Proyecto de Inteligencia Artificial

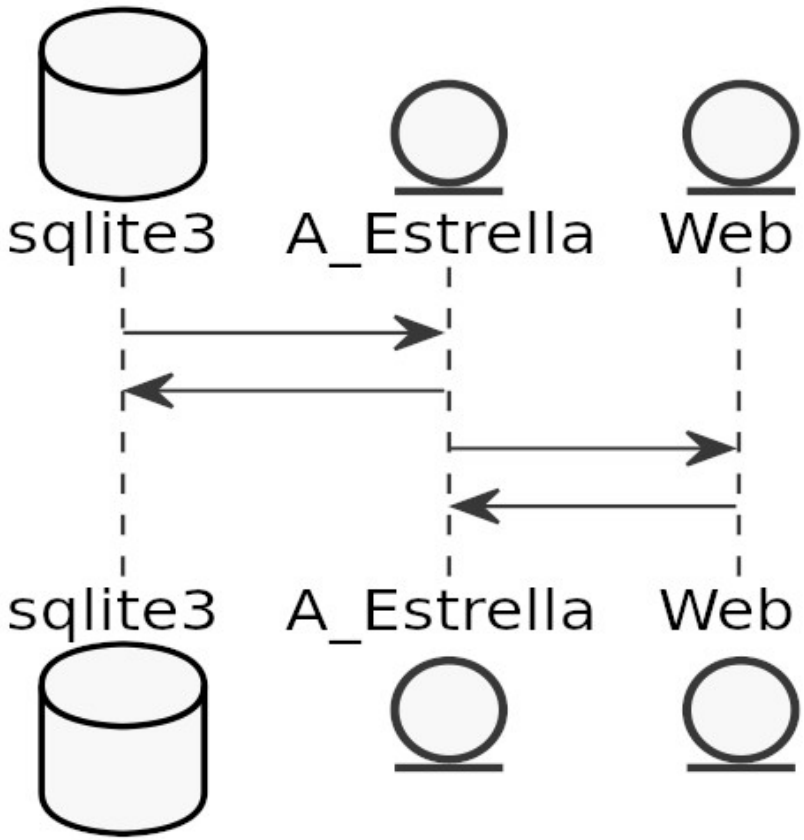
Rubn Aguado Cosano - z170284
Younes Idrissi Boulid - z170155
Paula Pousa Martinez - z170068
Jorge Sol Gonzalez - z170212

Universidad Politecnica de Madrid

December 16, 2019

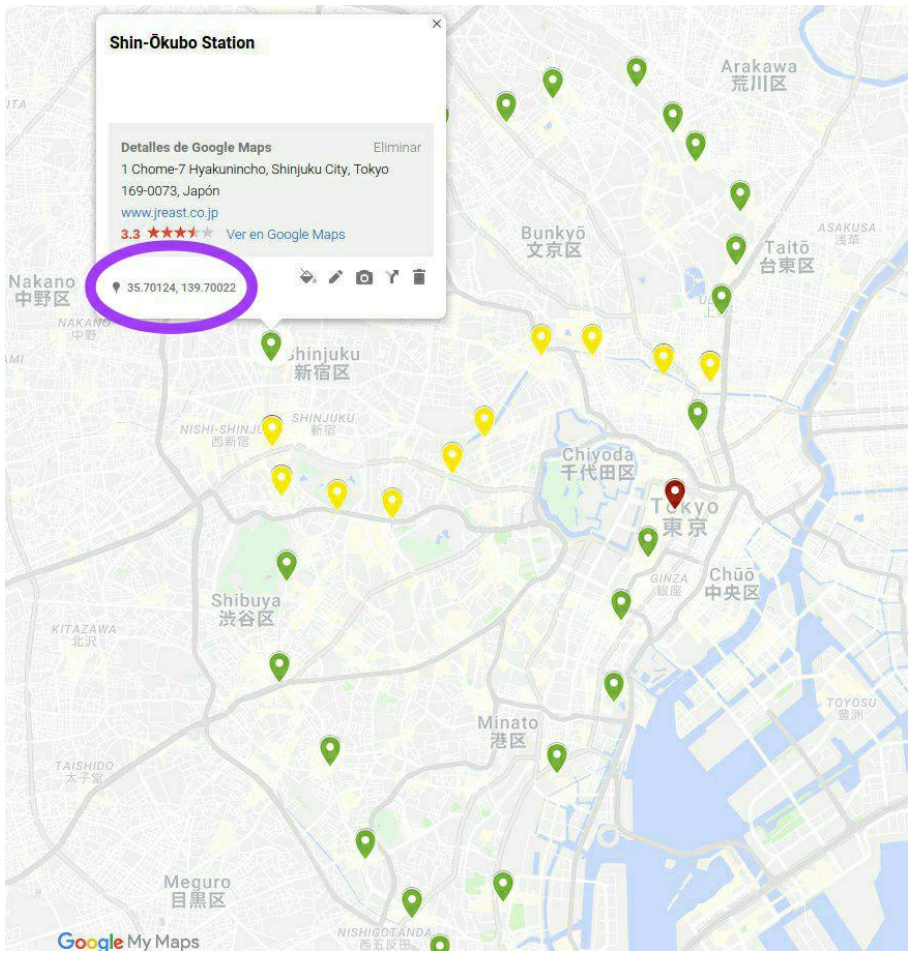
Indice

Arquitectura del Proyecto



Recogida de Datos

- La magnitud de medida son los **metros**.
- Para la obtencin de las **coordenadas** de las paradas se utiliz Google Maps.



Recogida de Datos (Formula de Haversine)

```
from math import radians, cos, sin, asin, sqrt

def haversine(lon1, lat1, lon2, lat2):

    coord = open("../../coordenadas.txt", 'r')
    respueta = open("../../recta.txt", 'w')
    data = coord.readLines()

    listaDeDatos = []

    for line in data:
        myLine = line.split()

        lon1 = myLine[0]
        lat1 = myLine[1]
        lon2 = myLine[2]
        lat2 = myLine[3]

        # convertimos grados en radianes
        lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
        c = 2 * asin(sqrt(a))
        r = 6371000 # Radio de la tierra en metros

        listaDeDatos.append(c*r)

    for i in listaDeDatos:
        respuesta.write(i)
```

Recogidas de Datos

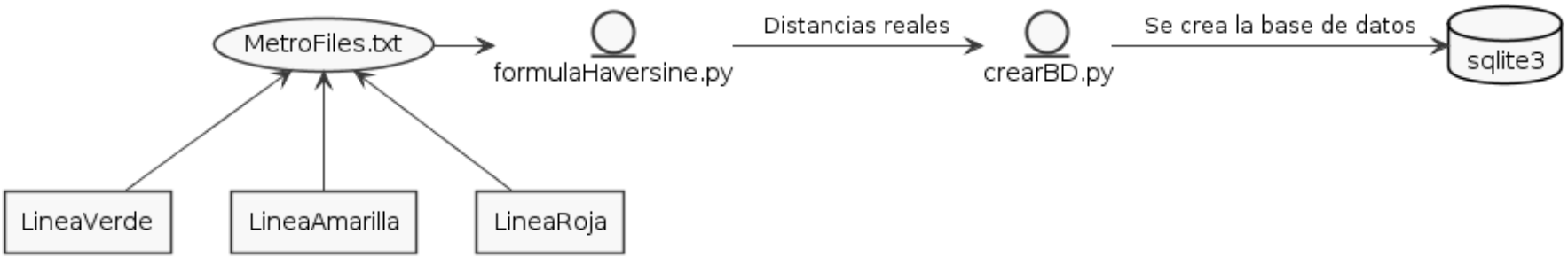
La recogida de las distancias reales se realiz a mano con ayuda de la pgina web **HyperDia**

Route1 Take time: 9 Minutes Transfer: 0 Times Distance: 7.7 km
Total:¥ 170(Ticket Fare:¥ 170 Seat Fee:¥ 0) Commuting ticket(1mon:¥ 5,270 3mon:¥ 15,010 6mon:¥ 25,290)

Commuting ticket

Time	Route	Fare	Seat Fee	UsefulLink
22:08	SHINJUKU(JR) [Departure track No.8] Station timetable | Add to favorite			Map Rent-a-car Hotel Restaurant
[9 Min]	JR Chuo Line Rapid Service for TOKYO Train timetable | Interval timetable	¥170		
22:17	OCHANOMIZU Add to favorite			Map Rent-a-car Hotel Restaurant

Base de Datos




```
└─± python3.7 crearBD.py

Tabla 1 - Creando la tabla con las relaciones entre paradas: EXITO

Tabla 2 - Creando la tabla con las lineas rectas: EXITO

Tabla 3 - Creando tabla de datos: EXITO

Insertando datos en la tabla 3: EXITO

Insertando los datos en la tabla 2: EXITO

Insertando datos en la tabla 1: EXITO
```

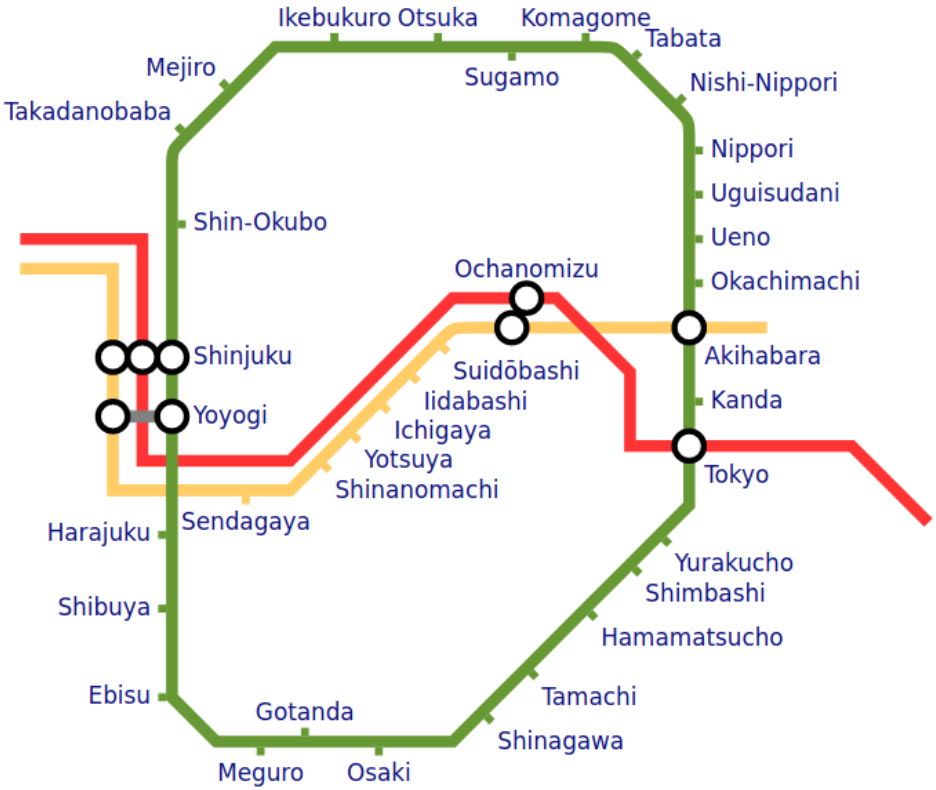
Base de Datos

```python
import sqlite3 # Libreria de la BDD que vamos a usar.

def getDistanciaTren(nodo):
    db = sqlite3.connect('metroDataBase.db')
    diccio = {}
    cursor = db.cursor()

    nodo = str(nodo)

    cursor.execute("SELECT DESTINO, DISTANCIA FROM tren WHERE ORIGEN = ?", (nodo,))
    for i in cursor:
        diccio[i[0]] = i[1]

    cursor.execute("SELECT ORIGEN, DISTANCIA FROM tren WHERE DESTINO = ?", (nodo,))
    for i in cursor:
        diccio[i[0]] = i[1]

    return diccio


def getDistanciaRecta(start, end):
    db = sqlite3.connect('metroDataBase.db')
    resultado = -1
    cursor = db.cursor()

    cursor.execute("SELECT DISTANCIA FROM recta WHERE ORIGEN = ? AND DESTINO = ?", (start, end))
    resultado = cursor.fetchall()

    if(resultado == []):
        cursor.execute("SELECT DISTANCIA FROM recta WHERE DESTINO = ? AND ORIGEN = ?", (start, end))
        resultado = cursor.fetchall()

    return resultado[0][0]


def getLinea(nodo):
    db = sqlite3.connect('metroDataBase.db')
    cursor = db.cursor()

    cursor.execute("SELECT LINEA FROM ids WHERE ID = ?", (nodo,))
    resultado = cursor.fetchall()

    return resultado[0][0]
```

Algoritmo A Estrella

```python
def algoritmo(inicio, fin, transbordo):
    initialDistance = metro.getDistanciaRecta(inicio, fin) if inicio != fin else 0
    listaAbierta = {inicio: {"g": 0, "h": initialDistance, "f": initialDistance, "padre": -1}}
    listaCerrada = {}  # {idNodo: idNodoPadre}
    finalWeight = 0

    # f(n) = g(n) + h(n)
    while(fin not in listaCerrada.keys()):
        thisNodeId = sorted(listaAbierta, key=lambda elem: listaAbierta[elem]["f"])[0]
        thisNode = listaAbierta[thisNodeId].copy()
        thisNodeLines = set(decodeLineNumber(metro.getLinea(thisNodeId)))

        if(thisNodeId == fin):
            finalWeight = thisNode["f"]

        listaCerrada[thisNodeId] = thisNode["padre"]
        del listaAbierta[thisNodeId]

        vecinos = metro.getDistanciaTren(thisNodeId)  # [{idVecino: distanciaAel}, ...]

        vecinos = dict(filter(lambda vecino: vecino[0] not in listaCerrada, vecinos.items()))
        if(len(vecinos) == 0):
            continue

        for idVecino, distanciVecino in vecinos.items():
            vecinoLines = set(decodeLineNumber(metro.getLinea(idVecino)))
            prevNodeLines = set(decodeLineNumber(metro.getLinea(thisNode["padre"]))) if thisNode["padre"] != -1 else set([1, 2, 3])
            specialCase = thisNode["padre"] in [5, 20] and idVecino in [5, 20]  # Caso especial no detectable de otra manera cuando se va por la linea roja

            g = thisNode["g"] + distanciVecino + 300  # Cuantas menos paradas, mejor, cada parada añade un delay de 1/3 de trayecto entre estaciones

            if(len(thisNodeLines & vecinoLines & prevNodeLines) == 0 or specialCase):
                g += 1250 if not transbordo else 1000000  # Penalización equivalente a 1 parada y cuarto

            h = 0 if idVecino == fin else metro.getDistanciaRecta(idVecino, fin)
            f = g + h

            if(idVecino not in listaAbierta or listaAbierta[idVecino]["f"] > f):
                listaAbierta[idVecino] = {"g": g, "h": h, "f": f, "padre": thisNodeId}

    pathList = []
    while(fin != -1):
        pathList.append(fin)
        fin = listaCerrada[fin]

    result = list(reversed(pathList))
    return result, lineasMetro(result), finalWeight
```

Demo



Rubn Aguado Cosano - z170284
Younes Idrissi Boulid - z170155
Paula Pousa Martinez - z170068
Jorge Sol Gonzalez - z170212