Práctica 1: Integración de Datos

Datos Masivos y Encadenados

Alberto Royo Valle (100481813)
Alejandro Sanz Fernández (100490797)
Arash Kazemi Díaz (100384030)
Julian Quintero Bejarano(100460445)
Pablo González Tamames (100483806)
Pablo Peñuela Rodríguez (100488691)

Computer Science and Engineering Deparment

Índice

1. Resumen Ejecutivo

El objetivo del caso práctico es trabajar en un escenario de integración de datos, diseñando una arquitectura de software e implementándola en una prueba de concepto.

En este proyecto concretamente, se buscará crear un servicio que sea capaz de ofrecer al usuario un sistema integral de búsqueda y selección de destinos para realizar viajes, incluyendo diversas recomendaciones, tanto generales, como adaptadas a cada usuario. El objetivo sería proveer al usuario de toda la información relevante que le ayude a tomar la decisión de cuál es su lugar ideal para viajar en un momento determinado y, además, poder reservar los vuelos desde la propia aplicación.

Por tanto, la principal ventaja que aporta esta aplicación es la integración, en un mismo sistema, de la búsqueda de destinos, comparador de vuelos, recomendaciones, planes de destinos e incluso mapas interactivos para poder planificar mejor el viaje. Y, por supuesto, en última instancia, la adquisición de los vuelos. Todo eso provee al usuario un entorno cómodo para la planificación, sin tener que andar navegando en decenas de páginas a la vez, consultando foros, páginas de aerolíneas, etc. y, además, la recomendación de sitios específicos que sera de gran ayuda para los más indecisos.

Para desarrollar esta idea, es necesario crear una base de datos que contenga información de vuelos y aeropuertos, localizaciones, actividades, condiciones climáticas, información de hoteles y opiniones de viajeros. Para acceder a estos datos se utilizará un *Data Warehouse* con una actualización diaria de los datos que no requieran ser actualizados a cada momento y, por otro lado, los datos que cambien con una frecuencia alta serán descargados en el momento que el cliente realice las consultas.

En esta memoria se explica con detalle en qué consiste este sistema y, además, se presenta una prueba de concepto en la que se implementan algunas de las funciones mencionadas previamente. También se dan las indicaciones para poder usar el programa realizado y subido a Aula Global.

2. Análisis y diseño del sistema propuesto

En esta sección se explican todas las herramientas y fuentes de datos que forman el sistema ideado para dar solución al problema previamente explicado.

En primer lugar, en la sección 2.1 se presenta la arquitectura general del sistema, introduciendo y explicando las fuentes de datos necesarias para el desarrollo de este.

En segundo lugar, en la sección 2.2 se explica detalladamente el modelo de integración de datos, explicando las metodologías usadas para integrar las fuentes, así como las herramientas usadas.

Por último, tras explicar el modelo de integración, en la sección 2.3 se explica la arquitectura funcional del sistema, describiendo el manejo de información y la conexión entre las distintas capas que lo forman.

2.1. Descripción y arquitectura general del sistema

El objetivo de este proyecto es el diseño y la implementación de un sistema que integre todo lo necesario para elegir y planificar un viaje de forma óptima, de forma que la experiencia para el cliente sea la mejor posible, simplificando al máximo los trámites que este deba realizar, así como aportando las ideas que más se ajusten a cada persona.

La idea es que un paciente entre a la aplicación pensando en realizar un viaje, pero con dudas de fechas y/o destinos. Desde el sistema diseñado se accederá a la localización del cliente para ver cuál es el aeropuerto más cercano y las opciones que hay desde ahí en diferentes periodos que el cliente plantee. Los destinos también serán personalizados, ya que se ajustarán al tipo de viaje que el cliente desee hacer (si es con la familia, con amigos, viaje de negocios...). También se ajustará a otras preferencias que el cliente pueda tener, como clima, seguridad o planes concretos. Una vez que el cliente se decida, podrá reservar los vuelos y el hotel usando esta misma aplicación, sin necesidad de tener que andar navegando en distintas webs y comparando precios.

La aplicación no requerirá de ningún conocimiento previo, ya que la interfaz será atractiva visualmente y se detallará al cliente todos los pasos que debe realizar de forma detallada.

Para llevar a cabo estos objetivos se usarán las siguientes fuentes de datos:

- Nearest Airport: Esta fuente de datos nos proporcionará aquellos aeropuertos y ciudad más cercanos, haremos uso de su API para extraerlos, transfórmalos y posteriormente cargarlos en memoria. Dicha información es dinamica y depende de la localización actual por lo que lo ideal es mantenerla en un esquema virtual.
- AviationStack: Es una de las fuentes principales para el proyecto, de esta plataforma se podrá extraer un amplio conjunto de datos de aviación, incluido el estado de vuelos en tiempo real, vuelos históricos, horarios, rutas de aerolíneas, aeropuertos, aeronaves, etc. La extracción de estos datos será por medio de la API o en su defecto accediendo directamente a la base de datos con el método de pago. Esta fuente forma parte del esquema virtual precisamente por la dinámica de los vuelos y la precisión requerida para el funcionamiento de la aplicación.
- Skyscanner: Esta fuente de datos servirá como un comparador de precios el cual nos proveerá de las mejores ofertas de viaje al destino deseado por el usuario. Para acceder a los datos, se hará uso de su API de la que se extraerán las diferentes ofertas del vuelo relacionado con el destino, es por esto que esta fuente debe de tener una integración virtual.
- Aemet Web que da información sobre las condiciones meteorlógicas de los diferentes destinos.
- Weather Underground API Esta fuente de datos esta desarrollada con más de 250,000 estaciones meteorológicas (la más grande de este tipo) y nos brinda pronósticos locales basados en puntos de datos meteorológicos reales. Se usará esta fuente de datos por medio de una API que proporcionará información meteorológica en tiempo real del posible destino. Dada la volatilidad de los pronósticos meteorológicos es necesario usar estos datos dentro de un enfoque virtual.
- Weather Spark:. Esta fuente de datos proporcionará información sobre el clima, para viajes programados en fechas lejanas. La extracción de estos datos se realizará por medio de técnicas de web scraping. Esta fuente de datos proporciona informes meteorológicos con información completa del clima típico de cualquier lugar específico. Es un excelente recurso para planificar viajes y eventos.
- Tripadvisor: Mundialmente conocida y con más de 8 millones de lugares entre hoteles, restaurantes, atracciones y puntos de interés, esta fuente de datos será de gran utilidad principalmente por su mas de 1 billón de reseñas sobre sitios de interés. La extracción de la información será por medio de su API, esta será parte del esquema virtual.
- Booking: Esta fuente de datos nos proporcionará información relacionada a los precios y disponibilidad en tiempo real de alojamientos, descripciones de propiedades, información de instalaciones, nombres de ciudades, tipos de hoteles y otros elementos que cambian con frecuencia, por lo tanto esta fuente de datos se contempla en el esquema virtual.
- Google maps: Esta fuente de información se utilizará para realizar la búsqueda de aeropuertos cercanos, proporcionandolla mejor ruta al aeropuerto seleccionado.
- Índice de la Calidad del Aire: Esta fuente de datos nos proporcionará información sobre la calidad del aire en los 5 mejores destinos para el usuario. Esta fuente de datos forma parte del esquema virtual ya que el índice proporciona datos en tiempo real.
- Airport database. Esta fuente proporciona una base de datos de todos los aeropuertos y sus conexiones.
- World Coastal Cities, Coastal settlements of Mediterranean sea, ,Coastal settlements America: Estas fuentes de datos nos proporcionará información sobre las ciudades costeras mas populares del mundo, ciudades costeras del mar mediterráneo y ciudades costeras en América del Norte respectivamente. Esta, al ser información estática y relativamente invariante se trabaja bajo un esquema de DWH y cuya periodicidad de carga es anual.
- Airport codes: Esta fuente de información proporciona una lista de los aeropuertos del mundo con codigos IATA (la Asociación Internacional de Transporte Aéreo) que servirán como identificadores, también nos proporcionará información con respecto al tipo de aeropuerto (pequeño, mediano, grande), ciudad, país y

coordenadas de ubicación y nos permitirá identificar aeropuertos que se encuentren cerrados. Estos datos pasaran por la ETL cada 3 meses.

- IPC: Esta fuente de datos nos proporcionará información sobre la economía del lugar de destino, permitirá al usuario encontrar aquellos lugares en donde es mas y menos rentable viajar. Esta fuente de información tendrá una periodicidad de carga de 1 año.
- Base de datos del Gobierno de España: Esta fuente de datos nos proporcionará un histórico del grado de ocupación media por apartamentos en fin de semana por provincias y meses, que enriquecerá la visualización por parte del usuario para obtener información sobre los niveles de ocupación del destino. Dicha extracción será directa al descargar cada año el archivo .JSON, además de esto, dicho archivo será sometido a transformación y carga en el DWH.
- Safety Index: Esta fuente de datos proporcionará los índices de seguridad del destino, esto, al igual que otras fuentes, enriquecerá la visualización del usuario mostrándole aquellos destinos con mejor seguridad con respecto a los niveles de criminalidad. Este índice se extraerá por medio de técnicas de web scraping y se actualizará cada año en el DWH.
- unhabitat.org:: Esta fuente de datos proporcionará información sobre la proporción media de espacio publico y zonas verdes en cada una de las ciudades, esta información enriquecerá la visualización de los posibles destinos del usuario, esta información se cargará con en periodos de 6 meses.

2.2. Descripción de la integración de datos

En esta sección se explicará la metodología usada para integrar las fuentes de datos que se han explicado en la sección 2.1. Para ello, se propone un sistema híbrido (representado en la figura 1), en el que integramos dos tipos de enfoque distintos:

- Por un lado, un enfoque de integración **virtual**, haciendo uso de *Wrappers*, que son programas que se comunican con las fuentes de datos, enviando consultas a las fuentes y realizando unas transformaciones básicas de estos. Este esquema se usará para las fuentes de datos que varían rápido. La principal ventaja de usar el esquema virtual es la capacidad de tener acceso en tiempo real a los datos que son más volátiles.
- Por otro lado, un enfoque de **almacenes de datos** (*Dataearehouse*). Este esquema se usará para datos que sean más estables y varíen menos con el tiempo, tales como el índice de seguridad, los datos sobre ocupación de viviendas del gobierno... En este esquema no se hará uso de *wrappers*, sino de un proceso de **Extracción, Transformación y Carga** (**ETL**, por sus siglas en inglés). Este proceso se encargará de extraer los datos de las fuentes y los cargará en un almacén de datos con una cierta periodicidad (más alta que la del esquema virtual). Estas herramientas limpiarán, agregarán y transformarán valores almacenados, que posteriormente al integrarse con los datos de salida del esquema virtual se presentarán en forma de información relevante para el usuario.

2.3. Arquitectura Funcional

La arquitectura funcional del sistema (representada en la figura 2 se podría dividir en los siguientes tres niveles: nivel de **servidor de datos**; nivel de **servidor de aplicaciones** y nivel de **clientes**.

Por un lado, hay dos tipos de servidores, los de aplicaciones y los de datos. El de aplicaciones será el encargado de proporcionar la interfaz gráfica al usuario con la información extraída del servidor de datos. Por tanto, el servidor de datos se encarga de recabar todos estos datos y proveerlos al servidor de aplicaciones. Estos datos los saca de los *wrappers* mencionados previamente y también se encarga de realizar consultas periódicas a las fuentes de datos del esquema *datawarehouse*.

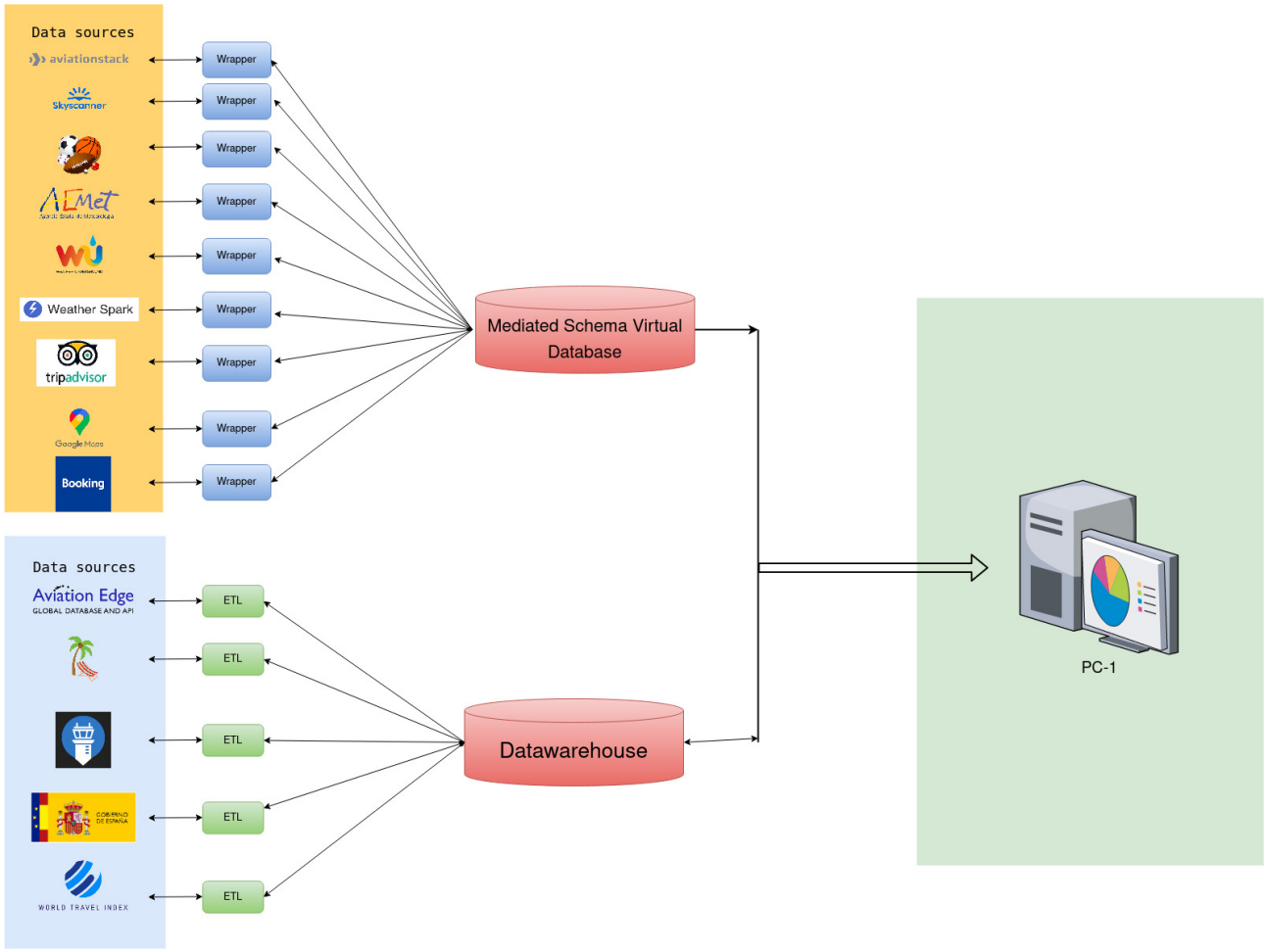


Figura 1: Representación de la integración de las distintas fuentes de datos en el sistema.

2.3.1. Nivel de servidor de datos

Este nivel se encarga de realizar toda la gestión de los datos que lleva a cabo el sistema. De este nivel forman parte el servidor de datos, la base de datos y los *wrappers*.

La base de datos será desarrollada usando **MySQl** y en ella se volcarán todos los datos que se almacenan en el *datawarehouse*, es decir, los que se consultan de forma periódica y necesitan ser almacenados.

Por otro lado, el servidor se encargaría de gestionar la base de datos, incluyendo en sus funciones tanto la realización de consultas a esta como de mantener los datos que están incluidos con el nivel de actualización requerido.

2.3.2. Nivel de servidor de aplicación

Este servidor será desarrollado con **Angular** y su función principal consistirá en mostrar de forma rápida y sencilla la información que requiera el cliente. Para esto, realizará peticiones que el cliente requiera al servidor de datos. Resumidamente, su función se puede ver como la de mediar entre el cliente y el servidor de datos.

2.3.3. Nivel de clientes

Este nivel está formado por los usuarios de la aplicación. El acceso a esta aplicación sera vía web, por tanto los clientes podrán acceder desde cualquier dispositivo que disponga de navegador, tanto móviles, como ordenadores, etc.

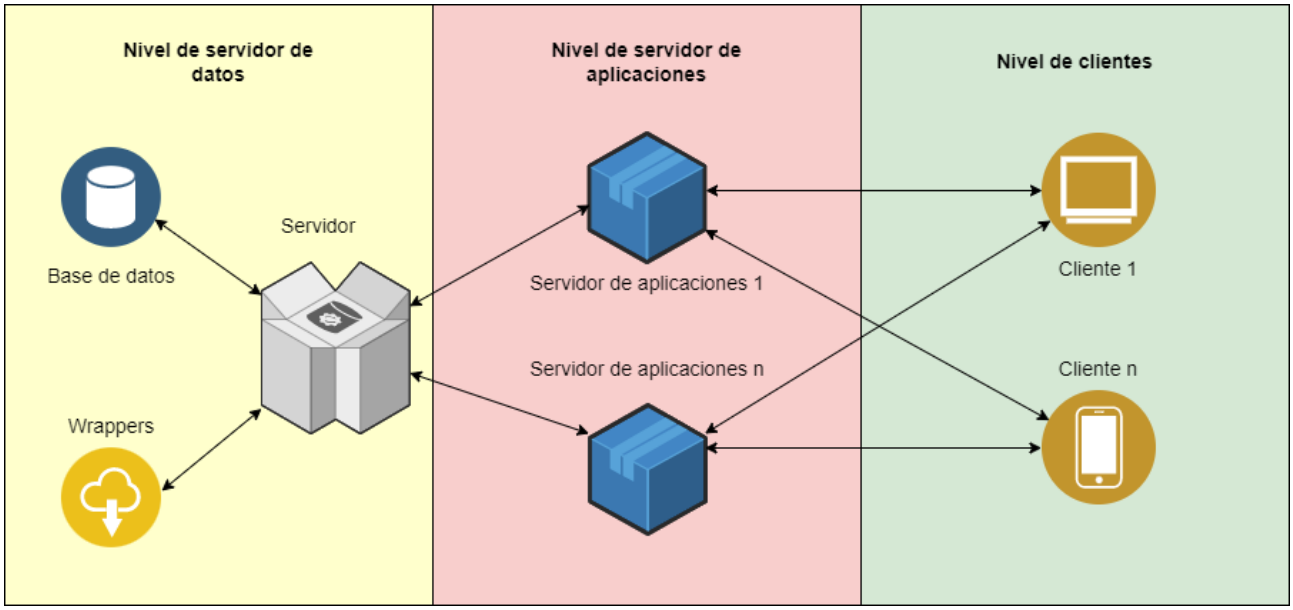


Figura 2: Arquitectura funcional del sistema.

2.4. Descripción del sistema en funcionamiento

PASO 1: PROPORCIONAR DESTINOS:

Lo primero será encontrar el **aeropuerto más cercano** para el usuario utilizando la API Nearest Airport. Con la información del aeropuerto de origen accederemos a nuestra fuente principal: Airport database. Esta fuente proporciona una **base de datos de todos los aeropuertos** del mundo así como de las rutas entre estos aeropuertos. por lo que la extracción de datos se realizará con un ETL. También se pueden extraer las compañías con las que trabajan estos aeropuertos (por si el usuario tiene algún tipo de preferencia por compañías "premium" o "lowcost"). A cada **aeropuerto se le asocia un código** y a través de este código, se puede obtener la ciudad de destino a través de Airport codes, que también nos indica si el aeropuerto es grande o pequeño.

PASO 2: BUSQUEDA DE POSIBLES PLANES A UTILIZAR COMO FILTRO PARA ELEGIR DESTINO:

Otra fuente que se integrará de forma materializada será **Tripadvisor**: Empresa que proporciona reseñas de contenido relacionado con viajes. La extracción de datos se realizará empleando su API. Los datos de Airport database nos servirán como entrada para extraer datos sobre los planes/monumentos/lugares mas representativos de nuestros hipotéticos destinos. Esta información es de gran utilidad ya que nos permitirá encontrar los destinos mas interesantes según los parámetros seleccionados por el usuario. Estos datos junto con los de Airport Database nos permitirán mostrar una pequeña lista de sugerencias con una serie de planes que puedan ayudar al usuario a decantarse por una opción u otra.

El usuario tendrá capacidad de elegir una **serie de opciones o filtros** sobre los planes en los destinos. Para ello utiliza tanto información de Tripavisor como de otras fuentes heterogeneas:

- Otra cosa que podrá elegir el usuario es si quiere ir a **zonas de costa**. Se emplearán fuentes como World Coastal Cities , Coastal settlements of Mediterranean sea ,Coastal settlements America, para determinar qué destinos de los ofrecidos por Airport database tienen playa.
- Otro criterio que puede ser determinante tiene que ver con la información de los mejores planes disponibles en su destino. Además de la API de tripadvisor se sacará información de próximos **eventos deportivos** en la zona empleando a la API de GoalServe

- Uno de los posibles criterios de búsqueda que puede elegir el usuario seria una búsqueda filtrada según el **índice de seguridad del destino** Índices de seguridad del destino: Safety Index se extraerán estos índices sobre los posibles destinos(extraídos de Airport database). Esta información se almacenará en una base de datos que se actualizará con cierta periodicidad.

PASO 3: UNA VEZ CONOCIDOS LOS MEJORES DESTINOS ⟶ BUSCAMOS VUELOS/HORARIOS/PRECIOS

Una vez conocidas las mejores opciones de viaje se emplearán AviationStack y Skyscanner (fuentes integradas en esquema virtual): Estas 2 fuentes de datos nos permitirán checkear los horarios y precios de vuelo a los destinos sugeridos. Ambas tienen API por lo que la extracción de información se realizará mediante el uso de estas. Con esa información el usuario podrá elegir una ordenación por preció para obtener los 5 mas baratos o los 5 mas caros etc... Estas dos fuentes se tratarán en un esquema virtual puesto que se trata de información cambiante (los precios y horarios de vuelo cambian de forma rápida).

PASO 4: ENRIQUECER LA CAPA DE VISUALIZACIÓN:

Una vez tengamos esa lista con los mejores destinos según los criterios del usuario es momento de enriquecer la capa de visualización. Esta sección trata de dar información accesoria sobre esos 5 mejores destinos. De esta forma facilitamos la decisión de elegir un viaje u otro.

- Se podrá mostrar información sobre los vuelos sugeridos y su duración del vuelo, la cual extraemos con la API Aviation Edge: Flight distance API en un esquema virtual.
- Se hará uso de Google Maps empleando su API para sacar la ruta de nuestra localización actual al aeropuerto de salida. También se podría utilizar para buscar en los destinos una serie de empresas de **alquiler de coches**.
- Se mostrará información sobre el **tiempo meteorológico** Weather Underground API (para fechas de viaje cercanas) y/o información sobre el **clima**: Weather Spark (para viajes programados en fechas lejanas). Esta información puede ser crucial para el usuario a la hora de elegir destino. La extracción de datos de Weather spark se realizará con un scrapper mientras que la de Eather Undergound se hara uso de una API.
- Se mostrará una valoración de la **calidad del aire** (en los 5 mejores destinos) Buscador calidad del aire Buscador calidad del aire o la cantidad de **zonas verdes en la ciudad** unhabitant.org. La extracción de información de esta fuente se realizara utilizando el buscador de la fuente y un scrapper para extraer los datos de los destinos (esquema virtual).
- En esta etapa final se podría mostrar el acceso a **paginas web de reservas** con las fechas seleccionadas por el usuario Booking . Aquí puede que sea de utilidad un **índice de ocupación de hoteles** y apartamentos Base de datos del Gobierno de España (el cual se almacena en un JSON).
- También se mostrará un breve **resumen de los destinos** que será extraído gracias a un scrapper de la Wikipedia del lugar. Si fuera necesario un **cambio de moneda**, la fuente de datos Conversor de Divisas dispone de una API la cual nos permite extraer información sobre los valores de las divisas.

3. Prueba de concepto

En esta sección se presenta la prueba de concepto realizada sobre el esquema planteado en los apartados anteriores. El objetivo es el de ejemplificar y plantear de forma sencilla el objetivo y la utilidad del sistema propuesto en los apartados anteriores.

3.1. Alcance

Como requisito para la prueba de concepto se deberán integrar al menos **dos** fuentes de datos y desde el punto de vista de la integración funcional será necesario integrar al menos **un servicio de terceros**. Es decir

- 2 Fuentes de datos
- 1 Servicio de terceros

Como se ha indicado en la descripción del proyecto se lleva a cabo un esquema de integración híbrido. La prueba de concepto de este proyecto es ejemplificar a pequeña escala nuestra idea de una aplicación de viajes que sugiera una serie de destinos.

El flujo de trabajo llevado a cabo en la implementación de este proyecto es el siguiente:

1. Obtención de los datos: Destinos, aeropuerto de salida
2. Toma de decisiones sobre que tipo de destinos quiere el usuario y filtrado de los datos obtenidos (para quedarnos con las partes relevantes).
3. Capa de visualización con una serie de sugerencias.

3.2. Objetivos

Se limita el proyecto original a las siguientes fuentes de datos:

- Aena (DATAWAREHOUSE)
- Wikipedia (DATAWAREHOUSE)
- Travelpayouts Data:APITravelpayouts (VIRTUAL)
- Tripadvisor (VIRTUAL)
- API Google maps (como servicio externo) (VIRTUAL)
- API Nominatim (VIRTUAL)

A grandes rasgos se restringe el aeropuerto de salida al Aeropuerto Adolfo Suárez Madrid-Barajas y los destinos a capitales europeas. La sugerencia de los mejores destinos se realizará a partir de los vuelos mas económicos. Se mostrará también información accesoria que ayude al usuario de la aplicación a decidirse por un destino u otro (planes que hacer, mapas con recorridos ...). Todo el proyecto fue implementado en el lenguaje de programación Python

3.3. Esquema/Arquitectura de la prueba de concepto implementada

Se mantiene la idea de una arquitectura híbrida entre el Esquema Datawarehouse y el Esquema virtual, que se puede ver en la figura 3

Para la sección de la capa de visualización se ha empleado Jupyter Notebooks

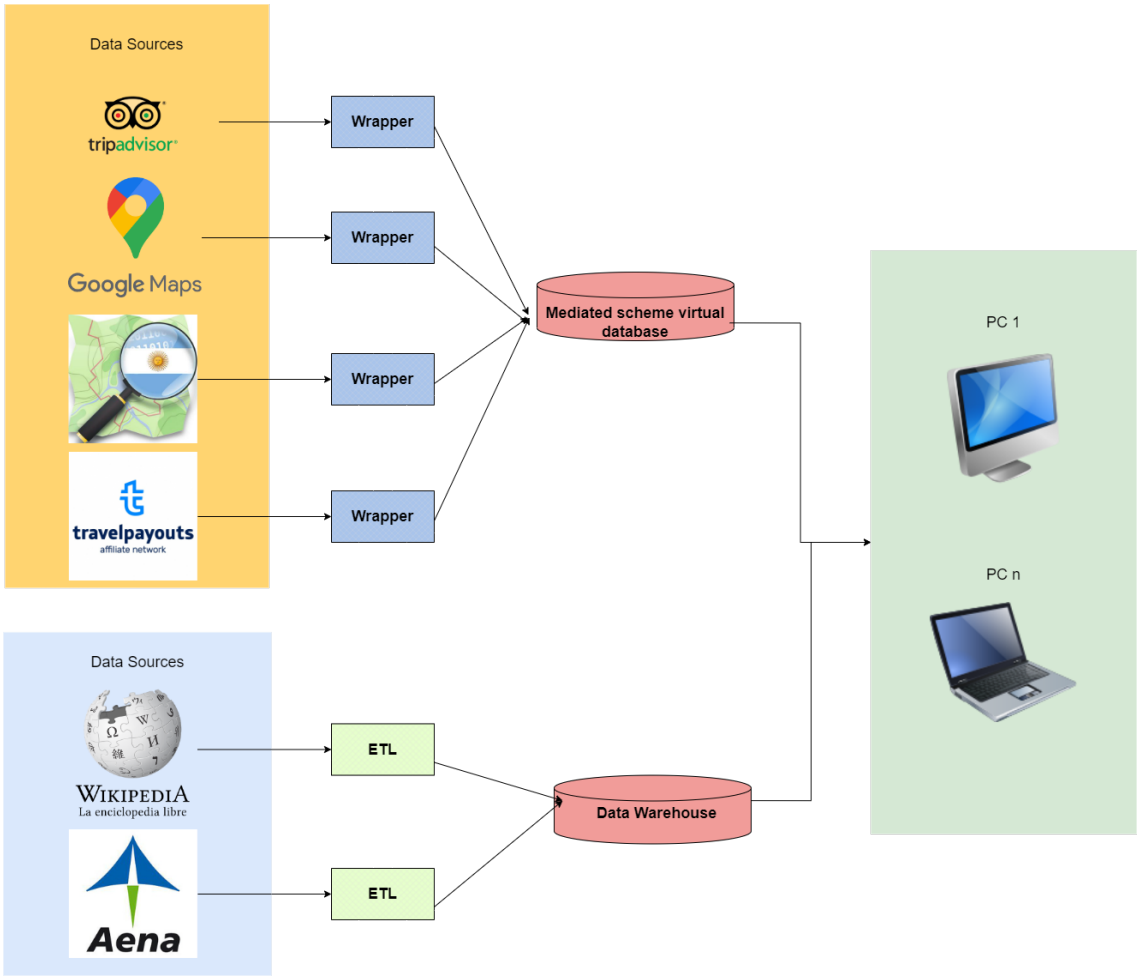


Figura 3: Representación de la integración definitiva de las fuentes de datos en el sistema.

3.4. Implementación

En esta sección se describen los códigos y servicios de terceros que se han utilizado para la implementación. Estos códigos se podrán encontrar dentro de la máquina virtual en la carpeta situada en el escritorio.

3.4.1. Esquema Datawarehouse

3.4.1.1. AENA Será la fuente de datos principal de este proyecto, proporcionando datos sobre aeropuertos y conexiones. Aena permite conocer todos los destinos posibles desde un aeropuerto de origen. Esta fuente de datos sirve como alternativa a fuentes propuestas como "Airport Database" la cual es una fuente de pago y con un gran volumen de datos.

La extracción de los datos de Aena se realiza mediante un "Scrapper" con la librería Beautifulsoup. Se decide extraer los destinos del aeropuerto de Barajas en Madrid, de esta forma se acota el volumen de datos para así presentar de forma clara un ejemplo sobre la prueba de concepto. Los datos obtenidos en Aena consisten del nombre completo del aeropuerto y su código de iniciales. La información se modifica convenientemente para su exportación en una fuente de datos CSV con dos columnas: una para los nombres de los destinos y otra para las iniciales de estos. CSV es un formato de archivo que se utiliza para almacenar datos en forma de tabla, donde cada línea del archivo representa una fila en la tabla y las celdas de la tabla están delimitadas por comas. CSV es un formato muy común en el mundo de la computación, ya que permite a los usuarios intercambiar datos de una aplicación a otra de manera fácil y sencilla. Los archivos CSV son fáciles de crear y editar, es por ello que se eligen

en este proyecto como formato para almacenar nuestros datos.

3.4.1.2. Wikipedia La idea de este proyecto es ofrecer o sugerir destinos de distintas características: destinos de playa, montañas, culturales... En la prueba de concepto se decide dar la opción de destinos dentro del género "Capitales europeas". La obtención de la lista de capitales europeas se realiza mediante la extracción de datos vía Scrapper de Wikipedia (con BeautifulSoup). Los datos se guardarán en un archivo CSV.

Los módulos o programas necesarios para la extracción y guardado de estas dos fuentes son los denominados ETL. Estos se ejecutaran con una cierta periodicidad . Al ser un tipo de datos que no varían mucho a lo largo del tiempo, las actualizaciones de estas bases de datos se podrán realizar una vez al año. Debido a que estos datos están enmarcados dentro de el esquema materializado se almacenarán en memoria de forma permanente, al ser bases de datos no muy grandes esto no supondrá ningún inconveniente a nivel de memoria.

3.4.1.3. Script unificador: El objetivo es crear una ultima fuente de datos que mantenga solo las capitales europeas dentro de los destinos de Aena. Este proyecto requiere de de una serie de opciones de destinos y/o planes con el que basar las búsquedas y sugerencias ofrecidas al usuario. Para ello se lleva a cabo un filtrado de la información de acuerdo a las preferencias del usuario.

Dentro del esquema Datawarehouse se desarrolla un programa/función encargada de mezclar o filtrar los datos utilizando de las fuentes Aena y Wikipedia. La intención de la prueba de concepto es acotar el numero de decisiones que se puedan tomar. Por el momento el usuario sólo podrá elegir como destino o tipo de plan las capitales europeas. Se crea una ultima fuente de datos que mantenga solo las capitales europeas dentro de los destinos de Aena. De este modo solo virtualizaremos en la aplicación final los datos que se necesiten utilizar para ofrecer una sugerencia. Aunque por el momento sólo se pueda acceder al género "Capitales", la idea es que en un proyecto profesional se tenga una base de datos con una serie de distintos tipo de subgéneros de destinos.

Con las preferencias del usuario ya filtradas llega el momento de ofrecer una sugerencia. La sugerencia de la prueba de concepto tiene que ver con el precio del vuelo (se sugerirán los precios más baratos).

Fuentes	**Periodicidad**
Aena	1 vez al año
Capitales/Wikipedia	1 vez al año
Script unificador	1 vez al año

Cuadro 1: Frecuencia de actualización del módulo DATAWAREHOUSE

3.4.2. Esquema Virtual

En esta sección se va a describir las fuentes de datos que se han utilizado dentro del esquema virtual, estas son las que contienen datos que van variando o que puede interesar acceder en el momento.

3.4.2.1. Travelpayouts: Dentro del esquema virtual se hace uso de una fuente de datos capaz de obtener los precios de una serie de vuelos. **Travelpayouts Data APITravelpayouts Data API** la cual obtiene el vuelo mas barato de un origen y destino dado. La obtención de los datos se realiza a partir de un servicio API proporcionado por Travelpayouts. A partir de la información materializada en el esquema datawarehouse (capitales a las que se puede viajar desde Barajas) se podrán obtener un listado de precios competitivos que se pueden sugerir al usuario.

3.4.2.2. Tripadvisor Se decide sugerir una serie de 5 destinos recomendados en función del precio de sus vuelos, se va a integrar una descripción de la ciudad y una lista de planes que se extraerá de **Tripadvisor**. Ante la imposibilidad de acceder a su API, esta función realizará una llamada a distintos enlaces URL dentro del portal de Tripadvisor para cada ciudad. Esto se realizará mediante un diccionario que asocia a cada capital europea, una web distinta. Sin embargo, todas estas páginas tienen la misma estructura, lo que nos hará posible extraer la descripción de la ciudad y una lista de planes (en este caso, de cinco) recomendados con la misma función de integración para todas las URL correspondientes a cada ciudad.

3.4.2.3. API google maps La API de google maps es un conjunto de APIS desarrollado por google que nos permiten obtener datos relacionados con sitios, mapas y demás. Esta api es de pago pero funciona a través de un servicio de créditos donde todos los meses se dispone de 200$ de crédito de forma gratuita, además al crear la cuenta por primera vez te dan 300$ de crédito extra. Esto está hecho para que no se abuse de está api ya que tiene muchas funcionalidades, en nuestro caso para esta implementación se va a considerar que es gratuita ya que los créditos se gastan cuando se hacen miles de llamadas a la api. Dentro de este conjunto de apis se han utilizado las siguientes.

- Geolocation API
- Places API
  - Find place
  - Place details
- Rutas

Geolocation El primer paso de nuestro programa es encontrar el aeropuerto más cercano al usuario, para ello, con el consentimiento del usuario, se accede a su ubicación a través de esta api. Este servicio te permite determinar tu ubicación con mucha precisión si se le aportan datos de torres de teléfono cercanas, como nosotros no disponemos de esa información la api se basa en la dirección IP del usuario para determinar su ubicación, es por eso que la ubicación no es precisa y es más una aproximación de la localidad en la que se encuentra. La respuesta de está api es un json (diccionario en python) en el que se encuentran la latitud y la longitud.

Places Este servicio se utiliza de dos formas diferentes, para obtener un sitio buscando por nombre, y para obtener más información de un sitio utilizando un place id. El place id es uno de los parametros que arroja la respuesta de la api para buscar un sitio a partir de texto, es un identificador único de cada sitio. Al pedir que busque sitios puedes introducir un parametro en forma de coordenadas para que los resultados estén basados en esa ubicación, de está forma se encuentra el aeropuerto más cercano entre otras cosas. Para encontrar los detalles de un sitio basta con pasar el id del sitio sobre el que quieres informarte y los parámetros de los que se quiere obtener información, al igual que antes estos parametros son devueltos en formato json. En este proyecto se utiliza para obtener el nombre y la localización de un lugar.

Rutas Esta api devuelve la ruta que hay que seguir para ir de un lugar a otro. Los parámetros de entrada son el origen (ubicación del usuario), destino (el aeropuerto con el que trabajemos más cercano) y el modo de ir (conduciendo). La respuesta de esta api es un json en el que está la información necesaria, esta información está codificada mediante puntos en el mapa, es decir, lo que hace está api es arrojar los pasos que se tienen que realizar para llegar a nuestro destino. Es por eso que al representar la ruta en el mapa (como se explicará más adelante) aparecen una serie de lineas rectas que aparentemente no siguen ninguna carretera. Estas lineas rectas unen los puntos que arroja la llamada a la api que son los puntos en los que habría que tomar decisiones, por ejemplo, si llegamos a un cruce y tenemos que seguir recto la api no considerará ese cruce como un punto mientras que si en ese cruce tenemos que cambiar la dirección se considerará un punto desde el que partirá una linea hacía la siguiente encrucijada. Esto se puede observar claramente en los trayectos sobre las autopistas en los que tenemos que estar durante un tiempo conduciendo por la misma carretera.

3.4.2.4. Nominatim Esta es una api más simple que lo que permite obtener la localización de una calle, ciudad, etc. El uso que se le da en este proyecto es obtener la ubicación del usuario cuando no nos da acceso a su ubicación. Se obtiene la ubicación de su calle de forma más precisa que con la api de google ya que no depende de dónde esté el ordenador en ese momento.

3.5. Capa de visualización.

La visualización de los datos, como por ejemplo los destinos de viaje disponibles en nuestra aplicación o los destinos más recomendables en el momento, es simple ya que se imprimen por pantalla. Estos datos dentro del programa están recogidos en un dataframe que es una estructura de datos propia de la librería pandas de Python.

Esta estructura de datos se podría considerar una tabla con filas y columnas. Por lo tanto la visualización de estos datos es la visualización de una tabla.

Folium Folium es una librería de Python equivalente a la librería leaflet.js de JavaScript que sirve para la visualización de mapas interactivos. Se utiliza en dos ocasiones ya que proporciona información muy visual. El Scrapper de Tripadvisor (3.4.2.2) devuelve, entre otra información, una lista con los mejores planes para realizar en esta ciudad. Haciendo uso de la api de google maps places y de Nominatim se consiguen las coordenadas de estos lugares y se representan dentro de un mapa folium (que es objeto HTML). Dentro de este mapa se muestran como marcadores y si se pulsa dentro de la chincheta aparece el nombre del sitio que representan.

También se utiliza para representar la ruta que tiene que seguir el usuario para llegar al aeropuerto disponible más cercano, para ello se utiliza la respuesta que nos arroja la api de google maps rutas. Sobre el mapa se representan cada uno de los puntos que nos indica esta respuesta y se unen entre si con polylineas, que no dejan de ser lineas rectas que unen dos puntos de un mapa. En este mapa se muestra con chinchetas tanto como el origen (ubicación del usuario) como el destino (aeropuerto).

3.5.1. Interfaz del usuario

La visualización completa de los datos se ha realizado en Jupyter Notebooks. Se opta por este método ya que es un entorno que es conocido de antemano por todos los integrantes del grupo y permite la utilización de widgets que dinamizan la interacción con el usuario. Dentro del archivo de Jupyter se ejecutan llamadas a funciones creadas por los integrantes donde se ha implementado todo el código, de tal forma que el notebook queda limpio. Se utiliza también widgets que permiten al usuario interactuar con los datos de una manera más intuitiva y visual que la simple ejecución de celdas. Este widget es un menú desplegable que al seleccionar una opción llama al Scrapper de Tripadvisor que aporta información extra sobre el destino que seleccionemos.

En este notebook es donde se refleja el valor añadido que se aporta en el proyecto ya que se muestra información que no puede ser obtenida de una sola fuente de datos a la vez. La información de nuestra ubicación, el aeropuerto más cercano, y la ruta que seguir hacia este son datos que se pueden encontrar haciendo una búsqueda en google. Los vuelos con precio a todas las capitales europeas llevaría varias búsquedas en distintas fuentes de datos al igual que para obtener la información de los mejores planes para cada ciudad habría que hacer una llamada a Tripadvisor. En este proyecto se proporciona todo esto en un mismo programa.

4. Instrucciones para la ejecución de la prueba de concepto

Se ha creado una máquina virtual que se puede ejecutar en VirtualBox. Las maquinas virtuales se utilizan como una forma de ejecutar un sistema operativo adicional dentro de otro sistema operativo principal. Se instala un sistema operativo Ubuntu, sistema operativo de código abierto basado en Linux. Ubuntu viene con una variedad de aplicaciones y herramientas pre-instaladas, incluyendo una interfaz de usuario de escritorio fácil de usar y una tienda de aplicaciones que le permite instalar aplicaciones adicionales de forma fácil y rápida. Es por todo esto que resulta un sistema operativo idóneo para la presentación de la prueba de concepto.

4.1. Instrucciones para instalar maquina virtual:

Para instalar la maquina virtual primero debe descargarse el archivo de la página: https://drive.google.com/file/d/1S3_Efnn8CkTiII6sWuXjNH5G_VQfwsIB/view?usp=sharing:

- En la barra de herramientas seleccionar: Archivo (esquina superior izquierda)
- Dentro de herramientas "Archivo" seleccionar: Importar servicio Virtualizado
- Cargar el archivo con el nombre: ProyectoDatos.ova
- Dejar todas las opciones por defecto

La maquina deberá tener acceso a internet (es necesario para ejecutar la aplicación).

4.2. Ejecutar la prueba de concepto

Iniciar maquina virtual en VirtualBox, ingresar con nombre de usuario y contraseña al sistema operativo.

- Usuario: ProyectoDatos
- Contraseña: 1234

Pasos dento del sistema operativo:

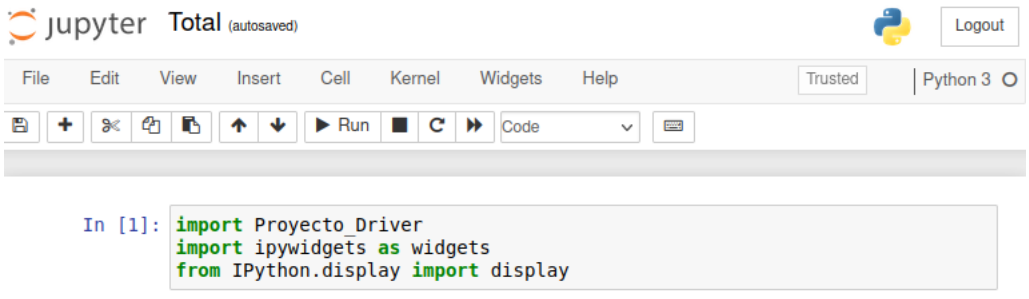
1. Abrir el terminal: ctrl+alt+t
2. Teclear:

```
cd Desktop/Codigos_Datos_masivos/Codigos/
```

3. Teclear:

```
jupyter notebook &
```

4. Abrir jupyter notebook llamado: Total.ipyb
5. Ejecutar celdas por orden pulsando `SHIFT+ENTER`



En la primera celda el usuario tiene que decidir si da permiso al programa para acceder a su ubicación, en caso de denegarla deberá introducir el nombre de su calle, en caso de que esta calle no sea única se deberá introducir el nombre de la localidad donde esté dicha calle, por ejemplo si estamos hablando de una calle en Leganés se deberá introducir `Leganés`.

Detectar el aeropuerto

```
In [2]: mapa_aeropuerto = Proyecto_Driver.step_1()

Me das acceso a tu ubicación: Si o No: Si
El aeropuerto Aeropuerto Adolfo Suárez Madrid-Barajas (MAD) se encuentra
a una distancia de 26,9 km.
Actualmente tardaría 21 min en llegar.
```

En la siguiente celda se muestran los destinos más destacables que salen desde el aeropuerto disponible más cercano. A continuación se pide al usuario que decida si quiere ver los destinos más recomendables, que son los más baratos. Esta ejecución tarda un poco ya que se llama a la api que obtiene los vuelos más económicos.

Destinos desde el aeropuerto

Ahora se le mostrarán 20 de los destinos más destacables que hemos encontrado desde el aeropuerto más cercano.

```
In [3]: lista, destinos = Proyecto_Driver.step_2()

A continuación se le mostrarán 20 de los destinos más destacables que hemos encontrado desde Adolfo Suárez Madrid-Ba
rajas (MAD).
                            Destino Aeropuerto
0   BERLIN-BRANDERBURG WILLY BRANDT         BER
1                    ROMA /FIUMICINO         FCO
2             PARIS /BEAUVAIS-TILLE         BVA
3          PARIS /CHARLES DE GAULLE         CDG
4                       PARIS /ORLY         ORY
5                          BUCAREST         OTP
6                             VIENA         VIE
7                           VARSOVIA         WAW
8                   VARSOVIA/MODLIN         WMI
9                           BUDAPEST         BUD
10                             PRAGA         PRG
11                             SOFIA         SOF
12                          BRUSELAS         BRU
13              BRUSELAS /CHARLEROI         CRL
14               ESTOCOLMO /ARLANDA         ARN
15                            ATENAS         ATH
16                          HELSINKI         HEL
17                              RIGA         RIX
18                         COPENHAGUE         CPH
19                            DUBLIN         DUB
¿Desea ver qué destinos son los más recomendables?Si
          Recomendaciones Aeropuertos  Precios      Fechas
0                    LISBOA          LIS       12  2023-01-16
1  PARIS /BEAUVAIS-TILLE          BVA       17  2023-01-17
2                BRUSELAS          BRU       18  2023-01-26
3                  DUBLIN          DUB       19  2023-01-17
4              PARIS /ORLY          ORY       20  2023-01-11
5          ROMA /FIUMICINO          FCO       21  2023-01-15
```

Detalles de las ciudades

El siguiente paso del programa es seleccionar una de las opciones de la lista con los mejores destinos. De forma predeterminada se muestra el elemento más barato de la tabla pero seleccionando otro destino en el menú desplegable se ejecuta automáticamente la función que muestra la información correspondiente a la ciudad seleccionada. Esta actualización tarda unos segundos.

```
In [4]: drop = widgets.Dropdown(
            options=lista,
        )
        out = widgets.interactive_output(
            lambda ciudad: Proyecto_Driver.step_3(ciudad, destinos),
            {'ciudad': drop}
        )
        widgets.VBox([drop, out])
```

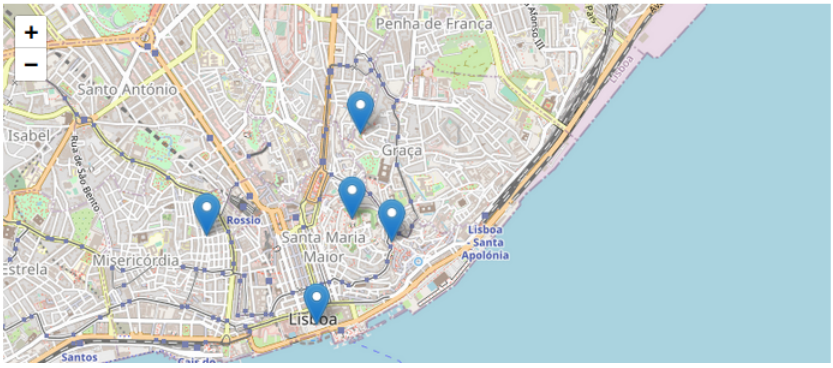
LISBOA

```
Ciudad:  LISBOA
Esplendor europeo poco conocido

Situada en un enclave entre siete colinas, esta ciudad ideal para caminar se erige como una
alternativa fascinante a las capitales más populares de Europa. La animada vida nocturna de
Lisboa, sus alegres mercados y sus vibrantes museos proporcionan montones de opciones entre
las que no pueden faltar unas copas de vino de Oporto, una buena ración de bacalao y pastéi
s de nata.

Lista de planes:

Castillo de San Jorge
Miradouro da Senhora do Monte
Barrio Alto
Praca do Comercio (Terreiro do Paco)
Alfama
```

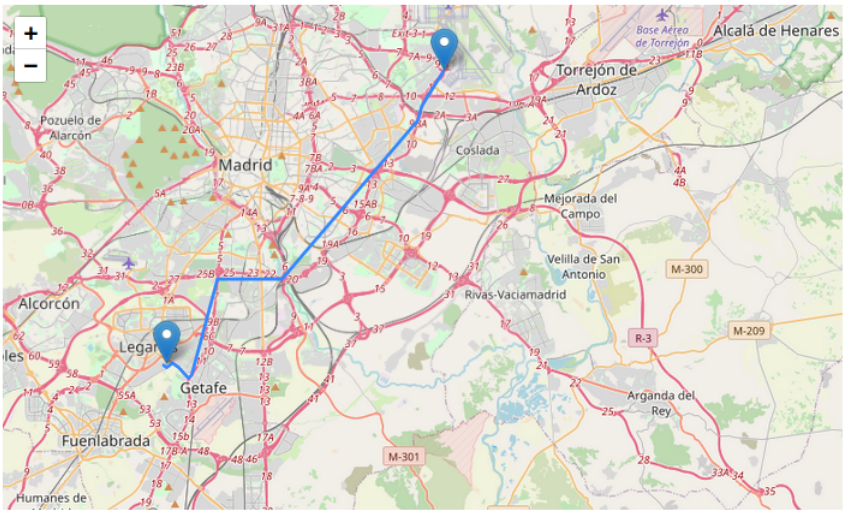


En la última celda se muestra en el mapa la ruta que deberá seguir el usuario para llegar al aeropuerto de salida.

Ruta hacía el aeropuerto

```
In [5]: Proyecto_Driver.step_4(mapa_aeropuerto)
```

```
La ruta hacia el aeropuerto es la siguiente:
```



4.2.1. Ejecución de los módulos Datawarehouse

Si se desean actualizar los datos enmarcados dentro del esquema datawarehouse se deben seguirse los siguientes pasos

1. Abrir el terminal: ctrl+alt+t
2. Teclear:

```
cd Desktop/Codigos_Datos_masivos/Codigos/ETL/
```

3. Teclear:

```
python3 AENA_lanzador.py &
python3 lanzador_capitales.py &
```

Se actualizaran las 3 tablas que tienen que ver con el módulo ETL. Se puede encontrar en el directorio que se accede con el comando:

```
cd Desktop/Codigos_Datos_masivos/Codigos/CSVs/
```

5. Escalabilidad del proyecto

En proyectos futuros sería interesante implementar una base para el almacenamiento ordenado de la información sobre nuestros destinos, aeropuertos de salida, etc. Para este tipo de información seria muy útil una base de datos relacional. Una base de datos relacional es un tipo de base de datos que se basa en el modelo relacional de datos, en el que se utilizan tablas para representar los datos y las relaciones entre ellos. Esto permite una organización más lógica y eficiente de la información, lo que a su vez ofrece varias ventajas.

Una de las principales ventajas de una base de datos relacional es que permite la integración y el intercambio de datos entre diferentes aplicaciones y sistemas. Esto facilita la interoperabilidad y permite a los usuarios acceder y utilizar los datos de manera más sencilla. Además, una base de datos relacional permite la realización de consultas complejas a los datos, lo que facilita su análisis y permite obtener información valiosa de manera más eficiente.

En resumen, las principales ventajas de una base de datos relacional son su capacidad para integrar y compartir datos, garantizar la integridad de los datos y facilitar su análisis mediante consultas complejas. Todo esto hace a bases de datos del estilo a MySQL una herramienta ideal para este tipo de trabajos (además de ser un software de código abierto ampliamente utilizado).

De cara a realizar la integración de las fuentes explicadas en el diseño propuesto, algunas de ellas requieren APIs de pago, como pueden ser AviationStack o Google maps, cuyos pagos están asociados a un aumento en el posible número de llamadas a su API, lo cual sería necesario para un proyecto de una envergadura mayor. Estos servicios de pago, que suponen una limitación en el desarrollo del proyecto, pueden ser un motivo para cobrar un pequeño precio por el uso de la aplicación a sus usuarios. Este pago se puede realizar a través de Smart Contracts, como se va a desarrollar en el siguiente trabajo.

Otro aspecto que debería tratarse en proyectos futuros es la utilización de Scrappers. Aunque pueden ser una herramienta útil para recopilar datos a gran escala, también pueden presentar ciertos inconvenientes. Uno de ellos es que puede ser ilegal o inapropiado si se utiliza para acceder a contenido protegido por derechos de autor o que viola las condiciones de uso del sitio web. Además, la recopilación de datos a gran escala puede generar tráfico en el sitio web que se está "scrappeando", lo que puede afectar el rendimiento del sitio y en algunos casos incluso causar que se caiga. Además, el uso indebido de un web scraper puede ser considerado una forma de "spamming" y puede dañar la reputación del usuario o de la empresa que lo utiliza. Por lo tanto, es importante utilizar un web scraper de manera responsable y asegurarse de tener permiso para acceder y recopilar los datos que se deseen. Por otro lado el gran inconveniente de los scrappers (además de su lentitud) es que dependen de la estructura HTML de la pagina web. Si la pagina web se actualiza, lo mas habitual es que los scrappers dejen de funcionar, es por ello que tiene una "vida util" relativamente corta. Para un proyecto serio/profesional, se deberían buscar alternativas eficaces y

robustas para la obtención de los datos. Si se decidiera utilizar scrappers en un proyecto profesional se debería controlar exhaustivamente los cambios en las páginas web utilizadas y cambiar la estructura de los scrappers para que se adapten a la nueva estructura lo más rápido posible (esto podría ser un trabajo muy laborioso e incluso costoso económicamente).

6. Conclusiones

A lo largo de este proyecto, se han podido estudiar y realizar los distintos tipos y técnicas de integración de datos que se han estudiado durante el curso realizando una propuesta de diseño de un programa de un programa de búsqueda y recomendación de vuelos, que pueda proveer al usuario información sobre los posibles destinos, además al de información adicional personalizada.

Se ha llevado a cabo el estudio de un diseño propuesto de arquitectura y posible implementación del servicio/app mencionado anteriormente, en el que se han seleccionado distintas fuentes de datos a integrar que lleven al programa a cumplir su función. El modelo o arquitectura de integración por el que se ha optado es un esquema híbrida almacenando los datos necesarios en un esquema DataWarehouse o virtualizando datos en tiempo real sin almacenarlos en memoria, según requiera la naturaleza de los datos, si bien, son datos invariables con el tiempo o datos que se actualizan constantemente, como se ha explicado detalladamente en cada fuente de datos. En esta parte, además se ha podido observar los distintos tipos de integración posibles en función del tipo de web y de datos que se quieran integrar, si la página a la que se llama posee API o, por el contrario, se necesita un WebScrapper que realice la llamda a la página web y extraiga la información necesaria . Se ha podido también notar las limitaciones que tiene realizar proyectos de este calibre debido a que algunas APIs pueden ser de pago, otras tienen limitaciones en los datos que se quieren integrar o algunas páginas pueden tener una estructura realmente difícil de integrar.

Partiendo de esta idea, se ha realizado la prueba de concepto, en la que se integran unas fuentes de datos seleccionadas de cara a obtener una app que cumpla el objetivo de la idea planteada pero de manera limitada en algunas funciones. En la implementación se ha seguido por lo general la arquitectura propuesta en la fase de validación del proyecto teniendo en cuenta una serie de limitaciones de tiempo y económicas. Se han seleccionado fuentes que que ofrecieran un servicio gratuito. Aunque estas fuentes a priori puedan ser menos potentes que las originales han servido para ejemplificar de forma sencilla nuestro planteamiento del proyecto, además de para estudiar de manera práctica los principios teóricos de la integración de datos y el uso de técnicas vistas en clase como la librería *BeautifulSoup* de Python, llamadas a distintas APIs y para la visualización de contenidos se ha empleado Jupyter Notebooks para mostrar los datos y la creación de mapas y rutas a partir de datos de nuestras fuentes mediante librerías externas, además de introducir inputs y *widgets* que permitan al usuario recibir información personalizada y seleccionar lo que desea visualizar.

En conclusión, tras la realización de este trabajo, se ha podido diseñar una idea ambiciosa de una aplicación explicada a lo largo del proyecto y realizar una implementación reducida de ella, donde se ha estudiado las ventajas y limitaciones que pueden surgir en un proceso de integración de datos de diversas fuentes con resultados satisfactorios en la realización de este proyecto pudiendo alcanzar los objetivos planteados al principio del trabajo. Además, en nuestro caso, este trabajo nos ha dado la oportunidad de conocer, llevar a la práctica y desarrollar habilidades informáticas en un tipo de programación que puede resultar muy útil para múltiples aplicaciones, que es distinta a la que podemos estar habituados y que nos ha resultado una experiencia muy satisfactoria.