ENSSAT LANNION | Université de Rennes

Rapport de projet de JEE

Réalisation d'un site web

de

BEKHEDDA Loueï
CROIZET Thomas
KHALOUI Oussama
DELAUNAY Gurwan

INFO2 - Promotion 2024
25 mai 2023

Table des matières

1 Introduction

1.1 Environnement de départ

Ce projet du module Services Web est destiné à la découverte et à la création d'une application web avec JEE (Java Entreprise Edition). Pour créer ce projet, seront utilisés également l'environnement de développement Eclipse ainsi que le serveur d'applications Apache Tomcat. Nous avons pris la décision de ne pas surcharger cet environnement et de ne pas utiliser de framework supplémentaire.

1.2 Contexte du projet

Un an avant le début des Jeux Olympiques de Paris 2024, le comité d'organisation des Jeux Olympiques nous ont confié la tâche de développer un site web permettant en premier lieu à gérer le planning des épreuves.

Le site comporte deux parties distinctes :

- une partie publique et accessible à tous : elle permet d'afficher les athlètes participant aux jeux mais également les différentes sessions disponibles et comporte également une partie statistique sur les jeux.
- une partie administrative, protégée par une authentification chiffrée, permettant aux différents utilisateurs de gérer les disciplines, les sites (lieux des sessions) et les sessions en CUD (création, mise à jour et suppression). Depuis cette partie, certains utilisateurs pourront également importer les athlètes.

Plusieurs contraintes et règles sont également prévues et seront détaillées au fur et à mesure de ce rapport.

2 Architecture : modèle MVC

Au niveau de l'architecture, nous avons décidé d'adopter une architecture logicielle MVC (Modèle-vue-contrôleur) qui est une manière très courante de construire une application web. Ce modèle MVC est surchargé avec une couche d'accès aux données (DAO).

Présentation de l'architecture :

- Le modèle contient les données que l'on souhaite afficher dans la vue donc dans le cas présent les différentes classes utilisées dans l'application : Discipline, Site, Session, Athlète, User et toutes les énumérations permettant de stocker des données précises (CategorieSession, CategorieSite, CategorieUser, Genre et TypeSession).
- La vue contient toutes nos interfaces graphiques donc nos pages HTML, nos scripts Javascript, nos images et nos feuilles de style. Les fichiers Javascript vont permettre de faire plusieurs appels à nos contrôleurs afin de récupérer les données et de les afficher dans les vues HTML.
- Le contrôleur va permettre de faire le lien entre la base de données (par le DAO) et la vue. Cette partie représente notre partie API et sera accessible par le biais de GET, POST, PUT et DELETE.

- Le DAO (Data Access Object) va permettre, comme son nom l'indique, de récupérer les données dans la base de données MySQL et de les transmettre aux contrôleurs. Nous avons un DAO par modèle à savoir AthleteDAO, DisciplineDAO, SessionDAO, SiteDAO et UserDAO. Ces DAO contiennent généralement des méthodes permettant d'ajouter, de mettre à jour, de supprimer ou de chercher dans la base de données (CRUD). Dans notre package DAO, nous retrouvons également des classes permettant d'accéder à notre base de données.

3 Accès à la base de données

La base de données utilisée pour ce projet fonctionne avec le SGBD MySQL et est hébergée sur un serveur OVH. Nous utilisons pour gérer cette base de données l'interface de programme JDBC. Afin d'intégrer cette base de données à notre application web, nous avons un fichier config.properties situés à la racine du répertoire contenant nos fichiers Java et contient l'URL de la base, le driver utilisé et les identifiants de connexion.

Pour récupérer la connexion à la base de données, nous avons une classe nommée DBManager dans notre package DAO. Cette classe (singleton) va permettre de récupérer la connexion grâce à la classe Connection de java.sql, elle va également permettre de fermer la connexion lorsque l'application s'éteindra. Dans DBManager, nous avons également une méthode permettant de créer les tables dans la base (initialisation si la base de données est vide). Cette méthode est déclenchée par la classe DBInitializer au démarrage du serveur.

Cet accès à une base de données, va nous permettre de remplir notre application avec les informations nécessaires à la bonne gestion des Jeux Olympiques 2024.

4 Traitement des données

Une fois la connexion avec la base de données effectuée, nous pouvons commencer à traiter les données et cela grâce tout d'abord aux modèles (Model). Le package Model contient toutes les classes utiles pour stocker nos données. Nous avons décidé dans notre implémentation du problème de modéliser cinq classes principales :

- Athlète avec l'énumération Genre (homme,femme,autre)
- User avec l'énumération CategorieUser (admin, session handler (gestionnaires sessions), better handler (gestionnaires compétitions, sites et disciplines))
- Discipline
- Session avec les énumérations CategorieSession (homme,femme,mixte) et TypeSession (qualifications, médailles)
- Site avec l'énumération CategorieSite (stade, salle de spectacle, lieu public, centre aquatique, salle, gymnase, golf)

L'autre package permettant notamment d'accéder à nos données est le package DAO contenant pour chaque classe du modèle un fichier DAO lié. Chaque DAO va permettre d'accéder aux données par le biais de méthodes comme findAll(), findByString(), findById(), d'ajouter des données (par exemple : addAthlete(Athlete athlete)), de mettre à jour des données (par exemple : editSite(Site site)) ou de supprimer des données (par exemple : removeSession(String code)). Ces appels à la base de données sont gérés grâce aux classes de java.sql à savoir PreparedStatement, Statement et ResultSet.

Cependant les DAO et les modèles ne communiquent pas directement avec l'affichage de notre application web. C'est à ce moment-là que nous avons besoin des contrôleurs.

5 Contrôleurs

Les contrôleurs vont nous permettre de faire le lien entre nos données récupérées par les DAO et misent dans nos modèles et nos vues qui vont permettre de les afficher. Pour cela, nous avons un fichier AppConfig qui va s'étendre de la classe Application de Jakarta et qui va nous permettre d'accéder à notre API donc à nos contrôleurs par le biais d'un ApplicationPath("/api").

Chacun de nos contrôleurs va être accessible par un chemin spécifié au-dessus de la déclaration de la classe du contrôle par le biais de l'annotation @Path (par exemple @Path("discipline-controller")). Dans chaque classe, une instanciation des DAO nécessaires sera effectuée et les instances seront stockées dans des variables globales.

Les méthodes des contrôleurs sont des méthodes liées à des méthodes REST HTTP (GET, POST, PUT et DELETE). Une annotation est disponible pour chacune de ces méthodes. Nous avons également une annotation de type @Produces (type de données retournées (JSON généralement)) et @Consumes (type de données reçues par la méthode). Suite à cela, l'annotation @Path est spécifiée pour pouvoir appeler cette méthode en particulier lors d'un fetch. La méthode du contrôleur va ensuite généralement appeler un DAO et retourner les données trouvées.

Nous utilisons également des servlets dans notre application. Les servlets sont des classes Java spéciales utilisées pour étendre les fonctionnalités des serveurs web. Ils sont particulièrement adaptés à la création de services web en utilisant le modèle requête-réponse. Dans notre application, nous avons créé trois servlets : DisconnectServlet, UserController et UserInfoController. Chaque servlet a une fonctionnalité spécifique et est accessible via un chemin spécifié dans l'URL.

DisconnectServlet gère la déconnexion des utilisateurs. Lorsqu'un utilisateur accède à l'URL "/admin/disconnect" en utilisant une requête GET, cette servlet invalide la session actuelle de l'utilisateur, le déconnectant ainsi de l'application. Ensuite, l'utilisateur est redirigé vers la page de connexion ("login.html"). Cela garantit que l'utilisateur est correctement déconnecté et peut se connecter à nouveau si nécessaire.

UserController gère les requêtes relatives à la gestion des utilisateurs. Lorsqu'un utilisateur envoie une requête GET à l'URL "/admin", cette servlet récupère les paramètres "username" et "password" fournis dans la requête. Ensuite, elle utilise un UserDAO pour obtenir les informations de l'utilisateur correspondant au nom d'utilisateur fourni. Si les informations d'identification sont valides (c'est-à-dire que le mot de passe correspond), la servlet crée une session et stocke le nom d'utilisateur et la catégorie de l'utilisateur dans cette session. Ensuite, l'utilisateur est redirigé vers une page spécifique ("admin/index.html") pour accéder à des fonctionnalités réservées aux utilisateurs authentifiés. En cas d'informations d'identification invalides, l'utilisateur est redirigé vers la page de connexion avec un message d'erreur approprié.

UserInfoController renvoie les informations de l'utilisateur connecté. Lorsqu'un utilisateur envoie une requête GET à l'URL "/admin/getUserInfo", cette servlet accède à la session en cours et récupère le nom d'utilisateur et la catégorie de l'utilisateur à partir des attributs de session. Ensuite, elle crée un objet User contenant ces informations, ainsi qu'une indication sur l'état de connexion de l'utilisateur. Pour transmettre ces informations au format appropriées, la servlet utilise la bibliothèque Gson pour convertir l'objet User en JSON (JavaScript Object Notation). Finalement, le JSON résultant est écrit dans la réponse, ce qui permet à l'application cliente d'accéder facilement aux informations de l'utilisateur.

6 Lien entre les vues et les contrôleurs

En ce qui concerne la partie qui fait le lien, nous avons mis en place des fonctions Javascript pour accéder aux données via des requêtes fetch. Ces fonctions utilisent les URLs spécifiées pour récupérer les différentes données depuis le serveur.

Par exemple pour notre partie statistique, nous avons mis en place une fonction intitulée *load* qui via des requêtes fetch accède aux données sur les sites et les disciplines. Une fois que les données sont récupérées avec succès, elles sont envoyées à des fonctions de traitement spécifiques (processSites et processDisciplines) afin de remplir des tableaux sur notre partie frontale du site.

```
1  /**============================================================
2   *                          load
3   * ? Loads data from the specified URLs and processes them.
4   * @param {string} urlSites
5   * @param {string} urlDisciplines
6   * @return {void}
7   *============================================================**/
8  function load(urlSites, urlDisciplines) {
9      // Show the loader
10     document.getElementById("loader").style.display = "block";
11     fetch(urlSites)
12         .then(response => response.json())
13         .then(response => processSites(response))
```

```
        .catch(error => console.log("Erreur : " + error));

    fetch(urlDisciplines)
        .then(response => response.json())
        .then(response => processDisciplines(response))
        .catch(error => console.log("Erreur : " + error));
}
```

L'appel de ces fonctions est fait avec les liens définis par les contrôleurs évoqués précédemment.

```
/**------------------------------------------------------------------------
 *                              CONST & LOAD
 *------------------------------------------------------------------------**/
load('/jee-project/api/site-controller/get-top-five-sites',
 '/jee-project/api/discipline-controller/get-disciplines-duration')

```

7 L'affichage

Pour ce qui est de l'affichage, nous nous sommes orientés vers un framework populaire nommé **Flowbite** qui est en réalité basé sur Tailwind. C'est un choix pertinent à notre sens car il offre une multitude de composants déjà utilisables via leur site qui offre une multitude d'exemple d'application de ces derniers. Il permet également de définir des styles très facilement en ayant juste à ajouter une class à un élément (ex : mt-5 au lieu d'écrire dans du css margin-top ...).

(Exemple de composants login pré-faits)

Qui plus est, nous avons incorporé à notre site afin de faciliter l'utilisateur une barre de recherche pour les athlètes et les sessions et ce afin de pouvoir offrir une expérience plus satisfaisante et facile, avec l'aide encore une fois de notre framework de style. La recherche est assurée en arrière-plan par des scripts Javascript qui accèdent à nos contrôleurs via des requêtes Fetch.

En voici un exemple pour la recherche d'athlète par nom, le code Java ci-dessous :

```
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("/get-athletes-name")
public String getAthletesName(@QueryParam("name") String name) {
    List<Athlete> athletes = null;
    if(name != null && name.length() > 0) {
        athletes = athleteDAO.findByName(name);
    }

    if(athletes == null || athletes.isEmpty()) {
        athletes = new ArrayList<Athlete>();
```

```
    }

    GsonBuilder builder = new GsonBuilder();
    Gson gson = builder.create();
    String json = gson.toJson(athletes);
    return json;
}
```

ainsi que le code Javascript y accédant :

```
/**============================================================
 **                    executeSearch
 * ? Executes a search based on the provided search term.
 * @return {void}
 **============================================================**/
function executeSearch() {
    const searchTerm = searchInput.value;
    //par souci de clarté sur le compte rendu on définit :
    const url1 =
     ↪  '/jee-project/api/athlete-controller/get-athletes-name?name='
    const url2 =
     ↪  '/jee-project/api/athlete-controller/get-athletes?limit=12&page=1'
    if (searchTerm.length > 0) {
        loadAthletes(url1 + searchTerm);
    } else {
        loadAthletes(url2);
        page = 1;
    }
}
```

Cette implémentation est viable et facilement modifiable.

8 Fonctionnalités diverses

8.1 Import des athlètes sous format CSV

Une des fonctionnalités demandée par les organisateurs est la possibilité de pouvoir importer les athlètes présents dans un fichier CSV (document tableur) en fournissant le fichier à l'application web.

8.1.1 Côté client

Afin de transmettre le fichier CSV au site web, nous avons utilisé deux façons de faire menant aux mêmes opérations :

- Glisser-déposer le fichier dans une zone visible à l'écran (avec un input de type file et des événements drop et drag).

- Cliquer sur la zone afin d'ouvrir l'explorateur de fichier à la recherche du fichier CSV voulu.

Une fois le fichier selectionné ou glissé, le fichier est lu comme un texte grâce à la fonction readAsText de la classe FileReader de Javascript. Ce contenu (fichier CSV sous format texte String) est transmis au contrôleur d'athlète par un fetch en POST (fonction sendCSV dans index.js). Si l'importation est un succès (le serveur nous transmet cette information), une alerte sera affichée sur le navigateur alors que si l'import échoue, un message d'erreur sera affiché dans la console.

8.1.2 Côté serveur

Afin de développer cette fonctionnalité, importante pour la bonne gestion de l'application web, nous avons utilisé la librairie OpenCSV permettant notamment de lire des CSV avec le langage de programmation Java (lien vers le repository Maven). La méthode permettant la conversion de CSV à une liste d'athlètes se trouve dans le contrôleur d'athlète. Dans cette méthode, nous allons utiliser StringReader en passant en argument le CSV sous format String (mis sous cette forme auparavant en Javascript). Nous allons ensuite passer ce StringReader dans un CSVReader, méthode de OpenCSV, et nous allons sauter les quatre premières lignes synonymes d'entête du fichier et nom des colonnes. Nous récupérons ensuite toutes les lignes du fichier et procédons à la répartition de chaque colonne en l'attribut qui lui est propre. Après création de chaque objet, nous les passons au DAO pour pouvoir les ajouter en base de données.

8.2 Règles de gestion sur les dates

Nous devions, au niveau de l'ajout de session, respecter plusieurs consignes fournies par les organisateurs :

- Ne pas avoir de sessions de la même discipline se superposant dans le temps
- Avoir au minimum une heure de différence entre la fin d'une session et le début d'une autre de la même discipline.

Afin de répondre à ces règles, nous avons implémenté une méthode dans SessionDAO qui va permettre de vérifier avant l'ajout ou mise à jour de nouvelles données dans Session que les emplois du temps coïncident. Cette méthode est nommée slotsAvailable et prend un objet Session en argument. Elle repose sur une requête SQL qui va demander à la base de données de vérifier si une session de la même discipline existe déjà dans cet intervalle de temps (vérifier d'une heure avant le début de la session passée en arguments jusqu'à une heure après). Si la requête retourne une valeur, on annule l'ajout ou la modification de la session en base de données.

9 Conclusion

Pour conclure, notre projet répond bien aux différentes attentes des organisateurs des Jeux Olympiques qui vont pouvoir désormais pouvoir gérer l'organisation des Jeux depuis notre application web. Nous avons réussi à implémenter les différentes fonctionnalités comme l'import d'athlètes avec un fichier CSV ou la vérification d'emploi du temps avant l'insertion des nouvelles sessions. Nous avons également tenu à mettre l'accent sur l'affichage du site afin de rendre sa navigation agréable et son utilisation ergonomique. Enfin, nous avons tenu à ajouter diverses fonctionnalités non demandées dans le cahier des charges (et qui peuvent être retirées sous demande des organisateurs) comme la barre de recherche dans les pages accessibles au public des sessions et des athlètes. Nous avons également ajouté un widget sur la page d'accueil du site avec une section "athlète aléatoire" qui aurait pu être amélioré avec la photo des athlètes. Enfin, nous avons tenu à ajouter un dark mode afin de rendre la navigation toujours plus agréable.

Ce projet nous a permis de découvrir une nouvelle manière de bâtir une application web et est un bon complément à la partie NodeJS de ce module de Services Web. Nous avons également eu l'occasion de travailler nos compétences sur le domaine des bases de données intégrées à un projet, notre utilisation de Git, de l'architecture MVC et du front-end.

⋆ ⋆ ⋆