Assignment 2 – COMP9321 24T1 UNSW Sydney

A smart API for the Deutsche Bahn

V1.5 written 25 March 1800

You have been contracted by Deutsche Bahn, the national railway company of Germany, to explore how APIs can enhance the experience of passengers across the country. You will need to develop a Flask-RESTX API in Python using the provided APIs:

- v6.db.transport.rest is a REST API for the public transportation system in Germany. Please refer to the "Getting Started" and "API Documentation" link on their site. You do not need to register for an account.
- Gemini API is a REST API for using Google's AI Model named "Gemini", which is similar to Chat-GPT. You are required to use a Google Account to create an API key here, which will be ingested in the template code.

You are expected to explore the provided Deutsche Bahn API to see how it can be used to fulfil the specifications for all questions. You are also only permitted to use the provided Gemini API for Question 6 and 7.

Note that this assignment explores external APIs. Students will interact with real-life web services to add to the learning experience. We are not responsible nor liable for the wording or inclusion/exclusion of entities and descriptions within the APIs. This is a "building your REST API" exercise and should be treated as such.

Question 1: Import a stop [3 marks]

Use the **PUT** method to add new stop(s) to your Sqlite database that match a query string. The data needs to be persisted for the remainder of the assignment in a file named **z[your zID].db.** The first time your script is executed, it will need to create this database.

Example query to your API:

```
PUT /stops?query=hbf
```

Example response from your API:

```
201 CREATED
[{
    "stop_id": 8000085,
    "last_updated": "2024-03-08-12:00:40",
    "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/stops/8000085"
        }
    }
},
{
     "stop_id": 8000152,
     "last_updated": "2024-03-08-12:00:40",
      "_links": {
         "self": {
           "href": "http://[HOST_NAME]:[PORT]/stops/8000152"
          }
     }
},
{
     "stop_id": 8002549,
     "last_updated": "2024-03-08-12:00:40",
      "_links": {
         "self": {
```

```
            "href": "http://[HOST_NAME]:[PORT]/stops/8002549"
          },
      }
},
{
      "stop_id": 8010159,
      "last_updated": "2024-03-08-12:00:40",
       "_links": {
          "self": {
            "href": "http://[HOST_NAME]:[PORT]/stops/8010159"
          },
      }
]
```


Note that:

- “stop_id” refers to the id of the location in the Deutsche Bahn REST API. Throughout this assignment, do not change the stop_id of a resource. The response results need to be sorted by “stop_id”.
- “last_updated” (yyyy-mm-dd-hh:mm:ss) refers to the local computer time that the database entry was last updated or created if first added.
- “_links” is a set field and includes URLs (str) to which the imported collection can be retrieved. It shall consist of the "self” link for this question.
- You should replace [HOST_NAME]:[PORT] with the values in your flask configuration. Refer to this page for further information.
- PUT is idempotent. Submitting the same query multiple times will not add new entries of the same “stop_id”s to the database, but update all of the fields including the “last_updated” time. An update would return 200 OK.

Your code should call **GET /locations** on the Deutsche Bahn API, which returns both stops and stations. The Deutsche Bahn API provides a fuzzy search, which provides more results than an exact query. You are only required to add the maximum number of stops returned when 5 results are requested to the Deutsche Bahn API.

You may store the data in the database in any format you wish, however, you should review the whole assignment and the Deutsche Bahn API to understand what attributes are required.

Requirements:

[1] “stop_id” is ordered and correct for all stops.

[1] “_links” and “last_updated” are correct for all stops.

[0.5] Query method/parameters and Swagger docs are correct and functional.

[0.25] The only status codes returned are 200, 201, 400, 404, and 503.

[0.25] The 500 status code is never returned.

Question 2 / Question 3: Retrieve a stop [6 marks]

Use the **GET** method to retrieve information about a stop from your database.

Example query to your API:

GET /stops/8002549

Example response from your API:

200 OK


```
{
     "stop_id": 8002549,
     "last_updated": "2024-03-08-12:00:40",
     “name": “Hamburg Hbf”,
     “latitude": 53.553533,
     “longitude": 10.00636,
     “next_departure": “Platform 4 A-C towards Sollstedt”,
      "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/stops/8002549"
        },
        "next": {
          "href": "http://[HOST_NAME]:[PORT]/stops/8010159"
        },
       "prev": {
          "href": "http://[HOST_NAME]:[PORT]/stops/8000152"
        }
    }
}
```


Example query to your API:

GET /stops/8000152?include=name,latitude

Example response from your API:

200 OK


```
{
```

```
    "stop_id": 8000152,
    “name": “Hannover Hbf”,
    “latitude": 52.377079,
     "_links": {
       "self": {
         "href": "http://[HOST_NAME]:[PORT]/stops/8000152"
       },
       "next": {
         "href": "http://[HOST_NAME]:[PORT]/stops/8002549"
       },
      "prev": {
         "href": "http://[HOST_NAME]:[PORT]/stops/8000085"
       }
   }
}
```


Note that:

- You will need to query the GET /stops/:id/departures Deutsche Bahn API endpoint to retrieve “next_departure”.
- You may assume that the next departure is at the top of the returned list. Select the first departure where the “platform” and “direction” field are both valid strings and not null up to a maximum of 120 minutes away. If this is not available, return a 404 error.
- “_links” will always consist of “self” and will include “prev” and/or “next” depending on whether there is an ID number before and/or after it. The order of these three fields is not important. The stop with the highest ID number (numerically) in the database would not return a "next" link, and the stop with the lowest ID number in the database (numerically) would not return a "prev" link."
- If a “stop_id” is the minimum or maximum, the “prev” or ”next” field should not be returned.
- The latitude and longitude decimal places should comply with what is provided by the Deutsche Bahn API, or updates in the database initiated by Question 5.
- It is optional to store the departure data in your database, though it may be useful for future questions.
- Requests for stops that are not already in your database should be rejected.

- The query string “include” can include zero to many of the strings “last_updated”, “name”, “latitude”, “longitude”, and “next_departure”.
- “_links” and “stop_id” are not permitted parameters in the “include” query string, as it shall always be returned. The request should be rejected with an appropriate error code if it is requested.
- The values in “include” are separated by a comma in the query string.
- Requests for stops that are not already in your database should be rejected.

Requirements:

[1] Filtering behaviour works correctly.

[1] “name”, “latitude”, and “longitude” are correct.

[1] “next_departure” is correct.

[1] “_links” is correct.

[0.5] “stop_id” and “last_updated” are correct.

[0.5] The only status codes returned are 200, 400, 404, and 503.

[0.5] The 500 status code is never returned.

[0.5] Query method/parameters and Swagger docs are correct and functional.

Question 4: Delete a stop [2 marks]

Use the **DELETE** method to delete stops from your database.

Example query to your API:

```
DELETE /stops/8010159
```

Example response from your API:

```
200 OK
{
    "message": "The stop_id 8010159 was removed from the database.",
    "stop_id": 8010159
}
```

Alternatively, if the stop_id does not exist:

```
404 NOT FOUND
{
    "message": "The stop_id 8010159 was not found in the database.",
    "stop_id": 8010159
}
```

Note that:

- You should not have to query any external APIs.

Requirements:

[0.5] "message" and “stop_id” are correct.

[0.5] “_links” is updated for the whole database.

[0.5] Query method/parameters and Swagger docs are correct and functional.

[0.25] The only status codes returned are 200, 400, and 404.

[0.25] The 500 status code is never returned.

Question 5: Update a stop [2 marks]

Use a **selected** method to update the field(s) of a stop.

Example query to your API:

```
     CHOSEN_METHOD /stops/8010159
{
     “name”: “Morty’s House”,
     “latitude”: -33.918859,
     “longitude”: 151.231034
}
```

Example response from your API:

```
200 OK
{
    "stop_id": 8010159,
    "last_updated": "2024-03-09-12:00:40",
    "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/stops/8010159"
        }
    }
}
```

Note that:

- Your API shall accept one to many fields in the request body to modify. This is explicitly different from Question 3, where it uses a GET request and a query string.
- A request with empty fields is considered invalid, and an appropriate error code should be returned.
- Fields are expected to be verified. Specifically, these include non blank strings for “name” and “next_departure”, valid “latitude” and “longitude” values, and correctly formatted “last_updated” values parsable as yyyy-mm-dd-hh:mm:ss.
- The updatable fields include the strings “last_updated”, “name”, “latitude”, “longitude”, and “next_departure”.
- “_links” and “stop_id” are not permitted parameters in the request body, as it shall always be returned. The request should be rejected with an appropriate error code if it is requested.

- You should not have to query any external APIs.
- If the “last_updated” field is requested to be updated, this should be complied with. Else, it should be updated to the current time of an update.

Requirements:

[0.5] All values are updated in the database.

[0.5] All invalid requests are rejected.

[0.5] Query method/parameters and Swagger docs are correct and functional.

[0.25] The only status codes returned are 200, 400, and 404.

[0.25] The 500 status code is never returned.

Question 6: Retrieve operator profiles [3 marks]

Use the **GET** method to return a profile for each operator who is operating a service departing from a desired stop within 90 minutes.

Example query to your API:

GET /operator-profiles/8010159

Example response from your API:

200 OK

{

"stop_id": 8010159,

“profiles”: [

{

“operator_name”: “DB Fernverkehr AG”,

“information”: “DB Fernverkehr AG is a subsidiary of Deutsche Bahn that operates long-distance passenger trains in Germany. It offers both domestic and international routes, travelling to 14 European countries. Their trains are known for being a fast, comfortable, and environmentally friendly way to travel. They also manage DB Lounges in major train stations for passenger comfort.”

},

...

{

“operator_name": “DB Regio AG Südost”,

“information": “DB Regio AG Südost is a regional passenger transport company in central Germany, operating in the states of Saxony, Saxony-Anhalt, and Thuringia. Founded in 2001/2002, it's a subsidiary of DB Regio AG and is responsible for operating trains on over 4,400 kilometers of track. With a workforce of over 2,400 employees, they move around 63.4 million passengers annually.”

}

]

}

Note that:

- “information” should be populated via the Gemini API.

- The requested stop_id must already exist in the database; else the request is not valid.
- To avoid spending too long with the Gemini calls, you are only required to return either all of operators, or the first 5 departure operators, whichever is lesser.
- You will not be penalised for including duplicate profiles of all operators, but are strongly encouraged to only return unique operator profiles.

Requirements:

[1] Substantial information is returned for all operators.

[0.5] All operators from the list of departures are included.

[0.5] Profiles response is structured correctly.

[0.5] Query method/parameters and Swagger docs are correct and functional.

[0.25] The only status codes returned are 200, 400, 404, and 503.

[0.25] The 500 status code is never returned.

Question 7: Create a tourism guide [5 marks]

Use the **GET** method to return a TXT file to help a tourist explore points of interests around a journey using stops from your database. You must include substantial information about at least two points of interest – one at the source, and one at the destination.

To further enhance the tourism guide, you may explore possible points of interest between the source and destination, or any other data via the Gemini API that you believe is helpful to a tourist.

If there are less than two stops in the database, or if there is no valid public transport route between any two stops in the database, then the request should be rejected. Otherwise, you may choose any two stops from the database, with a valid route between, to act as the source and destination.

You are expected to use the Gemini API to generate content about the locations.

Example query to your API:

```
GET /guide
```

Example response from your API:

200 OK

```
{

}
```

(TXT file z[your zID].txt is returned)

Note that:

- The TXT file in the format **z[your zID].txt** needs to be explicitly returned by the API, rather than only being saved to the local filesystem.
- The response body will be empty.
- Substantial information is defined where the user receives some tangible learning benefit as a tourist.

Requirements:

[1] Includes substantial information about at least one point of interest at the source.

[1] Includes substantial information about at least one point of interest at the destination.

[1] Includes other substantial information to enhance a tourist's experience using the guide.

[1] A TXT file is successfully returned via the API.

[0.5] Query method/parameters and Swagger docs are correct and functional.

[0.25] The only status codes returned are 200, 400, and 503.

[0.25] The 500 status code is never returned.

Important notes:

- Where an example response body is not provided, you should return an empty response body with an appropriate error code.
- Your data should be persisted in an Sqlite database named as **z[your zID].db**.
- You may only use libraries used in the labs or mentioned in the assignment specification.
- Your submission will be marked manually, where the tutor uses your Swagger doc to test your endpoints. If your Swagger doc is not functional, you will receive a zero for that question, and potentially other subsequent questions if your database cannot be modified. It is your responsibility to ensure your Swagger doc can be used for marking, and remarks due to non-functioning Swagger docs will not be permitted. Postman, Curl, or other similar tools will not be used for marking.
- You should adhere to the best design guidelines for REST APIs mentioned in the lectures and labs, such as appropriate responses with JSON format and proper status codes, and full API documentation from a Swagger doc that is fully self-explanatory with a summary, parameter descriptions, default values, and response codes.
- You should consider cases such as invalid inputs or any invalid attempts to use the endpoint, for example 400 for invalid requests, or 404 when an object is not found.
- You should also consider cases where other servers that you rely on are not available, such as 503 when an external API returns an error or is unavailable.
- Your code must be implemented in flask-restx and automatically generate a Swagger doc for testing the endpoints.
- Your code must be executable on CSE machines.
- Your code must not require installing any software, except for the permitted Python libraries.
- Your API endpoints must return appropriate responses in JSON format, and an appropriate HTTP response code. Do not return 500 Internal Server Error, as this signifies poor error handling and does not help the user.
- Ensure to comply with the correct types per the example responses. For example, do not use a string for a year “2024”.
- You are required to use the provided template, and to submit the **z[your zID].py** file. Please refer to WebCMS for up to date information about the due date and late penalties.
- You are also required to submit your **requirements.txt** (standard file containing all modules you require for running your script).