Titre : Documentation Simplifiée - Plateforme d'Aide au Choix de Cours

Sous-titre : Guide rapide pour l'équipe projet

Date : 2025-12-03

Pour l’implementation et les tests nous les avons mis dans un sous dossier nommé «code »

---

1. Objectif du projet

Aider les étudiants de l'Université de Montréal à choisir leurs cours en centralisant toutes les informations utiles au même endroit.

---

2. Utilisateurs et accès

Rôle	Ce qu'ils peuvent faire
Visiteur (non connecté)	• Voir les informations basiques des cours • Rechercher des cours
Étudiant connecté	• Tout ce que fait le visiteur • Comparer plusieurs cours (2-5) • Voir les avis et notes des cours • Personnaliser son profil avec ses préférences
Administrateur	• Gérer les informations des cours • Modérer les avis des étudiants • Générer des rapports d'utilisation

---

3. Fonctionnalités principales

3.1 Consulter un cours

- **Pour qui** : Tout le monde

- **Comment** : Recherche → Sélection → Affichage
- **Affiche** : Code, titre, crédits, horaire, préalables

3.2 Comparer des cours

- **Pour qui** : Étudiants connectés seulement
- **Limite** : 2 à 5 cours maximum
- **Comparaison** : Charge travail, difficulté, horaires, compatibilité
- **Plus** : Calcul automatique de la charge totale

3.3 Voir les avis

- **Pour qui** : Étudiants connectés
- **Règle IMPORTANTE** : Minimum 5 avis nécessaires avant d'afficher les résultats
- **Si moins de 5 avis** : Message "Avis insuffisants pour consultation"
- **Affiche** : Note moyenne, commentaires, charge de travail estimée

3.4 Personnaliser son profil

- **Pour qui** : Étudiants connectés
- **Paramètres** :
    - Niveau préféré (théorie/pratique)
    - Centres d'intérêt
    - Programme d'études
    - Contraintes horaires

---

4. Architecture technique

Choix : Application monolithique avec API REST

Étudiants → Site Web / Application → API REST → Base de données

↓

API Planifium (données officielles)

↓

Bot Discord (collecte avis)

Modules principaux :

- Module Cours (données Planifium)
- Module Avis (collecte Discord)
- Module Profil (préférences étudiants)
- Module Administration

---

5. Risques et solutions

Risque #1 : Biais dans les avis Discord

- **Problème** : Seuls les étudiants très contents ou très mécontents donnent leur avis
- **Solution** :
    - Seuil de 5 avis minimum avant affichage
    - Indicateur de fiabilité
    - Pondération des avis récents

Risque #2 : Planifium indisponible

- **Solution** : Cache des données + mode secours

Risque #3 : Faible adoption

- **Solution** : Interface simple, promotion via associations étudiantes

---

6. Phases de développement

Phase 1 (MVP - Minimum)

- Page de recherche de cours
- Affichage des informations depuis Planifium
- Système de connexion basique

- Interface responsive

Phase 2

- Comparaison de cours
- Gestion du profil étudiant
- Intégration du bot Discord

Phase 3

- Interface d'administration
- Modération des avis
- Rapports statistiques

7. Diagrammes de classes

- Justification des choix du design

Cohésion forte

Chaque classe a une **responsabilité unique et bien circonscrite** :

- Les modèles représentent les données métier
- Les services encapsulent la logique
- Les contrôleurs gèrent les entrées/sorties HTTP
- Les utilitaires s'occupent des préoccupations transversales

Couplage réduit

- Les services dépendent de **classes concrètes simples** plutôt que d'interfaces complexes
- Le GestionnaireFichiers centralise toutes les opérations de persistance
- Le ClientPlanifium isole les communications externes

Encapsulation

- Tous les attributs sont **privés** (-), protégeant l'état interne
- Les méthodes exposent uniquement les opérations métier nécessaires

- Les détails de persistance et de communication réseau sont **masqués** derrière des APIs simples

Simplicité pragmatique

Conformément au feedback reçu, l'architecture **évite les interfaces inutiles** pour les opérations de base, privilégiant des classes concrètes directes tout en maintenant une structure organisée.

Avantages pour les cas d'utilisation

Pour la Recherche de cours (CU1)

- ServiceCours.rechercher() utilise le cache pour les performances
- ClientPlanifium fournit les données fraîches lorsque nécessaire
- Architecture optimisée pour les requêtes fréquentes

Pour la Consultation détaillée (CU2)

- ServiceCours.obtenirAvecDetails() agrège cours, avis et statistiques
- ServiceAvis fournit les évaluations associées
- Séparation claire entre données de base et données dérivées

Pour la Comparaison de cours (CU3)

- ServiceCours.comparer() réutilise les méthodes existantes
- Structure flexible pour l'ajout de nouveaux critères de comparaison
- Gestion centralisée de la logique de comparaison

Évolutivité et maintenabilité

- **Extension facile** : Ajout de nouveaux critères de recherche ou de comparaison via modification localisée de ServiceCours
- **Testabilité** : Classes aux responsabilités claires, facilement mockables pour les tests unitaires
- **Remplacement aisé** : Le GestionnaireFichiers peut être remplacé par une base de données sans affecter les services
- **Support multi-canal** : L'architecture supporte déjà web et Discord, pouvant être étendue à d'autres interfaces

Correspondance avec l'implémentation réelle

Cette conception théorique se traduit en pratique par une implémentation **allégée** dans le code, où les principes de séparation sont maintenus tout en **évitant la sur-ingénierie**. Les services implémentent directement les trois CU requis, utilisant les utilitaires fournis pour la persistance et la communication externe.

8. Données clés par cours

Informations de base (toujours affichées) :

- Code du cours (ex: IFT2255)
- Titre complet
- Nombre de crédits
- Horaire des sessions
- Préalables et corequis

Informations conditionnelles (si connecté et disponibles) :

- Compatibilité avec votre profil
- Charge de travail estimée (si ≥5 avis)
- Difficulté moyenne (si ≥5 avis)
- Commentaires d'étudiants (si ≥5 avis)

---

9. Valeur ajoutée

- **Gain de temps** : Plus besoin de chercher sur Planifium + Discord + forums
- **Meilleures décisions** : Comparaison objective avec données réelles
- **Personnalisation** : Suggestions adaptées à chaque profil
- **Fiabilité** : Seuil minimum d'avis pour éviter les biais

---

10. Oracle de tests

✔ 1. Test searchCoursesBySigles_returnsCourses

Entrée : sigles = "IFT2255,IFT2015", name = null.

Sortie attendue : une liste non nulle et non vide, contenant au minimum les cours dont les identifiants sont IFT2255 et IFT2015.

État du système après l'appel : un appel HTTP GET est envoyé à l'API Planifium via

/api/v1/courses?courses_sigle=IFT2255,IFT2015&response_level=min.

Aucune modification de fichiers, de base de données ou d'état interne.

Type : Succès.

Description : Si des sigles valides sont fournis, la méthode doit retourner les cours correspondants.

✔ 2. Test searchCoursesWithoutParams_throwsException

Entrée : sigles = null, name = null.

Sortie attendue : une exception IllegalArgumentException.

État du système après l'appel : aucun appel HTTP n'est effectué, et l'objet CourseService reste inchangé.

Type : Échec.

Description : La méthode doit refuser une recherche sans aucun paramètre et signaler une erreur de validation.

✔ 3. Test getCourseDetails_validId_returnsDetails

Entrée : courseId = "IFT2255".

Sortie attendue : un objet CourseDetails non nul, avec id = "IFT2255" et un name non vide.

État du système après l'appel : un appel HTTP GET est envoyé à

/api/v1/courses/IFT2255.

Aucune modification de données locales.

Type : Succès.

Description : Avec un identifiant valide, la méthode doit retourner les détails complets du cours.

✔ 4. Test compareCourses_emptySigles_throwsException

Entrée : sigles = " " (une chaîne vide ou composée uniquement d'espaces).

Sortie attendue : une exception IllegalArgumentException.

État du système après l'appel : aucun appel HTTP n'est effectué, aucun état interne n'est modifié.

Type : Échec.

Description : Si la liste de sigles est vide ou invalide, la comparaison doit être refusée.