CAHIER DES CHARGES

Plateforme de Centralisation et Recommandation d'Opportunités Tech (Hackathons & Certifications)

Technologie : Spring Boot / Web Scraping / IA

---

1. CONTEXTE ET OBJECTIF

Le projet vise à développer une application web backend (avec API) capable d'agréger automatiquement des événements technologiques (Hackathons) et des certifications depuis diverses sources externes (Oracle, NASA, etc.). L'objectif est de centraliser l'information pour les étudiants et professionnels et d'offrir un moteur de recommandation personnalisé.

---

2. BESOINS FONCTIONNELS

A. Module d'Acquisition de Données (Web Scraping)

1. **Scraping Automatisé :** Le système doit se connecter périodiquement à une liste de sites cibles.
2. **Extraction Ciblée :** Pour chaque événement, le système doit extraire :
    - Le titre.
    - La description.
    - La date (début/fin).
    - Le prix (ou gratuité).
    - Le lieu (Ville/Pays ou "En ligne").
    - Le lien original (URL).
    - L'organisateur (ex: Oracle).
3. **Nettoyage :** Le système doit formater les données brutes (ex: convertir "150 USD" en `150.0` et `USD`).
4. **Dé-duplication :** Éviter d'enregistrer deux fois le même événement.

B. Module Utilisateur & Consultation

1. **Catalogue :** Afficher la liste des événements stockés en base de données.
2. **Filtrage Avancé :**
    - Par **Proximité/Lieu** (ex: "Maroc", "Paris", "En ligne").
    - Par **Prix** (ex: "Gratuit", "< 50€").
    - Par **Domaine** (ex: "Java", "Data Science", "Cybersecurity").
3. **Profil Utilisateur :** L'utilisateur doit pouvoir indiquer ses préférences.

C. Module de Recommandation

1. **Suggestion :** Sur la base du profil utilisateur, le système doit proposer un "Top 5 des événements pour vous".

2. **Feedback (Optionnel) :** L'utilisateur peut "liker" ou "sauvegarder" un événement pour améliorer les recommandations futures.

---

3. BESOINS NON FONCTIONNELS (Contraintes techniques)

1. **Robustesse du Scraping :** Si le site de la NASA change de structure, l'application ne doit pas planter entièrement. L'erreur doit être loggée.
2. **Performance :** Le scraping doit se faire en arrière-plan (Asynchrone) sans ralentir la navigation des utilisateurs.
3. **Respect des Sites Tiers :** Le robot ne doit pas envoyer plus d'une requête toutes les 2 secondes vers le même site (Politeness policy).
4. **Extensibilité :** Il doit être facile d'ajouter une nouvelle source sans réécrire tout le code.

---

4. ARCHITECTURE TECHNIQUE & STACK

A. Backend : Spring Boot (Java)

- **Pourquoi ?** C'est le standard de l'industrie pour les applications robustes. Il gère l'injection de dépendances et simplifie la configuration.
- **Utilisation :** Il sera le chef d'orchestre. Il recevra les requêtes HTTP (API) et lancera les tâches de scraping.

B. Base de Données :

- **Pourquoi ?** Pour stocker les données structurées (Relations entre Utilisateurs et Événements).
- **Utilisation avec JPA/Hibernate :** Tu n'écriras pas de SQL pur. Tu utiliseras des entités Java (`@Entity`) qui seront automatiquement traduites en tables.

C. Module de Scraping : Approche Hybride (Jsoup & Selenium)

- **Choix Technologique :** Utilisation conjointe de **Jsoup** et **Selenium WebDriver**.
- **Pourquoi cette stratégie ?**
  - **Jsoup** sera utilisé pour les sites "statiques" (HTML classique) car il est extrêmement rapide et consomme peu de mémoire.
  - **Selenium** est indispensable pour les sites "dynamiques" (Single Page Applications comme React/Angular) où le contenu n'apparaît qu'après l'exécution de JavaScript. Il permet aussi de simuler des actions utilisateur (scroller vers le bas, cliquer sur "Suivant").
- **Configuration :** Selenium sera configuré en mode **"Headless"** (sans interface graphique) pour tourner en arrière-plan sur le serveur Spring Boot.

D. Planification : Spring Scheduler (@Scheduled)

- **Pourquoi ?** Pour que le scraping se lance tout seul (ex: chaque nuit à 03h00).
- **Utilisation :** Une simple annotation sur une méthode Java.

E. Recommandation : LibRec (Java Library)

- **Pourquoi ?** Librairie spécialisée contenant déjà les algorithmes mathématiques (Collaborative Filtering, etc.).
- **Utilisation :** Elle prendra en entrée la matrice des préférences utilisateurs et sortira des IDs d'événements recommandés.

---

5. FEUILLE DE ROUTE D'IMPLÉMENTATION (Guide de démarrage)

Ce projet est complexe. Il est impératif de travailler par itérations (méthode Agile).

Phase 1 : Le "Crawler"

- **Objectif :** Remplir la base de données.
- **Tâches :**
    1. Configurer Spring Boot et MySQL.
    2. Créer l'entité `Evenement`.
    3. Implémenter Jsoup pour scraper **un seul site** (le plus simple).
    4. Vérifier que les données arrivent en base.

Phase 2 : L'API et les Filtres

- **Objectif :** Rendre les données accessibles.
- **Tâches :**
    1. Créer les Controllers REST (`@GetMapping`).
    2. Implémenter les filtres dynamiques dans le Repository (ex: `findByPriceLessThanAndLocation(...)`).

Phase 3 : Utilisateurs & Recommandation

- **Objectif :** Ajouter l'intelligence.
- **Tâches :**
    1. Ajouter une entité `User` avec des préférences.
    2. Intégrer **LibRec** : Créer un service qui prend un `User`, interroge LibRec, et renvoie une liste d'événements.

---

6. ARCHITECTURE DE LA BASE DE DONNÉES (Modèle Conceptuel)

Pour que le système fonctionne, nous avons besoin de trois entités principales interconnectées : les **Utilisateurs**, les **Événements** (Hackathons/Certifs), et les **Interactions** (le lien entre les deux qui nourrit l'IA).

A. Les Tables (Entités)

Voici la structure détaillée des tables à créer. En Spring Boot, chaque table correspond à une classe Java annotée `@Entity`.

1. Table `AppUser` (Les Utilisateurs)

- *Rôle :* Stocker qui se connecte et ses préférences de base.
- **Colonnes :**
  - `id` (Long, Primary Key) : Identifiant unique.
  - `username` (String) : Nom d'utilisateur.
  - `email` (String) : Pour le contact.
  - `preferences` (String) : Mots-clés séparés par des virgules (ex: "JAVA, DATA, SPACE") -> *Utile pour le démarrage à froid (Cold Start).*

2. Table `Event` (Les Ressources Scrapées)

- *Rôle :* Stocker tout ce que le robot récupère. On utilise une seule table pour Hackathons ET Certifications pour simplifier la recherche.
- **Colonnes :**
  - `id` (Long, Primary Key).
  - `title` (String) : Titre de l'événement.
  - `description` (Text) : Description longue (pour la recherche par mot-clé).
  - `type` (Enum/String) : "HACKATHON" ou "CERTIFICATION".
  - `price` (Double) : 0.0 si gratuit.
  - `eventDate` (Date) : Date de l'événement.
  - `location` (String) : "Paris", "Casablanca", "Remote".
  - `sourceUrl` (String) : Le lien original vers Oracle/NASA.
  - `provider` (String) : Le nom du site scrapé (ex: "Oracle University").

3. Table `Interaction` (Le Cœur de la Recommandation)

- *Rôle :* C'est la table la plus importante pour **LibRec**. Elle enregistre l'intérêt d'un utilisateur pour un événement.
- **Colonnes :**
  - `id` (Long, Primary Key).
  - `user_id` (Foreign Key) : Qui a fait l'action ?
  - `event_id` (Foreign Key) : Sur quel événement ?
  - `rating` (Integer) : Une note implicite ou explicite (ex: 1 à 5).
    - *Logique :* Si l'utilisateur clique sur "Voir détails", on met 3/5. S'il clique sur "Favoris", on met 5/5.
  - `timestamp` (DateTime) : Quand cela s'est passé.

B. Les Relations (Associations JPA)

Il est crucial de bien définir les liens entre les tables dans Spring Boot :

1. **User <-> Interaction (One-to-Many) :**
   - Un utilisateur peut avoir *plusieurs* interactions (clics, likes).
   - Une interaction appartient à *un seul* utilisateur.
2. **Event <-> Interaction (One-to-Many) :**
   - Un événement peut avoir *plusieurs* interactions (de différents utilisateurs).
   - Une interaction concerne *un seul* événement.