TIC-API3 / myAPI

Modalités

Type	Description
Dépôt	https://rendu-git.etna-alternance.net/module-9220/activity-50271/group-994467
Correction	Soutenance
Durée	1 Run
Effectif	Groupe de 3
Langages autorisés	Tous sauf Node.js

Objectifs

- Comprendre et restituer les 6 contraintes définies par une architecture RESTful.
- Développer une API RESTful respectant une architecture logiciel défini par un ensemble de contraintes utilisées pour la création d'un service Web.

Consignes

Dans le cadre d'une nouvelle plateforme de diffusion vidéo, vous devez mettre en place une application permettant la gestion des informations et la gestion des vidéos d'un utilisateur.

Pour commencer cette nouvelle plateforme, vous devez mettre en place une API REST (cf. Wikipédia).
Dans notre cas, l'API délivre ses réponses au format JSON.

Une base de données est fournie en ressource de ce projet.
Cette dernière suffit pour réaliser le projet, mais vous n'êtes pas obligé de l'utiliser. Ne perdez pas de temps à faire de la modélisation de bases de données.

Cahier des charges

Votre API doit comporter 14 endpoints. Pour définir un endpoint, nous utiliserons une syntaxe "classique".

Chaque endpoint sera défini par :

- une méthode HTTP
- un URI
- le type d'authentification
  - `false` => facultatif si l'utilisateur est connecté cela ne change rien
  - `__type__` => facultatif, mais si l'utilisateur est identifié le retour ne sera pas le même
  - `__type__*` => obligatoire
- des paramètres obligatoires ou non identifiés par un `*`, dans le body ou dans la *query string* (à vous d'utiliser le bon système)
- un code de retour
- un format de retour

Vous devez utiliser la méthode d'authentification JWT (cf Wikipédia).

Certaines ressources sont dynamiques.
Elles doivent être identifiées par `:` dans l'URL. Les ":" ne doivent pas être présent dans l'URL finale.

Si un endpoint nécessite des paramètres obligatoires ou typés, vous devez les vérifier et renvoyer une erreur dans le cas contraire.

Le format de retour est un format spécifique, le format `[]` n'est pas le même que `{}` et `[{}]` n'est pas le même format que `{}`.
Si le retour comporte "...", cela signifie que c'est une liste. Par conséquent, il ne s'agit pas d'un seul élément.

JSON (application/json | form/data | x-www-form-urlencode)

Dans certaines requêtes, vous devez inclure des paramètres.

Dans les exemples donnés au travers des sujets, nous préférons utiliser le format JSON pour ces paramètres.
Cependant, il reste libre de choix. Vous pouvez choisir d'utiliser le format `form/data` ou `x-www-form-urlencode`.

Ces trois formats sont gérés par défaut par la grande majorité des frameworks.

Attention

Le format de données `file` devra être `form/data` dans le but d'envoyer un binaire au lieu d'un fichier au format `base64` dans le JSON.

Ressources

Pour simplifier les endpoints, vous trouverez ci-après les différentes ressources que vous devez renvoyer.

Pour chaque champ, vous disposez du type (`int` / `string` / `float` / etc.).

Le type *datetime* est un string formaté au format *ISO-8601*.

Un `*` dans le champ de la ressource signifie qu'elle est visible que si l'utilisateur est le propriétaire de la ressource.

Retour des endpoints *User*

```
{
    "id": int,
    "username": string,
    "pseudo": string,
    "created_at": datetime,
    "email*": string
}
```

Retour des endpoints *Video*

```
{
    "id": int,
    "source": string,
    "created_at": datetime,
    "views": int,
    "enabled": boolean,
    "user": user,
    "format": {
        "1080": string,
        "720": string,
        "480": string,
        "360": string,
        "240": string,
        "144": string
    }
}
```

Retour du endpoint *Token*

```
{
    "token": string,
    "user": user
}
```

Retour des endpoints *Comments*

```
{
    "id": int,
    "body": string,
```

```
    "user": user
}
```

Pagination

La pagination fonctionne toujours de la même façon lorsque vous avez des listes.

Par défaut, elle renverra la première page avec 5 enregistrements toujours par ordre d'`id` décroissant.
L'ordre ne change jamais.

Cependant, le nombre et la page peuvent changer en fonction des paramètres de requête suivants :

- `page:int`
- `per_page:int`

Attention

Si le paramètre `page` est égale à `0` ou qu'il est supérieur au nombre de page existantes, vous devez retourner un code HTTP `400`.

Si il n'y a pas de données à afficher en `page 1` (exemple : lors d'une recherche), vous devez retourner un tableau vide.

Gestion d'erreurs

Une API bien faite se doit d'avoir une bonne gestion d'erreurs.
Pour cela, tous vos endpoints devront gérer les différentes erreurs possibles.

Attention, la gestion des erreurs fait partie intégrante de la note.

Erreur de ressource

Vous devez renvoyer cette erreur si l'endpoint n'existe pas ou que l'identifiant d'une ressource n'existe pas.

| Code HTTP | 404 |
| JSON |

```
{
    "message": "Not found"
}
```

|

Erreur d'authentification

Vous devez renvoyer cette erreur si un endpoint requiert un jeton d'authentification, mais que celui-ci n'est pas présent.

| Code HTTP | 401 |
| JSON |

```
{
    "message": "Unauthorized"
}
```

|

Erreur d'invalidité

Vous devez renvoyer cette erreur si le traitement de votre requête ne fonctionne pas ou que l'utilisateur rentre des informations invalides (dans le formulaire).

| Code HTTP | 400 |
| JSON |

```
{
    "message": "Bad Request",
    "code": xxx,
    "data": []
}
```

|

Le code permet d'identifier le type d'erreur (ex: code 10001 => formulaire invalide). Le champ "data" est disponible pour avoir la pile d'erreur.