Rapport SEV: web services and information systems

**Table des matières**

[***Introduction :***](#_heading=h.gjdgxs) ***1***

[***1. Schéma de la base de données relationnelle :***](#_heading=h.30j0zll) ***1***

[***2. Peuplement de la base de données :***](#_heading=h.1fob9te) ***2***

[***3. Requête :***](#_heading=h.3znysh7) ***2***

[**3.a. GET :**](#_heading=h.2et92p0) **3**

[**3.b. POST :**](#_heading=h.tyjcwt) **3**

[**4. Conclusion:**](#_heading=h.99nstvo4fun7) **4**

# Introduction :

Dans le cadre de cette unité, nous avons réalisé un projet dans lequel nous créons une base de données relationnelle à partir d’un dossier contenant des mesures pour plusieurs étudiants, pour plusieurs examens ainsi que plusieurs capteurs comme par exemple la température. Toutes ces données sont stockées dans différents fichiers CSV. L’objectif de ce projet une fois la production de la base de données terminée est de correctement gérer et implémenter des requêtes sur celle-ci en prenant en compte les droits des utilisateurs, en passant par une authentification, les headers devant être corrects, ainsi que la méthode qui elle aussi doit être reconnue.

# Schéma de la base de données relationnelle :

Cette base de données relationnelle contient cinq tables qui représentent les informations sur les étudiants, les examens, les capteurs, les mesures effectuées par les capteurs pendant les examens et les notes obtenues par les étudiants pendant les examens.

-La table Student contient les informations sur les étudiants avec leur identifiant unique Id\_Student généré automatiquement, un numéro d'étudiant Number qui est unique et est utilisé comme clé primaire.

-La table Exam contient les informations sur les examens avec leur identifiant unique Id\_Exam généré automatiquement et un nom d'examen Name qui est unique.

-La table Sensor contient les informations sur les capteurs avec leur identifiant unique Id\_Sensor généré automatiquement, un nom abrégé Short\_Name et un nom complet Name qui sont tous deux uniques.

-La table Measurement contient les informations sur les mesures effectuées par les capteurs pendant les examens, y compris l'identifiant de l'étudiant Id\_Student, l'identifiant de l'examen Id\_Exam, l'identifiant du capteur Id\_Sensor, l'heure d'acquisition Acquisition\_time et la valeur de mesure Measurement\_value. Cette table utilise des clés étrangères pour faire référence aux informations dans les tables Student, Exam et Sensor.

-La table Grade contient les notes obtenues par les étudiants pendant les examens avec l'identifiant de l'étudiant Id\_Student, l'identifiant de l'examen Id\_Exam et la valeur de la note Grade\_value. Cette table utilise également des clés étrangères pour faire référence aux informations dans les tables Student et Exam.

En résumé, cette base de données relationnelle est conçue pour stocker les informations sur les étudiants, les examens, les capteurs, les mesures effectuées pendant les examens et les notes obtenues par les étudiants. Les relations entre les tables sont définies à l'aide de clés étrangères pour garantir l'intégrité des données.

# Peuplement de la base de données :

Afin de créer notre base de données SQL, nous avons créé un fichier python nommé data.py. Pour ce faire, nous avons dans un premier temps utilisé l’outil ChatGPT. Nous lui avons donné le contexte de notre projet ainsi que l'arborescence des dossiers et fichiers de notre base de données afin de créer ce fichier python qui parcourt ces dossiers et récupère les informations. Naturellement, le code produit par ChatGPT n’était pas totalement conforme et prêt à l'emploi mais celui-ci nous a fait gagner beaucoup de temps.

Après avoir réalisé ce parcours de dossier, nous créons une requête SQL pour chaque donnée avec un “INSERT IGNORE INTO”. Etant donné la quantité de données nous avons rajouté une condition pour ne récupérer qu’une petite quantité de données pour chaque capteur. De plus, nous avons ajouté une donnée ‘acquisition\_time’ calculée à partir du timestamp initial de la mesure pour chaque capteur auquel nous avons ajouté la fréquence de mesure à chaque itération. La dernière étape à modifier était de changer le format de date sur nos données temporelles afin de correspondre au format SQL, ce que nous avons réalisé avec la fonction .strftime()

# Requête :

Nous avons implémenté dans ce projet deux requêtes différentes permettant d'interagir avec la base de données. La première est GET, elle nous permet, si nous avons les droits, de lire les données contenues dans notre base afin d’accéder aux mesures prises sur chaque élève ainsi que leurs notes. La seconde est la méthode POST, elle nous permet d’ajouter des données à la base, dans le cas ou de nouvelles mesures ont été réalisées. Il est important pour cette requête aussi, de vérifier que l’utilisateur possède les droits nécessaires pour réaliser cette opération. Toutes les vérifications sont réalisées dans le code python messages.py dans lequel nous appelons les fonctions du fichier wslib.py.

## 3.a. GET :

La requête GET va nous permettre de récupérer des données sur la table ‘Measurements’. Nous avons implémenté le fait que l’utilisateur doit fournir l’élève (S02), l’examen et le capteur pour récupérer les valeurs du capteur ainsi que l’heure à laquelle la mesure a été faite. Dans notre client il faut donc donner toutes ces informations dans l’URI, pour faire cela on crée un dictionnaire avec le nom de ces variables et les valeurs données par l’utilisateur. Ce dictionnaire sera utilisé en tant que paramètre *‘params’* de la fonction ‘*requests.get’.* Le serveur recevra donc une requête GET avec le précédent dictionnaire, pour exécuter la requête il est nécessaire de traiter correctement ce dictionnaire.

Pour faire cela on a créé une fonction *‘readRessource’* qui récupère le dictionnaire, dans un premier temps récupère les ID de l’élève, l’examen et le capteur tout en vérifiant que cela existe, si une des données n’existe pas alors on retourne (termine), sinon on construit une requête SQL que on va envoyer au serveur et on retournera la réponse du serveur qui contient les données demandées, ainsi l’utilisateur pourra voir les valeurs.

## 3.b. POST :

Comme dit précédemment une requête POST est utilisée pour ajouter des données. Nous voulons donc envoyer des données dans la table ‘Measurements’, il faut donc donner les champs de la cette table qui sont: ‘Id\_Stuent’, ‘Id\_Exam’, ‘Id\_Sensor’, ‘Acquisition\_time’, ‘Measurment\_value’.

Dans notre client il faut donc donner toutes ces informations dans l’URI, pour faire cela on crée un dictionnaire avec le nom de ces variables et les valeurs données par l’utilisateur. Ce dictionnaire sera utilisé en tant que paramètre *‘params’* de la fonction ‘*requests.post’.* Le serveur recevra donc une requête POST avec le précédent dictionnaire, pour exécuter la requête il est nécessaire de traiter correctement ce dictionnaire.

Nous avons donc créé une fonction *‘PostRessource’* dans laquelle on utilise la fonction *‘readRessource’*, si cette fonction retourne quelque chose c’est que les valeurs données par l’utilisateur sont déjà dans la base de données et que donc il est inutile de faire le POST, on retourne donc un message le précisant. Actuellement notre code ne vérifie que pour les paramètres ‘Id\_Stuent’, ‘Id\_Exam’, ‘Id\_Sensor’, il devrait aussi vérifier ’acquisition\_time’ car il est logique qu'on veuille rajouter des données d’un étudiant, d’un capteur sur un exam mais du coup avec un timestamp différent.

Ensuite, on vérifie que toutes les données du dictionnaire sont présentes. Si ce n’est pas le cas, on retourne un message d’erreur.

Après cette vérification on met en forme la requête SQL avec les données, on se connecte à la base de données, et on exécute la requête. Si quelque chose se passe mal on à une vérification qui retourne un message d’erreur sinon on retourne un code de réussite.

Pour améliorer l’expérience utilisateur, on pourrait rajouter des transformations automatiques des informations que l’utilisateur donne, par exemple la on impose le format du timestamp mais on pourrait faire en sorte que peu importe la façon dont on le donne on le transforme de la bonne façon. Ou alors lui demander les informations une par une et les manipuler à notre guise.

Les utilisateurs peuvent aussi avoir des fichiers .csv avec les données, dans notre projet on ne traite pas ces fichiers. Cependant pour le faire on peut imaginer plusieurs façons. Par exemple, on peut créer une interface pour que l’utilisateur puisse déposer son fichier pour l’envoyer au serveur et que le serveur le traite de la même manière que le fichier *data.py* pour exécuter toutes les requêtes possibles. Sinon on pourrait faire la même chose dans le fichier client et ainsi faire plusieurs requêtes POST issues du fichier.

# Conclusion:

Pour conclure sur ce projet, nous sommes heureux d’avoir réussi à tout implémenter et rendre un produit pleinement fonctionnel et répondant aux objectifs que nous nous étions fixés avec la gestion des headers, des droits et des requêtes. En revanche, beaucoup d’améliorations peuvent être apportées à ce projet. Par exemple, nos requêtes sont assez basiques, on pourrait rajouter des GET pour chaque table afin de récupérer toutes les informations que l'on veut plus facilement ou encore faire des POST pour rajouter un type d’examen uniquement, il faudrait donc créer une fonction pour chaque possibilité de requête. Il faudrait aussi préparer une input différente pour le client dans chaque cas. Les scripts python deviendraient alors extrêmement longs, ce qui n’était pas forcément l’objectif ici. Grâce au format de ce projet, nous avons pu explorer et développer les bases de ce genre de systèmes tout en comprenant jusqu’où nous pouvions aller avec plus de temps. Notamment après avoir développé les méthodes GET et POST, nous pouvons imaginer d’autres requêtes comme par exemple les méthodes DELETE ou PUT.