Rapport de TP

Développement d'une API pour la DGSE

19 octobre 2025

Ewen DENIAU
Nathan LEROUGE

CentraleSupélec
FISA
SIP

Professeur
Sébastien KILIAN
Fanny DIJOUD
Erwan FASQUEL
Lucas GIORDANI

CentraleSupélec

Sommaire

1 Introduction

Ce document est le rapport de rendu du TP de SIP. TP dans lequel nous devions développer une API où nous collections et assemblons plusieurs informations concernant des vols et des aéroports.

Le travail a été effectué par Ewen Deniau et Hakaroa Vallée dans le cadre du cours de SIP. Les scripts ont été programmés en Python, avec les librairies suivantes :

- Pandas
- SQLite
- Flask

Vous pourrez retrouver le code sur le repo suivant : Bismuth10K/flights-api

2 Database design

2.1 Question 1

La première tâche de ce projet consiste à élaborer un diagramme entité-relation (ER) pour identifier les différentes entités et les relations qui existent entre elles. Ce diagramme nous permet de visualiser les interactions entre des éléments essentiels comme les compagnies aériennes, les vols, les aéroports, les utilisateur et les pays. Par exemple, un vol est caractérisé par un aéroport de départ, une heure de départ, un aéroport et une heure d'arrivé, le type d'avion utilisé pour ce vol et son prix. Une fois ce modèle ER conçu, nous le traduisons en un modèle relationnel sous forme de tables dans une base de données relationnelles.

Chaque table doit contenir une clé primaire pour garantir l'unicité des enregistrements (par exemple, `flight_id` pour les vols). Les clés étrangères assurent la cohérence des relations entre les tables (comme un champ `isao_country_code` dans une table d'aéroports pour lier chaque aéroport au pays dans lequel il est situé). Nous adoptons la troisième forme normale (3NF), une norme de structuration qui permet d'éviter les redondances et de minimiser les anomalies lors des insertions, suppressions ou mises à jour.

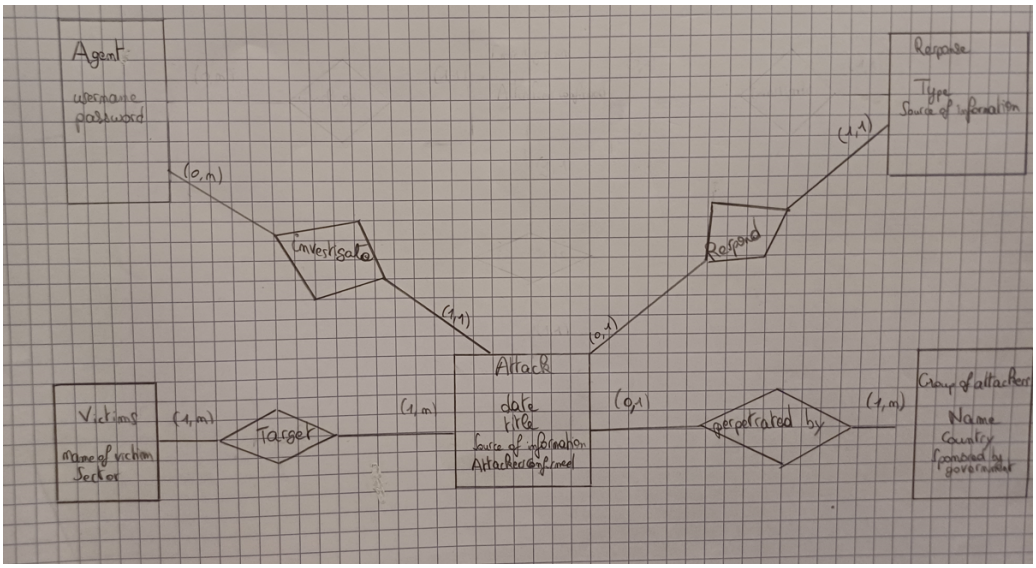

Figure 1. Diagramme entité-relation

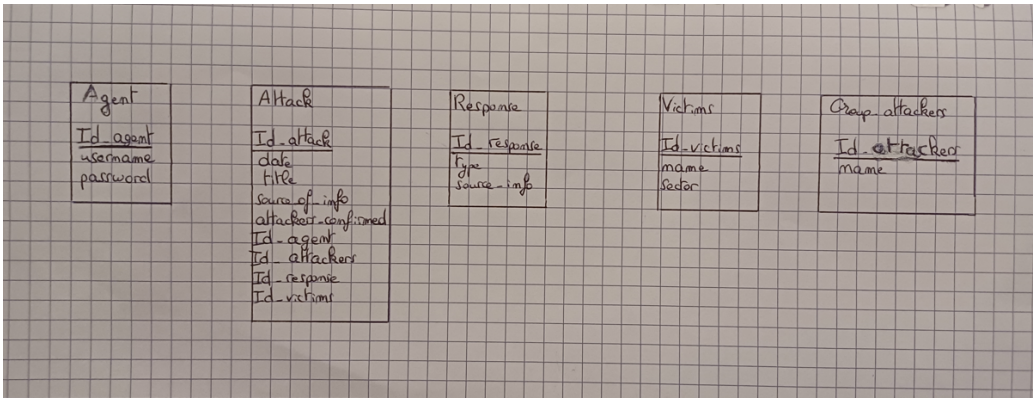

Figure 2. Modèle physique

3 *Create the database*

3.1 Question 2 et 3

Ce script permet de séparer le fichier all.csv en différentes tables sous la forme d'un *dataframe* pandas afin de structurer les données. Ainsi ce script renvoie une liste de *dataframes* pandas représentant chacun une table calquant le diagramme ER.

3.2 Question 5

Ce script permet de créer la base de données qui sera utilisée plus tard. L'organisation de celle-ci se base toujours sur le Diagramme ER en rajoutant des tables si nécessaire (*e.g.* la table `Booking` liant `User` et `Flights`).

3.3 Question 6

À l'aide des *dataframes* pandas et des tables SQL vides dans notre database, nous remplissons les tables des informations des *dataframes*. Pour cela, nous utilisons la commande SQL : `INSERT INTO ... VALUES ...`.

Pour certaines tables, une clé primaire "..._id" était nécessaire pour distinguer tous les vols, tous les utilisateurs ou toutes les réservations. Pour garantir l'unicité des identifiants, nous avons utilisé un incrémenteur automatique, ici le numéro de ligne.

3.4 Question 7

Dans la fonction `init_database()`, nous devons maintenant appeler correctement la fonction `populate_database()`. Cette fonction a besoin de `cursor` afin d'exécuter des requêtes SQL, et de `conn` qui permet la connexion à la BDD, ainsi que la liste des *dataframes* pandas contenant toutes les informations à remplir dans les tables SQL.

4 *First communications with the database*

4.1 Question 8

Nous avons lu, ce fut très fructueux.

4.2 Question 9

Il manquait l'appel de la méthode `conn.commit()`. Le *commit* sert à confirmer les modifications effectuées sur la base de données.

4.3 Question 10

Nous sommes dans le regret de vous annoncer qu'aucune erreur nous a marqué à l'heure où nous écrivons ces lignes. Nous écrivons le rapport après avoir fini le code, et nous nous sommes organisés de façon à ce que nous ignorions complètement ce qui était demandé dans cette question.

En vous remerciant de votre compréhension.

4.4 Question 11

Nous vous laissons lire l'implémentation. Nous avons aussi pris le soin de de hasher le mot de passe comme demandé plus tard.

4.5 Question 12

Quand nous essayons de modifier le mot de passe d'un utilisateur inexistant, cela ne marche pas. L'utilisateur n'est pas trouvé dans la BDD.

5 *Database manipulation*

5.1 Question 13

Cette série de fonction permet d'utiliser au mieux la database pour des demandes récurrentes. Par exemple : connaître le nom d'une ville lorsqu'on ne connaît que son code ISO, connaître toutes les informations d'un aéroport simplement avec son id. Cela permet d'utiliser la database efficacement.

6 *Create the first API routes*

6.1 Question 14

Ce script créé l'application (i.e. le serveur web) sur le port 5000 du serveur où sera hébergé le code. Le serveur web est géré par la librairie Flask.

6.2 Question 15

Pour obtenir tous les `users`, il faut juste aller à la racine du blueprint `users` comme en témoigne la ligne suivante :

```
1 @users_bp.route("/", methods=["GET"])
2 def get_all_users():
3     <code>
```

Le premier paramètre de la méthode route définit la suite de l'url, en l'occurence un slash simple donc la racine. Nous devons donc accéder à l'url suivant : `http://localhost:5000/users/`

Et enfin, il s'agit d'une requête `GET`.

6.3 Question 16

Pour les questions de ce genre, nous vous redirigeons vers le script `test_requests.py` et d'appeler les fonctions de votre choix.

6.4 Question 17

Nous vous redirigeons vers le script correspondant pour voir l'implémentation.

7 *Help to debug*

7.1 Question 18

Nous vous redirigeons vers le script correspondant pour voir l'implémentation. Nous avons aussi implémenté des fonctions supplémentaires afin de nous simplifier le travail et de compléter des fonctions non réalisés dans la bibliothèque db.

8 *Remaining routes*

8.1 Question 19

Même réponse que pour la 18, et nous y avons aussi créés des fonctions supplémentaires.

Idéalement, toutes ces fonctions devraient être confirmées par des tests unitaires. N'étant pas le sujet principal de ce TP, nous continuons ainsi.

9 *Securing the API*

9.1 Question 20 - 21

Nous vous laissons aussi voir l'implémentation. Elle n'est pas très poussée et suis globalement la documentation de la librairie.

Ensuite, nous appelons les fonctions adaptées dans la fonction `insert_user()`.

9.2 Question 22 - 24

Les tokens sont implémentés et fonctionnent correctement. Nous vous laissons voir l'implémentation encore une fois.

Pour le décorateur, nous avons surtout décidé de le mettre sur des fonctions ayant besoin d'une forme d'authentification. Par exemple, nous ne pouvons pas laisser n'importe qui observer la liste des voyageurs d'un vol, car la protection des données n'est pas respectée. Nous sommes aussi partis du principe que les agents allaient aussi avoir des *tokens*.

Certaines fonctions n'ayant pour but que de chercher tous les vols (par exemple) n'ont pas besoin de *tokens*. Puisqu'ils permettent à une personne d'observer les vols avant d'acheter, et donc de ce faire un compte.

Nous souhaitons vous faire remarquer deux choses :

- Premièrement, il serait pertinent de modifier les fonctions afin de lire l'utilisateur dans le *token*. Ainsi, nous pouvons lui assigner directement une réservation à son nom.
- Secondement, nous avons mentionné les agents. Nous pensons que rajouter des droits et des privilèges en fonction des gens rajouteraient une couche de sécurité.

Remarque : Lors de l'implémentation du *token*, nous avons remarqué que le code fourni appliquait un `split` et prenait le second élément pour récupérer le *token*. Après concertation avec madame Dijoud, j'ai eu son accord pour modifier cette partie afin de récupérer directement le *token* sans opérations supplémentaires.

10 Crédits

Nous voulons remercier les professeurs de leurs aides et de leurs cours lors du développement de ce TP. Nous remercions aussi tous les forums et tutos qui existent en ligne et qui nous ont aidé.

10.1 Document

Le template du document provient de Vincent Labatut (vincent.labatut@univ-avignon.fr) tel que modifié par Simon Leclercq (leclercq.e2102543@etud.univ-ubs.fr). Les modifications supplémentaires ainsi que la page de garde proviennent d'Ewen Deniau (ewen.deniau@student-cs.fr).