Cégep de Sainte-Foy – Hiver 2026
Développement de services web – 420-W41-SF

Travail Pratique 1 (30%)

API REST de gestion de location d'équipements sportifs

23 février 2026

Préparé par
Karine Filiatreault
--
Présenté par
Stéphane Delêtre

OBJECTIF DU TRAVAIL

Contexte de réalisation : Individuel

Ce travail vise à produire un API REST pour une application de gestion de location d'équipements sportifs en Laravel. Conjointement au développement de l'API, une suite de tests unitaires et la documentation *Swagger* de l'API sera à joindre au travail. En plus d'une remise du travail dans LEA, un dépôt Git devra être maintenu.

Le professeur se réserve le droit de convoquer l'étudiant afin de lui poser des questions sur son travail et d'ainsi valider sa compréhension.

DÉCOUPAGE DU TRAVAIL

L'API devra être bâti à partir du schéma de base de données fourni en annexe. Afin de faciliter le travail soutenu et la qualité des requis, ce TP est découpé en plusieurs remises qui seront expliquées en détails par la suite :

Partie 1	Dimanche 1 mars	Migrations Modèles de l'application Fabriques (*factories*) *Seeds* Dépôt Git
Partie 2	Dimanche 8 mars	Ressources Contrôleurs Routes Logique applicative Collection *Postman* de requêtes pour chaque route
Partie 3	Dimanche 15 mars	Tests unitaires Documentation de l'API
Partie 4	Lundi 16 mars	En classe - quiz sur le TP (*format papier*)

PARTIE 1 – CONSTRUCTION DE LA BASE DE DONNÉES

Dans cette partie, le travail à effectuer est le suivant :

Tâches	✓
Créer un dépôt sur **GitHub** (*ou GitLab*) nommé **TP1_Nom_Prenom**. Mon utilisateur Gitlab est « sdeletre », merci de m'ajouter comme « Collaborator » ou « Maintainer ». Une utilisation régulière doit être faite! Une fonctionnalité par branche.	
Étudier attentivement le schéma (*voir annexe fournie*) et les relations (*ne pas négliger cette étape)*.	
Développer les migrations afin de créer la BD selon le schéma de BD fourni en annexe (*attention à l'ordre de création des tables et aux spécifications du schéma, types, longueurs, etc.)*. **Attention !** *equipment* est un mot qui ne se compte pas en anglais donc la table sera au singulier. **Note** : Laravel vient par défaut avec une migration pour la table User, modifier celle-ci pour l'adapter au schéma.	
Développer les modèles (`$fillable` *et relations*). La table composite `equipment_sport` n'aura pas de modèle. **Note** : Même commentaire que la section précédente pour le modèle User, il existe déjà, donc modifier celui-ci. **Attention !** La table composite `equipment_sport` vise à représenter une relation « **many-to-many** » dans le modèle (*un sport peut utiliser plusieurs équipements et vice-versa).* Voici la documentation afin de construire les modèles impliqués dans cette relation : https://laravel.com/docs/master/eloquent-relationships#many-to-many	
Quatre (4) fichiers `.sql` de *Seed* sont donnés pour les tables **`sports, categories, equipement et equipment_sport.`** 1) Créer un *seeder* pour ces 4 tables en implantant le *seed* via le fichier sql. 2) Coder le **`DatabaseSeeder`** afin de lancer ces *Seeders* en séquence, encore une fois, attention à l'ordre.	

Créer une fabrique (**`factory`**) pour les autres tables de la BD. Les données injectées doivent utiliser la libraire **`Faker`** et respecter les spécifications du schéma (*types, longueur, etc.*). **Note** : Même commentaire pour la **`factory User`** **Important** : Pour les tables **`rentals`** et **`reviews`**, assurer une variété de valeurs sur les clés étrangères, on ne voudra surtout pas « hardcoder » toutes les locations comme étant pour 1 seul équipement.	
Après le bloc d'instructions de *seed*, effectuer un *seed* à l'aide des *factories* afin de *seeder* les tables restantes. Entre autres, insérer au moins 10 sports qui ont au moins 2 équipements chacun.	

Remise

Date	**Dimanche 1 mars, 23h55**
Objectifs	• BD qui peut être installée à l'aide de migrations, et « seedable » en sa totalité. • Couche de modèle complète et prête à être utilisée pour l'API.
Modalités	• Dossiers *vendor* et *node_modules* supprimés et archive zippée (.zip) sur LEA. • L'archive devra contenir un document texte nommé « git.txt » qui contient le lien vers votre dépôt Git. Je devrai avoir les accès *Collaborator* ou *Maintainer* sur celui-ci.

PARTIE 2 – LOGIQUE D'AFFAIRE

Dans cette partie, nous allons créer la logique d'affaire afin de :

Cas	Description
1	Recevoir l'information de tous les équipements
2	Recevoir l'information d'un équipement en particulier
3	Recevoir l'indice de popularité d'un équipement (*expliqué plus bas*)
4	Créer un utilisateur
5	Mettre à jour un utilisateur (*mise à jour complète et non partielle*)
6	Supprimer une critique
7	Recevoir la moyenne du prix total de location d'un équipement (*expliqué plus bas*)

Tâches	✓
Créer les ressources d'API pour tous les modèles développés dans la partie 1. **Note** : Dans les ressources, on n'inclura pas les ressources avec lesquelles nous sommes en relation, nous nous en tiendrons aux données de base du modèle en prenant soin d'inclure les valeurs de clés étrangères. Les relations seront exploitées via les contrôleurs.	
Définir les routes de l'API selon ce qui a été présenté ci-haut. Cette étape mérite une réflexion afin de : • Bien définir la route selon les standards vus en classe; • Cibler le bon contrôleur et la bonne méthode pour effectuer le travail.	
Coder les méthodes dans les contrôleurs afin de mettre en place la logique d'affaire demandée. On n'oubliera pas : • De gérer les codes de statut manuellement; • D'utiliser un validateur (*de champs requis*) sur les routes de création et de mise à jour; • D'appliquer les bonnes pratiques de code (*en particulier, l'utilisation de constantes)*. **Pour le cas 3 ci-haut** : Calculer un **indice de popularité** basé sur : • Le nombre total de locations; • La note moyenne (*si existante*). Règles : • Popularité = (nombre de locations × 0.6) + (note moyenne × 0.4) • Si un équipement n'a aucune évaluation, la note moyenne est considérée comme 0. **Pour le cas 7 ci-haut** : On pourra spécifier (en paramètre **`get`**) jusqu'à 2 critères de recherche : • `minDate` (*utiliser les locations uniquement à partir de cette date*) • `maxDate` (*utiliser les locations uniquement avant cette date*) (*Suite page suivante*)	

➢ Les dates inscrites sont incluses. ➢ On peut utiliser une seule des 2 dates, les 2 en même temps ou aucune. ➢ `minDate` doit être inférieur à `maxDate`. ➢ Le format de date doit être valide. https://laravel.com/docs/master/queries#where-clauses **Spécification** : On demande de paginer les résultats de la recherche à **20** enregistrements.	
Tester toutes les routes, sans exception, avec *Postman* **Vous devez partager votre collection de requêtes avec le professeur.**	

Remise

Date	**Dimanche 8 mars, 23h55**
Objectifs	• API et logique applicative fonctionnelles • Gestion des statuts (*codes*)
Modalités	• Collection de requêtes *Postman* : partager l'url • Dossiers *vendor* et *node_modules* supprimés et archive zippée (.zip) sur LEA

PARTIE 3 – TESTS ET DOCUMENTATION

Dans cette dernière partie, nous allons finaliser l'API avec une suite de tests et une documentation en *Swagger*. Si vos tests soulèvent des erreurs de conception dans votre logique d'affaire, il s'agit évidemment de les corriger.

Tâches	✓
Codez les *features tests* pour les cas **1 à 6** de la partie 2 en n'oubliant pas : • Les cas généraux • Les cas d'erreur avec une gestion des codes de statut applicables selon le type de route (*200, 201, 204, 404, 422*) **Note** : Il est conseillé de « seeder » la base de données au début de chaque test.	

Produisez la documentation *Swagger* de votre API, assurez-vous de vérifier celle-ci. **Attention !** Étant donné sa nature, le cas **#9** devra avoir des paramètres documentés. https://blog.quickadminpanel.com/laravel-api-documentation-with-openapiswagger/ **Important** : Je devrai être en mesure de tester votre API entièrement dans *Swagger*	

Remise

Date	**Dimanche 15 mars, 23h55**
Objectifs	• *Feature* tests complets • Documentation *Swagger* • API fonctionnelle à 100%
Modalités	• Dossiers *vendor* et *node_modules* supprimés et archive zippée (*.zip*) sur LEA

PARTIE 4 – QUIZ

Un quiz sur le travail pratique aura lieu en classe.

Date	**Lundi 16 mars, durant le cours**
Objectifs	• Vérifier la compréhension • Préparer à l'examen
Modalités	Quiz sur papier

COLLABORATION ET PLAGIAT

Une évaluation sommative ne peut **en aucun temps** être le produit d'une collaboration ou de partage de code entre des personnes ou équipes différentes. De plus, vous devez obligatoirement **citer vos sources** si vous vous inspirez de code trouvé sur internet ou provenant de Chat GPT ou autre. Pour les **IA**, indiquez le ***prompt*** utilisé pour interroger l'IA. Une utilisation non citée entrainera la note de 0. **L'essentiel de votre travail ne doit pas provenir de l'IA.** Si de tels comportements sont observés, le professeur se réserve le droit de mettre fin à l'évaluation des étudiants fautifs en cours de réalisation et d'attribuer la note 0. Voir la clause 6.1.12 du plan de cours. Chaque cas de plagiat, de fraude ou de tricherie sera signalé à la Direction des études.

Source : https://sites2.csfoy.ca/presentationtravaux/limportance-de-citer-ses-sources/

(*Annexe page suivante*)

ANNEXE – MODÈLE DE DONNÉES