**Question 1**

Un diagramme entité-relation, également nommé diagramme ER est un schéma graphique avec les entités et les relations liant ces entités. Dans cette première étape, nous avons commencé par modéliser la base de données sous la forme d’un schéma Entité-Relation (ER). Il était essentiel d’identifier les entités principales et les relations pertinentes entre elles afin de structurer correctement les informations sur les incidents de cybersécurité et les agents qui s'en occupent. Nous avons déterminé que les entités suivantes devaient figurer dans notre modèle : les agents, les incidents, les attaquants, les victimes et les réponses aux attaques. Un agent est responsable de la collecte d’informations sur les incidents, et chaque incident peut contenir des détails tels que la date, le titre, le type d’attaque, une description et les sources d’information. Il est aussi possible de documenter les secteurs industriels touchés par l’attaque ainsi que les attaquants impliqués, en précisant le pays d’origine et s’ils sont parrainés par un gouvernement.

Après avoir identifié ces entités, nous avons défini les relations entre elles. Chaque incident est enregistré par un agent, et il peut impliquer plusieurs attaquants et cibler plusieurs victimes. Il arrive aussi que des attaquants soient suspectés mais non confirmés, ce qui nécessite une gestion particulière de cette information. En ce qui concerne les réponses, chaque attaque peut susciter une réaction de la part des victimes, composée d’un type de réponse et de la source d’information associée. Le modèle traduit en diagramme ER a permis de clarifier ces relations et de poser les bases de notre travail sur la base de données relationnelle.

Une fois le schéma ER établi, nous avons procédé à la traduction du modèle en tables relationnelles. Chaque entité est devenue une table avec une clé primaire unique pour garantir l’unicité des enregistrements. Nous avons veillé à respecter la 3ème forme normale (3NF), une étape cruciale pour éviter les redondances et maintenir l’intégrité des données. Par exemple, la table des incidents comprend un identifiant unique pour chaque incident, des champs pour le titre, la date, la description, ainsi que des clés étrangères pointant vers les table des agents, des victimes, des attaquants et de la réponse.

**Question 2**

Après la conception de la structure de la base de données, nous avons reçu un fichier CSV contenant des informations sur les incidents. Nous avons constaté que la colonne "Response" de ce fichier regroupait deux informations distinctes : le type de réponse et la source de cette réponse. Afin de rendre la structure des données plus claire et plus cohérente, nous avons décidé de diviser cette colonne en deux nouvelles colonnes, intitulées respectivement "type of response" et "source of response". Cette séparation permet une manipulation plus précise des données et facilite leur exploitation ultérieure.

Le processus de transformation a consisté à lire le fichier CSV, extraire les parties pertinentes avant et après le point-virgule pour former les deux nouvelles colonnes, puis générer un nouveau fichier CSV avec les modifications. Une fois la transformation terminée, ce nouveau fichier sera utilisé pour alimenter la base de données avec les informations correctement structurées. Cette séparation garantit une plus grande clarté dans l’exploitation des données, notamment lors des requêtes spécifiques sur le type ou la source de réponse.

**Question 3**

En examinant plus en détail le fichier CSV, nous avons également remarqué que certaines informations sur les attaquants n’étaient pas toujours confirmées. Cela a posé un problème, car il est essentiel de savoir si un attaquant est formellement identifié ou s’il s’agit simplement d’une hypothèse. Pour résoudre ce problème, nous avons décidé d’ajouter une nouvelle colonne binaire intitulée "confirmé", qui prend la valeur 1 si l’implication du groupe attaquant est confirmée, et 0 dans le cas contraire. Cette nouvelle colonne permet de mieux suivre l’état des enquêtes et de distinguer les cas avérés des cas suspects.

Le processus de modification a consisté à parcourir le fichier CSV, identifier les groupes d'attaquants pour lesquels il n’y avait pas de certitude, et ajouter une entrée correspondante dans la nouvelle colonne "confirmé". Cette information est précieuse pour les agents, car elle facilite le suivi des cas suspects et la production de rapports sur l’évolution des enquêtes. Une fois la colonne ajoutée, nous avons généré un nouveau fichier CSV mis à jour qui sera utilisé pour peupler la base de données avec ces informations structurées.

**Question 4**

Nous avons commencé l’étape d’initialisation de la base de données en modifiant la fonction load\_config() dans le fichier utils.py. Cette fonction a pour rôle de lire le fichier de configuration situé dans ./config/config/, qui contient les paramètres nécessaires à la configuration de la base de données. Ces paramètres incluent le chemin vers la base de données, les options de journalisation ou encore les modes de connexion. En automatisant cette lecture, nous garantissons que la base est toujours initialisée de manière cohérente et en accord avec les spécifications attendues, évitant ainsi des erreurs de configuration manuelle. Cette approche est particulièrement utile pour garantir la portabilité de l’application, en permettant des ajustements rapides selon l’environnement (développement, test ou production).

Une fois cette étape accomplie, nous avons implémenté la fonction create\_database() dans le fichier ./db/init.py. Cette fonction est essentielle puisqu’elle crée toutes les tables de la base à partir du modèle relationnel que nous avons défini. Chaque table a été créée avec une structure précise : par exemple, les colonnes d’identifiants sont du type INTEGER et définies comme PRIMARY KEY pour assurer l’unicité des enregistrements. De plus, les mots de passe sont stockés sous forme binaire avec le type BINARY(256) pour garantir la sécurité des données sensibles. Les champs booléens, quant à eux, utilisent le type INTEGER avec 0 ou 1 pour représenter respectivement des valeurs fausses ou vraies.

Après avoir implémenté la création des tables, nous avons exécuté le fichier test\_create\_db.py, ce qui nous a permis de vérifier que le fichier incidents.db avait bien été généré dans le répertoire ./data/. Cette validation est cruciale, car elle garantit que toutes les tables ont été créées correctement et que la base est prête à recevoir des données.

**Question 5**

Pour mettre en place une base de données fonctionnelle, il est essentiel de disposer d'une fonction capable d'intégrer les données contenues dans un fichier CSV au sein de la base. Cette tâche repose sur la fonction create\_database() située dans le fichier init.py. Son rôle est non seulement de lire les informations présentes dans le CSV, mais également de s'assurer que chaque donnée soit insérée correctement dans les tables SQL appropriées.

Le fichier CSV fourni par les superviseurs contient des informations diverses, telles que les agents impliqués, les incidents observés, et les réponses apportées. Une fois les informations extraites, elles doivent être distribuées aux bonnes tables de manière cohérente. Par exemple, si un enregistrement mentionne un agent responsable, celui-ci doit être inséré dans la table des agents, et son nom doit être référencé dans les incidents liés via une clé étrangère. Chaque table suit un schéma strict, avec des contraintes d'intégrité référentielle pour garantir la cohérence entre les données.

Le traitement de ces données repose sur des requêtes paramétrées, une méthode qui permet d’exécuter des commandes SQL en séparant les valeurs des instructions. Cela renforce la sécurité du système en évitant les attaques par injection SQL. Par ailleurs, chaque insertion est gérée à l’intérieur d’une transaction : si une erreur survient au cours du processus, toutes les opérations sont annulées afin de ne pas laisser la base de données dans un état partiel ou incohérent.

Une fois l'insertion terminée, il est crucial de valider l'ensemble des modifications en utilisant une commande de commit. Cette étape permet d'enregistrer définitivement les changements effectués. Des tests ont ensuite été menés pour vérifier que les informations du fichier CSV étaient correctement reflétées dans la base et que toutes les relations entre les tables étaient respectées.

**Question 6**

L’étape suivante consistait à automatiser la procédure d’initialisation complète de la base de données, ce qui comprend la création des tables et l'insertion des données. Cela a été réalisé en modifiant la fonction principale init\_database() afin qu’elle enchaîne automatiquement ces opérations sans intervention manuelle. L’objectif était de s’assurer que, dès le lancement de l’application, la base soit prête à fonctionner avec les données requises.

Un point important de cette procédure était de respecter l'ordre logique d'insertion des informations. Certaines entités, comme les incidents, dépendent d'autres informations (par exemple, les agents ou les cibles) via des clés étrangères. Si les agents ou les cibles n’étaient pas présents avant d’insérer les incidents, des erreurs de contraintes auraient bloqué le processus. Ainsi, nous avons d'abord inséré les agents, les attaquants, et les cibles, avant de finaliser avec les incidents et les réponses.

Par ailleurs, une gestion fine des erreurs a été mise en place. Si une erreur d’insertion ou une violation de contrainte survenait, un message d’erreur clair est généré pour faciliter le diagnostic. En outre, pour éviter la duplication des informations, nous avons introduit des vérifications supplémentaires : avant d’insérer un enregistrement, nous vérifions s'il est déjà présent. Cette logique préventive permet de garantir l’intégrité et la cohérence de la base.

**Question 7**

Nous avons ensuite étudié le fonctionnement des requêtes SQL paramétrées à travers un exemple concret d'insertion d’un agent dans la base. Cette opération est effectuée par une fonction située dans agents.py. Son fonctionnement repose sur l’utilisation de requêtes avec des paramètres, ce qui signifie que les valeurs sont introduites de manière sécurisée en dehors de l'instruction SQL principale. Cela empêche toute tentative d’injection SQL et améliore la sécurité générale du système.

La fonction doit également gérer les erreurs liées aux contraintes d’unicité. Par exemple, si l’on essaie d’ajouter un agent dont le nom existe déjà dans la table, une violation de contrainte se produit. Le code intercepte cette erreur et renvoie un message approprié, indiquant que l'agent est déjà enregistré.

Lors des tests, nous avons tenté d'insérer un agent nommé "Hubert". Cependant, cet agent n’apparaissait pas dans la liste après l’insertion. L’analyse du problème a révélé que la cause principale était l’absence de commit après l'opération d'insertion. Sans validation explicite, les modifications restent en mémoire temporairement et sont annulées automatiquement à la fin de la session. Après avoir corrigé cette erreur en ajoutant la commande de commit, nous avons confirmé que l'agent "Hubert" était bien visible dans la base de données.

**Question 8**

Une fois ce problème résolu, nous avons procédé à d’autres tests et rencontré une nouvelle erreur liée aux contraintes de clé étrangère. Cela s'est produit lorsque nous avons tenté d'insérer un incident sans que l'agent correspondant soit déjà présent dans la table des agents. Cette erreur illustre l’importance de respecter l'ordre d’insertion des entités. Les relations entre les tables étant définies par des clés étrangères, il est impératif que chaque entité référencée soit déjà créée au moment de l’insertion.

Pour corriger cette erreur, nous avons réorganisé le processus d’initialisation de manière à insérer toutes les entités de base (comme les agents, attaquants, et cibles) avant de passer aux incidents. De plus, nous avons ajouté des vérifications préalables dans le code : avant d'insérer un incident, nous vérifions si toutes les entités mentionnées existent dans la base. Si une dépendance est manquante, un message d’erreur explicite est affiché, ce qui permet de détecter rapidement l’origine du problème et d'éviter des incohérences.

**Question 9**

Nous avons également travaillé sur la mise à jour des mots de passe des agents. La fonction concernée prend en paramètre le nom de l’agent et le nouveau mot de passe. Elle vérifie ensuite si l'agent existe dans la base avant de procéder à la mise à jour. Si l'agent est trouvé, son mot de passe est remplacé par le nouveau et un message de succès est affiché.

Cependant, nous avons constaté qu’il était possible qu’un agent inexistant soit ciblé pour une mise à jour. Dans ce cas, aucune modification n'est apportée à la base. Ce comportement est correct, car il évite de créer des entrées fantômes ou de corrompre des enregistrements. Pour améliorer l'expérience utilisateur, nous avons également ajouté un message indiquant si la mise à jour a échoué parce que l'agent n'existait pas. Cela permet de mieux comprendre le résultat de l’opération et d'éviter toute confusion.

**Question 10**

Après avoir corrigé les erreurs précédentes et effectué une nouvelle tentative de mise à jour du mot de passe d’un agent, nous avons rencontré une nouvelle erreur liée à la gestion des transactions. Cette erreur survient car la mise à jour était tentée sans que la connexion à la base de données soit maintenue jusqu'à la fin de l'opération, ou encore parce que la commande de *commit* n'était pas exécutée après la mise à jour.

Cela souligne l'importance d'une gestion correcte des transactions dans une base de données. Chaque opération d'insertion, de mise à jour ou de suppression doit être encadrée par un *commit* pour s’assurer que les modifications soient définitivement appliquées. En l’absence de cette validation explicite, SQLite (comme d’autres systèmes de gestion de base de données) considère que les changements ne doivent pas être enregistrés, et ils sont annulés.

Nous avons également découvert que l'erreur pouvait être liée à une mauvaise gestion des exceptions. Si une exception est levée (par exemple, une tentative de mise à jour pour un agent inexistant), la transaction doit être annulée proprement avec une commande *rollback* pour rétablir l'état initial de la base. Après l'ajout de cette gestion d'erreurs améliorée, nous avons confirmé que l'application réagissait correctement même en cas de problème, sans compromettre l’intégrité des données.

**Question 11**

La tâche suivante portait sur l’amélioration de la fonction permettant de mettre à jour les mots de passe des agents. Cette étape a demandé une modification de la fonction pour garantir que les informations sensibles soient stockées de manière sécurisée. Nous avons donc opté pour un hachage cryptographique des mots de passe avant de les enregistrer. Cela permet de protéger les données, même en cas de fuite de la base.

Le choix de l’algorithme de hachage a été crucial. Nous avons utilisé SHA-256, une méthode de hachage reconnue pour sa robustesse. Ainsi, lorsqu’un mot de passe est fourni par un agent, celui-ci est d’abord converti en une empreinte cryptographique qui est ensuite comparée à celle stockée en base. Cela garantit qu'un mot de passe en clair n'est jamais directement enregistré.

Une autre amélioration importante consistait à vérifier que le mot de passe respecte certaines règles de sécurité (par exemple, une longueur minimale et la présence de caractères spéciaux) avant d’autoriser sa mise à jour. Cette vérification renforce la sécurité du système et empêche les agents de définir des mots de passe trop faibles.

**Question 12**

Lorsque nous avons tenté de mettre à jour le mot de passe d’un agent inexistant dans la base, nous avons constaté que la fonction de mise à jour ne retournait aucune erreur explicite. Cela aurait pu conduire à une confusion pour l'utilisateur, qui pourrait penser que la mise à jour a réussi. Pour corriger cela, nous avons modifié la fonction afin qu’elle vérifie d’abord l’existence de l’agent avant de procéder à la mise à jour.

Si l'agent n'existe pas, un message d'erreur est désormais généré, signalant que la mise à jour n’a pas pu être effectuée. De plus, nous avons décidé de renvoyer une valeur booléenne indiquant le succès ou l'échec de l'opération. Cela permet aux développeurs qui utilisent cette fonction de gérer correctement l’échec de la mise à jour. En utilisant cette approche, nous nous assurons que l'application soit plus fiable et que les erreurs soient facilement identifiables par les utilisateurs et les développeurs.

**Question 13**

La dernière partie du projet consistait à implémenter une série de fonctions permettant de gérer les différentes entités de la base de données. Ces fonctions devaient être réparties dans plusieurs fichiers correspondant aux entités spécifiques : attaquants, cibles, réponses, sources, et incidents.

Dans le fichier attackers.py, nous avons implémenté trois fonctions essentielles. La première, insert\_attacker(), permet d’ajouter un nouvel attaquant dans la base en s’assurant que ses informations (comme le nom et le pays) soient correctement enregistrées. La deuxième fonction, update\_attacker\_sponsor(), permet de mettre à jour les informations concernant le parrainage éventuel de l’attaquant. Cette fonction est particulièrement utile pour refléter des changements d’informations au fur et à mesure que l’enquête progresse. Enfin, la fonction get\_attackers() récupère la liste de tous les attaquants enregistrés dans la base.

Dans targets.py, deux fonctions ont été ajoutées. La fonction insert\_target() permet d'insérer une nouvelle cible dans la base, en spécifiant son secteur d'activité. La fonction get\_targets() renvoie quant à elle toutes les cibles enregistrées, ce qui est crucial pour les rapports d'incidents.

Dans le fichier responses.py, nous avons implémenté insert\_response(), qui ajoute une réponse à un incident en spécifiant le type de réponse et sa source. La fonction get\_response() permet de récupérer les informations de réponse pour un incident donné.

De manière similaire, dans le fichier sources.py, nous avons ajouté deux fonctions. La première, insert\_source(), permet d’ajouter une nouvelle source d’information liée à un incident. La seconde, get\_sources(), retourne toutes les sources associées aux incidents, offrant une visibilité sur l'origine des informations collectées.

Enfin, dans le fichier incidents.py, nous avons développé une série de fonctions pour gérer les incidents. La fonction insert\_incident() permet d’enregistrer un nouvel incident en liant les informations pertinentes (comme l'attaquant, la cible, la date et la description). get\_incident() récupère les détails d’un incident donné. D'autres fonctions, comme update\_incident\_attacker() et update\_incident\_response(), permettent de mettre à jour respectivement l’attaquant et la réponse associés à un incident.

De plus, nous avons implémenté des fonctions pour gérer les relations entre incidents et autres entités. Par exemple, add\_incident\_target() et remove\_incident\_target() permettent d'ajouter ou de retirer des cibles associées à un incident. Des fonctions similaires, comme add\_incident\_source() et remove\_incident\_source(), gèrent les sources d’informations. Ces fonctions garantissent que l'application reste flexible et capable de s'adapter à des informations changeantes au fil de l'enquête.Passer de 3 à 6 tables, pour la table victimes séparer les victimes par une virgule ou un and (pas grave si parfois pas de sens).

**Question 14**

En examinant le fichier app.py, nous découvrons qu'il sert de point d'entrée pour l'application Flask. Lorsque nous exécutons la commande python app.py, cela initialise le serveur web qui écoute les requêtes entrantes sur le port 5000 de localhost. Ce fichier joue un rôle crucial, car il configure l’application et définit les routes qui permettent d’interagir avec les données des agents.

Lorsque l'application est lancée, elle crée une instance du serveur Flask qui se charge de traiter les requêtes HTTP. Les utilisateurs peuvent alors effectuer des actions comme récupérer des informations sur les agents, ajouter de nouveaux agents, ou même mettre à jour leurs détails. L'application, dans son état actuel, se concentre principalement sur les informations relatives aux agents, ce qui en fait un outil de gestion utile pour la DGSE afin de suivre et gérer leurs agents affectés aux enquêtes sur les incidents de cybersécurité.

**Question 15**

En se penchant sur le fonctionnement de l'application, nous notons qu'elle se concentre principalement sur la gestion des agents. Les routes correspondantes sont partiellement définies dans le fichier ./routes/agents.py. Pour obtenir la liste de tous les agents, les utilisateurs doivent effectuer une requête GET sur le chemin d'accès spécifié : /agents/. Ce type de requête est particulièrement adapté pour récupérer des informations sans les modifier, ce qui est exactement le besoin ici.

Le choix de la méthode GET est pertinent car il s'agit d'une méthode standard dans le développement d'API REST pour récupérer des données. Lorsqu'un utilisateur fait une telle requête, le serveur interroge la base de données et renvoie la liste des agents enregistrés au format JSON, facilitant ainsi l'intégration avec d'autres systèmes ou outils.

**Question 16**

Pour illustrer l’interaction avec l’API, nous avons conçu un script en Python utilisant la bibliothèque requests, qui permet d'envoyer des requêtes HTTP facilement. En envoyant une requête de type GET à l'adresse http://localhost:5000/agents/, le script interroge le serveur pour obtenir des informations sur les agents.

Le fonctionnement de ce script repose sur l'envoi d'une requête à l'API, suivie de la vérification de la réponse. Si la requête réussit, le serveur renvoie une réponse avec un code de statut 200, signifiant que les informations ont été récupérées avec succès. Cela permet de s'assurer que le serveur fonctionne comme prévu et que les agents sont accessibles. Dans le cas contraire, le script informe de l'échec de la requête en affichant le code d'erreur correspondant.

**Question 17**

Dans le fichier ./routes/agents.py, nous avons enrichi l'API avec plusieurs fonctions essentielles. La fonction get\_agent() a été développée pour permettre aux utilisateurs de récupérer les informations d'un agent spécifique basé sur son nom d'utilisateur. En plus de cela, elle fournit la liste des incidents qui ont été assignés à cet agent, facilitant ainsi le suivi de ses activités.

La fonction patch\_password() a été conçue pour permettre la mise à jour du mot de passe d'un agent. Ce processus implique la validation des nouvelles informations et leur enregistrement dans la base de données de manière sécurisée, en s'assurant que le mot de passe est toujours stocké sous forme de hash.

Enfin, la fonction add\_agent() permet l'ajout d'un nouvel agent dans la base de données. Cela inclut la collecte des informations nécessaires à l'enregistrement, telles que le nom d'utilisateur et le mot de passe, tout en s'assurant que le nom d'utilisateur est unique pour éviter les doublons. Ces ajouts contribuent à la gestion efficace des agents au sein de l'organisation.

**Question 18**

Nous avons développé plusieurs fonctions dans le fichier ./routes/data.py pour faciliter le débogage de la base de données. Ces fonctions sont cruciales pour identifier et résoudre les problèmes potentiels au sein des données. Elles permettent de vérifier la cohérence des relations entre les différentes entités de la base, comme les incidents et les agents, et s'assurent que toutes les entrées nécessaires sont présentes.

Par exemple, une fonction pourrait vérifier si tous les incidents référencés dans la base de données ont un agent attribué, ce qui est essentiel pour maintenir l'intégrité des données. D'autres fonctions pourraient aider à lister les enregistrements manquants ou erronés dans des colonnes clés, facilitant ainsi le nettoyage et l'organisation des données. Cela garantit que la base de données reste fiable et opérationnelle, ce qui est primordial pour le bon fonctionnement de l’application.

**Question 19**

Dans le fichier ./routes/incidents.py, nous avons mis en œuvre toutes les fonctions nécessaires pour gérer efficacement les incidents. Cela inclut des fonctions telles que insert\_incident(), qui permet d'ajouter un nouvel incident à la base de données. Cette fonction enregistre des détails essentiels tels que la date de l'incident, sa description et les parties impliquées, ce qui est fondamental pour le suivi des incidents de cybersécurité.

La fonction get\_incident() est conçue pour récupérer les détails d'un incident spécifique, facilitant ainsi l'accès à l'historique des incidents. D'autres fonctions, comme update\_incident\_attacker() et update\_incident\_response(), permettent de mettre à jour les informations relatives à l'attaquant et à la réponse à l'incident, respectivement. En outre, des fonctions permettent d'ajouter ou de supprimer des cibles et des sources associées à un incident, assurant ainsi que toutes les informations pertinentes restent à jour.

**Question 20**

Pour renforcer la sécurité de l'application, nous avons introduit des fonctions utilisant la bibliothèque bcrypt, qui permet de gérer le stockage sécurisé des mots de passe. Au lieu de conserver les mots de passe en clair, nous avons modifié le processus d'enregistrement pour créer un hash sécurisé du mot de passe à l'aide de l'algorithme bcrypt. Cela signifie que même si la base de données est compromise, les mots de passe des utilisateurs restent protégés.

La fonction hash\_password() génère un hash pour un mot de passe donné, tandis que check\_password() compare un mot de passe en clair avec un hash existant pour vérifier sa validité. Enfin, la fonction check\_agent() utilise ces outils pour valider les identifiants d'un agent lors de la connexion, garantissant ainsi que seuls les utilisateurs authentifiés peuvent accéder à l'application.

**Question 21**

Nous avons modifié la fonction insert\_agent() pour stocker les mots de passe sous forme de hash dans la base de données. Cela implique de transformer le mot de passe fourni par l'utilisateur en un hash avant de l'enregistrer. Cette approche permet d'améliorer considérablement la sécurité des mots de passe, car même si la base de données est accessible, les mots de passe en clair ne peuvent pas être récupérés.

**Question 22**

Pour résoudre le problème d'authentification, nous avons mis en place un système de tokens JWT (JSON Web Tokens). Ce mécanisme fonctionne en exigeant que l'utilisateur effectue une requête initiale avec ses identifiants, ce qui, en cas de succès, génère un token unique. Ce token peut ensuite être utilisé pour authentifier les requêtes futures à l'API.

Le processus est conçu pour renforcer la sécurité des communications entre l'utilisateur et le serveur. Chaque fois qu'un utilisateur souhaite accéder à une ressource protégée, il doit inclure le token dans l'en-tête de sa requête, ce qui permet au serveur de vérifier son identité et d'accorder l'accès aux informations demandées.

**Question 23**

Pour permettre aux utilisateurs d'obtenir un token, nous avons créé une route dédiée /login. Cette route gère la vérification des identifiants fournis par l'utilisateur. Si les identifiants sont valides, le serveur génère un token JWT qui est renvoyé à l'utilisateur. Ce token peut ensuite être utilisé pour effectuer des requêtes sécurisées à l'API, facilitant ainsi l'accès aux fonctionnalités de l'application tout en garantissant la sécurité des données.

**Question 24**

Enfin, nous avons introduit un décorateur appelé token\_required(), qui assure que les routes critiques de l'application sont protégées par une vérification de token. Ce décorateur est ajouté à certaines routes pour s'assurer que seules les requêtes contenant un token JWT valide peuvent accéder à des informations sensibles, telles que les données des agents et les incidents.

Cette protection est essentielle pour garantir que seuls les utilisateurs authentifiés peuvent interagir avec les données critiques, ce qui renforce considérablement la sécurité de l'application. Les routes protégées ne seront accessibles qu’après validation du token, empêchant ainsi tout accès non autorisé aux informations sensibles.