Université du Québec à Rimouski

Travail Pratique II

INF37407 - Technologies de l'inforoute

Professeur – Yacine Yaddaden, Ph. D.

→ Goulet, Bastien

TABLE DES MATIÈRES

INTRODUCTION

Tout système sérieux se retrouvant en production et offrant des fonctionnalités à des utilisateurs se doit d'implémenter une persistance des données. En effet, une application s'exécutant dans un navigateur ne possède pas les capacités pour enregistrer des données d'une manière permanente. Cette fonction doit être offerte par un système en arrière-plan exposé à l'internet. De plus, il est souhaitable, pour des raisons de conformité, de sécurité et de réduction des investissements requis dans le développement, qu'une API REST standardisée soit offerte. Grâce à cette API, les clients d'une entreprise ou même ses applications possédant une interface comme des applications web, mobiles ou *IoT* peuvent interroger les mêmes points d'entrées documentés de l'API et ainsi éviter à l'entreprise qu'elle ne doive développer un '*back-end*' spécifique à chaque application. Telle était notre mission dans le cadre de ce deuxième travail pratique dans le cadre du cours INF37407 et du projet de recherche sur l'aide à l'apprentissage de la lecture pour les enfants en situation de déficience intellectuelle. Il nous était demandé de développer des APIs REST pour l'application React créée précédemment durant le premier travail.

TECHNOLOGIES UTILISÉES

La technologie principale utilisée durant ce travail fut le langage Python. Il s'agit d'un langage interprété très populaire de nos jours. Il est multiplateforme et peut facilement s'installer sur n'importe quelle machine. Python est facile d'approche et offre une syntaxe épurée et simplifiée, comparativement avec bien d'autres langages orientés objet.

Cependant, le simple langage Python n'offre pas de moyen facile pour faire du développement web. Pour parvenir à cette fin, il est fortement recommandé d'utiliser un cadre de programmation. C'est ce que j'ai fait dans le cadre de ce travail. En effet, j'ai utilisé le cadre de développement web *Django*. Ce cadre, très populaire sur le marché du travail, est un ensemble d'outils et de librairies en Python offrant au développeur une manière précise de travailler afin de créer une application web. Le développeur peut utiliser les classes et les fonctions fournies avec Django pour facilement créer son application, s'il sait comment les utiliser. Le cadre offre les mécanismes nécessaires afin de démarrer un serveur web exposant l'application Django en local, ce qui est pratique pour le développement.

Afin de créer mes APIs REST, la librairie Django *djangorestframework* fut nécessaire. Je l'ai installée avec le gestionnaire de *packages PIP* et j'ai pu bénéficier directement des annotations *@api_view* ajoutées au-dessus des fonctions de vue de chaque application du projet Django pour les transformer en API REST. Des annotations particulières peuvent être utilisées afin d'informer *Swagger*, un outil de tests d'API REST, de la documentation de chaque point d'entrée.

De plus, le package *django.contrib.auth*, installé par défaut avec Django, m'a permis une gestion des utilisateurs simplifiée. En effet, ce package permet de générer les tables nécessaires à l'enregistrement des informations des utilisateurs. Il aurait été suffisant, par lui-même, si je m'étais contenté de l'authentification par mot de passe. Cependant, il nous était demandé d'utiliser l'authentification par l'entremise de jetons de sécurité. Pour y parvenir, j'ai dû installer le package *rest_framework.authtoken*. Ce package offrait des fonctionnalités de génération et de validation de jetons d'authentification.

Bien entendu, mon projet d'API Django aurait été inutile sans persistance des données. Heureusement, le cadre de développement offre, par défaut, un *ORM* (un système de liaison de modèles Python avec une base de données relationnelle) facile d'utilisation. Dans mon cas, j'ai utilisé une simple base de données *SqlLite*. Avec ce système de gestion de base de données, les données ne sont conservées que dans un simple fichier. Cependant, ce système de persistance des données était suffisant pour cette application. Les modèles représentant les différentes entités de mon application, tels les questions ou encore les quiz, pouvaient donc être facilement enregistrés dans la base de données.

Enfin, le package *gTTS* fut utilisé afin de générer automatiquement des fichiers audio à partir du texte envoyé à l'API. Ainsi, le consommateur de l'API n'a pas besoin de générer un fichier lui-même et de le téléverser sur le serveur.

Fonctionnement de l'API REST

L'application qui fut développée durant ce travail était une application web en Python utilisant le cadre Django et le package *djangorestframework* afin d'exposer des APIs REST utiles à la gestion des utilisateurs et aux objets de l'application React pour l'aide à la lecture.

Tout d'abord, deux APIs permettent la gestion des utilisateurs. Le premier, l'API *Authentification*, permet à n'importe qui de créer un compte de participant dans le système en fournissant, à travers une requête HTTP POST, un nom d'utilisateur, un mot de passe, le nom et le prénom et le mot de passe du participant. Une fois le compte créé, il est possible d'appeler le point d'entrée *login* en fournissant nom d'utilisateur et mot passe. L'API, si les informations sont valides, retournera un jeton d'authentification qui peut être utilisé pour représenter l'utilisateur et pour accéder aux autres points d'entrée.

Une fois authentifié, le consommateur peut consulter la liste des quiz, des questions, des textes, des mots de question, ses propres informations de compte ou encore ses résultats de tentatives de quiz. Il n'a accès à rien d'autre, cependant.

Pour accéder aux autres points d'entrée, il est nécessaire de s'authentifier en tant qu'administrateur. Ce rôle permet de gérer tous les comptes de participants, de gérer les questions, les quiz, les textes et les mots de questions et également de consulter tous les résultats des tentatives de quiz. L'administrateur peut donc, grâce à des requêtes HTTP POST, GET ou PUT, ajouter, modifier ou supprimer chacune des entités de l'application. Il faut noter que la suppression des entités n'effectue pas une requête *DELETE* auprès de la base de données. Le champ *is_active* est plutôt changé pour la valeur *false* pour l'enregistrement correspondant. La création ou encore la modification des questions, des textes, des mots de question et des réponses exécute systématiquement du code utilisant la librairie *gTTS* afin de traduire le contenu du champ *statement* en un fichier audio MP3 qui sera sauvegardé dans le dossier statique de l'application web et accessible depuis le réseau.

De plus, en appelant les points d'entrée *add_answer* de l'API *Question* et *create* de l'API *Words*, il est possible d'associer une réponse ou un mot avec une image qui

sera téléversée automatiquement sur le serveur et sauvegardée dans le dossier *media* du serveur. Toutes les images sont ainsi accessibles depuis l'internet étant donné que mon projet d'API REST en Django et mon application React ont été déployés sur des services en ligne dédiés à cet effet. Les liens vers ces applications sont détaillés dans le fichier *README.md* fourni avec le code source de ce travail.

Ce que vous avez aimé et moins aimé dans le projet

Ce que j'ai particulièrement aimé dans ce projet fut l'élaboration des différents points d'entrée de l'API, ainsi que la gestion des fichiers. En effet, créer chacune des vues avec les spécifications *Swagger* fut une révision des concepts d'API REST très efficace. Ce processus me permit d'apprendre les différentes fonctionnalités du cadre Django de fond en comble. De plus, il s'agissait de la première fois que j'utilisais le téléversement de fichiers à travers des requêtes HTTP. Ce fut extrêmement intéressant à mettre en place et je crois maîtriser le monde des requêtes HTTP un peu plus désormais.

Ce que j'ai moins aimé dans ce projet fut l'intégration de l'application React existante avec l'API REST. Étant donné que les représentations des entités dans le premier travail n'étaient pas nécessairement toutes pensées pour être communiquées et enregistrées à travers des requêtes HTTP et des API REST, la conversion des données fut un processus extrêmement frustrant et pénible. Contrairement à la gestion des fichiers, je n'ai pas l'impression d'avoir appris quoi que ce soit durant cette étape.

Problèmes et Difficultés rencontrées

Bien que le cadre de programmation web Django fut utilisé dans le cadre de ce travail, il va sans dire que le développement de cette application ne fut pas aisé ! En effet, j'ai rencontré trois problèmes notables durant le développement.

Tout d'abord, toute la portion portant sur le téléversement d'images du développement de l'application fut difficile bien qu'initialement, ce processus sembla simple. En effet, une fois les champs *image* enregistrant le lien vers l'image dans les modèles l'utilisant transformés en champ *models.ImageField*, je n'avais besoin que d'appliquer la migration sur la base de données, de spécifier dans les *serializers* que ces champs étaient des champs d'images et d'ajouter l'annotation *@parser_classes([MultiPartParser, FormParser])* au-dessus des vues offrant la fonction de téléversement d'images et le tour était joué ! En réalité, ce n'était pas aussi simple puisque *Swagger* ne supportait pas le type de contenu *application/multipart-formdata* pour les collections d'entités enfants, ce que j'utilisais pour représenter les mots et les réponses. Il a donc fallu que j'ajoute des points d'entrée afin de permettre le téléversement d'images pour un mot ou une réponse spécifique.

Les deux prochains problèmes se sont présentés durant l'intégration de l'API REST avec l'application React existante. En effet, durant l'intégration de la lecture et de la publication des résultats de tentatives de quiz, je me suis rendu compte que les résultats, dans la version précédente de l'application (celle du premier travail), n'étaient pas représentés sous une forme traduisible facilement en entités relationnelles. Mon coéquipier ayant programmé cette partie injectait les résultats dans l'entité user directement et représentait les associations entre les questions et les réponses comme des champs dans un objet javascript unique et non un tableau, ce qui était problématique. J'ai donc dû faire un effort considérable de conversion des résultats retournés par l'API en cette représentation étrange utilisée par l'application React.

Enfin, le téléversement d'images depuis l'application React fut très problématique. En effet, la librairie utilisée pour la gestion de l'état des formulaires, *react-hook-form*, ne gère pas nativement les champs de type fichier. De ce fait, seule une

chaîne de caractères étrange était insérée dans le champ *image* fourni avec la requête POST effectué auprès de l'API. J'ai dû parcourir le *DOM* dans la console de mon navigateur afin de déterminer quels traitements devaient être effectués afin d'extraire les fichiers de chacun des champs de type fichier et de construire l'objet Javascript *FormData* nécessaire pour ce type de requête.

Conclusion

En conclusion, ce travail fut une occasion d'or pour apprendre un nouveau langage, un nouveau cadre de programmation ainsi que divers mécanismes spécifiques aux requêtes HTTP. Bien que le travail fut très demandant ; en effet, j'ai dû le compléter seul, j'ai grandement apprécié d'enfin utiliser le langage Python dans le cadre d'un projet concret. De plus, les difficultés rencontrées m'ont contraint à comprendre réellement les mécanismes des requêtes HTTP, tels les en-têtes et les transferts de formulaires ou encore de blocs de données avec le *multipart-formdata*. Somme tout, je suis très satisfait de mon travail et j'espère qu'il pourra bénéficier Mme Jolicoeur, ne serait-ce qu'un iota.

Répartition du travail parmi les membres de l'équipe

- Bastien Goulet : J'ai tout fait le travail.

RÉFÉRENCES

[1] Yaddaden Y. INF37407 - Cours 08 - Initiation à Django - Introduction au Langage Python - V1.0

[2] Yaddaden Y. INF37407 - Cours 09 - Initiation à Django - Routage et gestion des requêtes (Contrôleur) - V1.0

[3] Yaddaden Y. INF37407 - Cours 10 - Initiation à Django - Initiation aux moteurs de templates (Vue) - V1.0

[4] Yaddaden Y. INF37407 - Cours 11 - Initiation à Django - Présentation des ORM (Modèle) - V1.0

[5] Yaddaden Y. INF37407 - Cours 12 - Initiation à Django - Création d'une API REST - V1.0

[6] Yaddaden Y. Django Demonstration - Temperature Converter. Repéré à https://github.com/yyaddaden/django-intro-demo

[7] Zolghadr T. (2022. janvier). Uploading Images to Django REST Framework from Forms in React. Repéré à https://dev.to/thomz/uploading-images-to-django-rest-framework-from-forms-in-react-3jhj