

RTP-API3 / Rattrapage YouTube

Modalités

Fichiers requis	Tous les fichiers nécessaires au lancement et au déploiement de l'API
Correction	À distance
Durée	1 RUN
Taille de groupe	Seul

Objectifs










Notion	Description
Architecture	Création d'une architecture orientée micro service et la comparer avec des architectures déjà vues type MVC / Monolithique
Outils et langages	Analyse des besoins et choix des technologies en fonction des contraintes de fonctionnalités
Interconnexion	Faire communiquer un ou plusieurs services via des APIs
Déploiement / Configuration (Facultatif)	Mise en place des services via des outils de déploiement (Docker / Vagrant / Ansible / ...)

Consignes

Le projet comporte 8 endpoints.

Pour définir un endpoint, nous utiliserons une syntaxe "classique".




Chaque endpoint sera défini par :

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

Pour récupérer un token, l'utilisateur devra envoyer son login / password via le endpoint "Authentification" qui enregistrera dans la base un token unique pour l'utilisateur. (Attention vous n'avez pas à utiliser de JWT pour cette API)

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

Informations

-  Certaines ressources sont dynamiques, elles seront identifiées par ":" dans l'URL (les ":" ne sont bien sûr pas dans l'URL finale)
-  Le format de retour est un format spécifique, donc attention [] n'est pas {} et [{}] n'est pas la même chose que {}
-  Si dans le retour on met "...", cela signifie que c'est une liste et que, par conséquent, il ne s'agit pas d'un seul élément.

Ressources

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

Pour chaque champ, nous vous donnerons le type (int / string / float / ...).

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

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

User

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

File

```
{
  "id": int,
  "name": string,
  "created_at": datetime,
  "path": string
}
```

Token

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



Gestion des erreurs

Une API bien faite se doit d'avoir une gestion d'erreurs la plus propre possible.

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

Retour 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

Retour JSON :

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

Erreur d'autorisation

Vous devez renvoyer cette erreur si vous avez un jeton d'authentification mais que l'utilisateur authentifié n'a pas le droit d'interagir avec la ressource :

Code HTTP => 403

Retour JSON :

```
{  
  "message": "Forbidden"  
}
```

Erreur d'autorisation

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).

Le code permet d'identifier le type d'erreur (ex: code 10001 => formulaire invalide).

Le champ "data" est disponible pour avoir la stack d'erreurs :

Code HTTP => 400

Retour JSON :

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

Endpoint : création d'utilisateurs

Méthode	POST
---------	------

URI	/user
-----	-------

Paramètres	<pre>{ "username*": string([a-zA-Z0-9_-]), "pseudo": string, "email*": string(email), "password*": string }</pre>
------------	---

Authentication false

JSON

```
{  
  "message": "OK",  
  "data": User  
}
```

Code HTTP retour 201

Endpoint : authentication

Méthode POST

URI /auth

Paramètres

```
{  
  "login*": string,  
  "password*": string  
}
```

Authentication false

JSON

```
{  
  "message": "OK",  
  "data": Token  
}
```

Code HTTP retour 201

Endpoint : suppression d'utilisateurs

Méthode	DELETE
URI	/user/:id
Paramètres	-
Authentification	Token*
JSON	-
Code HTTP retour	204

Endpoint : modification d'utilisateurs

Méthode	PUT
URI	/user/:id
Paramètres	<pre>{ "username": string([a-zA-Z0-9_-]), "pseudo": string, "email": string(email), "password": string }</pre>
Authentification	Token*
JSON	<pre>{ "message": "OK", "data": User }</pre>
Code HTTP retour	200

Endpoint : création de fichiers

Méthode	POST
URI	/user/:id/file

Paramètres

```
{  
  "name": string,  
  "source": file  
}
```

Authentification

Token*

JSON

```
{  
  "message": "OK",  
  "data": File  
}
```

Code HTTP retour

201

Endpoint : liste de fichiers utilisateurs

Méthode

GET

URI

/user/:id/file

Authentification

false

JSON

```
{  
  "message": "OK",  
  "data": [  
    File,  
    ...  
  ]  
}
```

Code HTTP retour 200

Endpoint : mise à jour de fichiers

Méthode PUT

URI /file/:id

Paramètres

```
{  
  "name": string,  
  "source": file  
}
```

Authentication Token*

JSON

```
{  
  "message": "OK",  
  "data": File  
}
```

Code HTTP retour 200

Endpoint : suppression de fichiers

Méthode DELETE

URI /file/:id

Paramètres -

Authentification Token*

JSON -

Code HTTP retour 204