ENSSAT LANNION | Université de Rennes

Projet développement java web
INFO2 - 2024

TEAM LOKI

Mathis ANELKA
Samuel BOLLOT
Camille BOUTHOLEAU
Vincent BUENO

12 janvier 2024

Sommaire

Introduction

Dans le cadre de nos cours de Programmation Web Serveur et Services Web nous avons étudié la programmation Web coté Serveur et l'architecture type REST. Après avoir étudié la partie "front" lors du module Programmation Web Client l'année dernière, nous nous focalisons ici sur la partie "back". Afin de mettre en application ce que nous avons vu en JEE, nous avons réalisé un projet de développement en java web.

L'objectif est de créer un site de gestion des tournées de DJ. Grâce à ce site, les DJ pourront consulter les événements auxquels ils participent. Ils pourront également ajouter de nouveaux événements, regarder la liste des lieux pouvant accueillir des événements et modifier leurs informations personnelles.

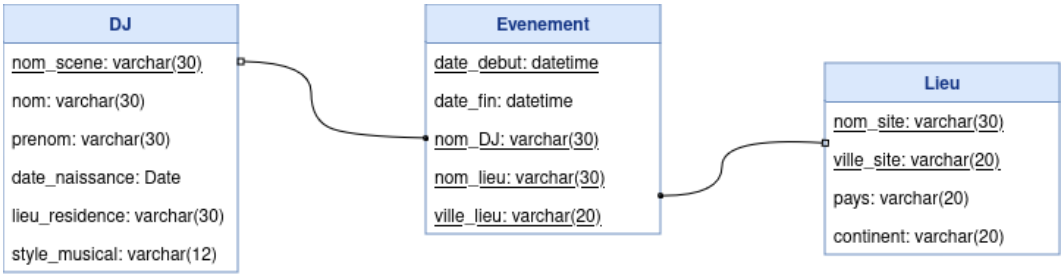
1 Cas d'utilisation

2 Base de données

2.1 Schéma

Pour stocker les informations concernant les DJ, les lieux et les événements, nous avons besoin d'une base de données contenant 3 tables. Voici le schéma relationnel de cette base :

FIGURE 1 – Schéma relationnel de la base de données



Pour la table Evenement, nous avions au départ crée une colonne date et deux colonnes heure_debut et heure_fin. Seulement, cela voulait dire qu'un événement ne devait pas se terminer après minuit. De plus, c'était plus compliqué pour créer des triggers. En effet, pour vérifier que des événements d'un même DJ étaient bien séparés de 24 ou 48h, nous devions soustraire les deux dates, ce qui ne prenait pas en compte les heures de début et de fin de ces événements. Nous avons donc préféré utiliser les colonnes date_debut et date_fin de type datetime qui prennent une date et une heure.

Pour la table Lieu, nous avons remarqué que certains lieux avaient des noms identiques, nous ne pouvons donc pas mettre le nom en clé primaire. Ainsi, la clé primaire est le couple (nom_site, ville_site).

2.2 Triggers

Nous avons ensuite créé des triggers pour que les tables respectent les contraintes souhaitées.

Salle_prise : Ce trigger fait en sorte que deux événements animés par deux DJ différents ne se tiennent pas au même endroit au même moment.

Mauvaises_horaires : Ce trigger fait en sorte que lors de la création d'un événement, la date de début soit avant la date de fin.

Intervalle_evenements_24_post : Ce trigger fait en sorte que le nouvel événement d'un DJ ne débute pas dans les 24h après un événement de ce même DJ ayant lieu dans le même continent.

Intervalle_evenements_24_ante : Ce trigger fait en sorte que le nouvel événement d'un DJ ne finisse pas dans les 24h précédant le début d'un événement de ce même DJ ayant lieu dans le même continent.

Intervalle_evenements_48_post : Ce trigger fait en sorte que le nouvel événement d'un DJ ne débute pas dans les 48h après un événement de ce même DJ ayant lieu dans un autre continent.

Intervalle_evenements_48_ante : Ce trigger fait en sorte que le nouvel événement d'un DJ ne finisse pas dans les 48h précédant le début d'un événement de ce même DJ ayant lieu dans un autre continent.

Evenements_concurrents : Ce trigger fait en sorte que les événements d'un même DJ ne se déroulent pas sur des périodes qui se chevauchent.

2.3 Procédures

Nous avons enfin crée des procédures pour alléger les requêtes dans le code Java. Nous utilisons par exemple la procédure pour insérer un événement dans la table ou alors pour générer le top 5 des DJ.

3 Architecture côté serveur

3.1 Présentation

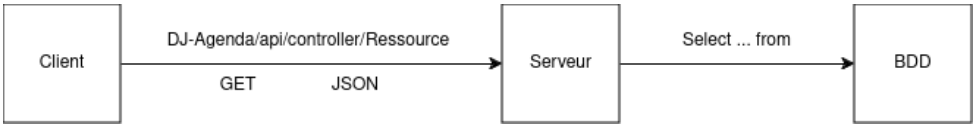
L'API est géré par un serveur Tomcat et s'articule autour d'un contrôleur qui gère les demandes liées aux évènements, aux dj et aux lieux. L'architecture adoptée pour cet API est une architecture REST. Le client envoie des requêtes HTTP à un URL contenant le nom de la ressource à laquelle il souhaite avoir accès et l'API réalise l'opération correspondante. Ainsi pour accéder aux ressources évènements, lieux ou DJ, il faut envoyer une requête aux endpoints suivants :

[GET|POST|PUT|DELETE] DJ-Agenda/api/controller/[Ressource]

où les ressources sont {'event', 'lieu', 'dj'}.

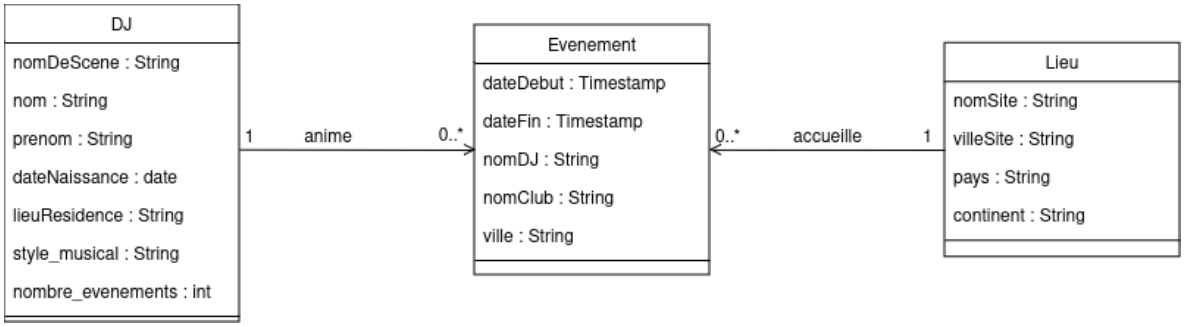
Les données transmises et reçues sont au format JSON.

Figure 2 – Schéma de la requête



Pour mettre en place cette architecture, les classes java s'organisent comme indiqué sur le diagramme de classe ci-dessous :

Figure 3 – Diagramme de classe



Le contrôleur utilise les méthodes des DAO afin d'interagir avec la base de donnée pour réaliser l'action demandée. Nous avons choisi d'instancier un seul contrôleur pour l'ensemble des ressources car l'accès à une ressource différente se fait simplement par le changement du nom de la ressource à la fin de l'URL. L'existence de plusieurs contrôleurs rendrait l'utilisation de l'API moins claire et moins intuitive. C'est pourquoi la classe 'controller' contient tous les endpoints de notre API.

3.2 Implémentation Java

Pour chaque table (DJ, Lieu et Evenement), il y a côté serveur une classe dont les attributs sont les colonnes de la table. Pour afficher le top 5 des DJ, nous avons également afficher le nombre d'événements qu'ils animent. Dans la classe DJ, nous avons donc rajouté un entier nombre_evenements. Pour chacune de ses classes nous avons crée une interface et programmé une implémentation de cette interface. Nous avons ensuite codé les requêtes GET, POST, PUT et DEL. Pour les la liste des lieux, nous voulons juste l'afficher, donc une requête GET suffit. Pour les événements, nous voulons les afficher et en créer, nous avons donc besoin des requêtes GET et POST. Enfin, pour les DJ, nous devons les afficher, en créer, en modifier et en supprimer. Nous avons donc besoin des quatre types de requêtes.

3.3 Pages HTML

La page d'accueil du site (index.html) propose quatre boutons pour accéder à quatre pages : DJ, lieux, événements et top 5 des DJ. Nous avons décidé de mettre sur une même page le calendrier par mois des événements des DJ ainsi que la fonctionnalité d'ajout d'un événement. Ainsi, nous pouvons voir directement l'ajout dans le calendrier. Quand un DJ tente d'ajouter un événement, il est amené sur la page échec ou succès en fonction du résultat de sa demande. En effet, il y a beaucoup de triggers pour l'ajout des événements (car beaucoup de contraintes), nous avons donc décidé de procéder à une légère redirection permettant de traiter les cas d'erreurs.

En se lançant, les pages DJ et topfive appellent la fonction loadDJ(), mais avec une ressource différente. La première fait un fetch sur "/DJ-Agenda/api/controller/djs ?topfive=true" tandis que la seconde le fait sur "/DJ-Agenda/api/controller/djs ?topfive=false". C'est ensuite le Controller qui va appeler la bonne fonction implémentée dans DJDAOMockImpl en fonction de la valeur de topfive.

Conclusion

En conclusion, nous sommes plutôt satisfaits de ce que nous avons réalisé, et nous avons pu implémenter tout ce que nous avions prévu de faire. Néanmoins, nous aurions aimé aller un peu plus loin. Nous ne pensions pas avoir le temps d'implémenter un accès restreint par login/mot de passe, mais si nous avions eu plus de temps, nous aurions fait en sorte que le DJ indique son nom et qu'il puisse n'accéder qu'à son calendrier, la liste des lieux et le formulaire de modification de ses informations personnelles. Une personne administrative pourrait elle, avoir accès au calendrier de chaque DJ, à la liste des DJ, à la possibilité de suppression et au top 5 des DJ. Nous aurions pu également prévoir qu'un DJ puisse supprimer/modifier ses événements.

Ce projet nous a permis de mieux comprendre l'élaboration d'une API Rest sous java avec un serveur Tomcat. Il nous a également permis de travailler en équipe et de mieux maîtriser la résolution de problèmes liés à Git et à la mise en commun des travaux de chacun. Enfin, il nous a permis de mieux comprendre ce que nous avions fait l'année dernière en programmation web client car cette fois nous voyons ce qui se passe derrière, et nous avons également pu appliquer les notions vues dans le module de Bases de données relationnelles.