Rapport de projet – PlanifiumHelper

API d'aide à la planification de cours (UdeM) + bot Discord (collecte d'avis)

Date : 30/12/2025

1. Introduction

PlanifiumHelper est une application qui expose une API HTTP (JSON) permettant d'aider un étudiant à rechercher des cours, consulter leur fiche, comparer des cours, consulter des horaires, vérifier une éligibilité (prérequis/cycle), et produire un horaire pour un ensemble de cours en détectant des conflits. Le projet inclut aussi un bot Discord (Python) qui extrait des avis depuis un canal Discord et les transmet à l'API d'avis.

L'application serveur est écrite en Java avec Javalin et Jackson. Elle consomme une API externe Planifium (planifium-api.onrender.com) pour les données académiques (cours, programmes, horaires, etc.).

2. Contexte, objectifs et portée

Objectifs principaux :

- Fournir une interface HTTP simple pour interroger des informations sur les cours.
- Automatiser des opérations utiles à la planification (comparaison, éligibilité, horaire multi-cours).
- Centraliser des avis d'étudiants (provenant de Discord) et les rendre consultables par code de cours.

Portée (scope) :

- Inclut : API REST, modèles de domaine, services métier, contrôleurs, tests automatisés JUnit.
- Inclut : bot Discord de collecte et export des avis.
- Exclut : interface Web complète (front-end), authentification/autorisation avancée, persistance DB (avis stockés via flux/implémentation du service).

Hypothèses / contraintes :

- Dépendance à une API externe (Planifium) : disponibilité réseau requise pour certains cas.
- Les codes de cours (sigles) sont normalisés (espaces/majuscules) dans la logique d'éligibilité.
- Le format d'échange est JSON (Content-Type par défaut configuré).

3. Besoins (exigences)

3.1 Besoins fonctionnels

ID	Besoin	Composant	Priorité	Critères d'acceptation (oracle attendu)
BF-01	Rechercher des cours par sigle et/ou nom (recherche simple).	Service CourseService.searchCourses(sigles, name).	Must	Retourne une liste non vide pour un sigle valide; retourne liste vide si sigle inconnu; lève IllegalArgumentException si aucun critère.
BF-02	Rechercher des cours de manière avancée (sigle partiel / nom / description).	Service CourseService.searchCoursesAdvanced(siglePartial, name, description).	Should	Retourne une liste (potentiellement vide) cohérente avec les filtres fournis.
BF-03	Consulter la fiche détaillée d'un cours (crédits, description, prérequis, disponibilités).	Service CourseService.getCourseDetails(courseId).	Must	Retourne un CourseDetails; champs essentiels non nuls lorsque disponibles.
BF-04	Comparer plusieurs cours (charge, difficulté, nb d'avis, etc.).	Service CourseComparisonService.compareCourses(courses).	Should	Retourne une liste de ComparedCourse (une entrée par cours demandé).
BF-05	Consulter l'horaire d'un cours pour un trimestre (sections, type, jour, heures).	Service CourseScheduleService.getCourseSchedule(courseId, semester).	Must	Retourne une liste non vide pour un couple (cours, trimestre) valide; lève une exception pour

				paramètres invalides ou cours inexistant.
BF-06	Obtenir les cours d'un programme.	Service ProgramService.getCoursesForProgram(programId).	Should	Valide les paramètres et signale les erreurs (IllegalArgumentException ou RuntimeException selon le cas).
BF-07	Obtenir les cours offerts pour un programme et un trimestre.	Service CourseService.getCoursesForSemesterAndProgram(semester, programId).	Should	Lève IllegalArgumentException si semester est null/vide; signale RuntimeException si l'API externe échoue.
BF-08	Vérifier l'éligibilité à un cours selon les cours complétés et le cycle.	Service EligibilityService.checkEligibility(courseId, completedCourses, cycle).	Must	Retourne EligibilityResult : eligible, missingPrerequisites, reason (ex: "Prérequis manquants", "Cycle insuffisant").
BF-09	Soumettre un avis et consulter les avis par code de cours.	OpinionService.addOpinion(opinion) et OpinionService.getOpinions(courseCode).	Should	Après ajout, getOpinions(course) ne retourne que les avis liés au course_code; retourne liste vide si aucun avis.
BF-	Générer un horaire pour un ensemble	CourseSetService.createCourseSetSchedule(courseIds, semester) +	Should	Retourne CourseSetResult

10	de cours et détecter les conflits.	detectConflicts(...).		(schedules + conflicts); respecte un maximum de cours (ex: 6).
BF-11	Consulter un résultat académique agrégé pour un cours (moyenne/score/participants).	ResultService.getResultForCourse(courseId).	Could	Retourne AcademicResult ou signale une erreur en cas de paramètres invalides ou d'échec de récupération.

3.2 Besoins non fonctionnels

- BNF-01 – Robustesse : validation des paramètres (null/vide) et messages d'erreur explicites.
- BNF-02 – Interopérabilité : API REST JSON, compatible avec clients (HTTPie/Postman/bot Discord).
- BNF-03 – Maintenabilité : séparation Controller / Service / Model / Util (HttpClientApi).
- BNF-04 – Observabilité minimale : logs via slf4j-simple; messages console au démarrage.
- BNF-05 – Dépendance réseau : prévoir des erreurs/retours non-200 de l'API externe.

4. Cas d'utilisation

Acteur principal : Étudiant (utilisateur). Acteurs secondaires : API Planifium (externe), bot Discord (pour la collecte d'avis).

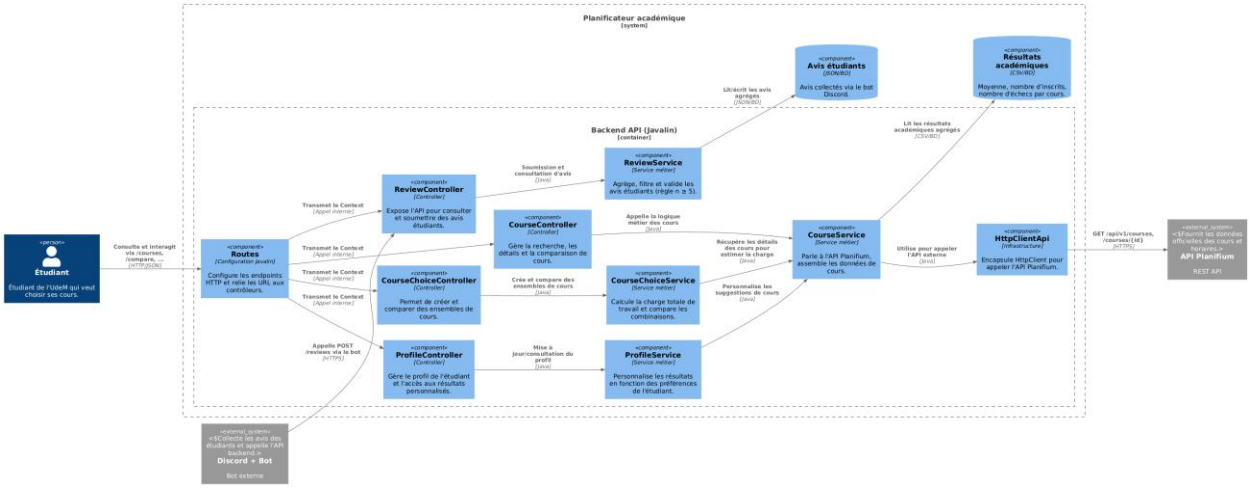
ID	Nom	Résumé
UC-01	Rechercher des cours	L'étudiant fournit un sigle et/ou un nom; le système retourne une liste de cours.
UC-02	Consulter la fiche d'un cours	L'étudiant choisit un cours; le système retourne la fiche détaillée (prérequis, description, disponibilités).
UC-03	Comparer des cours	L'étudiant fournit une liste de cours; le système renvoie un comparatif

		(charge/difficulté/avis).
UC-04	Consulter l’horaire d’un cours	L’étudiant fournit un cours et un trimestre; le système renvoie les plages horaires par section.
UC-05	Vérifier l’éligibilité	L’étudiant fournit ses cours complétés et son cycle; le système indique si l’inscription est possible et pourquoi.
UC-06	Construire un horaire multi-cours	L’étudiant fournit un ensemble de cours; le système renvoie un horaire et les conflits détectés.
UC-07	Soumettre / consulter des avis	Les avis sont soumis (via API/bot Discord) et consultés par code de cours.
UC-08	Lister les cours d’un programme (et éventuellement par trimestre)	Le système retourne la liste selon programme et trimestre.

5. Architecture

Technologies : Java, Maven, Javalin (serveur HTTP), Jackson (JSON), JUnit 4/5, Mockito. Bot Discord en Python (discord.py, aiohttp).

Figure 1 – Vue d'architecture (C4) fournie dans le dépôt.



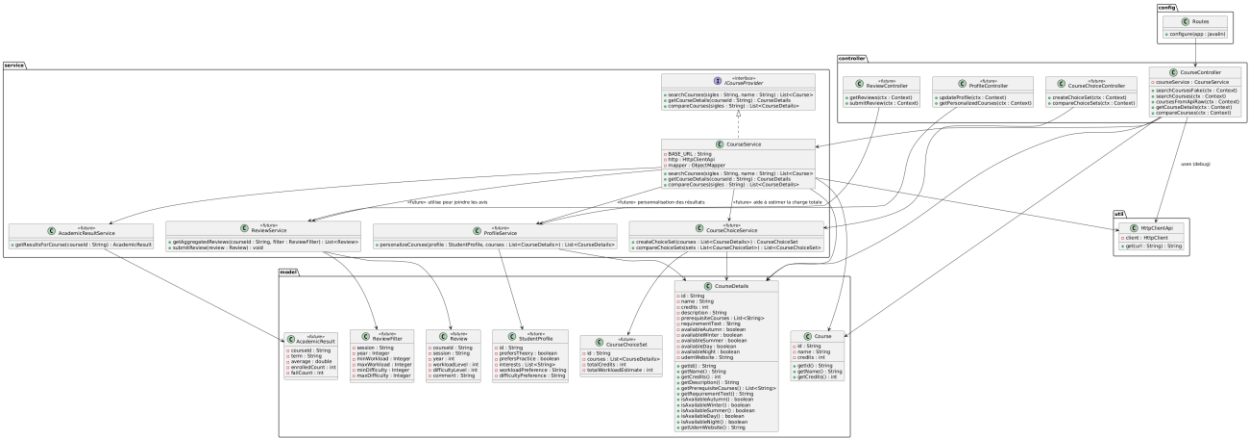
Découpage logique :

- Controller : endpoints Javalin (adaptation HTTP → appels service).
- Service : logique métier (validation, appel API externe, agrégation).
- Model : objets de domaine (Course, CourseDetails, CourseSchedule, Opinion…).
- Util : HttpClientApi (client HTTP standard java.net.http).

API externe : planifium-api.onrender.com/api/v1 (cours, programmes, horaires, résultats).

6. Diagramme de classes

Figure 2 – Diagramme de classes (fourni dans le dépôt).



Modèle de domaine (principales entités) :

Course : id, name, credits

CourseDetails : id, name, credits, description, prerequisiteCourses, requirementText, availableAutumn/Winter/Summer, availableDay/Night, udemWebsite

CourseSchedule : section, activityType, day, startTime, endTime

EligibilityResult : eligible, missingPrerequisites, reason

Opinion : course_code, text, professor_name (+ métadonnées Discord)

CourseConflict : courseA, courseB, day, startTime, endTime

CourseSetResult : schedules (CourseScheduleWithCourse), conflicts (CourseConflict)

AcademicResult : sigle, nom, moyenne, score, participants, trimestres

7. Diagrammes de séquence

Les diagrammes ci-dessous décrivent les scénarios principaux (UC).

Figure 3 – Séquence UC-01 : Recherche de cours.

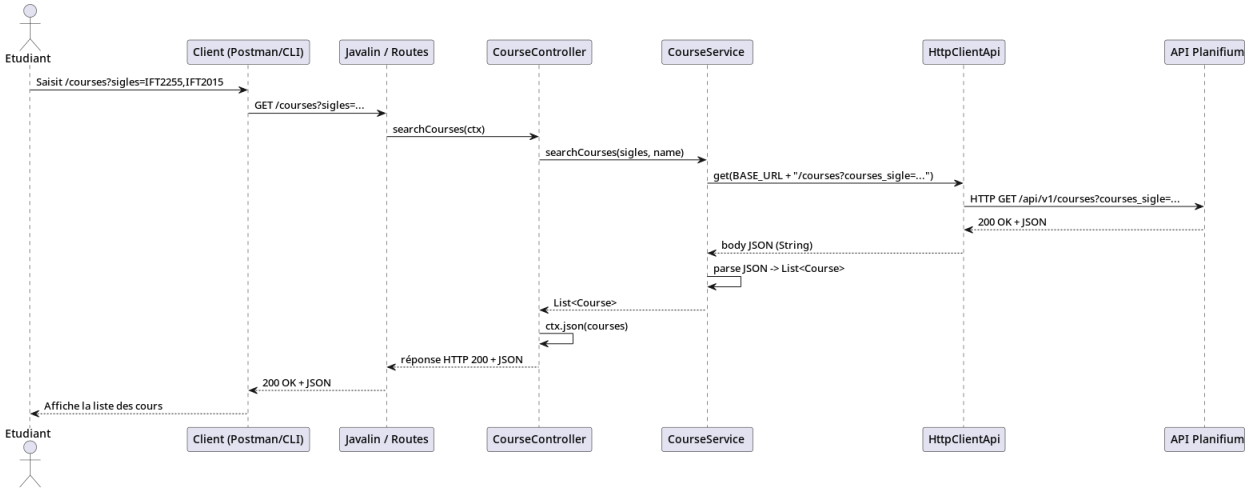


Figure 4 – Séquence UC-02 : Consultation fiche cours.

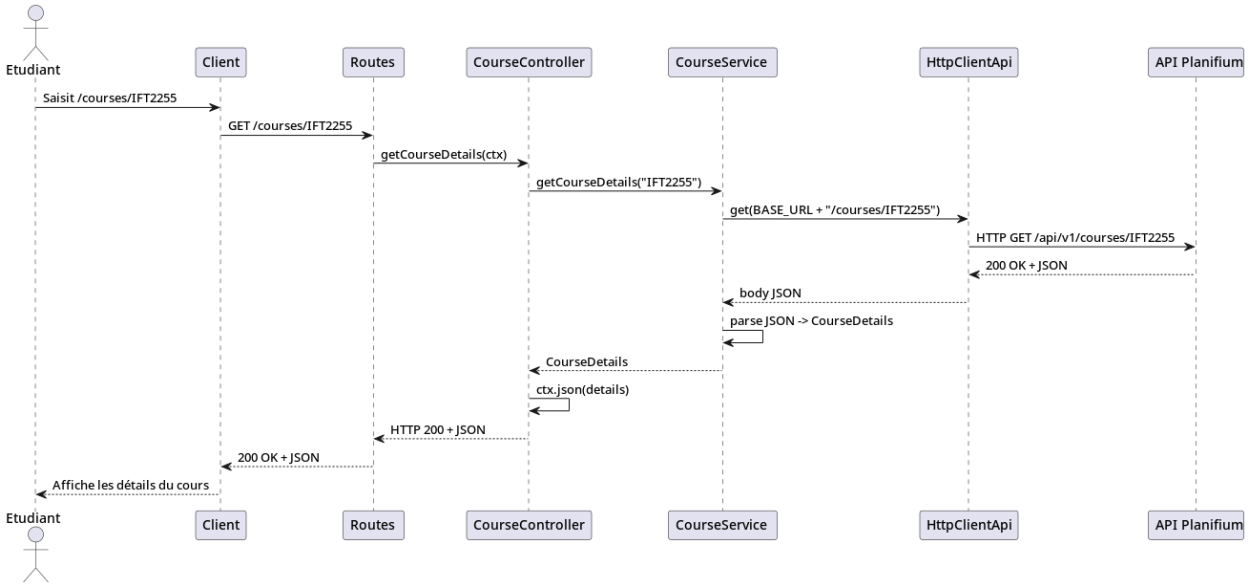


Figure 5 – Séquence UC-03 : Comparer cours.

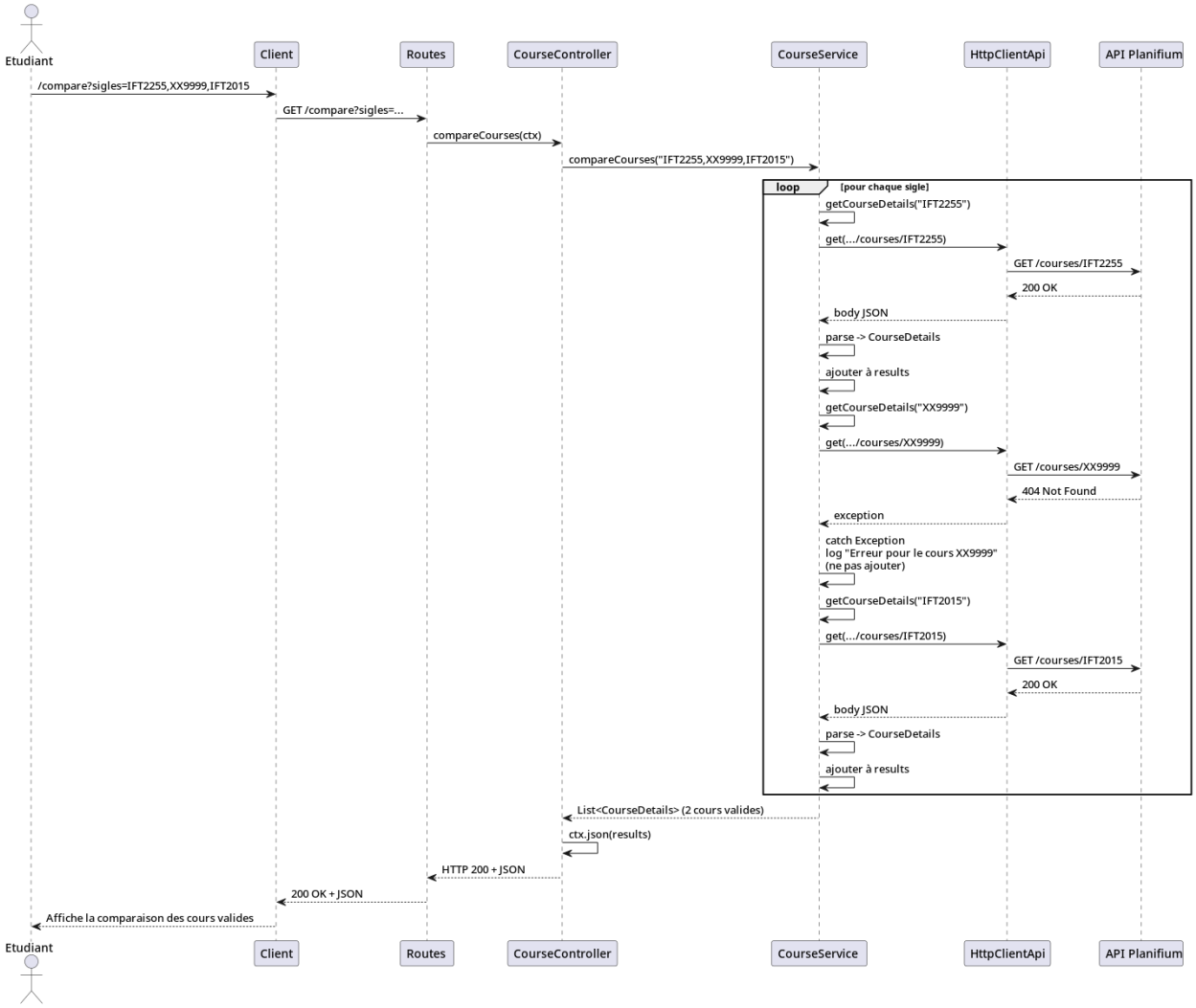
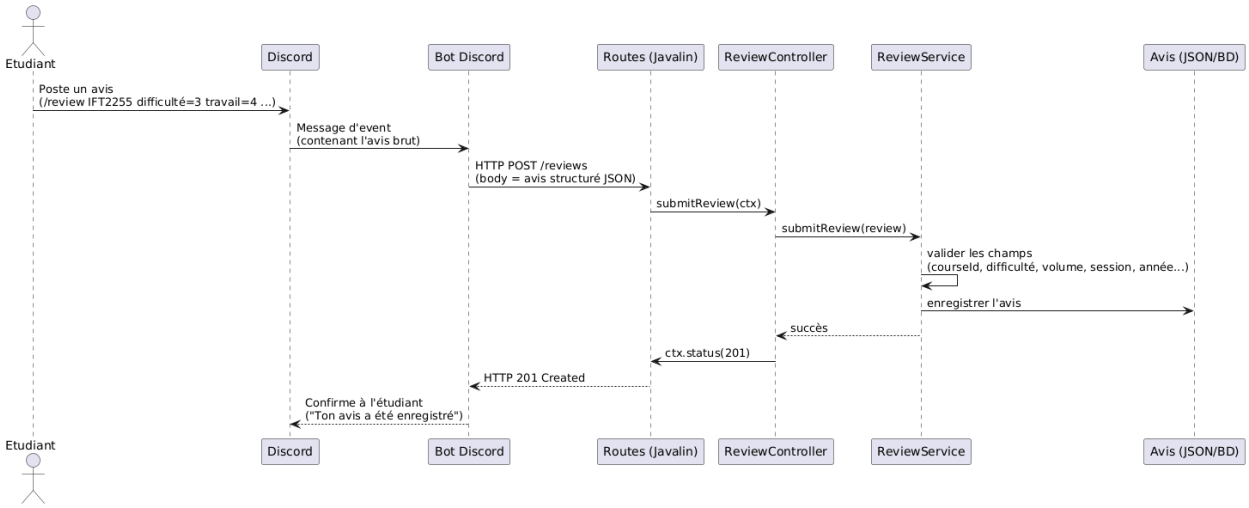


Figure 6 – Séquence UC-07 : Soumettre un avis.



Séquences additionnelles (PlantUML) :

8. Tests

8.1 Stratégie

Les tests sont principalement des tests unitaires de services (JUnit 4 et JUnit 5). Ils valident la gestion des erreurs (paramètres null/vide), ainsi que des scénarios métiers (éligibilité, filtrage d'avis, récupération d'horaires). Une collection HTTP (src/test/http) permet aussi des tests manuels d'endpoint.

8.2 Oracles (résultats attendus)

Test Oracle 1 : Recherche par Sigle Exact

Objectif : Vérifier qu'un cours peut être trouvé par son sigle exact.

Description : Ce test envoie une requête avec le sigle exact "IFT2255" à l'API de recherche. Il vérifie que la réponse contient exactement un cours, que ce cours a le bon sigle, un nom non vide, et un nombre de crédits valide (entre 1 et 15). Ce test valide que le système peut identifier précisément un cours par son identifiant unique.

Test Oracle 2 : Échec de Recherche Sans Critères

Objectif : Vérifier que la recherche nécessite au moins un critère.

Description : Ce test tente d'effectuer une recherche sans fournir ni sigle ni mot-clé. Il vérifie que le système rejette cette requête en levant une exception IllegalArgumentException. Ce test garantit que l'API n'accepte pas les requêtes vides qui surchargeraient inutilement le système.

Test Oracle 3 : Prérequis Obligatoire pour IFT2255

Objectif : Vérifier que IFT1025 est un prérequis obligatoire pour IFT2255.

Description : Ce test vérifie l'éligibilité d'un étudiant qui n'a pas complété IFT1025 pour s'inscrire à IFT2255. Il doit retourner "non éligible" avec la raison "Prérequis

manquants" et lister IFT1025 comme prérequis manquant. Ce test valide une règle métier fondamentale du département d'informatique.

Test Oracle 4 : Validation du Format des Trimestres

Objectif : Vérifier que seuls les trimestres au format standard sont acceptés.

Description : Ce test tente de récupérer un horaire avec un trimestre invalide "Automne2024". Il vérifie que le système rejette ce format en levant une IllegalArgumentException. Seuls les formats A24, H25, E25 (lettre suivie de 2 chiffres) doivent être acceptés. Ce test assure la cohérence des données temporelles.

Test Oracle 5 : Isolation des Avis par Cours

Objectif : Vérifier que les avis sont correctement associés à leur cours.

Description : Ce test ajoute des avis pour différents cours (IFT2255, IFT2015), puis récupère uniquement les avis pour IFT2255. Il vérifie que seuls les avis de IFT2255 sont retournés, et aucun avis d'autres cours. Ce test valide l'intégrité des données et la séparation des informations par cours.

.3 Exécution des tests

Commande Maven :

```
mvn test
```

9. Exécution et endpoints

Démarrer le serveur :

```
mvn -q exec:java  (ou exécuter la classe com.diro.ift2255.Main)
```

Le serveur écoute par défaut sur http://localhost:7000 (voir Main.java).

Endpoints documentés dans le code :

- GET /ping
- GET /courses-fake
- GET /courses?sigle=... | /courses?name=... | /courses?description=...
- GET /courses/:id
- GET /courses/semester-program?semester=...&program=...
- GET /courses/{id}/schedule?semester=...
- GET /compare-courses?courses=IFT2255,IFT2015,...
- POST /course-sets/schedule
- POST /api/opinions
- GET /api/opinions?course_code=...

Bot Discord :

Le bot (discord-bot/bot.py) lit un canal Discord et poste les messages/avis vers une URL d'API configurée par OPINION_API_URL dans le fichier .env. Il peut aussi exporter les avis en NDJSON (discord-bot/exports/opinions_stream.ndjson).

10. Conclusion

PlanifiumHelper fournit une base cohérente pour la planification de cours : recherche, fiche, comparaison, horaires, éligibilité, et avis. Les tests automatisés décrivent les oracles clés (validation, erreurs attendues et résultats). Les améliorations futures naturelles incluent une persistance durable (DB), une couche d'authentification, et un front-end utilisateur.

Javadoc (hors-ligne) : Le Javadoc est inclus dans le livrable. Ouvrir apidocs/index.html (index des classes : apidocs/allclasses-index.html).

Pour le BOT de collecte d'informations. Voici le lien : https://discord.gg/JRUGrmuQ