PROJET - JO Tickets

Compétences mobilisées

- Utilisation d'un gestionnaire de versions pour son code
- Usage de l'HTML et du CSS pour créer des pages statiques
- Utilistation d'une base de données pour stocker les inforlations importantes
- Utilisation de Python avec le framework Django pour interagir avec la BDD et proposer une interface d'administration

Contexte

Le Comité International Olympique fait de nouveau appel à vos talents ! En effet il a besoin d'un POC (Proof Of Concept) plus avancé pour sa compétition de football.

Il vous demande de créer une interface d'administration pour gérer les matchs de la compétition mais surtout une API utilisable depuis plusieurs application.

La première application mobile permettra de voir la liste des match, se connecter, acheter des billets et générer des qr codes pour les billets achetés.

La seconde application à destination des stadiers permettra simplement de scanner le QR Code d'un billet afin d'autoriser l'entrée, et indiquer la place au supporter.

Pour cela le CIO vous fourni une ébauche de début de projet avec 3 dossier pricipaux :

- `/admin/` qui contient le projet Django (déjà créé)
- `/mobile/` qui contiendra l'application pour les supporters
- `/scanner/` qui contiendra l'application pour les stadiers

Objectifs

Créer **3 applications** !

① Projet Django

Premièrement le projet Django, c'est la base. C'est lui qui accède à la base de donnée et fourni une API afin d'envoyer ces données aux autres applications. Il contient également une interface d'administration permettant de modifier les évenements et ajouter les scores de chaque matchs.

Important, seul les supers utilisateurs peuvent se connecter à l'interface d'administration.

Pour la base de donnée, le CIO vous à fourni une base, plusieurs fichier de `Model` et des données en SQL pour la garnir. Il vous faudra tout de même ajouter un model Ticket pour stocker les tickets achetés par les supporters.

Mise en place de l'API

Pour rappel la mise à disposition d'une **API** consiste à proposer des points d'entrée (**endpoint**) dans votre application, ces points d'entrés sont en réalité des **adresses URL**.

Par exemple si votre application tourne en local (`http://127.0.0.1:8000/`), vous pouvez proposer un *endpoint* correspondant à l'*URL* `/api/teams` qui renvera les équipes présente dans votre BDD. Il faudra donc inscrire dans le *fichier* `urls.py` que l'*URL* `/api/teams` correspond à une vue que vous aurez créée et qui renvois une *JsonResponse* contenant les équipes de votre BDD. Ainsi une autre application appellant `http://127.0.0.1:8000/api/teams/` aura en réponse un JSON contenant la liste des équipes.

---

② Application Mobile Supporter

L'application mobile supporter permettra de voir la liste des matchs dans leur ordre de diffusion, les supporters pourront acheter des billets pour chacun des matchs **SI il sont connectés**. Il devront donc pouvoir s'inscrire, et se connecter à l'application.

Important : L'inscription et la connexion se fait à travers l'API fourni par le projet Django. L'application mobile est un simple frontend, elle ne se connecte donc pas directement à une BDD. On enregistre donc les utilisateurs dans la BDD du projet Django.

Lors de l'achat du billet, le supporter peut choisir sa catégorie, le prix varie en fonction de la catégorie choisi. Les catégories sont les suivantes :

- Silver (100€)
- Gold (200€)
- Platinium (300€)

Le supporter peut acheter plusieurs tickets, de différentes catégories si il le souhaite.

Une fois ses billets achetés, le supporter peut les retrouver dans son espace personnel. On regroupe les billets par match pour simplifier les choses au moment de les présenter au stade. On génère un QR Code par billet, celui ci contient l'id du billet (un UUID), ainsi quand il sera scanné par l'application scanner, on aura immédiatement la place et le nom du supporter. Un bouton permettant de télécharger le QR Code peut être un plus.

---

③ Scanneur de billets

En plus de la petite application mobile, une autre page web indépendante aura pour unique but de scanner le QR Code inscrit sur les billets de la compétition. Après avoir scanné le QR Code contenant l'identifiant du billet, il faudra appeler **l'URL API** permettant de savoir si un billet est valide ou non, et en afficher les informations afin de comparer avec ce qui est imprimé sur le billet.

Pour la détection d'un QR Code à l'aide de la caméra d'un téléphone, on utilisera la bibliothèque Javascript QR Scanner. Cette dernière nous renverra le contenu encodé dans le QR Code au moment où il réussira à être scanné. Pour simplifier les choses dans ce POC il est possible de simplement uploader une image du QR Code.

Appel d'API

Vous allez donc devoir appeler l'API que vous aurez mis en place dans le projet Django depuis les 2 applications mobile. Pour ce faire, vous devrez utiliser Javascript afin de faire une requête HTTP vers l'URL de votre endpoint. Il faudra pour cela probablement utiliser la méthode ***fetch()*** (Doc ici)

Cette méthode est **asynchrone**, cela signifie que l'on ne sait pas dans combien de temps elle va répondre (logique ça dépend du serveur, de la connexion, de la réponse...). Le traitement de ce genre de méthode est particulier, je vous conseille de vous intéresser au mot clé ***await*** et/ou à la méthode ***then()*** pour obtenir les résultats attendus.

C'est un des points de difficulté du projet, **prenez votre temps** pour comprendre ce que vous faite pour ne pas vous perdre.

Infos supplémentaires

Pour les petites applications front, vous êtes libres d'utiliser une bibliothèque front-end comme Vue.js, React, etc.

L'application contenant l'API et l'interface devra elle obligatoirement être écrite en Python avec le framework Django. On vous demande notament de bien utiliser **l'ORM de Django pour générer votre BDD**, et ne pas utiliser du SQL natif

Le CIO vous a fourni la police d'écriture `Paris2024.ttf` et vous demande de l'utiliser, mais pour le reste vous êtes totalement libre sur le style des applications. Il faut simplement que toutes les fonctionnalités soient là et toutes les informations lisible. Essayez d'être cohérent dans le parcours utilisateur, n'hésitez pas à proposer des choses (la fameuse *proactivité* si chère aux RH).

Pour l'interface d'administration pas besoin de beaucoup de style, il faut *juste* que ce soit utilisable

---

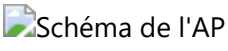
Schéma de l'AP

⚠ Point de difficulté

Gestion de CORS (Cross Origin Ressource Sharing)

Afin que vos URLs soient accessibles plus tard par l'application web mobile et le scanneur de billets, il vous faudra au préalable installer un middleware permettant de gérer les en-têtes CORS. Ils permettent d'autoriser les requêtes venant de votre page web, accédée depuis un autre appareil (téléphone...) jusqu'à votre projet Django qui est exécuté sur votre ordinateur.

1. Installer le middleware en exécutant `pip install django-cors-headers` (ou `pip3` selon votre système d'exploitation)
2. Ajouter la ligne suivante au sein de la variable `INSTALLED_APPS` du fichier de paramètres `base.py` en faisant en sorte que ce soit la **première ligne** du tableau

```
    INSTALLED_APPS = [
        'corsheaders',
```

```
        ...
    ]
```

3. Ajouter la ligne suivante au sein de la variable `MIDDLEWARE` du fichier de paramètres `base.py` en faisant en sorte que ce soit la **première ligne** du tableau

```
MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    ...,
]
```

4. Ajouter les variables suivantes dans le fichier de paramètres `base.py`

```
CORS_ALLOW_ALL_ORIGINS = True
CORS_ALLOW_CREDENTIALS = True
CSRF_TRUSTED_ORIGINS = [
    "http://127.0.0.1:5500", # Adresse URL local sur le port liveserver
]
ALLOWED_HOSTS = [
    '127.0.0.1',  # Pour les tests en local
]
```

Requête HTTP méthode POST vers l'API

Lors de l'achat d'un ticket par un supporter vous allez probablement devoir appeler votre API en envoyant des données. Pour cela vous devrez utiliser la fonction JS ***fetch()*** avec un ***body*** donc en `method: POST`. Problème le serveur ne vous laissera pas lui envoyer n'importe quoi sans montrer patte blanche.

Pour recevoir des données, le serveur vérifie l'*origin* de celles ci, par défaut les données arrivant d'une origin (URL) différente à celle du serveur sont rejettés. Dans le fichier de settings `base.py` nous avons également autorisé l'URL `127.0.0.1:5500` à envoyer des données. Cette URL correspond à un liveserver lancé en local. Pour faire simple je vous conseille donc de lancer vos fichier HTML via l'extention liveserver. Si vous utilisez autre chose (un framework) ou un autre port, il faudra modifier votre fichier `base.py`

Un collegue expérimenté vous fourni 2 morceaux de code qui devraient vous aider.

Le premier permet de récupérer le cookie laissé par le server interrogé, il contient le token CSRF permettant de lui envoyer des données

```
function getCookie(name) {
    let cookieValue = null;
    if (document.cookie && document.cookie !== "") {
        const cookies = document.cookie.split(";");
        for (let i = 0; i < cookies.length; i++) {
            const cookie = cookies[i].trim();
```

```
            if (cookie.startsWith(name + "=")) {
                cookieValue = decodeURIComponent(cookie.substring(name.length +
1));
                break;
            }
        }
    }
    return cookieValue;
}
```

Le second envoi une requete *POST* avec les informations nécéssaire à son bon fonctionnement dans l'en-tête

```
fetch(apiPath, {
    method: "POST",
    credentials: "include",
    headers: {
        "Content-type": "application/json",
        "X-CSRFToken": getCookie("csrftoken"),
    },
    body: JSON.stringify(body),
});
```

Débogage caméra téléphone

Cet exercice utilisant l'accès à la caméra depuis Javascript, nous allons nous heurter à quelques problèmes suite à la sécurité imposée par les navigateurs web.

Lorsque vous voudriez tester cette page web sur votre ordinateur, vous pourrez ouvrir directement le fichier dans le navigateur web ou encore créer à la volée un petit serveur web qui servira la page sur l'adresse `localhost` (par exemple à l'aide de Python ou de l'extension Live Server pour VS Code).

Pour tester votre page sur votre téléphone, il faudra soit la télécharger et l'ouvrir à la main, soit l'ouvrir depuis un site web servi strictement en **HTTPS** afin d'avoir l'accès à la caméra de fonctionnel. Aussi, vous ne pourrez pas accéder au serveur web de test qui tourne sur votre ordinateur depuis votre téléphone (en tapant l'IP locale de l'ordinateur) car dans ce cas le navigateur refusera d'afficher la caméra sur un serveur web n'utilisant pas **HTTPS**.

Lancer l'application

1. Installez django-cors-headers via la commande `pip install django-cors-headers`
2. Créez une base de données et modifiez le fichier `base.py` si besoin pour la connecter
3. Importer les tables et les données fourni par le CIO via le fichier data_jo.sql
4. Lancez l'application avec `py manage.py runserver`