SAE Web Service :

Questions API REST :

Comment avez-vous défini les endpoints de votre API REST pour répondre aux besoins fonctionnels de votre application ?

Les endpoints ont été définis dans les contrôleurs avec des annotations comme @GetMapping, @PostMapping, @PutMapping, et @DeleteMapping. Chaque endpoint correspond à une action spécifique sur la ressource en question. Par exemple pour Teacher, cela correspond à : récupérer tous les enseignants, récupérer un enseignant par son ID, ajouter un nouvel enseignant, mettre à jour un enseignant existant, et supprimer un enseignant.

Quelles sont les différentes méthodes HTTP que vous avez utilisées dans votre API REST et pourquoi ?

J'ai utilisé les méthodes HTTP suivantes :

GET pour récupérer des données, par exemple, pour récupérer tous les enseignants ou un enseignant par son ID.

POST pour créer de nouvelles ressources, par exemple, pour ajouter un nouvel enseignant.

PUT pour mettre à jour des ressources existantes, par exemple, pour mettre à jour les informations d'un enseignant.

DELETE pour supprimer des ressources, par exemple, pour supprimer un enseignant existant.

Comment avez-vous respecté le principe "give everything an id" dans votre API ?

J'ai attribué un identifiant unique à chaque instance des entités de mon API. J'ai utilisé l'annotation @Id pour marquer la variable id comme étant la clé primaire des entités. De plus, j'ai utilisé l'annotation @GeneratedValue(strategy = GenerationType.AUTO) pour indiquer à la base de données de générer automatiquement des valeurs d'identifiant uniques chaque fois qu'un nouvel enseignant est ajouté à la base de données. Ainsi, chaque ressource de mon API est clairement identifiée par un identifiant unique, ce qui facilite la manipulation et la référence des ressources.

Comment avez-vous géré la sérialisation et la désérialisation des données entre les clients et le serveur dans votre API REST ?

Spring Boot gère automatiquement la sérialisation et la désérialisation des données JSON entre les clients et le serveur. Les données JSON envoyées par les clients sont automatiquement converties en objets Java lors de la réception dans les méthodes du contrôleur, et les objets Java sont convertis en données JSON lors de l'envoi de réponses aux clients.

Quels sont les types de réponses que votre API REST renvoie en cas de succès ou d'échec d'une requête ?

En cas de succès, l'API REST renvoie une réponse HTTP 200 (OK) avec les données demandées. En cas d'échec, elle renvoie une réponse HTTP avec un code d'erreur approprié, par exemple, 404 (Not Found) pour une ressource non trouvée ou 400 (Bad Request) pour une requête invalide.

Quelles techniques avez-vous utilisées pour gérer les erreurs et les exceptions dans votre API REST et fournir des réponses HTTP appropriées ?

J'ai utilisé des exceptions personnalisées comme TeacherNotFoundException et TeacherBadRequestException pour gérer les erreurs et fournir des réponses HTTP appropriées avec les codes d'erreur correspondants.

Questions HATEOAS :

Quelles stratégies avez-vous utilisées pour incorporer les principes de HATEOAS dans votre API REST ?

Dans les méthodes de contrôleur, des liens hypertexte sont ajoutés aux réponses pour permettre la navigation entre les ressources. Par exemple, dans “getAllTeachers”, des liens vers chaque enseignant individuel sont inclus, ainsi qu'un lien vers la liste complète des enseignants.

Comment avez-vous défini les différentes ressources de votre API et les relations entre elles en utilisant HATEOAS ?

Chaque ressource est accompagnée de liens vers elle-même (self) ainsi que vers la liste complète de ces mêmes ressources (allRessources).

Comment avez-vous inclus les liens hypertexte dans les réponses de votre API pour permettre la navigation et la découverte des ressources ?

Les liens hypertexte sont ajoutés aux réponses en utilisant la classe EntityModel de Spring HATEOAS. Chaque lien est associé à une relation spécifique, comme self pour la ressource actuelle.

Questions Database/Repository :

Comment avez-vous conçu votre repository pour interagir avec la base de données dans votre API ?

J'ai conçu mon repository en utilisant Spring Data JPA. Pour cela, j'ai créé une interface qui étend l'interface CrudRepository. Cette interface générique fournit des méthodes prédéfinies pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur les entités.

Comment avez-vous implémenté les opérations CRUD (Create, Read, Update, Delete) dans votre API en utilisant le Repository ?

J'ai implémenté les opérations CRUD dans mon API en utilisant les méthodes fournies par le repository que j'ai créé. Par exemple pour Teacher, pour récupérer tous les enseignants, j'ai utilisé la méthode héritée findAll() du repository. Pour récupérer un enseignant par son ID, j'ai créé une méthode findById() personnalisée dans le repository. Pour ajouter un nouvel enseignant, j'ai utilisé la méthode héritée save(), et pour supprimer un enseignant existant, j'ai créé une méthode deleteById() personnalisée dans le repository. En utilisant ces méthodes, je peux facilement effectuer des opérations CRUD sur les données des enseignants dans la base de données à partir de mon API.