *fares* contiene las distintas tarifas que hay disponibles para cada empresa y tipo de transporte. Debido a que se va a introducir varios transportes nuevos será necesario insertar varias filas en esta tabla. En esta tabla también se muestra información de aspectos relativos a la reserva y cancelación de la misma y reglas propias de otros tipos de transporte, como por ejemplo la posibilidad de elegir asiento o el *check-in*. Todos los transportes tratados en el entorno urbano no requieren de una reserva y tampoco tienen este tipo de opciones por lo que se dejará el valor por defecto en cada uno de los campos que no aplique.

Más concretamente, de los 18 campos que contiene esta tabla, se hacen uso de 7 de ellos. El más importante y estrictamente necesario es el identificador de la tarifa. Gracias a este identificador será posible relacionar las tarifas de cada transporte con las rutas en las que aparezcan. El formato de este identificador consiste en el código del transporte seguido de un acrónimo o identificador correspondiente con la tarifa. Por ejemplo en el caso de la tarifa llamada *Adult Card Fare* del metro MRT de Singapur se introduciría el identificador MRT-ACF.

Los otros 6 campos se corresponden al propio nombre de la tarifa en los seis idiomas en los que trabaja Baolau. Al tratarse de transporte urbano no reservable, el uso de esta tabla se encuentra bastante limitada y su única función práctica es la de relacionar cada tarifa con su nombre en distintos idiomas. Sin embargo, rellenarla de forma adecuada es un paso necesario para que se muestren correctamente las tarifas.

La segunda tabla que se ha de modificar se llama *transports* y como su propio nombre indica, en ellos se almacena cada transporte. Al contrario que en la tabla anterior, se han de

rellenar la mayoría de los campos. En primer lugar se encuentra el código del transporte y consiste en una palabra de tres caracteres con el nombre o iniciales del mismo. Para seguir con el ejemplo anterior, en el caso del *Mass Rapid Transit* de Singapur se introduce MRT.

En los seis siguientes campos se almacenan el nombre en cada idioma y en otros seis más una breve descripción de este transporte. También existe otro campo para introducir la página web oficial del método de transporte en cuestión. Dos campos adicionales para introducir un código del país en el que se encuentra y otro para la ciudad (para Singapur el código del país es SG y el de la ciudad SIN). Para finalizar se tiene el campo *deleted*, que por defecto se encuentra a cero, y es de utilidad en los casos en los que se quiera deshabilitar el transporte sin eliminar la entrada en la tabla.

Para finalizar, el último campo de esta tabla es clave secundaria ya que relaciona esta tabla con la siguiente y consiste en un identificar del tipo de transporte. Se ha decidido que, aunque sean trenes y este identificador ya este creado, se debe crear un nuevo tipo de transporte para el entorno urbano. De esta forma, el identificador correspondiente con este nuevo tipo será el número 7 y se corresponderá con el tipo *urban-train*. Esta última información se almacena en la última tabla mencionada en esta sección (aparte de la de rutas y nodos) con el nombre *transport_types*. En esta tabla simplemente se encuentra el identificador como clave primaria y otros campos con el nombre del tipo de transporte en seis idiomas.

5.3.1 INSERCIÓN DE NODOS

Habiéndose hecho manualmente (ya que son pocas entradas) todas las modificaciones mencionadas en las tablas anteriores se procede con la inserción de nodos y rutas en sus correspondientes tablas. Como la inserción de nuevos nodos y rutas en la base de datos se repite constantemente en Baolau, este proceso se encuentra automatizado mediante funciones programadas en PHP. Este tipo de funciones, las cuales se utilizan como herramientas en el proceso de desarrollo se encuentran almacenadas en un fichero llamado Toolbox y es posible acceder a él a través del navegador para mayor comodidad.

Debido a que las rutas y nodos urbanos cuentan con información ligeramente distinta del resto de rutas se ha programado desde cero estas dos funciones. En el orden de ejecución de ambas, se debe comenzar con la inserción de nodos ya que la función de insertar rutas consulta si los nodos existen. A continuación se explica ambas funciones con ayuda de pseudocódigo y la sintaxis es ligeramente distinta debido a que se ha programado en PHP y no en Python (se ha querido preservar la mayor parte del código original sin complicar demasiado la lógica de la función).

```
public function insert_urban_nodes() {
    // Variables
    $file = file_get_contents('[PATH]/nodes_file_name');
    $country_code = "SG";
    $country_name = "singapore";

    foreach $json of $file { // Para cada nodo
        if(get_by('name_en',$json->name->en) == null){
            $row = array(
                'node_type' => "urban-train", // Tipo de transporte
                'urban' => $country_name,
                'name_en' => $json->name->en,
                ... // Resto de idiomas
                'name_tw' => $json->name->tw,
                'town_en' => $json->town->en,
                ... // Resto de idiomas
                'town_tw' => $json->town->tw,
                'country_code' => $country_code,
                'latitude' => $json->location->lat,
                'longitude' => $json->location->lng,
                'address' => $json->address,
                'deleted' => 0 // Nodo activo por defecto
            );
            insert($row); // Inserción de nodo
        }
    }
}
```

Pseudocódigo 8. Inserción de Nodos.

En primer lugar se encuentran las tres variables que se han de modificar dependiendo de los nodos que se estén insertando. Se debe introducir la ruta dónde se encuentra el fichero con las rutas, el código del país y el nombre del país. Para cada nodo la función comprueba si existe un nodo con ese mismo nombre en la base de datos. Si la respuesta es falsa genera un vector accediendo al fichero mencionado anteriormente y rellenando los campos que son necesarios en la tabla de nodos. Para cada iteración inserta el nodo y continúa con el siguiente.

Los campos de esta tabla son prácticamente los mismos que los atributos que se han extraído en el apartado anterior. Entre ellos se encuentra el nombre de la estación en los seis idiomas, el nombre de la ciudad dónde se encuentra en los seis idiomas, las coordenadas y la dirección. Además de estos, gracias a las variables anteriores se introduce el código del país y el valor del campo *deleted* que tiene la misma función que en la tabla de transportes.

Por último, también se introduce el campo *node_type* y el *urban*. Aunque estos se comentarán más adelante en el apartado de la implementación, el primer campo servirá principalmente para mostrar el icono correspondiente a este tipo de transporte (*urban-train*) y el segundo para detectar a que ciudad pertenece dicho nodo. Este último campo es necesario para diferenciar que estaciones mostrar en el autocompletar de cada *mini site*.

5.3.2 INSERCIÓN DE RUTAS

A continuación se muestra el pseudocódigo de la función que se encarga de insertar las rutas. Este se debe ejecutar después de haber introducido los nodos con la función previa.

```
public function insert_urban_routes() {
    // Variables
    $file = file_get_contents('[PATH]/routes_file_name');
    $currency = "SGD";
    $transport_code = "mrt";

    foreach $json of $file { // Para cada ruta
            $origin_id = get_by('name_en',$json->origin->name)->id; // Búsqueda del nodo
```

```
            $destination_id = get_by('name_en',$json->destination->name)->id;
            $duration = ($json->route->duration)/60; // A minutos
            $vessel = $json->route->vessel;

            // Tarifas
            $ACF = "Adult card fare";
            $ST = "Single trip";
            $prices = "ACF:".$json->route->fares->$ACF."#ST:".$json->route->fares->$ST;

            // Frecuencias
            $days = array('weekdays','no_weekdays');
            foreach $day of $days{
                $frequencies = (array)$json->route->schedule->$day->frequency;
                foreach($frequencies as $entry){
                    $json_temp->$day->$i = [$entry[1]/60, $entry[2]];
                }
            }
            $frequency = php.serialize($json_temp); // Serializado

            $row = array(
                'origin_id' => $origin_id,
                'destination_id' => $destination_id,
                'duration' => $duration,
                'vessel' => $vessel,
                'prices' => $prices,
                'currency' => $currency,
                'transport_code' => $transport_code,
                'frequency' => $frequency,
                'static' => 1
            );
            insert($row); // Inserción de ruta
    }
}
```

Pseudocódigo 9. Inserción de Rutas.

En el caso de las rutas, las variables que se deben introducir previa a la ejecución de la función son la ruta del fichero en el equipo, el código de la divisa de las tarifas de ese país (SGD para el Singapur Dólar) y el código del transporte. La función itera para cada ruta

que encuentra en dicho fichero y genera los campos necesarios de la tabla de rutas de la base de datos.

En primer lugar, realiza una búsqueda en la base de datos a través de la función *get_by* para determinar el código de identificación único de los nodos origen y destino que interconecta la propia ruta. Después pasa la duración de la ruta a minutos y almacena el código del transporte (*vessel*). Los dos siguientes campos deben seguir un determinado formato que se explican a continuación.

Respecto a las tarifas, a pesar de no ser solamente una, se deben almacenar en un solo campo. Para aumentar la compatibilidad del código ya existente se sigue el siguiente formato que es común a todas las rutas de la plataforma.

```
"Código_de_la_tarifa_1:precio_1#Código_de_la_tarifa_2:precio_2"
```

Por ejemplo, el campo de las tarifas es rellenado de esta forma para una ruta del metro de Singapur.

```
"ACF:77#ST:140"
```

Respecto a las frecuencias, se crea un JSON parecido al explicado en el apartado 5.2.7 pero emitiendo la hora en número de segundos por lo que solamente cada intervalo tiene la hora en texto y la frecuencia del tren en minutos. Tras generar esa variable, es necesario serializarla mediante una función propia de PHP ya que es una práctica recomendable al guardarla en la base de datos un JSON. De esta forma, a pesar de no ser legible directamente desde la base de datos se le añade a dicho campo una capa de seguridad.

Tras llevar a cabo la generación de estas variables, se aúnan en un vector incluyendo el campo *static* a 1 ya que se trata de una ruta estática en la base de datos y no va a cambiar con el paso del tiempo. Por último se introduce en la base de datos mediante una función ya generada.

La ejecución de ambas función se ha de repetir para cada línea de metro u otro transporte que se quiera introducir. Esto se ha hecho para cada ruta independientemente para prevenir errores y facilitar su solución en caso de que ocurran. Tras llevar a cabo la inserción de nodos y rutas en la base de datos se implementó los cambios necesarios en el actual motor de búsquedas para que se logren obtener resultados en el entorno urbano. Este proceso se explica en la siguiente sección.

5.4 IMPLEMENTACIÓN

Esta sección cubre la etapa principal del proyecto que consiste en la implementación de los cambios necesarios en el código de la página web para que funcionen correctamente las rutas urbanas extraídas anteriormente. Se incluye desde los cambios en el motor de búsqueda hasta el diseño e implementación del interfaz de usuario. Se trata de la parte más importante en términos de la planificación del proyecto ya que en esta etapa se ha invertido la mayor parte del tiempo total disponible.

La sección va a constar de cuatro apartados que engloban los principales puntos que se han de explicar de la implementación. En primer lugar se describe brevemente el objetivo principal de esta etapa con el fin de recordar las principales funcionalidades finales. Tras esta explicación se evalúan los efectos sobre la página que pueden producir los cambios a realizar y cómo se van a tener en cuenta durante la programación.

Más tarde, se procede a explicar la implementación de las rutas urbanas en el motor de búsquedas y el algoritmo de la página para lograr obtener los resultados esperados. Y para finalizar con la sección se explicará tanto el diseño como la implementación del interfaz de usuario y la forma de representar los resultados urbanos.

5.4.1 RESULTADOS ESPERADOS

Antes de proceder con la propia implementación de los cambios, en este apartado se explica el objetivo o resultado que se debe alcanzar tras este proceso. Al tratarse de rutas urbanas que no requieren de un reserva previa el uso que se le dará a estas rutas es distinta

al de las rutas tradicionales o interurbanas. Existen diferencias tanto al nivel interno del motor de búsqueda como a su propia representación en la página.

Como se ha explicado en varias ocasiones a lo largo de este documento, la introducción de rutas urbanas se planificó con el fin de otorgar más opciones de comunicación entre dos estaciones en aquellas rutas combinadas que fuese necesario desplazarse dentro de la ciudad. Hasta entonces la única opción que se estaba utilizando era el transporte vía taxis. Dado que en muchas ocasiones las principales estaciones de una ciudad están conectadas por otros medios de transporte públicos se barajó la posibilidad de introducir nuevos en el sistema.

Desde el primer momento se decidió que las rutas urbanas solo se debían mostrar a nivel informativo y, como al contrario que la mayoría de rutas del sistema, su reserva previa no fuera posible. Esto es muy importante a nivel de implementación ya que no es necesario llevar a cabo ciertos procesos de validación y consulta tan sólo para mostrar la ruta. Este hecho también conlleva cambios en el interfaz ya que ciertos elementos cómo el selector de tarifa o el botón de reserva no deben aparecer o tener la misma funcionalidad.

Al tratarse de rutas estáticas que no cambian con el paso del tiempo no es necesario realizar ninguna consulta durante el cálculo de la ruta óptima, al contrario de lo que ocurre con las rutas de plazas limitadas como los aviones o trenes. Otra diferencia de las rutas urbanas, es que no tienen una hora de salida determinada sino que esta debe ser calculada a partir de la hora de llegada a la estación y la frecuencia del tren. Estas son algunas diferencias a nivel lógico que será necesario tratar durante la implementación de las rutas urbanas para tener los resultados esperados.

Respecto a su visualización, es importante destacar cómo se mostrarán al usuario. Debido a que las rutas urbanas sólo se muestran a nivel informativo y como interconexión de dos grandes rutas, estás aparecen en un nivel inferior. De una forma similar, los nodos urbanos no se encuentran disponibles a la hora de hacer la búsqueda en el formulario principal ya que no son destino principal. Aunque en un futuro se pueda implementar esta función, de

momento se ha mantenido esta decisión y por tanto no se mostrarán en el autocompletar del origen y el destino del interfaz.

Cómo resumen, el uso principal de las rutas urbanas será el siguiente. Desde la página principal el usuario no notará su existencia ya que no se encuentra disponibles en la selección del origen y el destino. Solamente se mostrarán una vez que haga la búsqueda y le salgan como resultado rutas combinadas en las que haya una interconexión entre nodos de una misma ciudad. Dónde tradicionalmente sólo aparecería la opción del taxi, tras la implementación debiera aparecer también la opción del metro o tren ligero. En el apartado de la representación de resultados se define con más detalle la forma y elementos que tendrán estas rutas urbanas.

Una vez se tuviese una red de transporte urbano lo suficientemente compleja se podría dar un uso adicional a estas rutas urbanas. La segunda principal función de ellas sería crear un página especializada en el entorno urbano de dicha ciudad. De momento y para completar el desarrollo del proyecto se crearán las páginas *singapore.baolau.vn* y *Bangkok.baolua.vn*. Estas páginas serán independientes para cada ciudad y no tienen poseen la capacidad de trabajar con rutas de la página principal. Durante este documento se hace referencia a ellas con el término de *mini-site*.

Cómo es de esperar, los principales cambios en ambas páginas se realizarán en su interfaz y no en la lógica. Los únicos cambios lógicos que notará el usuario que utilice los *mini-sites* será tener acceso a los nodos urbanos en el formulario de búsqueda y la posibilidad de seleccionar la hora de salida.

Respecto a los cambios del interfaz, aunque se discuten más adelante, se producirán en la página principal con algunos elementos decorativos, imágenes, el mapa urbano y la eliminación de alguna sección. Cómo se ha dicho, también será necesario cambiar el formulario de búsqueda y en la página de resultados, el botón de reserva será sustituido por uno que sirva para mostrar la ruta seleccionada en el mapa de la ciudad. Estos son algunos cambios que se deben implementar y que se explican con más detalle en los siguientes

apartados. Todos las modificaciones en la interfaz se realizarán con un diseño de acorde con el actual.

Respecto a la usabilidad de los *mini-sites*, cuando se detecte que el usuario va a visitar una ciudad de la que se tengan rutas urbanas, se anunciará mediante un enlace el acceso al *mini-site*. De esta forma el usuario no abandona la plataforma una vez que ha finalizado su viaje principal y le da uso mientras se encuentra en dicha ciudad. De esta forma, los *mini-site* hacen que el usuario cuente con Baolau como solución única y no se vea obligado a usar otra aplicaciones.

Una vez el usuario accede a la página durante su estancia en dicha ciudad, con el nuevo interfaz sí tiene acceso a las estaciones que pretenda seleccionar y la posibilidad de salir en ese momento o a una hora determinada. De esta forma el usuario puede planificar su visita a los distintos lugares de la ciudad y ver las rutas fácilmente en el mapa. Aunque este tipo de rutas no sean reservables también se mostrará las distintas tarifas disponibles de manera informativa.

Cómo los dos usos principales de las rutas urbanas, la interconexión de nodos en rutas interurbanas y los *mini-site*, son muy importantes es necesario ambos no se interfieran. Para optimizar la convivencia de la página principal con los *mini-sites* de cada ciudad se resolverán los conflictos que puedan suceder durante su implementación. Al ser un aspecto fundamental de la implementación, la resolución de estos conflictos se estudian en el siguiente apartado.

5.4.2 COEXISTENCIA DE RUTAS URBANAS E INTERURBANAS

Al introducir nueva funcionalidad en la página y esta puede interferir en las funciones que realizaba previamente es necesario filtrar de algún modo dicho comportamiento. Este es el caso de la implantación de rutas urbanas que se ha llevado a cabo en el proyecto y por tanto, hay que buscar una serie de parámetro o identificadores únicos con el que se pueda diferenciar en qué casos se ejecuta el nuevo código. En este apartado se discute algunos ejemplos de situaciones en las que ha sido necesario y cómo se podido filtrar.

Ya sea para implementar cambios en el motor de búsqueda o para desarrollar el interfaz, se ha hecho uso del operador condicional *if*. Se han identificado solamente dos variables con las que se puede determinar que parte del código se ha de ejecutar. La primera variable se trata del código del transporte utilizado en la ruta urbana. Una vez se hace la consulta a la base de datos y se tiene todas las rutas que tendrá en cuenta el algoritmo, cada una posee un código correspondiente al transporte usado.

Para aclara esta posibilidad a continuación se muestra la condición para el ejemplo que se ha explicado en varias ocasiones. Suponiendo que el código llega a una iteración en la que la ruta es el metro de Singapur sería posible hacer lo siguiente.

```
// Si es hay que añadir una funcionalidad para el transporte mrt
If($r->transport_code == "mrt"){…

// Si es hay que quitar una funcionalidad para el transporte mrt
If($r->transport_code != "mrt"){…
```

De esta forma se puede diferenciar las situaciones dependiendo del tipo de transporte. Si se quisiera aplicar a todas las rutas urbanas insertadas en la base de datos se debería añadir a la condición los códigos del resto de transportes.

La otra variable binaria que se utilizará en otras situaciones es la que tiene el nombre de *urban-site*. Esta variable determina mediante la URL de la barra de direcciones del navegador si se trata de un *mini-site* o de la página principal. En el caso de entrar en el *mini-site* de Singapur o Bangkok toma el valor de verdadero. Se hará uso de ella mediante la sintaxis condicional mencionada anteriormente pero comprobando el valor de dicha variable. Por si fuera necesario diferenciar cada uno de los *mini-site*, también se ha creado la variable *subdomain* que toma de la URL el nombre de la ciudad del *mini-site*. En el apartado de la implementación del interfaz se retoma esta variable con uno de sus usos.

Por regla general, solamente se filtra por código de transporte cuando se trata de una funcionalidad propia de las rutas urbanas y que debe aparecer en la aplicación global.

Mientras que el uso de la variable *urban-site* se usará principalmente en los cambios que afectan exclusivamente al *mini-site* y al conjunto de todas rutas urbanas en dicho contexto. Haciendo uso de estas dos variables se ha podido independizar el código programado a lo largo del proyecto con el ya existente. Adicionalmente, como parte de buenas prácticas, todas las partes del código generado con este proyecto se ha comentado con una palabra clave única para saber en todo momento que código se escribió y pertenece a las rutas urbanas y a los *mini-site*.

Tras comentar las variables usadas para formar la condición, a continuación se explica brevemente algún momento en el que se ha tenido que usar esta solución para resolver un conflicto con la implementación anterior. En cada uno de ellos se muestra la variable usada para filtrar el comportamiento. Todos los cambios anunciados pertenecen solamente a la implementación del motor de búsqueda y se han omitido la parte del interfaz al ser demasiados. Muchos de ellos se discutirán en el apartado siguiente.

- **Error por mismo origen y destino**: En la página principal, si se pone como origen y destino dos aeropuertos o estaciones que se encuentren en la misma ciudad genera un error que te pide continuar con la búsqueda. Debido a que ya se tiene acceso a rutas urbanas, se debe quitar esta condición para los *mini-site* haciendo uso de la variable *urban-site*.

- **Botón de reserva**: Como se ha comentado anteriormente, el botón de reserva ha sido sustituido por un botón que aparece en el misma posición pero que te muestra la ruta determinada en un mapa. Cómo esta funcionalidad sólo se quiere en las páginas urbanas se utilizará *urban-site*.

- **Configuración de los mapas**: Para mostrar las rutas a través de un mapa se ha utilizado el servicio que proporciona Google de mapas. Dependiendo de la ruta se utiliza un determinado modo de transporte y coordenadas a la hora de mostrarla. Solamente se hará uso de esta configuración en el caso de los sitios urbanos que es dónde se hace uso del modo de transporte público proporcionado por estos mapas.

- **Sólo resultados a partir de 2 horas**: Por defecto y para tener el tiempo necesario para realizar la reserva, Baolau desecha las rutas que empiezan en menos de dos horas desde que se realiza la búsqueda y dependiendo del transporte más. Cómo en los *mini-sites* se podrá elegir comenzar la ruta a la misma hora a la que se realiza la búsqueda es necesario quitar esa limitación con la variable *urban-site*.

- **Filtro de precios en el formulario**: Concretamente en el *mini-site* de Singapur se ha querido implantar un filtro en el formulario de búsqueda para cambiar entre los dos precios disponibles. Para este caso en concreto se debe filtrar con la variable *urban-site* junto con el valor del subdominio ya que sólo se ha implementado para Singapur.

A continuación también se mencionan algunos conflictos de los cambios en la propia implementación del algoritmo de Dijkstra.

- **Hora de salida a partir de las frecuencias**: En el caso de los tipos de transporte urbanos se ha de sobre escribir la hora de salida del tren a partir de la hora de llegada a la estación y la frecuencia. Para filtrar este comportamiento se debe usar el código del transporte debido a que debe seguir funcionando en la página principal.

- **Tiempo de espera entre transportes**: Debido a que en el *mini-site* ya se ha tenido en cuenta con la frecuencia de los trenes el tiempo de espera medio que se espera al tren en las estaciones no es necesario añadir unos minutos entre trasbordos. Esto se hace en la página principal para añadir cierto margen de tiempo en cada intercambio y por tanto se ha de filtrar con la variable *urban-site*.

- **Cuenta de los pasos o trasbordos**: Otro caso particular de los *mini-site* es la forma de contar el número de pasos que contiene la ruta cuando se trata de una combinada. En el caso de la página principal se comprueba un cambio en el código del transporte y de ciudad, sin embargo, en esto se debe filtrar con *urban-site* para que se cuenten los trasbordos urbanos.

- **Identificador de la ruta**: De una forma similar a la anterior, el identificador de cada ruta se genera a partir de un conjunto de campos que no es excluyentes en el caso de las rutas urbanas. Dependiendo del código del transporte se generará haciendo uso de algún campo adicional para que este identificador sea verdaderamente único.

Esto ha sido tan sólo algunos ejemplos de los cambios que se han realizado a lo largo del proyecto que generan un conflicto con el código ya existente. Como se puede comprobar muchos de ellos son de gran importancia e incluso críticos para que funcionen las rutas urbanas. Por tanto, la gestión de conflictos gracias a estas dos variables es fundamental y se verá en muchos de los códigos mostrados en los dos apartados siguientes.

5.4.3 MOTOR DE BÚSQUEDA Y ALGORITMO DE DIJKSTRA

En este apartado se comentan las modificaciones principales que se han llevado a cabo en el motor de búsqueda y el algoritmo para lograr obtener los resultados esperados. Para explicar con claridad todos estos aspectos se presenta el pseudocódigo de estas partes del código y se detalla el funcionamiento y el contexto dónde se sitúa el mismo. En el pseudocódigo se ha omitido la información que no facilite la comprensión del código o que no sea estrictamente necesaria para llevar a cabo la función del mismo.

Como se ha explicado en el apartado anterior, ha sido necesario el uso de dos variables para diferenciar la funcionalidad dependiendo de la página introducida por el usuario en el buscador o de la ruta resultado. A continuación se explica de qué forma se determina el valor de la variable *urban_site*. Esta variable aparecerá constantemente a lo largo de los códigos mostrados en este y el siguiente apartado.

```
// -- core_controller.php
// Generación de la variable urban_site
$data['urban_site'] = True if $data['subdomain'] in ['singapore','bangkok'];
```

En la primer línea de todos los siguientes pseudocódigos aparecerá el fichero dónde se ha introducido el código para facilitar el entendimiento del mismo. En este caso, la definición de esta variable se ha introducido en el controlador principal de la aplicación ya que su código se ejecuta prácticamente en primer lugar. Muchas de las variables globales que se usan a lo largo de la aplicación se definen en este controlador.

Respecto a la variable *urban-site*, se trata de una variable booleana que tendrá un valor verdadero solamente cuando la variable *subdomain*, extraída del dominio de la página, coincida con las palabras definidas en un vector. De momento se ha implementado la ciudad de Singapur y Bangkok por lo que el vector sólo tiene dichos valores. En el caso de acceder a la página principal la variable *subdomain* se encuentra vacía o sin valor y por tanto *urban-site* toma el valor de falso.

El siguiente código se trata de una función llamada *get_next_urban* que será usada por el motor de búsqueda y el algoritmo. Como su propio nombre indica, su función es la de obtener la hora de salida del próximo tren de una determinada estación. Esta función se ejecutará en cada iteración del conjunto de rutas que tienen posibilidad de formar parte del resultado final. Como se ejecuta para cada ruta y con información extraída de ella, se ha definido en un archivo modelo en el que se definen atributos y funciones de las rutas. A continuación se muestra el pseudocódigo de ella:

```
// -- models/routes.php
public function get_next_urban($departure_time, $frequency) {
    $frequency = unserialize($frequency); // Se deserializa la frecuencia
    do{
        $weekday = date('w', $departure_date); // Día de la semana actual
        if($weekday == 0 || $weekday == 6){ // Si es fin de semana
            $frequencies = $frequency->no_weekdays;
        }else{ // Si es día de diario
            $frequencies = $frequency->weekdays;
        }
        $slot = $frequencies->{0}; // Primer intervalo de frecuencias
        $next_departure_time = $slot[0];
        $extra_wait_time = $slot[1];
        $next_day_secs = 0;
```

```
        foreach $slot of $frequency{ // Se recorren todos los intervalos
            // Por si el último tren pasa de las 12am
            if ($slot[0] < $prev_slot[0]) $next_day_secs = 86400;
            // Se obtiene el intervalo más cercano a la hora actual
            if($departure_time > ($slot[0] + $next_day_secs)){
                $next_departure_time = $departure_time;
                $extra_wait_time = $slot[1];
            }
            $prev_slot = $slot;
        }
        // Si no hay ninguno disponible para esa hora pasa al siguiente día
        if($extra_wait_time == 0){
            $departure_time = $next_day;
        }
    }while($extra_wait_time == 0);

    // Devuelve la hora del siguiente tren y el tiempo de espera
    return array($next_departure_time, $extra_wait_time);
}
```

Pseudocódigo 10. Función para calcular el siguiente tren urbano.

La función tiene como argumentos la hora a la que hay que buscar el siguiente tren y las frecuencias extraídas de la ruta que se está analizando. Esta función se llamará tanto si es el primer tren de la ruta (y entonces el primer argumento es la hora de salida desde el origen) como los siguientes (el primer argumento es la hora de llegada a la estación). Respecto al segundo argumento, la frecuencia, es necesario "deserializarla" ya que fue almacenada en la base de datos de esta forma.

En este punto es necesario recordar la estructura que tiene el campo de la frecuencia. En la extracción de datos se almacenó en un JSON cada intervalo de tiempo con un determinado tiempo de espera. De esta forma, es posible recorrer cada uno de esos intervalos hasta encontrar la hora a la que queremos saber la frecuencia del tren. Antes de iniciar este bucle, se determina a qué día de la semana pertenece la fecha y hora de entrada.

Se recorre cada uno de estos intervalos comparando la hora de búsqueda con la de inicio de dicho intervalo. Si la hora de búsqueda es mayor se almacena la frecuencia de dicho

intervalo y de esta forma, tras recorrerlos todos se obtiene el valor buscado. También hubo que tener en cuenta que en las ocasiones en las que el último tren saliera más allá de las 12. Para que esto no se traduzca en un problema se compara constantemente el intervalo actual con el anterior para detectar si se ha pasado de día.

Finalmente, la función devuelve la hora de salida del tren y el tiempo de espera o frecuencia de los trenes. Se ha decidido introducir tanto tiempo de espera como la propia frecuencia del tren en el resultado. Generalmente la hora de salida del tren suele ser la misma que se introdujo en la función si está se encuentra dentro del horario de apertura de la estación determinada. En su defecto está hora se corresponde con la salida del primer tren.

Una vez explicada esta función, las siguientes partes del código se corresponden con el motor de búsqueda. En primer lugar, y como una de las diferencias de los sitios urbanos, fue necesario hacer posible la búsqueda entre estaciones que se encuentran en una misma ciudad. Esto se ha llevado a cabo mediante la siguiente condición.

```
// -- engine.php
// Error mismo origen y destino
if(data['urban_site']){ // Solo para los páginas urbanas
    if ($params['origin'] == $params['destination']) { // Por nodo
        error = "error_same_origin_destination";
        return false;
    }
}else{ // Para la página principal
    if ($params['origin_town'] == $params['destination_town']) { // Por ciudad
        error = "error_same_origin_destination";
        return false;
    }
}
```

Pseudocódigo 11. Detección de error en el formulario.

Para permitir este tipo de búsquedas simplemente fue necesario hacer uso de la variable *urban-site* definida previamente. De esta forma se comprueban las ciudades origen y destino cuando el usuario esté visitando la página principal y los nodos cuando se

encuentre en las páginas urbanas. La siguiente modificación a realizar en el motor de búsqueda es la extracción de la hora de salida del primer tren a partir de la frecuencia gracias a la función explicada anteriormente. A continuación se muestra dicha funcionalidad.

```
// -- engine.php
// Modificación de la hora de salida de la ruta
if ($data['urban_site']){

    list($next_time,$extra_wait_time) =
          get_next_urban($params['travel_time'], $route->frequency);
    $next_time += $extra_wait_time*60; // Le añado el tiempo de espera
    $route->start_time = $next_time;

    // Sólo se aceptan las rutas que salen el mismo día
    if($route->start_date != $travel_date) continue;
}
```

Pseudocódigo 12. Hora de salida para rutas directas.

Este código corresponde al momento anterior en el que el motor de búsqueda detecta si la ruta directa parte del origen determinado y llega al destino correcto y si además la hora de inicio no haya pasado. Antes de dicha situación, se ejecuta la función mencionada anteriormente para obtener la hora de salida de dicho tren y el tiempo de espera medio que el usuario tendrá consumir. Ambos datos se sobrescriben en la ruta y se continúa si la hora de salida entra dentro del día actual. Esta última condición se ha tomado como decisión de negocio ya que no se han querido mostrar rutas del día siguiente a la fecha buscada.

Es necesario recordar que el motor de búsqueda, entre otras funciones, es el encargado de encontrar las rutas directas, mientras que el algoritmo de Dijkstra de las combinadas. Como se puede comprobar más adelante se ha implementado un código similar al anterior en el algoritmo para realizar la misma función. El siguiente código muestra una modificación propia de la página urbana y que se encuentra relacionado con la forma de representar las rutas resultado.

```
// -- engine.php
if($data['urban_site']){
    $result['alternative_action'] = 'map'; // Habilitado el botón de mostrar mapa

    $result['total_price'] = 0;
    foreach $step of $steps {
          // Utilizando el filtro de tarifas
        $step['price'] = $step['prices'][$params['fare_filter']];
        $result['total_price'] += $step['price']; // Actualizando el precio total
    }
    return $result;
}
```

Pseudocódigo 13. Últimos cambios en el motor de búsqueda.

Aunque en el apartado de la implementación del interfaz se explica con más detalle, la función del botón de reserva para las rutas urbanas deja de tener dicha función. Con la primera línea del código se ha hecho que este botón cambie de aspecto y su nueva función sea la de mostrar la ruta en el mapa de forma aclarativa. Además de esta modificación, debido a la implementación de un filtro en el formulario de búsqueda que afecta a las tarifas el siguiente código se encarga de recalcular el precio mostrado por defecto y el precio total de la ruta (si se tratase de una combinada).

Una vez explicados los cambios más significativos del motor de búsqueda, a continuación se muestran algunos realizados en el algoritmo de Dijkstra.

```
// -- dijkstra.php
// Modificación de la frecuencia de los trenes
if($route['transport_code'] == 'mrt' && ...){
    $arrive_time = $arrival_from_previous_node;

    // Hora del siguiente tren
    list($next_time,$extra_wait_time) = get_next_urban($arrive_time, $route->frequency);
    $next_time += $extra_wait_time*60;
    $route->start_time = $next_time;
}
```

Pseudocódigo 14. Hora de salida extraída de la frecuencia.

Este es el código equivalente al mostrado en el motor de búsqueda encargado del cálculo de la frecuencia y hora de salida del tren. La principal diferencia es que la hora pasada a la función no es la hora de inicio de la ruta sino la hora de llegada desde el nodo anterior. Esto es totalmente necesario para que se la frecuencia de los trenes sea precisa en las rutas combinadas. También es importante destacar que este código no sólo se ejecuta en los sitios urbanos sino que también en la página principal si se trata de un transporte urbano (como puede ser en el ejemplo MRT). En el código se ha omitido el resto de transportes.

El pseudocódigo mostrado a continuación se encarga del cálculo de una variable del que no depende las rutas resultado. Se trata de la variable *steps_count* y se encarga de contar en las rutas combinadas las veces que se tiene que cambiar de transporte y que aparece representada junto con el resto de información de la ruta. Para la página urbana se hace uso de este contador para contar el número de trasbordos que se han de realizar hasta llegar al destino.

```
// -- dijkstra.php
// Modificación para contar los pasos de la ruta
if($data['urban_site']){
          if (!in_array($step['transport_type'])) $steps_count++;
}else{
          if ($step['origin_town'] != $step['destination_town']) $steps_count++;
}
```

Pseudocódigo 15. Pasos de las rutas urbanas.

Para ello, el único cambio que hay que realizar es contar cuando se ha producido un cambio en el tipo de transporte y no de ciudad, que es lo que ocurre en la página principal. Por último, el siguiente código muestra una funcionalidad esencial para que se puedan llegar a representar las rutas en la página de resultados.

```
// -- dijkstra.php
// Modificación del id de la ruta
if($route['transport_code'] == 'mrt' && ...){
    $route['id'] = md5(implode($transport_codes+$vessels+$total_time+$total_price));
}
```

Pseudocódigo 16. Generación del código identificador de rutas urbanas.

Se trata de la forma en la que se calculan los códigos identificadores únicos de cada resultado. Mientras el transporte de la ruta en cuestión pertenezca a un tipo urbano el cálculo del identificador se genera a partir del código de transporte, del nombre de la línea y del tiempo y precio total de la ruta. Se han elegido este conjunto de atributos porque son capaces de determinar de manera única una ruta urbana.

El pseudocódigo mostrado en esta sección son las principales modificaciones que se han tenido que realizar tanto en el motor de búsqueda como en el algoritmo de Dijkstra para lograr mostrar los resultados urbanos teniendo en cuenta sus diferencias respecto al resto. De este modo, la página puede calcular las rutas entre el origen y el destino determinado por el usuario pero no es capaz de mostrarlas de forma adecuada. Para ello, ha sido necesario realizar las modificaciones mostradas en el apartado siguiente.

5.4.4 INTERFAZ DE USUARIO

Esta es la última sección de la implementación y es tan necesaria como la anterior. Aparte de que los resultados se muestren correctamente el diseño del interfaz también puede dar lugar a cambios en la funcionalidad de la página y la forma de actuar del usuario. En esta sección se presentan todas las modificaciones que se han realizado en el interfaz de la página y en las funcionalidades.

El aspecto final de todos los elementos mostrados en esta sección han sido resultado de varias iteraciones y reuniones con otros trabajadores. Esta ha sido una de las principales ventajas de la metodología seguida en este desarrollo en el que impera la comunicación. Según se iban desarrollando los cambios se subían a un servidor de pruebas para poder obtener el *feedback* necesario y continuar correctamente.

Al igual que en la implementación de la lógica, existen cambios que se comparten con la página principal y otros que son propios para las página urbanas de cada ciudad. Para llevar un cierto orden, se presentan en primer lugar los cambios comunes a ambos entornos y más adelante los propios de las páginas de Singapur y Bangkok. Para diferenciar ambos

comportamientos se han hecho uso de las mismas variables presentadas en el apartado anterior.

En primer lugar, para los cambios que se comparten en ambos entornos se filtrará a través de la variable de *transport_codes* propio de cada ruta. Esta variable contiene todos los códigos de los transportes que contiene la ruta (más de uno si se trata de una ruta combinada). En el caso de que dentro de dicha variable se encuentre un transporte urbano se llevarán a cabo sus respectivos cambios en su representación. Esto es posible porque la única parte común de ambos entornos se encuentra en la representación de las rutas.

Las otras variables que se usarán en las figuras mostradas en segundo lugar y que solamente se corresponden con los sitios urbanos son *urban_site* y *subdomain*. De esta forma es posible discernir fácilmente si el usuario se encuentra en la página principal o en la página local de una de las dos ciudades implementadas hasta el momento. Las tres variables obtienen su valor de la forma mencionada en el apartado anterior.

En lugar de representar el pseudocódigo correspondiente a cada modificación se ha decidido mostrar una figura de una captura de pantalla de dicha alteración. La principal razón es debido a que en la mayoría de los casos no tiene ninguna complejidad y son modificaciones de HTML. Adicionalmente, una de las ventajas del desarrollo web en PHP es que acepta la inclusión de HTML, compartiendo así la lógica y la vista un mismo fichero. Por tanto, en el caso de considerarse necesario se mostrará el pseudocódigo correspondiente. A continuación se muestra la parte común a ambos entornos.

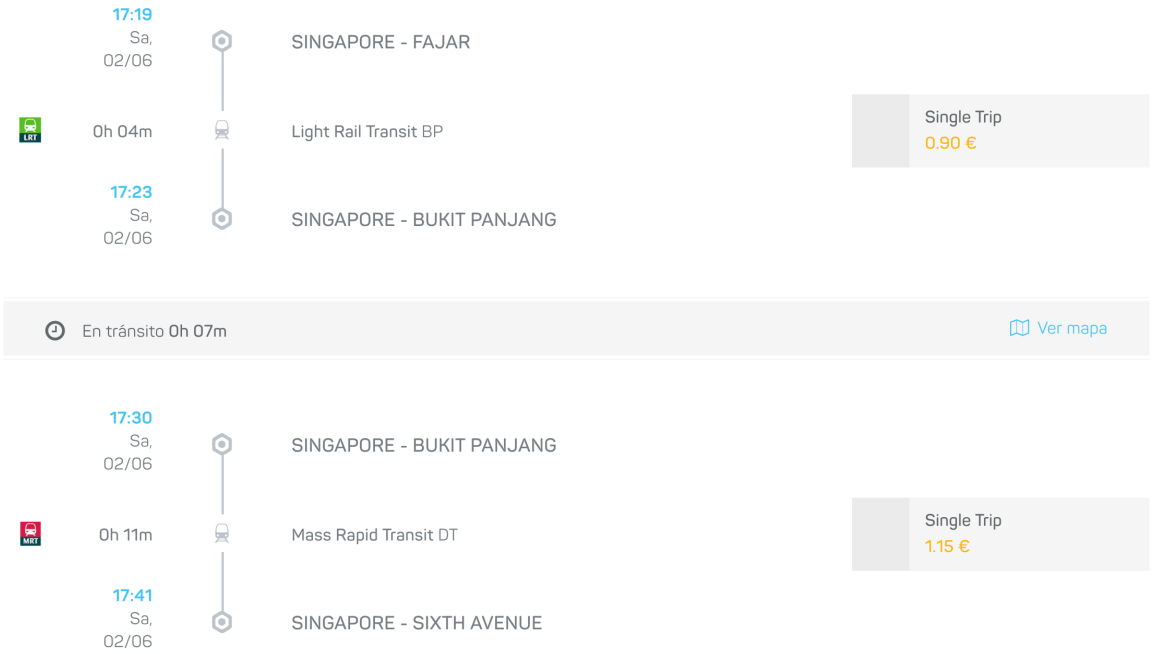

Figura 10. Ejemplo. Ruta combinada desplegada.

Este es el aspecto que posee cualquier ruta cuyo transporte se haya definido como urbano. En este ejemplo el usuario ha introducido en el sitio urbano de Singapur la estación de Fajar como origen y la de Sixth Avenue como destino en una fecha y hora determinada. El resultado es una ruta combinada compuesta por un trayecto en el tren ligero LRT seguido de otro tramo en el metro o MRT. Esta imagen corresponde a la ruta desplegada por el usuario ya que el resumen de la ruta no se mostraría de igual forma en ambos entornos. A continuación se describe brevemente que elementos han sido modificados en esta representación:

- **El número de tarifas**: a pesar de que en muchos casos los transportes urbanos tienen distintas tarifas para un trayecto, como era el caso de Singapur, se decidió mostrar tan sólo una tarifa. La principal razón de esta decisión era mantener simplificada la representación de la ruta lo máximo posible. De todos modos aunque sólo sea posible mostrar una tarifa al mismo tiempo, como se verá más adelante en el caso de Singapur se puede seleccionar el tipo de tarifa desde el formulario de búsqueda.

- **La tarifa**: Se muestra de una forma distinta en comparación con una ruta de la página principal. En las originales, al tener a disposición del usuario varias tarifas, se le da la opción de seleccionar la que más le interese y que más tarde podrá reservar. En el caso urbano, al mostrarse solamente de manera informativa se decidió eliminar dicho selector y el botón de información. Más adelante se muestra un ejemplo dónde se observa dicha funcionalidad.

- **La dirección de las estaciones**: en las rutas originales se muestra debajo del nombre la dirección de la localización de la estación. En esta representación se ha omitido para simplificar la ruta.

Varios elementos que puede parecer que fueron modificados es el logo de ambos transporte y el icono del tren urbano que aparece a la derecha de la duración de cada segmento. Ambos elementos gráficos fueron insertados en la base de datos pero para que se muestren como lo hacen no fue necesario programar ningún cambio sino tan solo nombrarlos de la manera adecuada.

La última modificación que aparece tanto en la página urbana como la página principal es el mapa que muestra de manera gráfica ambos segmentos de los que está compuesta la ruta. Se tiene acceso mediante el botón que aparece a la derecha junto al tiempo de espera. Debajo de la ruta, en la página principal se encuentra el botón de reserva la ruta, y en la página urbana ese botón se ha sustituido por uno que también da acceso al mapa. Al seleccionar dicha opción se muestra la siguiente figura.

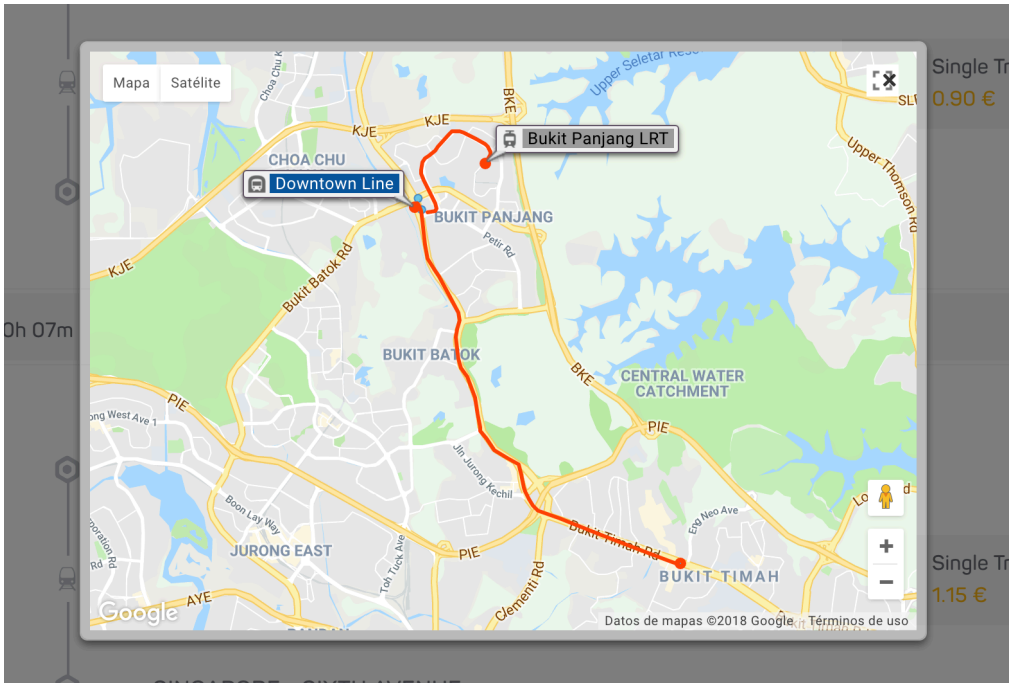

Figura 11. Ejemplo. Mapa de una ruta combinada.

Para mostrar la ruta en el mapa se hace uso de la Google Maps API. Gracias a ella se puede dibujar de forma cómoda y sencilla las rutas incluso teniendo en cuenta el tipo de transporte en el que se realiza. La llamada al mapa se realiza a través de Javascript y es la única modificación realizada en dicho lenguaje. A continuación se muestra el pseudocódigo que fue necesario introducir para que llevar a cabo esta funcionalidad.

```
// -- baolau.js
if(urban_site == true){
    map = new google.maps.Map($canvas[0], { // Inicialización del mapa
        zoom: 12,
        center: { lat: coordinates[0][0], lng: coordinates[0][1] },
    });
}

if(route_coordinates[3] == "Urban-train"){
    travel_mode = "TRANSIT"; // Transporte público
}else if(route_coordinates[3] == "Walking"){
    travel_mode = "WALKING"; // Andando
}else{
```

```
    travel_mode = "DRIVING"; // Por defecto
}
directionsService.route({
    origin: route_coordinates[i], // Origen de cada paso
    destination: route_coordinates[i + 1], // Destino
    travelMode: google.maps.DirectionsTravelMode[travel_mode] // Modo
```

Pseudocódigo 17. Representación de rutas urbanas en el mapa.

En el caso de que se esté accediendo a través de las páginas urbanas el mapa se inicializa centrado en las coordenadas de la primera estación y con un zoom determinado (más propio de un tamaño de ciudad y no de país). Pasada la inicialización, dependiendo del tipo de transporte del segmento que se va a dibujar en el mapa se determina el valor de la variable *travel_mode*. Este toma los valores de *transit* cuando se trata de transporte público, *walking* cuando se trata de un tramo andando y *driving* por defecto.

Más adelante se introducen dichos parámetros y el punto de origen y el de destino para que dibuje la ruta adecuadamente. Esta implementación, hace que las rutas urbanas tengan el trayecto verdadero que posee el tren e incluso aparecen etiquetas que informan sobre el color y el nombre de la línea de metro en cuestión. Esta forma de representar las rutas es muy cómoda ya que solamente son necesarios el punto de origen y destino, sin embargo, como se explica en el apartado de resultados puede dar algún problema en ciertas ocasiones.

Estos dos elementos del interfaz, la ruta desplegada y el mapa, son los únicos que se comparten en los dos tipos de páginas y son dependen del tipo de transporte que se esté representando. A continuación se explican los elementos gráficos que han sido expresamente diseñados para las páginas urbanas de Singapur y Bangkok.

Para finalizar con los resultados, las rutas se muestran de manera diferente cuando no están desplegadas. La siguiente figura muestra el resumen de una búsqueda realizada a través de la página urbana seguido de las rutas que ha obtenido el algoritmo de Dijkstra pero esta vez sin desplegar.

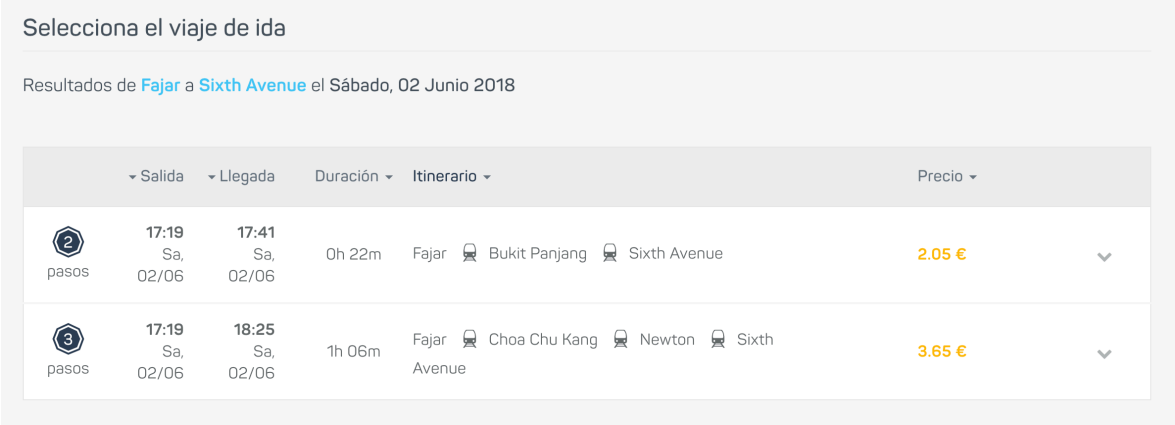

Figura 12. Ejemplo. Resumen de la búsqueda y rutas resultado plegadas.

Respecto al origen y el destino que aparece en la línea inicial, anteriormente aparecían las ciudades a las que pertenecen los nodos y en el sitio urbano son los nombres de las estaciones. Lo mismo ocurre con los nombres en las propias rutas ya que en el sitio urbano no se han escrito los nombres de las ciudades. De esta forma, es posible observar una descripción bastante detallada de la ruta sin necesidad de desplegarla.

Por último, un cambio que se ha comentado en la implementación de la lógica pero no se mostró su efecto es el número de pasos. En la página principal los pasos se cuentan por cada cambio de ciudad en la ruta mientras que en esta representación depende solamente del cambio de estación, es decir, los trasbordos.

Figura 13. Formulario de la página urbana de Singapur.

En esta figura se muestra el formulario de búsqueda de la página de Singapur. El formulario tiene varios cambios significativos con respecto al original:

- **Origen y destino**: Las entradas dónde se introducen el origen y el destino solamente deja seleccionar aquellos nodos pertenecientes a la red urbana de la ciudad y genera un error si la estación origen y destino es la misma.

- **Filtro de tarifas**: Para Singapur, en dónde se han almacenado dos tipos de tarifas para cada ruta, el formulario te deja seleccionarla antes de lanzar la búsqueda. En este caso, las posibilidades son la tarifa de un viaje sencillo y la tarifa de la tarjeta MRT.

- **Hora de salida**: Por último, un cambio importante tanto en la lógica como en el interfaz es la posibilidad de introducir la hora de salida a través de este campo. Por defecto este campo toma el valor de la hora actual en dicha ciudad.

Este formulario se ha introducido tanto en la página principal como en la de resultados, en dónde se puede cambiar la búsqueda que se hizo inicialmente. La correcta implementación, tanto del filtro de tarifas como el de la hora de salida, era un requisito necesario que se debía implementar.

Además de todos los cambios mencionados se diseñó ciertos elementos que debía tener la página de inicio de los sitios urbanos. Entre ellos, se introdujo un cambio en el icono de la página, añadiendo junto a él el nombre de la ciudad a la que se estaba accediendo. También se modificó la imagen de entrada, siendo ahora una foto de algún paisaje emblemático de la ciudad y por último se introdujo un mapa de la red de transporte público ferroviario. Este mapa se encuentra en una pestaña por si en un futuro se añaden otros transportes como los autobuses.

La siguiente imagen muestra una miniatura de la página principal de Singapur en dónde se pueden ver estos elementos.

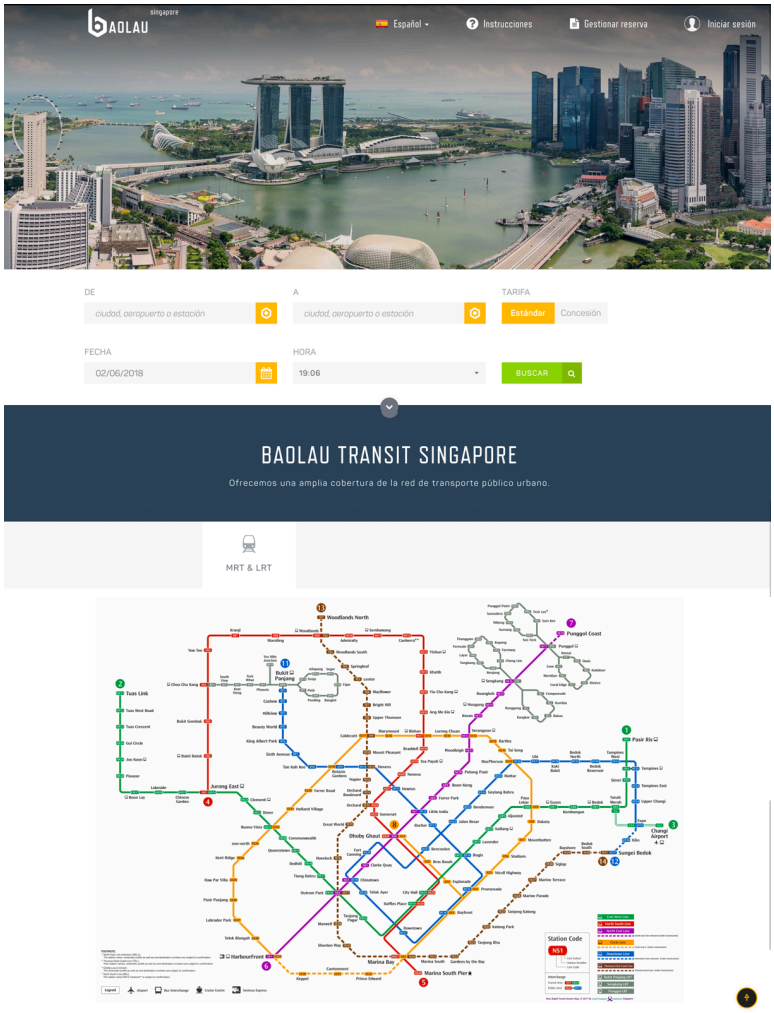

Figura 14. Miniatura de la página de entrada de Singapur.

Todos los cambios mencionados en este apartado se han tenido que repetir para distintos tamaños de pantalla dónde se muestren. Tener un diseño responsivo es muy importante ya que muchas de las reservas y accesos a la página se realizan a través de los dispositivos móviles de los usuarios. Como ejemplo a continuación se muestra la versión móvil de la página urbana de Singapur.

Figura 15. Versión móvil de la página urbana de Singapur.

Antes de finalizar con este apartado destacar que la página debe estar operativa en cinco seis idiomas distintos: español, inglés, chino simplificado, chino tradicional, vietnamita y japonés. Para ello, cada texto que se muestra por pantalla se ha almacenado en una variable que cuenta con variantes de ese mismo texto en los seis idiomas, de forma que cuando se accede con un determinado lenguaje solamente se muestra dicho idioma. Existe un fichero de idioma por cada idioma y por cada sección principal de la página.

Tanto las imágenes de los mapas cómo la de los logos de cada uno de los transportes han sido extraídos de las páginas oficiales. Las únicas modificaciones que se han realizado a estos ficheros son de tamaño y formato. Debido a que se ha extraído de las páginas oficiales estos mapas se encuentran muy actualizados. En el siguiente apartado se muestran los mapas de ambas ciudades y es posible observar las líneas que efectivamente se encuentran en construcción o planificadas para un futuro cercano.

Capítulo 6. ANÁLISIS DE RESULTADOS

En esta sección del documento se analizan los resultados tras llevar a cabo las tareas descritas en el apartado anterior en el tiempo disponible. En lo que refiere al proyecto, se analizará la medida en la que se han cumplido cada uno de los objetivos que se propusieron al inicio del proyecto. En primer lugar se encuentra el proceso de búsqueda de fuentes de información y extracción de datos. Respecto a la implementación es necesario analizar tanto la calidad de los resultados obtenidos como su representación en la página urbana y la principal, además de su integración con el resto de rutas. En los siguientes apartados se discuten estos puntos.

6.1 DATOS

Respecto a la extracción de datos, las fuentes de información encontradas y la forma de extraer los datos han sido muy positivo dadas las circunstancias. La búsqueda de datos no fue la ideal ya que se esperaba encontrar ciertas bases de datos o acceso a APIs de fuentes oficiales. El defecto de estas fuentes provocó un cambio de planes en lo que se refiere a la extracción de datos pero la solución desarrollada también cuenta con ciertas ventajas.

En primer lugar, al poder acceder a las fuentes de datos elegidas a través de varias APIs ha dado la posibilidad de generar unos scripts que automaticen el proyecto y no sea tan costoso como recolectar la información de manera manual. Sin embargo la principal ventaja de esta aproximación es la posibilidad de usar los mismos scripts para llevar a cabo la extracción de otros transporte públicos y de otras ciudades. Adicionalmente, dada la amplitud de la actividad de Google, es muy posible que se tengan los datos de prácticamente cualquier lugar.

Respecto a los scripts en particular, ninguno de ellos es excesivamente complejo y al ser independientes, facilita la obtención de diferentes atributos por separado. Todos ellos cuentan con mecanismos de detección de errores y son capaces de identificar dónde se ha

producido para que el usuario sea capaz de arreglar el problema con mayor facilidad. No se ha considerado necesario realizar pruebas de rendimiento de estos scripts ya que la mayoría de ellos son inmediatos dado el volumen de datos o dependen excesivamente de la velocidad de conexión con internet. En condiciones normales, el script que envuelve más peticiones puede llegar a tardar un minuto para extraer la información de toda una línea.

En el caso concreto de las dos ciudades de las que se pretendía obtener datos, se han logrado extraer todos los atributos de todas las rutas y estaciones de cada línea. Este éxito es principalmente a la base de datos actualizada que posee Google y que incluso en algún momento ha tenido registrada unas pequeñas obras en una estación de Singapur. Gracias a esta fuente de datos, se han logrado obtener un total de 157 nodos, 10 líneas y 4184 rutas. El siguiente mapa muestra la red completa de metro y tren ligero y de la cual se tienen toda su información (salvo de las estaciones que están en construcción).

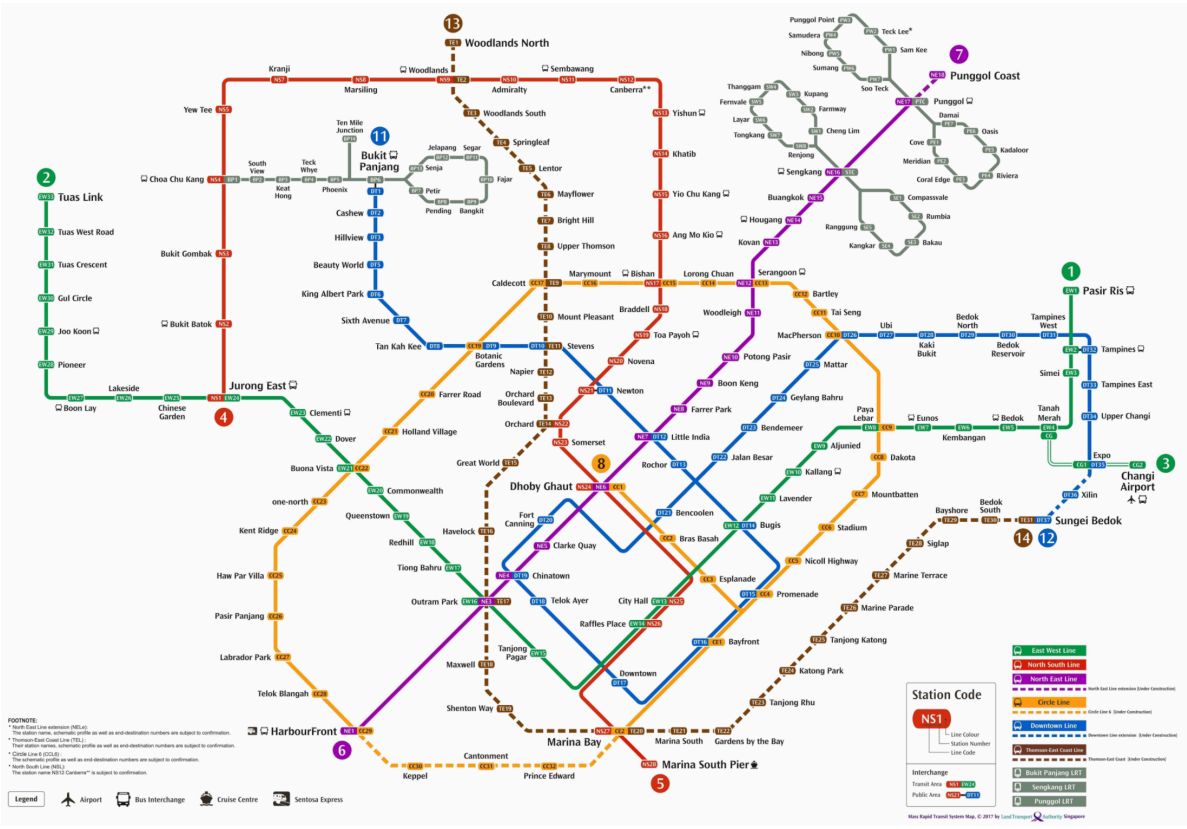

Figura 16. Mapa de Singapur. MRT y LRT.

Respecto a las rutas de Bangkok, el número total de estaciones almacenadas son 76, 5 líneas y 1319 rutas. El mapa de todas las líneas que se tienen en el sistema se muestra en la siguiente figura:

Figura 17. Mapa de Bangkok. MRT, BTS y ARL.

6.2 *INTERFAZ DE USUARIO*

Otro aspecto fundamental que debía cubrirse en la realización del proyecto era que el realizar las variaciones necesarias en el aspecto de las rutas urbanas con el fin de que el usuario las diferencia sin esfuerzo. Este aspecto, si bien no era complicado, era necesario definir y diseñar de qué modo y que información se debía mostrar de este nuevo tipo de rutas. Como resultado, se cree que el diseño implementado consigue realizar dicha función y ofrece al usuario la información necesaria sin llegar a sobrecargar dicha ruta. De este

modo, se ha logrado tener una rutas reutilizables tanto en la página principal como en las urbanas.

Respecto a las páginas urbanas de Singapur y Bangkok, ambas han logrado tener el aspecto adecuado. En primer lugar, el formulario de búsqueda, sin perder el estilo del de la página principal, añade la hora de salida y el filtro de tarifas, ambos fundamentales para la búsqueda de rutas urbanas. Quitando el resto de información de la página se ha dejado simplemente el mapa del que se tienen las rutas y ningún transporte público más para no confundir al usuario. En un futuro se añadirán más tipos de transporte y por tanto otros mapas en dicha sección.

Todos los cambios en el interfaz que se han realizado y están descritos en el apartado anterior han funcionado sin la detección de ningún problema. Aparte del aspecto estético y el diseño, al tratarse de una implementación no muy compleja, el interfaz como resultado no requiere de mayor análisis. Como prueba de funcionamiento se ha ido comprobando durante el desarrollo y al finalizar la implementación que no requería de ningún retoque o modificación y todo se mostraba según lo definido en el diseño.

Finalmente sólo queda destacar un resultado que no ha sido tan satisfactorio como puede llegar a ser y se trata de la implementación del mapa. Si bien la implementación ha sido muy sencilla y no ha sido necesario realizar grandes cambios debido a que se ha usado los mapas de Google, también se ha detectado un problema. La forma de testear la representación de las rutas ha sido probar muchas combinaciones en busca de un fallos llamativos.

El hecho de que solamente haya que pasar las coordenadas de origen y destino además del modo de transporte hace que sea el algoritmo de cálculo de rutas de Google el que determine el camino entre dichos puntos. Aunque en este caso las rutas siempre van a seguir una determinada línea de metro o tren, es posible que dependiendo de la distribución geográfica de las estaciones de dicha línea se consideré más eficaz salirse de la ruta espera y utilizar otro método de transporte público.

Realizando los test de funcionamiento del mapa se detectó que este comportamiento sólo aparecía en algunas combinaciones de la red de Singapur, debido a que a es una red muy condensada, con muchos cruces y con muchos autobuses disponibles. Para solucionar el problema, se implementó una opción en los parámetros de entrada de la API de Google Maps que indicara que se utilizaran los menos trasbordos posibles. De esta forma, se ha evitado este problema tan común encontrado en la red de Singapur.

Si bien el resultado no es el óptimo porque el resultado depende en gran medida de la API de Google Maps ha sido el utilizado debido a su simplicidad. De todos modos, cambiar esta forma de representación de rutas es una opción a considerar y se explicará en más detalle en el apartado de trabajos futuros.

6.3 MOTOR DE BÚSQUEDA Y RUTAS RESULTADO

Finalmente, el otro objetivo que se debía cubrir era que la información que mostrase la página fuera verdadera y lo más precisa posible. Sin tener en cuenta su representación, las rutas resultado se fueron testeando realizando variedad de búsquedas. Para facilitar el reconocimiento de posibles fallos, también se dejó de lado la integración de las rutas urbanas con la página principal y se probó en las páginas de Bangkok y Singapur.

Debido a la dificultad de comprobar la exactitud de los datos, se ha comparado tanto con los datos de Google como con la experiencia que han tenido los trabajadores de la empresa en dichos transportes. Lógicamente, dado que la información que forman las rutas se ha obtenido de la base de datos de Google, no debería haber ningún resultado inesperado al menos en las rutas directas. De todos modos se llevaron a cabo muchas combinaciones posibles y no se encontró fallo alguno.

Una de las ventajas de tener los datos almacenados en local, aparte de no depender de un tercero y el coste de la licencia, es poder hacer uso del algoritmo de rutas ya utilizado en la página principal. Es en este aspecto en dónde cobra más sentido realizar las pruebas y compararlas con las mostradas por Google. Cómo valor añadido, el algoritmo usado en

Baolau muestra más de una ruta y con posibilidad de ordenarla en función de las preferencias del usuario (rutas de menos duración o de menos coste).

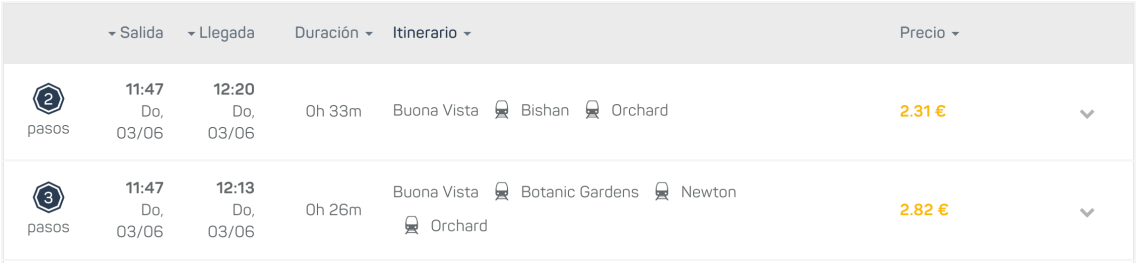

Figura 18. Ejemplo. Rutas combinadas para distintas preferencias.

En el ejemplo se puede observar como el algoritmo de Dijkstra ha obtenido dos rutas que interconectan los mismos origen y destino pero que sin embargo tienen distintas duraciones y precios. De este modo el usuario es capaz de ordenar en función de sus preferencias las rutas extraídas por el algoritmo. En este caso concreto, se tiene una ruta más larga pero de menor precio que la segunda opción, en la que se ahorra tiempo pero el precio es mayor.

En ese sentido hay que destacar que el primer resultado de rutas combinadas es muy similar al obtenido con Google y el experimentado por los trabajadores. Realizando estas pruebas se detectó que el máximo número de trasbordos que se tenían que dar para interconectar dos puntos cualquiera eran tres, tanto para la red de Bangkok como de Singapur. Esto es importante a nivel de rendimiento ya que determina el tamaño de la petición que se realiza a la base de datos.

El algoritmo actual no tiene ningún problema con esta cantidad de información pero puede resultar en un problema de tiempo de ejecución si se aumentan las rutas obtenidas con la petición. Un caso en el que es necesario hacer una petición mayor es cuando se añaden las rutas urbanas a la red principal y es en esos casos en dónde es posible observar que la velocidad con la que se obtienen las rutas disminuye de manera notable.

Aunque se aleje del alcance de este proyecto, aumentar el número de saltos en dicha petición implica obtener muchas más rutas de manera exponencial y la capacidad de que

sean manejadas con velocidad depende mucho de algoritmo de cálculo. Si bien Dijkstra no es el más lento cuando aumentan considerablemente el número de rutas, hay otros que se comportan mejor. Afortunadamente, como un proyecto paralelo a este, se ha realizado la implementación de nuevo algoritmo cuyo rendimiento no baja tanto. Hasta entonces, la implementación total con el resto de rutas de la red de Baolau no estará completa al nivel de rendimiento que se desea.

Quitando este pequeño inconveniente, la integración de las rutas urbanas en la página principal, el cual era uno de los objetivos principales, también se ha llevado a cabo con éxito. Las pruebas realizadas en ese aspecto ha sido comprobar la conexión entre los nodos de una misma ciudad. Lógicamente, como el número de nodos que tenían para ambas ciudades no es muy grande, ha sido posible comprobar todas las combinaciones y estas se han producido perfectamente.

En la mayoría de los casos se ha introducido una ruta en la que se interconectan andando el nodo urbano con el nodo ya existente en la red principal. En la mayoría de los casos la estación de metro se encuentra al lado de la estación del otro tipo de transporte y este recorrido andando se puede realizar en duraciones de entre 1 y 10 minutos. Además de realizarse todas las conexiones, la forma de representarse junto con el resto de tramos en un ruta combinada ha sido exactamente como se deseaba.

A continuación se muestra una captura de pantalla de la integración de una ruta urbana con una ruta combinada obtenida de la página principal. En este ejemplo también es posible observar la forma de representación de las rutas andando que interconectan las estaciones de ambos entornos.

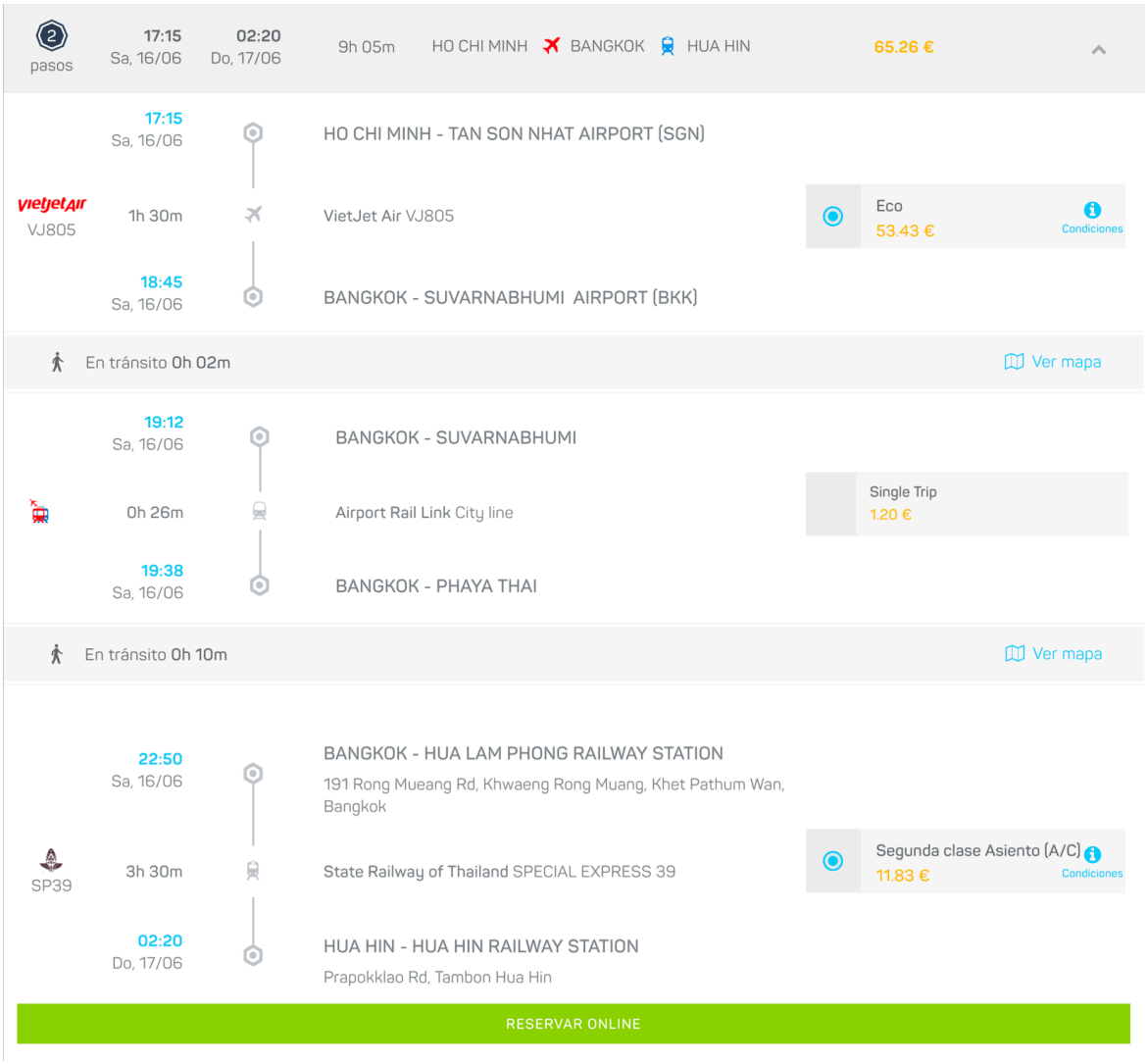

Figura 19. Integración de rutas urbanas en la página principal.

En definitiva, tanto en las rutas directas cómo en las combinadas e independientemente de si se producen en las páginas urbanas o en la principal junto con las originales, todas las combinaciones proporcionan al menos un resultado preciso y consistente. Estos resultados, junto con una representación diseñada expresamente para proporcionar el servicio requerido hace que el desarrollo del proyecto se pueda considerar un éxito.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Para finalizar este documento, en este apartado se explican las conclusiones a las que se ha llegado con la realización de este proyecto y se enumeran varias posibles modificaciones o trabajos futuros relacionados con este proyecto. Si bien no es necesario realizar ninguna modificación adicional para que funcione correctamente lo implementado, si se considera conveniente llevar a cabo los siguiente trabajos. El objetivo de estos trabajos es obtener un producto más completo y con más posibilidades de cara a un futuro. Respecto a los datos se tienen las siguientes mejoras.

- **Añadir más ciudades**: Al planificar este proyecto se pensó en las dos ciudades comentadas a lo largo del proyecto porque eran las que tenían una red de metro considerable y que se encontraba en funcionamiento. Sin embargo, Baolau también opera en otras ciudades en las que se tiene un transporte público similar. Con el paso del tiempo y tras la evaluación de la implementación realizada, si esta es positiva sería conveniente introducirla para más ciudades a través de los scripts de extracción de datos programados en este proyecto ya que no tendría prácticamente ningún coste para la empresa.

- **Añadir otros transportes públicos urbanos**: Siendo la red de autobuses el otro gran transporte público urbano en muchas ciudades, sería aconsejable introducir dichos transportes en las redes urbanas de las ciudades. La ventaja principal que conlleva este futuro trabajo es que se puede reutilizar la mayor parte del código de los scripts de extracción de datos y de la implementación de las páginas.

- **Actualización de los datos urbanos**: Para finalizar los trabajos futuros respecto a los datos, es conveniente actualizar de vez en cuando la información almacena en la base de datos por si ha sufrido algún cambio importante la red de metro. Esto se puede llevar a cabo ejecutando de nuevo los scripts de extracción de datos. Sería conveniente repetir este proceso siempre que se sepa que se han producido grandes

cambios en la red, que algún tramo se encuentra en obras o que han modificado considerablemente la frecuencia, horario o tarifa de las rutas.

Respecto a la implementación de cambios en el interfaz y algoritmo se han identificado los siguiente trabajos futuros:

- **Enlaces a las páginas urbanas**: Para completar la experiencia de usuario y que les sea más fácil a los usuarios encontrar las páginas urbanas, sería conveniente introducir una serie de enlaces o botones cerca de las rutas cuyo destino fueran una de las ciudades con una red de transporte urbano. De este modo el usuario sigue utilizando Baolau para trasladarse por la ciudad durante su estancia.

- **Petición a la base de datos y algoritmo**: Se debe determinar la solución óptima respecto al tema mencionado en el apartado anterior y que puede ser la causa de grandes problemas si aumenta el número de rutas de la red. Para ello, es conveniente implementar el nuevo algoritmo capaz de hacer uso de más rutas y así no quedarse corto con una petición a la base de datos escasa.

- **Representación de las rutas**: Aunque se haya decidido mostrar de esa forma los resultados, es posible que finalmente no sea el adecuado. Este problema puede darse con más facilidad si una ruta urbana con muchos trasbordos forma parte de una ruta combinada de la página principal. En ese caso es posible que la información del entorno urbano ocupe demasiado y distraiga al usuario de la ruta principal. Sería conveniente tener en cuenta este posible efecto del diseño actual y mejorarlo si fuese necesario.

- **La representación gráfica de las rutas**: Al representar las rutas en el mapa es posible que en algún momento se produzca el problema mencionado en el apartado anterior. Si esto se produjese no habría forma de solucionarlo con otro parámetro adicional de la API de Google Maps y una posible solución sería almacenar las coordenadas que forman la ruta y representar dichos puntos. El único problema de esta solución sería que a nivel de implementación sería muy costosa ya que habría

que modificar todas las rutas de la base de datos y en algunos casos encontrar dichas coordenadas puede ser complejo.

Habiéndose explicado los futuros trabajos, a continuación se explican las conclusiones del proyecto. En primer lugar, es necesario destacar la importancia que ha tenido la motivación del proyecto para hacer posible su implementación. Al tratarse de añadir una funcionalidad que podía proporcionar grandes beneficios en términos de experiencia de usuario, ha motivado la correcta implementación del proyecto.

Además de la motivación, tantos los objetivos marcados al principio del proyecto como la buena planificación de las tareas y el alcance del mismo han producido que la implementación se finalice a tiempo y correctamente. En ese aspecto, también ha sido fundamental la comunicación directa que se ha dado a lo largo del desarrollo con el equipo de trabajo. Estos son algunos de los motivos que reafirman positivamente la metodología seguida en este proyecto.

La implementación, llevada a cabo sin ningún problema que haya dificultado la planificación inicial, ha dado lugar a unos resultados muy satisfactorios. Por un lado, tanto el rendimiento del motor de búsqueda como la exactitud de los resultados es verdaderamente positivo. Una situación similar se encuentra en el interfaz de usuario, en dónde la página principal integra de manera adecuada las nuevas rutas urbanas. Finalmente, el diseño y funcionalidades implementadas en las páginas urbanas de Bangkok y Singapur completan el proyecto otorgando un valor añadido a los nodos y rutas urbanas.

Capítulo 8. BIBLIOGRAFÍA

[1] Página Oficial de Baolau: https://www.baolau.com

[2] Uber, Wikipedia: https://es.wikipedia.org/wiki/Uber

[3] Grab, Wikipedia: https://en.wikipedia.org/wiki/Grab_(company)

[4] Página Oficial de PHP: http://php.net/

[5] Google, Wikipedia: https://es.wikipedia.org/wiki/Google

[6] Google Maps: https://www.google.es/maps

[7] Página Oficial de Python: https://www.python.org/

[8] Página Oficial de Bitbucket: https://bitbucket.org

[9] Página Oficial de Sourcetree: https://www.sourcetreeapp.com/

[10] Página Oficial de MAMP: https://www.mamp.info/en/

[11] Página Oficial de Visual Studio Code, Microsoft: https://code.visualstudio.com/

[12] Javascript, Wikipedia: https://es.wikipedia.org/wiki/JavaScript

[13] SQL, Wikipedia: https://es.wikipedia.org/wiki/SQL

[14] HTML, Wikipedia: https://es.wikipedia.org/wiki/HTML

[15] Documentación de la Google Places API: https://developers.google.com/places/?hl=es-419

[16] Documentación de la Google Directions API: https://developers.google.com/maps/documentation/directions/intro?hl=es-419

[17] Documentación de la Google Geocoding API: https://developers.google.com/maps/documentation/geocoding/start?hl=es

[18] Java, Wikipedia: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[19] Página Oficial de mySQL: https://www.mysql.com/

[20] Git, Wikipedia: https://es.wikipedia.org/wiki/Git

[21] Mac Os, Wikipedia: https://es.wikipedia.org/wiki/MacOS

[22] Microsoft Windows, Wikipedia: https://es.wikipedia.org/wiki/Microsoft_Windows

[23] Página Oficial de Apache: https://httpd.apache.org/

[24] Página Oficial de Perl: https://www.perl.org/

[25] MRT Singapur, Wikipedia: https://en.wikipedia.org/wiki/Mass_Rapid_Transit_(Singapore)

[26] LRT Singapur, Wikipedia: https://en.wikipedia.org/wiki/Light_Rail_Transit_(Singapore)

[27] MRT Bangkok, Wikipedia: https://es.wikipedia.org/wiki/Metro_de_Bangkok

[28] BTS Bangkok, Wikipedia: https://es.wikipedia.org/wiki/Metro_A%C3%A9reo_de_Bangkok

[29] ARL Bangkok, Wikipedia: https://es.wikipedia.org/wiki/Airport_Rail_Link

[30] JSON, Wikipedia: https://es.wikipedia.org/wiki/JSON

[31] MyTransport: https://www.mytransport.sg/content/mytransport/home.html

[32] Página Oficial de LTA: https://www.lta.gov.sg/content/ltaweb/en.html

[33] Data.gov.sg: https://data.gov.sg/

[34] Bangkok MRT: http://www.bangkokmrt.com/

[35] Bangkok BTS: http://www.bangkokbts.com/

[36] Transit Bangkok: https://www.transitbangkok.com/mrt.html

[37] Railway Tailandia: http://www.railway.co.th/checktime/checktime.asp