MyDigitalSchool — Évaluation 2/2 — API, ORM et GraphQL

Date: 25/11/24

Auteur : Paul Schuhmacher

Version : 1

Comment réaliser ce travail

- Vous devez faire cette évaluation **par groupe de 2** idéalement, ou seul ;
- **Chaque membre du groupe doit contribuer** de manière significative au projet (commits) ;
- Vous pouvez **utiliser les technologies de votre choix**. Vous devez seulement **concevoir et implémenter une web API RESTful**, utiliser un **ORM** et **une base de données relationnelle**.

Comment rendre votre travail

> *Merci de **lire attentivement** les consignes !*

Déposer votre travail **sur un dépôt git public** (GitHub, Gitlab, BitKeeper, etc.) et fournir **le lien du dépôt**. Le dépôt contiendra le code source de l'API et un fichier `README` donnant les instructions pour installer et lancer le projet.

> *Penser à utiliser un `.gitignore` pour éviter de pousser sur votre dépôt le contenu de `node_modules` et du dossier `db` contenant le volume docker de la base de données !*

Envoyer votre travail dans **un e-mail** à l'adresse suivante :
paul.schuhmacher.ext@eduservices.org, **ayant le sujet suivant** :

```
rendu - orm api graphql
```

Dans l'e-mail :

- **Mettre le lien de votre dépôt Github/Gitlab qui héberge votre projet** ;
- **Mettre le nom et prénom de chaque membre du groupe** ;

> *Vous devez rendre votre travail **avant la date butoir fixée ensemble sous peine de pénalité** : 1 point le premier jour de retard, 2 points le deuxième jour de retard, et ainsi de suite jusqu'à un minimum de 0.*
>
> *Merci de vérifier que le dépôt est **public** !*
>
> *Vous êtes encouragé·e à mettre votre collègue en copie du mail de rendu.*

Notation

Le projet **est noté sur 20**, **coefficient 4**.

- **Documentation du projet** : **7 points**. Le projet doit être correctement versionné avec git (et le `.gitignore` bien défini) et hébergé sur Github/Gitlab (ou autre). Le dépôt doit contenir un `README` bien formé (voir les instructions ci-dessous).
- **Conception de l'API** : **11 points**. L'API doit être RESTful et la conception doit être documentée comme demandé :
  - URL bien formées et implémentation de l'interface uniforme (3pt)
  - Choix pertinent des codes status (1pt)
  - Réponses hypermédias respectant la spécification HAL (3pt)
  - OAD utilisable pour tester l'API avec Swagger Editor (4pt)
- **Implémentation** : **11 pts**
  - Votre projet doit répondre correctement aux besoins exprimés (7pt)
  - Respecter les contraintes (4pt) :
  - Exposer une ressource GraphQL (2pt)

À rendre

L'URL de votre dépôt. Votre dépôt **doit contenir** *a minima* :

- Un fichier `README.md` **bien formé** avec les sections suivantes :
  - **Table des matières** (vous pouvez la générer automatiquement), pour naviguer facilement dans le document (1pt)

    - **Lancer le projet** : instructions pour installer les dépendances, lancer votre projet, donner l'URL d'entrée du service, insérer le jeu de données test en base, lancer Swagger Editor, etc. (1pt)
    - **Utiliser le service** : présenter le cas nominal d'utilisation du service pour guider l'utilisateur (1pt)
    - **Conception** :
        - **Dictionnaire des données** (2pt)
        - **Un tableau récapitulatif avec les 5 colonnes suivantes** : (3pt)
            - **Ressource** : le nom de la ressource
            - **URL** : l'URL de la ressource
            - **Méthodes HTTP** : la liste des méthode HTTP implémentées sur cette ressource
            - **Paramètres d'URL/Variations** : indiquer les paramètres d'URL et les valeurs possibles s'il y'en a
            - **Commentaires** : description supplémentaire, contraintes sur la ressource
    - **Sécurité** : discuter des mesures de sécurité mises en place dans votre système ; (1pt)
    - **Remarques** : si vous voulez faire des remarques sur votre travail, difficultés rencontrées
    - **Références** : la liste des références (sites web, cours, livre, article, billet de blog, etc.) qui vous ont aidé à concevoir et développer votre système ;
- **Le code source de votre API**
- **Un Open API Description File (OAD)** au format **yaml**, décrivant votre API

> *Inspirez-vous du travail réalisé ensemble sur le système de billetterie et relire les supports de cours. Si vous le souhaitez utiliser le kit de développement mis à votre disposition pour l'implémentation.*
>
> *Un dépôt sur le web **est un site web** et le fichier `README.md` en est sa page d'accueil. Soignez-le pour aider vos utilisateur·s (et vous-même !) à utiliser votre projet.*

Contraintes

Le projet d'API **doit respecter les contraintes suivantes** :

- **Suivre une approche *design-first***, la conception doit être formalisée avec la spécification OpenAPI. Un fichier OAD au format yaml doit être présent et utilisable via Swagger UI ou Swagger Editor pour parcourir et tester votre API ;
- Retourner des **représentations hypermédias** au format `JSON` et respectant la spécification `HAL` ;
- **Utiliser un ORM et une base de données relationnelle** (laissés au choix) ;
- **Le projet doit disposer d'une base de données avec un schéma opérationnel ainsi qu'un jeu de données minimal** au démarrage pour être utilisable. L'idéal étant d'utiliser **les migrations et seeders** (valorisé) de votre ORM ;

- **Bonus : Implémenter une mesure de sécurité additionnelle au choix** (protection anti brute-force sur le login, rating-limit, white-list/black-list, gestionnaire de processus Node, gestion aboutie des secrets/variables d'environnement, etc.) (+2pt)

  Ne pas répondre à une question bonus n'est pas pénalisant. Cela n'apporte que des points en plus.

Expression des besoins : Système de réservation de terrains de badminton

Une association de badistes souhaite créer un **service web** permettant de réserver des terrains de badminton pour ses adhérents·es. La municipalité a mis à disposition de l'association **une salle de 4 terrains** (A, B, C et D). **La salle est ouverte du lundi au samedi de 10 h à 22 h**. Il est possible de réserver des créneaux de **45 minutes**. Pour effectuer une réservation, il faut être **identifié** par le système (par un simple pseudo). Une réservation peut être annulée.

> *Par souci de simplification, **il ne sera possible d'effectuer des réservations que sur la semaine en cours**.*

La toiture du stade souffre de quelques problèmes d'étanchéité. Lors de fortes pluies, le terrain B est souvent impraticable les deux jours suivants. En attendant que des travaux soient effectués, l'association aimerait pouvoir rendre un terrain au choix **indisponible** de manière temporaire. **Un terrain indisponible ne peut accueillir de nouvelles réservations**.

Seul·e un·e administrateur·ice du système peut rendre indisponible un terrain. Cette ressource doit donc être protégée par authentification. L'administrateur·ice est identifié·e par le pseudo réservé `admybad` et le mot de passe `astrongpassword`.

On voudrait rapidement développer une application mobile de réservation qui utiliserait le service. Pour cela, exposer une ressource capable de **résoudre la requête GraphQL** suivante :

```
query GetAvailableSlots {
  availableSlots(date: "2024-11-27", terrain: "A") {
    time
    isAvailable
  }
}
```

Bien démarrer et conseils

> *Rappel : **Ne développez pas un site web**, mais **un service web**, destiné à être utilisé par des programmes !*

Suivez la méthode utilisée en cours (rappelée ici) :

1. **Déterminer** l'ensemble des données

2. **Décomposer** l’ensemble des données en *ressources*

   À mener en parallèle avec la conception de la base de données (dictionnaire des données, MCD)

3. **Pour chaque ressource** :
    1. **Nommer** la ressource avec des URI (et ses variations)
    2. **Implémenter** un sous-ensemble de l’interface uniforme (`GET`, `POST`, `DELETE`, `PUT`)
    3. **Étudier** la ou les représentations acceptées par les clients
    4. **Concevoir** la ou les représentations à mettre à disposition des clients
    5. **Intégrer** la ressource parmi celles qui existent déjà, en utilisant la hiérarchie entre ressources et les hypermédias
    6. **Envisager** la progression typique des évènements : qu’est-ce qui est censé se produire ?
    7. **Considérer** les cas d’erreurs : qu’est-ce qui peut mal se passer ?
4. Formaliser le design de votre API avec un OAD (spécification OpenAPI)
5. *Mocker* la base de données dans un premier temps ;
6. Utiliser la CLI de votre ORM pour gérer les entités et les migrations ;
7. Répartissez-vous bien le travail ;
8. Répondez aux spécifications de la manière la plus simple possible ;
9. Prenez le temps de vérifier vos propres instructions pour installer et lancer le projet en les suivant vous-même !