

Louis MASSICARD

Projet M2 SID – API

Table des matières

Analyse.....	3
Introduction.....	3
Fonctionnalités utilisateur.....	3
Fonctionnalités administrateur.....	3
Données nécessaires.....	4
Conception.....	5
Architecture micro-services.....	5
Développement.....	6
Configuration et mise de place de la solution.....	6
Ports utilisés.....	7
Interaction de l'admin avec backoffice-gateway.....	8
Interaction de l'utilisateur avec frontoffice-gateway.....	9
Exemples d'opérations de l'API.....	10
Problèmes rencontrés.....	19

Analyse

Introduction

On veut réaliser une API permettant à des utilisateurs inscrits d'acheter des cours vidéo, comme peuvent le faire des sites tels que Udacity ou Coursera.

L'utilisateur doit donc être inscrit pour profiter des cours payant (via CB : numéro, code et code de validation) ou non. Les cours sont associés à un thème et possède une description. Chaque cours est composé d'épisode(s) (vidéo).

Une fois acheter le cours et définitivement accessible.

L'API doit fournir certaines métriques : cours visionnés par un utilisateur, etc.

Il y a également une partie administrateur qui permet de gérer les différentes entités utilisateurs, cours et épisodes à partir de fonctionnalités CRUD. L'administrateur a également accès à différentes métriques : nombre d'utilisateurs inscrits, nombre d'utilisateurs supprimés, etc.

Fonctionnalités utilisateur

Liste des fonctionnalités accessible aux utilisateurs :

- s'inscrire
- lister les cours
- acheter un cours
- visionner un cours payant (s'il a été acheté) ou gratuit
- visionner les épisodes associés

Fonctionnalités administrateur

Liste des fonctionnalités accessible aux administrateur :

- connaître les users inscrits (select all)
- modifier les users
- supprimer des users

- créer des cours
- modifier les cours
- supprimer des cours

- créer des épisodes
- modifier les épisodes
- supprimer des épisodes

Données nécessaires

Utilisateur :

- id_user
- nom
- prénom
- adresse mail => identification
- supprime => true || false

Cours :

- id_cours
- nom => theme
- description => description_theme
- statut => gratuit || payant
- prix
- []id_episode => ensemble/liste d'épisodes vidéo

Épisode/Vidéo :

- id_episode
- concept => concept (du cours) qui est présenté
- href

Payement (on parlera par la suite plutôt d'abonnement) :

- id_user
- id_cours
- numero_CB
- code
- code_de_validation
- consulte => visonné || non_visonné

Ex : Un cours sur Kubernetes, on aura les épisodes Introduction, Concepts de base, Concepts avancés, Création d'un cluster, Bonnes pratiques.

Conception

Architecture micro-services

J'ai de créer des services en fonction des problématiques métiers.

Pour cela, j'ai choisi de réaliser un service pour gérer les cours et les épisodes associés aux cours : **cours-service**. J'ai également un service pour gérer les utilisateurs et ce qui se rattache aux utilisateurs comme leurs abonnements à des cours par exemple : c'est le service **utilisateurs-service**.

Ces deux services permettent de manipuler les objets métiers via une API REST. On y trouve donc les routes traditionnelles « /cours », « /épisodes », « /utilisateurs » et « /abonnements ». Pour chacune on va avoir les actions de bases GET, POST, PUT, PATCH et DELETE. En fonction des besoins, j'ai ajouté des routes plus spécifiques comme GET « /cours/{coursID} » pour récupérer par exemple le contenu d'un seul cours.

Nous avons deux types d'utilisateurs à gérer : des utilisateurs « normaux » de l'application et des administrateurs. Pour moi, cela ressemble à une solution applicative avec un Front-office pour les utilisateurs et un Back-office pour les administrateurs. J'ai donc décidé de mettre en place deux Gateway, permettant respectivement aux utilisateurs et aux administrateurs de réaliser les fonctionnalités qui leur sont propres. C'est pourquoi j'ai les Gateway suivante : **backoffice-gateway** et **frontoffice-gateway**.

Pour accompagner c'est quatre repos, j'ai choisi d'utiliser **consul** comme Service Discovery. Mes deux services et mes deux Gateways s'enregistrent donc auprès de consul afin de mieux gérer les interactions entre services et Gateways.

Les Gateways communiquent avec les deux services via une **queue de messages** grâce à **RabbitMQ**.

Chaque service utilise sa base de données qui lui est propre. Le service **utilisateurs-service** utilise une base de données **Postgres**. Le service **cours-service** utilise une base de données **H2**. Il n'y a pas eu de raison particulière pour le choix de ces deux bases de données, je me suis seulement inspiré du « TP Full ».

De même, basé sur le « TP Full », il y a un **config-service** mais qui n'est pas réellement utilisé mais qui pourrait l'être par la suite, si besoin est. Ce service est associé au **repo config** qui stocke les différentes configurations.

J'utilise également **Hystrix** comme **Circuit Breaker** au cas où un service ne répond pas.

Pour montrer cette architecture micro-services, plusieurs outils sont à disposition :

- ◆ **consul** : permet de voir tous les services enregistrés et le nombre d'instances pour chaque,
- ◆ le service **spring-boot-admin** : utilise les informations émises par **Actuator** depuis les autres services,
- ◆ le service **monitor** : utilise les informations émises par **Hystrix** depuis les autres services, pour voir les requêtes en attentes et leur charge,
- ◆ **zipkin** : permet de voir .

Développement

Configuration et mise de place de la solution

Un README récapitule la mise en place de tous les services. De manière générale il faut réaliser dans l'ordre suivant :

(pour les services de monitoring, ils sont tous expliqués dans le README.md)

```
docker run --name consul -p 8500:8500 consul
```

```
sudo pkill -u postgres
```

```
docker run --name postgres -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=riovas -p 5432:5432 -d postgres
```

```
docker run -it --rm --name rabbitmq -p 5672:5672 -e RABBITMQ_DEFAULT_USER=user -e RABBITMQ_DEFAULT_PASS=password -p 15672:15672 rabbitmq:3-management
```

```
cd cours-service  
mvn clean package -DskipTest  
java -jar target/cours-service-1.0.jar
```

```
cd utilisateurs-service  
mvn clean package -DskipTest  
java -jar target/utilisateurs-service-1.0.jar
```

```
cd frontoffice-gateway  
mvn clean package -DskipTest  
java -jar target/frontoffice-gateway-1.0.jar
```

```
cd backoffice-gateway  
mvn clean package -DskipTest  
java -jar target/backoffice-gateway-1.0.jar
```

Ports utilisés

— Nécessaire :

config-service : 8888

cours-service : 8080

utilisateurs-service : 8082

backoffice-gateway : 9000

frontoffice-gateway : 9001

— Nécessaire :

consul : 8500

postgres : 5432

rabbitMQ : 5672

rabbitMQ :3-management : 14 672

— Facultatif :

spring-bout-admin : 8762

monitor : 9903

zipkin : 9411

Interaction de l'admin avec backoffice-gateway

Je n'ai pas eu le temps de rediriger les requêtes du backoffice-gateway vers cours-service et utilisateurs-services. Cependant, dans un premier, l'administrateur peut directement utiliser cours-service et utilisateurs-services pour gérer la solution applicative. Il peut réaliser les actions suivantes :

Cours API		Utilisateur API	
12 requests	ooo	12 requests	ooo
└ cours	ooo	└ abonnements	ooo
└ {cours ID}	ooo	└ {abonnement ID}	ooo
GET get Cours		GET get Abonnement	
PUT update Cours		PUT update Abonnement	
DEL delete Cours		DEL delete Abonnement	
PATCH update Cours Partiel		PATCH update Abonnement Partiel	
GET get All Cours		GET get All Abonnements	
POST save Cours		POST save	
└ episodes	ooo	└ utilisateurs	ooo
└ {episode ID}	ooo	└ {utilisateur ID}	ooo
GET get Episode		GET get Utilisateur	
PUT update Episode		PUT update Utilisateur	
DEL delete Episode		DEL delete Utilisateur	
PATCH update Episode Partiel		PATCH update Utilisateur Partiel	
GET get All Episodes		GET get All Utilisateurs	
POST save Episode		POST save 1	

Interaction de l'utilisateur avec frontoffice-gateway

Inscription

► inscription

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** {{baseUrl}}/users
- Body (JSON):**

```
1 {
2   "nom": "Louis",
3   "prenom": "Massicard",
4   "mail": "louis.massicarde@orange.fr"
5 }
```
- Headers (5):**

KEY	VALUE
connection	keep-alive, keep-alive
date	Fri, 05 Feb 2021 21:04:21 GMT
keep-alive	timeout=60
location	http://localhost:8082/utilisateurs/aac1e613-9ff2-43bc-bd44-e35e3e1de74b
Content-Length	0
- Status:** 201 Created

Exemples d'opérations de l'API

Parmi les opérations possibles de l'API (attention, liste non exhaustive), on trouve :

GET /utilisateurs
GET /utilisateurs/?statut=...
GET /utilisateurs/{id}
POST /utilisateurs
PUT /utilisateurs/{id}

GET /cours
GET /cours/{id}
GET /cours/{id}/episodes
GET /utilisateurs/{id}/cours

...

Routes disponibles :

utilisateurs-service :

GET /utilisateurs => toutes les informations pour tous les utilisateurs, retour :

```
{  
    "_embedded": {  
        "utilisateurs": [  
            {  
                "id": "de7d9052-4961-4b4f-938a-3cd12cbe1f82",  
                "nom": "MASSICARD",  
                "prenom": "Louis",  
                "mail": "louis@netlor.fr",  
                "statut": "actif",  
                "_links": {  
                    "self": {  
                        "href": "http://localhost:8082/utilisateurs/de7d9052-4961-4b4f-  
938a-3cd12cbe1f82"  
                    }  
                }  
            },  
            ...  
        ]  
    },  
    "_links": {  
        "self": {  
            "href": "http://localhost:8082/utilisateurs"  
        }  
    }  
}
```

GET *utilisateurs/{id}* => toutes les informations pour l'utilisateur dont on passe l'ID, retour :

```
{  
    "id": "de7d9052-4961-4b4f-938a-3cd12cbe1f82",  
    "nom": "MASSICARD",  
    "prenom": "Louis",  
    "mail": "louis@netlor.fr",  
    "statut": "actif",  
    "_links": {  
        "self": {  
            "href": "http://localhost:8082/utilisateurs/de7d9052-4961-4b4f-938a-3cd12cbe1f82"  
        },  
        "collection": {  
            "href": "http://localhost:8082/utilisateurs"  
        }  
    }  
}
```

POST /utilisateurs => crée un utilisateur dont les informations sont passées dans le body :

GET /cours & GET /cours?statut=&?acces=

▶ get All Cours

GET [{{baseUrl}}/cours?statut=&?acces=](#)

Params Authorization Headers (7) Body Pre-request Script Tests Settings Examples 1 ▾ BUILD

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> statut				
<input checked="" type="checkbox"/> acces				
Key	Value	Description		

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "_embedded": {  
3     "cours": [  
4       {  
5         "id": "372049f0-647c-11eb-ae93-0242ac130002",  
6         "nom": "Docker",  
7         "description": "Docker... Une alternative aux VM !",  
8         "statut": "ACTIF",  
9         "acces": "GRATUIT",  
10        "prix": 0,  
11        "episodes-id": [  
12          "181003d4-648b-11eb-ae93-0242ac130002",  
13          "c758595a-648a-11eb-ae93-0242ac130002",  
14          "ddc2ced8-6489-11eb-ae93-0242ac130002"  
15        ],  
16        "_links": {  
17          "self": {  
18            "href": "http://localhost:8080/cours/372049f0-647c-11eb-ae93-0242ac130002"  
19          }  
20        },  
21      },  
22      {  
23        "id": "3d96ela4-647c-11eb-ae93-0242ac130002",  
24        "nom": "NoSQL",  
25        "description": "Les NoSQL, une réponse au Big Data.",  
26      }  
27    },  
28  }  
29
```

Status: 200 OK Time: 1045 ms Size: 2.46 KB [Save Response](#) ▾

▶ get All Cours

GET [{{baseUrl}}/cours](#)

GET /cours?statut=ACTIF & GET /cours?statut=SUPPRIME

get All Cours

GET [\({{baseUrl}}/cours?statut=ACTIF\)]({{baseUrl}}/cours?statut=ACTIF)

Params	Authorization	Headers (7)	Body	Pre-request Script	Tests	Settings
Query Params						
KEY	VALUE					
<input checked="" type="checkbox"/> statut	ACTIF					
Key	Value					

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1
2   "embedded": {
3     "cours": [
4       {
5         "id": "372049f0-647c-11eb-ae93-0242ac130002",
6         "nom": "Docker",
7         "description": "Docker... Une alternative aux VM !",
8         "statut": "ACTIF",
9         "acces": "GRATUIT",
10        "prix": 0,
11        "episodes-id": [
12          "181003d4-648b-11eb-ae93-0242ac130002",
13          "c758595a-648a-11eb-ae93-0242ac130002",
14          "ddc2ced8-6489-11eb-ae93-0242ac130002"
15        ],
16        "_links": {
17          "self": {
18            "href": "http://localhost:8080/cours/372049f0-647c-11eb-ae93-0242ac130002"
19          }
20        }
21      },
22    ],
23    "_links": {
24      "self": {
25        "href": "http://localhost:8080/cours"
26      }
27    }
28  },
29  "cours": [
30    {
31      "id": "4346f850-647c-11eb-ae93-0242ac130002",
32        "nom": "UML",
33        "description": "Un super cours sur la modélisation !",
34        "statut": "SUPPRIME",
35        "acces": "GRATUIT",
36        "prix": 0,
37        "episodes-id": [
38          "dfa00858-648c-11eb-ae93-0242ac130002",
39          "24b04650-648e-11eb-ae93-0242ac130002",
40          "171la8894-648d-11eb-ae93-0242ac130002",
41          "2d6a5326-648e-11eb-ae93-0242ac130002",
42          "d07eb0cc-648c-11eb-ae93-0242ac130002",
43          "ac6226f0-648d-11eb-ae93-0242ac130002",
44          "d8833806-648c-11eb-ae93-0242ac130002",
45          "19f832b8-648e-11eb-ae93-0242ac130002"
46        ],
47        "_links": {
48          "self": {
49            "href": "http://localhost:8080/cours/4346f850-647c-11eb-ae93-0242ac130002"
50          }
51        }
52      },
53    ],
54    "_links": {
55      "self": {
56        "href": "http://localhost:8080/cours"
57      }
58    }
59  }
60 ]
61 
```

GET [\({{baseUrl}}/cours?statut=SUPPRIME\)]({{baseUrl}}/cours?statut=SUPPRIME)

Params	Authorization	Headers (7)	Body	Pre-request Script	Tests	Settings
Query Params						
KEY	VALUE					
<input checked="" type="checkbox"/> statut	SUPPRIME					
Key	Value					

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1
2   "embedded": {
3     "cours": [
4       {
5         "id": "4346f850-647c-11eb-ae93-0242ac130002",
6           "nom": "UML",
7           "description": "Un super cours sur la modélisation !",
8           "statut": "SUPPRIME",
9           "acces": "GRATUIT",
10          "prix": 0,
11          "episodes-id": [
12            "dfa00858-648c-11eb-ae93-0242ac130002",
13            "24b04650-648e-11eb-ae93-0242ac130002",
14            "171la8894-648d-11eb-ae93-0242ac130002",
15            "2d6a5326-648e-11eb-ae93-0242ac130002",
16            "d07eb0cc-648c-11eb-ae93-0242ac130002",
17            "ac6226f0-648d-11eb-ae93-0242ac130002",
18            "d8833806-648c-11eb-ae93-0242ac130002",
19            "19f832b8-648e-11eb-ae93-0242ac130002"
20          ],
21          "_links": {
22            "self": {
23              "href": "http://localhost:8080/cours/4346f850-647c-11eb-ae93-0242ac130002"
24            }
25          }
26        },
27      ],
28      "_links": {
29        "self": {
30          "href": "http://localhost:8080/cours"
31        }
32      }
33    }
34 ]
35 
```

GET /cours?acces=GRATUIT

get All Cours

GET [\({{baseUrl}}/cours?acces=GRATUIT\)]({{baseUrl}}/cours?acces=GRATUIT)

Params	Authorization	Headers (7)	Body	Pre-request Script	Tests	Settings
Query Params						
KEY	VALUE	DESCRIPTION				
<input type="checkbox"/> statut	SUPPRIME					
<input checked="" type="checkbox"/> acces	GRATUIT					
Key	Value	Description				

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

21  },
22  ],
23  "_links": {
24    "self": {
25      "href": "http://localhost:8080/cours"
26    }
27  }
28 }
29 
```

Examples 1 BUILD

Send Save

Status: 200 OK Time: 1683 ms Size: 1.55 KB Save Response

GET /cours?statut=ACTIF&acces=PAYANT

▶ get All Cours

GET {{baseUrl}}/cours?statut=ACTIF&acces=PAYANT

Params (1) Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> statut	ACTIF
<input checked="" type="checkbox"/> acces	PAYANT
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▾

```
1 [ {  
2   "_embedded": {  
3     "cours": [  
4       {  
5         "id": "4fc174ca-647c-11eb-ae93-0242ac130002",  
6         "nom": "Java-Spring",  
7         "description": "JEE, Spring et Maven un aperçue des dépendances de base.",  
8         "statut": "ACTIF",  
9         "acces": "PAYANT",  
10        "prix": 100,  
11        "episodes-id": [  
12          "5acbf4be-64ac-11eb-ae93-0242ac130002",  
13          "24d20960-64b7-11eb-ae93-0242ac130002"  
14        ],  
15        "_links": {  
16          "self": {  
17            "href": "http://localhost:8080/cours/4fc174ca-647c-11eb-ae93-0242ac130002"  
18          }  
19        }  
20      },  
21      {  
22        "id": "2cc2ba57-5ela-4ad0-8c3e-01702a46b102",  
23        "nom": "<string>",  
24        "description": "<string>",  
25      }  
26    ]  
27  }  
28}
```

GET /cours?statut=SUPPRIME&acces=PAYANT

▶ get All Cours

GET [\({{baseUrl}}/cours?statut=SUPPRIME&acces=PAYANT\)]({{baseUrl}}/cours?statut=SUPPRIME&acces=PAYANT)

Params (1) Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> statut	SUPPRIME
<input checked="" type="checkbox"/> acces	PAYANT
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "_embedded": {
3     "cours": [
4       {
5         "id": "4a9a0cfa-647c-11eb-ae93-0242ac130002",
6         "nom": "Java-JEE",
7         "description": "Java JEE n'aura plus de secret pour vous.",
8         "statut": "SUPPRIME",
9         "acces": "PAYANT",
10        "prix": 50,
11        "episodes-id": [
12          "90c72502-648e-11eb-ae93-0242ac130002"
13        ],
14        "_links": {
15          "self": {
16            "href": "http://localhost:8080/cours/4a9a0cfa-647c-11eb-ae93-0242ac130002"
17          }
18        }
19      }
20    ]
21  }
22}
```

POST /cours

▶ save Cours

POST {{baseUrl}}/cours

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {  
2   "acces": "GRATUIT",  
3   "description": "<string>",  
4   "nom": "<string>",  
5   "prix": 0,  
6   "statut": "ACTIF",  
7   "episodesID": []  
8 }  
9  
10 }
```

Body Cookies Headers (5) Test Results

KEY VALUE Status: 201 Created

KEY	VALUE	Status
Location	http://localhost:8080/cours/91cffbd4-956b-4f5f-9a82-31211582045b	201 Created
Content-Length	0	
Date	Fri, 05 Feb 2021 15:33:52 GMT	
Keep-Alive	timeout=60	
Connection	keep-alive	

GET /cours/:coursID

▶ get Cours

GET {{baseUrl}}/cours/91cffbd4-956b-4f5f-9a82-31211582045b

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": "91cffbd4-956b-4f5f-9a82-31211582045b",  
3   "nom": "<string>",  
4   "description": "<string>",  
5   "statut": "ACTIF",  
6   "acces": "GRATUIT",  
7   "prix": 0,  
8   "episodes-id": [],  
9   "_links": {  
10     "self": {  
11       "href": "http://localhost:8080/cours/91cffbd4-956b-4f5f-9a82-31211582045b"  
12     },  
13     "collection": {  
14       "href": "http://localhost:8080/cours?statut,acces",  
15       "templated": true  
16     }  
17   }  
18 }
```

JSON

Status: 200 OK

PUT /cours/:coursID

▶ update Cours

The screenshot shows a POST request in Postman. The URL is `{baseUrl}/cours/91cffbd4-956b-4f5f-9a82-31211582045b`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2     "nom": "<string>",
3     "description": "<TEST>",
4     "statut": "ACTIF",
5     "acces": "GRATUIT",
6     "prix": 0,
7     "episodes-id": []
8 }
```

The response status is 200 OK. The response body is a JSON object with fields like id, nom, description, statut, acces, prix, episodes-id, and links.

```
1 {
2     "id": "91cffbd4-956b-4f5f-9a82-31211582045b",
3     "nom": "<string>",
4     "description": "<TEST>",
5     "statut": "ACTIF",
6     "acces": "GRATUIT",
7     "prix": 0,
8     "episodes-id": [],
9     "_links": {
10         "self": {
11             "href": "http://localhost:8080/cours/91cffbd4-956b-4f5f-9a82-31211582045b"
12         },
13         "collection": {
14             "href": "http://localhost:8080/cours{?statut,acces}",
15             "templated": true
16         }
17     }
18 }
```

DELETE /cours/:coursID

▶ delete Cours

The screenshot shows a DELETE request in Postman. The URL is `{baseUrl}/cours/91cffbd4-956b-4f5f-9a82-31211582045b`. The 'Body' tab is selected, showing a JSON payload:

```
1
```

The response status is 204 No Content.

PATCH /cours/:coursID

▶ update Cours Partiel

PATCH [\({{baseUrl}}/cours/91cffbd4-956b-4f5f-9a82-31211582045b\)]({{baseUrl}}/cours/91cffbd4-956b-4f5f-9a82-31211582045b)

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {  
2   "statut": "ACTIF"  
3 }
```

Body Cookies Headers (5) Test Results Status: 200 OK Ti

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": "91cffbd4-956b-4f5f-9a82-31211582045b",  
3   "nom": "<strings>",  
4   "description": "<TEST>",  
5   "statut": "ACTIF",  
6   "acces": "GRATUIT",  
7   "prix": 0,  
8   "episodes-id": [],  
9   "_links": {  
10     "self": {  
11       "href": "http://localhost:8080/cours/91cffbd4-956b-4f5f-9a82-31211582045b"  
12     },  
13     "collection": {  
14       "href": "http://localhost:8080/cours{?statut,acces}",  
15       "templated": true  
16     }  
17   }  
18 }
```

Problèmes rencontrés

Manque de temps surtout... Je n'ai pas eu le temps d'implanter la totalité des fonctionnalités car j'ai perdu beaucoup de temps. En effet, j'ai préféré bien comprendre toutes les technos que l'on a vu en cours et réussir à les manipuler dans ce projet : Discovery Service, architecture micro-services avec Gatway, queue de messages, Laodbalancing, etc.

Je pense que j'ai réussi à comprendre et à implémenter chacune des technos présentées durant le « TP Full » et elles sont normalement fonctionnelles dans ce projet. J'ai seulement eu des difficultés avec les différents canaux de rabbitMQ, j'ai laissé de côté et je suis donc directement passé par @FeignClient. J'ai compris comment implémenter une API REST avec HATEOAS dans ce contexte du projet que j'ai mis en place : j'ai fait quelques exemples mais je n'ai pas eu le temps de faire toutes les fonctionnalités.

Comme nous l'avons vu l'année dernière avec le TP de la machine à café, j'ai essayé de rendre mon code le plus paramétrable possible. Par exemple, les statuts et les types de paiement sont gérés avec des Enum. Cela me permet de facilement ajouter un statut. Ces Enum sont directement utilisées par les validators des inputs.

Les fonctionnalités administrateurs sont disponibles même si elle ne passe pas par le backoffice-gateway. Ces fonctionnalités pourraient être enrichies à terme avec des statistiques.

Pour la partie front-office, l'utilisateur peut s'inscrire, se connecter, lister les cours. Il manque l'abonnement et le visionnement. D'un point de vue entités, il faudrait que je rajoute une entité « historique » dans le service utilisateurs-service pour savoir quels épisodes ont déjà été visionné par un utilisateur.