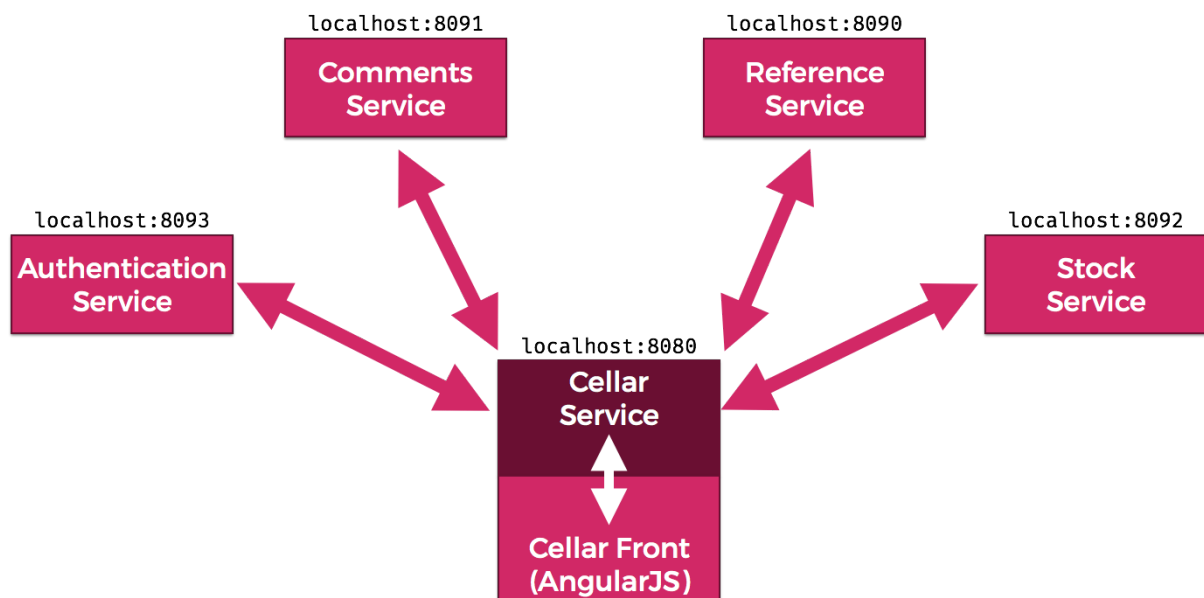


# Introduction

C'est votre premier jour votre client *BuzzwordInProdCorp*. Ci-dessous est la présentation que l'architecte vous fait de l'écosystème dans lequel vous aller devoir travailler.



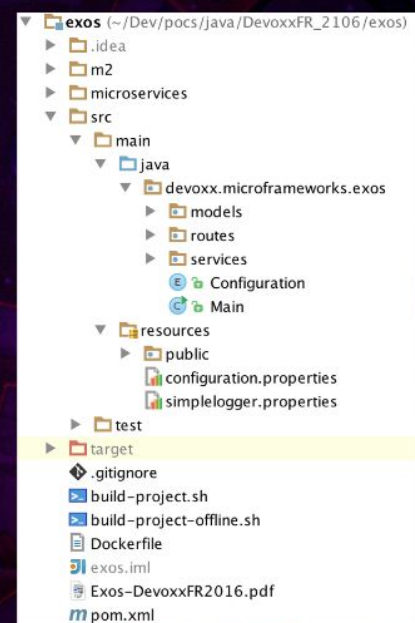
## Cellar Service

microservices: mockup services (authentication, reference, comment and stock)

src/main/java // devovx.microframeworks.exos

- **models**: Pojo used in the application
- **routes**: Cellar service REST routes implementation (**Spark Java**)
- **services**: REST clients services implementation (**Feign**)
- **Main**: Cellar service Main class (embed cellar front-end)

#DevoxxFR #feign #sparkjava



DEVVOX France

# Start others microservices

Open 4 terminals / consoles

```
./start-authentication-services.sh  
./start-reference-services.sh  
./start-comments-services.sh  
./start-stocks-services.sh
```

```
$ ./start-authentication-service.sh  
[Thread-0] [INFO] log - Logging initialized @363ms  
[Thread-0] [INFO] JettySparkServer - == Spark has ignited ...  
[Thread-0] [INFO] JettySparkServer - >> Listening on 0.0.0.0:8093  
[Thread-0] [INFO] Server - jetty-9.3.z-SNAPSHOT  
[Thread-0] [INFO] ServerConnector - Started ServerConnector@50f244a7{HTTP/1.1,  
[http/1.1]}{0.0.0.0:8093}  
[Thread-0] [INFO] Server - Started @450ms
```

#DevooxFR #feign #sparkjava

DEVVOX France

## Or if you <3 Docker

```
$ docker-compose up  
Creating sparkfeign-auth  
Creating sparkfeign-reference  
Creating sparkfeign-stocks  
Creating sparkfeign-comments  
Attaching to sparkfeign-auth, sparkfeign-reference, sparkfeign-stocks,  
sparkfeign-comments  
sparkfeign-auth | [Thread-0] [INFO] log - Logging initialized @578ms  
sparkfeign-reference | [main] [INFO] WineDao - Parse wines database :323 wines  
read.  
sparkfeign-auth | [Thread-0] [INFO] JettySparkServer - == Spark has ignited ...  
sparkfeign-reference | [Thread-0] [INFO] log - Logging initialized @565ms  
sparkfeign-auth | [Thread-0] [INFO] JettySparkServer - >> Listening on 0.0.0.0:  
8093  
...
```

#DevooxFR

DEVVOX France

On va commencer par travailler en **localhost** avec l'aide de scripts qui ont été préparés.  
On peut configurer les URL des micro services dans le fichier suivant:

```
/src/main/resources/configuration.properties.
```

On peut utiliser les URLs locale, sur heroku, ou sur ilaborie.org.

## Pré requis

- Avoir un JDK 8, on peut le [télécharger ici](#).
- Avoir un Maven 3+, voir le [site officiel](#).
- Avoir un IDE supportant Java 8 et Maven. Par exemple [Eclipse](#), [IntelliJ CE](#), [Netbeans](#)...
- Connaître les bases de [HTTP](#), et de Java bien sûr !

## Présentation de l'ensemble des services disponibles

Les services disponibles sont les suivants :

## Service d'authentification

Ce service est accessible aux URLs suivantes :

- `http://localhost:8093/`
- `http://dvxx-sparkfeign-auth.herokuapp.com/`
- `http://ilaborie.org:8093/`

Service	URL	Verbe HTTP	Paramètres du service
<b>Login</b>	http://localhost:8093/api/auth/login	POST	<p>Dans le Body de la requête HTTP</p> <pre>{"email": "toto@plop.fr", "password": "admin"}</pre>
	<p>Le service retourne un message JSON contenant un <a href="#">token JWT</a> de la forme suivante avec un code de retour HTTP 200 OK :</p> <pre>{   "token":   "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1Ym91IjoiZjoidG90b0BwbG9wLmZyIiwiaXNjaXNlbnQ9DU3ODY4MzA5LCJlbWFpbCI6InRvdG9AcGxvcC5mciJ9.x1U4oNH3BXgrzIWp4aWJec9K_llQi40p0eAP9svkx0w" }</pre>		

	Un code de retour HTTP 403 Forbidden et le message suivant si l'authentification n'a pas été réalisée avec succès : Invalid login/password		
<b>User</b>	http://localhost:8093/api/user/{token}	GET	Dans l'URL de la requête HTTP le token JWT obtenu auprès du service login
	<p>Le service retourne un message JSON de la forme suivante avec un code de retour HTTP 200 OK :</p> <pre>{   "name": "toto@plop.fr",   "email": "toto@plop.fr" }</pre> <p>Un code de retour HTTP 403 Forbidden et le message suivant si le token fourni n'est pas valide : Invalid JWT</p>		

## Service de référence

Ce service est accessible aux URLs suivantes :

- <http://localhost:8090/>
- <http://dvxx-sparkfeign-ref.herokuapp.com/>
- <http://ilaborie.org:8090/>

Service	URL	Verbe HTTP	Paramètres du service
<b>All Wines</b>	http://localhost:8090/api/wines	GET	Aucun paramètre
	<p>Le service retourne un message JSON contenant un tableau de vins de la forme suivante avec un code de retour HTTP 200 OK :</p> <pre>[   {     "id": "be147222-009d-446f-b4d2-ee0e80d3131c",     "produit": "CHATEAU HAUT MAZIERES 2014",     "millesime": 2014,     "appellation": "Bordeaux AOC",     "couleur": "Vin Rouge",     "region": "Bordeaux",     "pays": "France",     "cepageDominant": "64% Merlot, 22% Cabernet-sauvignon, 14% Cabernet-franc",     "alcool": "12% vol",     "contenance": "0,75 L",     "priceCategory": "MID",     "image": "chateau-haut-mazieres-2014.jpg",     "conservation": "A boire et à garder",     "aBoireAPartirDe": 2015,     "garderJusquA": 2019,   } ]</pre>		

	<pre> "auNez": "notes boisées", "enBouche": "Tanins équilibrés, arômes du nez", "aOeil": "Robe prune, reflets rubis", "service": "Ouvrir 30 min avant le service", "temperatureDeService": "16-18°C", "accordsMetsVin": "Charcuterie, Viande rouge, Viande blanche, Fromage" }, ... ] </pre>		
<b>Wine By Id</b>	http://localhost:8090/api/wines/{id}	GET	Dans l'URL de la requête HTTP l'identifiant du vin.
	<p>Le service retourne un message JSON contenant les caractéristiques du vins de la forme suivante si l'identifiant est valide avec un code de retour HTTP 200 OK :</p> <pre> {   "id": "be147222-009d-446f-b4d2-ee0e80d3131c",   "produit": "CHATEAU HAUT MAZIERES 2014",   "millesime": 2014,   "appellation": "Bordeaux AOC",   "couleur": "Vin Rouge",   "region": "Bordeaux",   "pays": "France",   ... } </pre> <p>Un code de retour HTTP 404 Not Found si l'identifiant n'a pas pu être trouvé et le message suivant : Wine not found: e2c3d7cb-a357-47f5-b100-501457e3d</p>		
<b>Search Wine</b>	http://localhost:8090/api/wines?q={query}	GET	Dans le paramètre q de la requête HTTP le terme à chercher.
	<p>Le service retourne un message JSON contenant un tableau de vins de la forme suivante correspondant au résultat de la recherche avec un code de retour HTTP 200 OK :</p> <pre> [   {     "id": "be147222-009d-446f-b4d2-ee0e80d3131c",     "produit": "CHATEAU HAUT MAZIERES 2014",     "millesime": 2014,     "appellation": "Bordeaux AOC",     "couleur": "Vin Rouge",     "region": "Bordeaux",     "pays": "France",     ...   },   ... ] </pre>		

## Service de commentaires

Ce service est accessible aux URLs suivantes :

- <http://localhost:8091/>
- <http://dvxx-sparkfeign-comment.herokuapp.com/>
- <http://ilaborie.org:8091/>

Service	URL	Verbe HTTP	Paramètres du service
<b>Get all comments for a wine</b>	http://localhost:8091/api/wines/{id}/comments	GET	Dans l'URL de la requête HTTP l'identifiant du vin.
	<p>Le service retourne un message JSON contenant un tableau des commentaires pour le vin identifié par id de la forme suivante avec un code de retour HTTP 200 OK :</p> <pre>[   {     "id": "89c7ac6a-907c-4b67-ba64-ee58274be3c1",     "author": "Romain.Royer30",     "email": "Romane.Morel@gmail.com",     "date": "2016-02-29T03:35:02.261Z",     "message": "Ipsa molestiae voluptas omnis quod est sit est voluptatem nesciunt.\nSimilique repellendus accusantium quo dolore quam voluptatum.\nDolorem ut qui facilis."   },   ... ]</pre> <p>Un code de retour HTTP 404 Not Found si l'identifiant n'a pas pu être trouvé et le message suivant : Wine not found: e2c3d7cb-a357-47f5-b100-501457e3d</p>		
<b>Post a comment for a wine</b>	http://localhost:8091/api/wines/{id}/comments	POST	Dans l'URL de la requête HTTP l'identifiant du vin et dans le body un message JSON de la forme suivante : <pre>{   "author": "Romain.Royer30",   "email": "Romane.Morel@gmail.com",   "date": "2016-02-29T03:35:02.261Z",   "message": "my comment" }</pre>
	<p>Le service retourne un message JSON contenant le commentaire du vin de la forme suivante si l'identifiant du vin est valide avec un code de retour HTTP 201 Created :</p> <pre>{   "id": "be147222-009d-446f-b4d2-ee0e80d3131c",   "author": "Romain.Royer30",   "email": "Romane.Morel@gmail.com",   "date": "2016-02-29T03:35:02.261Z",   "message": "my comment" }</pre> <p>Un code de retour HTTP 404 Not Found si l'identifiant n'a pas pu être trouvé et le message suivant : Wine not found: e2c3d7cb-a357-47f5-b100-501457e3d</p>		

## Service de gestion des stocks (magasin)

Ce service est accessible aux URLs suivantes :

- <http://localhost:8092/>
- <http://dvxx-sparkfeign-stock.herokuapp.com/>
- <http://ilaborie.org:8092/>

Service	URL	Verbe HTTP	Paramètres du service
<b>Get stock infos for a wine</b>	<a href="http://localhost:8092/api/wines/{id}/qty">http://localhost:8092/api/wines/{id}/qty</a>	GET	Dans l'URL de la requête HTTP l'identifiant du vin.
	<p>Le service retourne un message JSON contenant le nombre de bouteilles disponible et le prix unitaire pour le vin identifié par id de la forme suivante avec un code de retour HTTP 200 OK :</p> <pre>{  "stock": 32,  "price": 13.2}</pre> <p>Un code de retour HTTP 404 Not Found si l'identifiant n'a pas pu être trouvé et le message suivant : <code>Wine not found: e2c3d7cb-a357-47f5-b100-501457e3d</code></p>		
<b>Order a wine</b>	<a href="http://localhost:8092/api/wines/{id}/order?qty={number_bottles}">http://localhost:8092/api/wines/{id}/order?qty={number_bottles}</a>	POST	Dans l'URL de la requête HTTP l'identifiant du vin et dans le paramètre <code>qty</code> de la requête HTTP le nombre de bouteilles à commander.
	<p>Le service retourne un message <code>"Order accepted"</code> avec un code de retour HTTP 200 OK.</p> <p>Un code de retour HTTP 404 Not Found si l'identifiant n'a pas pu être trouvé et le message suivant : <code>Wine not found: e2c3d7cb-a357-47f5-b100-501457e3d</code></p> <p>Un code de retour HTTP 400 Bad request si le nombre de bouteilles commandées n'est pas un entier positif : <code>Invalid quantity</code></p> <p>Un code de retour HTTP 400 Bad request si le nombre de bouteilles commandées excède le stock disponible dans le magasin : <code>Not enough stock</code></p>		



# Exercices Feign

L'objectif général de ces exercices est de consommer depuis votre application écrite en Java des services REST existants. L'utilisation de la documentation des services existants fournie ci-dessus est très fortement recommandée.

## Exercice 1.1 - Encodeur/Décodeur JSON

Pour consommer des services REST il faut que Feign soit capable de décoder les messages JSON retournés par les services afin de les transformer en objets Java (par exemple lors du retour du service de recherche des vins); mais aussi d'encoder des objets Java au format JSON afin de les envoyer à un service REST en paramètre (par exemple lors de l'envoi d'une commande).

En particulier le service d'authentification est le premier service qui sera utilisé par l'application et ce dernier a besoin de savoir à la fois encoder et décoder des messages au format JSON.

### Cet exercice va m'apprendre à :

Configurer Feign afin de pouvoir lire et écrire des messages au format JSON.

### Travail à réaliser

- Le travail à réaliser est de configurer Feign pour qu'il puisse encoder / décoder les messages au format JSON.
- Dans le cadre de cet exercice le choix de la librairie de sérialisation / désérialisation JSON devra être Gson.
- Il faudra modifier la méthode suivante :  
`devoxx.microframeworks.exos.services.Services#createFeignService`

### Comment savoir si l'exercice est terminé ?

- Le test `devoxx.microframeworks.exos.exo1.Exo_1_1` doit passer.
- Regarder les logs dans la console pour voir les messages échangés en JSON

### Support et documentation

- La documentation de Feign est disponible sur [GitHub](https://github.com/Netflix/feign) (<https://github.com/Netflix/feign>). Pensez à l'utiliser.
- Si vous avez besoin d'explication sur le code Java8 fourni: Demandez nous !



- Attention, nous avons déjà ajouter la dépendance sur l'extension Feign pour l'intégration de Gson dans le pom.xml.

## Exercice 1.2 - Méthodes GET & Paramètres

L'application doit consommer les services REST qui permettent d'interroger le service de référence sur les vins afin de récupérer la liste de tous les vins, leurs caractéristiques et d'effectuer une recherche sur des critères. Pour se faire notre boulot est d'utiliser Feign pour consommer les services existants qui répondent à la spécification suivante sachant que l'interface à utiliser a déjà été développée.

### Cet exercice va m'apprendre à :

Ajouter les annotations Feign sur une interface déjà écrite par l'équipe de développement pour des requêtes de type GET tout en gérant des paramètres de type URL et de type Query.

### Travail à réaliser

- Il faudra modifier l'interface suivante afin de rajouter les annotations Feign:  
`devovx.microframeworks.exos.services.ReferenceService`

### Comment savoir si l'exercice est terminé ?

- Le test `devovx.microframeworks.exos.exo1.Exo_1_2` doit passer.
- Regarder les logs pour voir les messages échangés en JSON.

### Support et documentation

- Voir les slides de présentation de Feign.
- La spécification des services de référence à consommer
- La documentation de Feign est disponible sur GitHub (<https://github.com/Netflix/feign>). Pensez à l'utiliser.
- Si vous avez besoin d'explication sur le code Java8 fourni: Demandez nous !

## Exercice 1.3 - Méthodes POST & Body

L'application doit consommer les services REST qui permettent de récupérer l'ensemble des commentaires précédemment posté sur un vin et d'ajouter un nouveau commentaire sur un vin. On notera que la gestion et la persistance des commentaires est assuré par un micro service externe à notre application permettant ainsi de mettre en commun les commentaires des plusieurs applications.

## Cet exercice va m'apprendre à :

Ajouter les annotations Feign sur une interface déjà écrite par l'équipe de développement pour des requêtes de type POST et la prise en compte des paramètres envoyés dans le Body d'une requête HTTP.

## Travail à réaliser

- Il faudra modifier l'interface suivante afin de rajouter les annotations Feign:  
`devovx.microframeworks.exos.services.CommentService`

## Comment savoir si l'exercice est terminé ?

- Le test `devovx.microframeworks.exos.exo1.Exo_1_3` doit passer.
- Regarder les logs pour voir les messages échangés en JSON

## Support et documentation

- Voir les slides de présentation de Feign.
- La spécification des services de commentaires à consommer
- La documentation de Feign est disponible sur GitHub (<https://github.com/Netflix/feign>). Pensez à l'utiliser.
- Si vous avez besoin d'explication sur le code Java8 fourni: Demandez nous !

## Exercice 1.4 - On part de zéro...

L'application doit consommer les services REST qui permettent de s'interfacer avec le micro service de commande du magasin afin de récupérer le stock disponible et le tarifs des bouteilles à commander sur le site de vente en ligne d'une part. Et d'autre part de s'interfacer avec le service de commande de bouteilles pour enrichir notre cave. Cependant pour cette interface un stagiaire facétieux de l'équipe de développement n'a coder qui s'est essayé au TDD n'a codé qu'une classe "Mock" pour faire compiler ses tests.

## Cet exercice va m'apprendre à :

Écrire l'interface Feign à partir de zéro pour consommer des services REST existants sur la base de leurs spécifications.

## Travail à réaliser

- Il faudra dans un premier temps supprimer la classe Mock développée par le stagiaire afin de la remplacer par une interface Feign.
- Il faudra modifier l'interface suivante afin de rajouter les annotations Feign:  
`devovx.microframeworks.exos.services.StockService`

## Comment savoir si l'exercice est terminé ?

- Le test `devoxx.microframeworks.exos.exo1.Exo_1_4` doit passer.
- Regarder les logs pour voir les messages échangés en JSON.

## Support et documentation

- Voir les slides de présentation de Feign.
- La spécification des services de stock à consommer
- La documentation de Feign est disponible sur GitHub (<https://github.com/Netflix/feign>). Pensez à l'utiliser.
- Si vous avez besoin d'explication sur le code Java8 fourni: Demandez nous !

## Exercices Bonus (Série 1)

### Bonus 1.5 : Utilisation de Jackson à la place de GSON

Lors de l'exercice 1.1, nous avons utilisé Gson pour réaliser l'encodage et le décodage des messages au format JSON. Feign peut utiliser d'autres librairies pour cette tâche telle que Jackson.

Regardez la documentation de Feign et remplacez Gson par Jackson. Pensez à modifier le fichier `pom.xml` afin d'importer les bons JARs. Vous trouverez les extensions de feign dans le [repository central de maven](#).

### Bonus 1.6 : Ajoutez des logs

Il peut être utile de logger les différents échanges effectué à l'aide de Feign.

Regardez la documentation de Feign afin de voir comment mettre en place un système de log général sans avoir à le coder avant chaque utilisation d'un service via un `requestInterceptor`.

Notez que l'on peut aussi fournir un `Logger` pour le `FeignBuilder`, ou bien utiliser l'extension `feign-slf4j` pour ajouter des logs, mais l'objectif de l'exercice consiste à utiliser le couteau suisse de la customisation: le `requestInterceptor`.

### Bonus 1.7 : Changez la configuration des services afin de pointer sur les services Heroku

En éditant le fichier `configuration.properties` vous pouvez changer l'URL des services utilisé par Feign, tester avec les services en ligne sur Heroku ou sur [ilaborie.org](http://ilaborie.org).

**Vous avez bien mérité votre pause ! :-)**

# Exercices SparkJava

La première série d'exercices ont permis d'écrire les services permettant de consommer des services REST déjà existant depuis le backend de notre application à l'aide de la technologie **Feign**. Il est temps maintenant de construire les routes REST de ce backend afin qu'elles puissent à leur tour être consommées par l'application WEB développée par l'équipe de développement "Front". Pour ce faire nous utiliserons **SparkJava**.

Par défaut votre application sera accessible à l'URL suivante : <http://localhost:8080>

NB: Vous pouvez changer le port de votre service dans le fichier de propriétés suivant :

```
/src/main/resources/configuration.properties
```

Le serveur HTTP hébergera à la fois les services REST de votre backend ainsi que l'application WEB développée en AngularJS dont les ressources statiques seront exposées par SparkJava comme cela a été montré dans les slides.

Vous pouvez analyser comment c'est implémenté en regardant la classe :

```
devoxx.microframeworks.exos.Main
```

## Exercice 2.1 - Services pour la page 'catalogue'

L'application WEB pour notre cave s'appuie sur des services REST qui faut implémenter avec Spark Java.

On utilisera les services Feign développés dans la première série soit de manière directe soit en agrégeant plusieurs résultats issus de plusieurs services dans un seul objet à retourner. Il s'agit ici de traiter la problématique d'agrégation de plusieurs micro services au sein d'un service de plus haut niveau. Plusieurs approches sont possibles ;

On se contentera pour cet exercice d'implémenter l'approche la plus simple et la plus naïve consistant à récupérer les informations l'une après l'autre.

**Cet exercice va m'apprendre à :**

- Analyser comment SparkJava expose des ressources statiques (site WEB)
- Analyser le mapping des routes REST et l'utilisation avantageuse de Java8 pour le découpage du code
- Implémenter les routes de notre application en utilisant les services Feign déjà développés à l'exercice 1.2.

## Travail à réaliser

- Regarder la classe `devovx.microframeworks.exos.Main` afin de prendre connaissance des routes REST qui sont implémentées par classe `WineRoute`
- Éditer la classe `devovx.microframeworks.exos.routes.WineRoute` afin d'implémenter les méthodes de la route `handleFindById` et `handleSearch`.
- La route `handleFindById` doit construire un objet `WineDetail` en interrogeant les services REST nécessaires et en agrégeant le résultat des 3 requêtes dans un même objet qui sera enfin retourné.
- La route `handleSearch` doit retourner les 20 premiers résultats (objets `Wine`) de la recherche effectuées grâce au service REST de recherche du micro service Reference. Il n'y a pas d'agrégation ici, simplement la limitation à 20 éléments maximum pour les résultats. Vous pouvez par exemple utiliser un `Stream` et sa méthode `limit()` pour ne garder que les 20 premiers résultats de la recherche.

## Comment savoir si l'exercice est terminé ?

- Le test `devovx.microframeworks.exos.exo2.Exo2_1` doit passer.
- Regarder les logs dans la console pour voir les messages échangés en JSON
- Vous pouvez tester l'application dans votre navigateur. Au début de l'exercice la page d'authentification doit marcher pour tout adresse email valide avec le mot de passe **admin**. La page 'Catalogue' doit fonctionner. Le détail d'un vin ne va pas remonter les commentaires, les pages concernant 'ma cave' et 'mon panier' ne fonctionneront pas encore.

### Catalogue


<b>Côtes du Rhône AOC</b> COTES DU RHONE BLANC 2014 - E.GUIGAL €	2014 <a href="#">Detail</a>
<b>Côtes Catalanes IGP</b> CADIRETA 2014 - DOMAINE LAFAGE €	2014 <a href="#">Detail</a>
<b>Bourgogne AOC</b> LAFORET BLANC 2014 - JOSEPH DROUHIN €€	2014 <a href="#">Detail</a>
<b>Ardèche IGP</b> GRAND ARDECHE 2013 - LOUIS LATOUR €	2013 <a href="#">Detail</a>
<b>Costières de Nîmes AOC</b> SAINTE-CECILE BLANC 2014 - CHATEAU L'ERMITAGE €€	2014 <a href="#">Detail</a>
<b>Savoie AOP</b> GAMAY VIEILLES VIGNES 2014 - CAVE DE CHAUTAGNE €€	2014 <a href="#">Detail</a>
<b>Savoie AOC</b> APREMONT CUVÉE PRESTIGE 2014 - DOMAINE DE ROUZAN €€	2014 <a href="#">Detail</a>

### Côtes du Rhône AOC - 2014

Rhône - Vin Blanc

#### COTES DU RHONE BLANC 2014 - E.GUIGAL

€ [Acheter](#)



**Cépage:** Viognier, Roussanne, Marsanne, Clairette, Bourboulenc, Grenache blanc  
**Alcool:** 13,5% vol  
**Température de service:** 12°C

**Conservation:** A boire dans les 2 ans

**A l'oeil:** Robe jaune or brillante.  
**Au nez:** Arômes de fleurs blanches, d'abricot, d'acacias et de pêche blanche.  
**En Bouche:** Vin fruité avec beaucoup de gras et de richesse.

**Accords mets-vin:** Salades variées, volaille, poissons, plateau de fromage  
**Accords recommandés:** Terrines de volaille, poulet fermier

## Support et documentation

- La [Documentation de Spark](#) est disponible. Pensez à l'utiliser.
- Si vous avez besoin d'explication sur le code Java8 : demandez nous !

## Services d'affichage / recherche des Vins

Ce service est accessible aux URLs suivantes :

- <http://localhost:8080/>

Service	URL	Verbe HTTP	Paramètres du service
<b>Wine By Id</b>	<a href="http://localhost:8080/api/wines/{id}">http://localhost:8080/api/wines/{id}</a>	GET	Dans l'URL de la requête HTTP l'identifiant du vin.
	<p>Le service retourne un message JSON contenant l'agrégation de 3 services suivants :</p> <ul style="list-style-type: none"><li>• service de reference : <code>findById(wineId)</code></li><li>• service de stock : <code>findById(wineId)</code></li><li>• service de commentaires : <code>findById(wineId)</code></li></ul> <p>Ci-dessous un exemple de message retourné qui correspond à la classe <b>WineDetail</b> si l'identifiant est valide avec un code de retour HTTP 200 OK :</p> <pre>{   "wine": {     "id": "be147222-009d-446f-b4d2-ee0e80d3131c",     "produit": "CHATEAU HAUT MAZIERES 2014",     "millesime": 2014,     "appellation": "Bordeaux AOC",     "couleur": "Vin Rouge",     [...]     "temperatureDeService": "16-18°C",     "accordsMetsVin": "Charcuterie, Viande rouge, Viande blanche, Fromage"   },   "stock": {     "stock": 18,     "price": 13.2   },   "comments": [     {       "id": "89c7ac6a-907c-4b67-ba64-ee58274be3c1",       "author": "Romain.Royer30",       "email": "Romane.Morel@gmail.com",       "date": "2016-02-29T03:35:02.261Z",       "message": "Ipsa molestiae voluptas."     },     ...   ] }</pre> <p>Un code de retour HTTP 404 Not Found si l'identifiant n'a pas pu être trouvé et le message suivant : <code>Wine not found: e2c3d7cb-a357-47f5-b100-501457e3d</code></p>		

<b>Search Wine</b>	http://localhost:8080/api/wine?q={query}	GET	Dans le paramètre q de la requête HTTP le terme à chercher.
	<p>Le service retourne un message JSON correspondant à la classe <b>Wine</b> contenant un tableau de vins de la forme suivante correspondant au <u>résultat de la recherche limité à 20 résultats du service de référence sans transformation</u> avec un code de retour HTTP 200 OK :</p> <pre>[   {     "id": "be147222-009d-446f-b4d2-ee0e80d3131c",     "produit": "CHATEAU HAUT MAZIERES 2014",     "millésime": 2014,     "appellation": "Bordeaux AOC",     "couleur": "Vin Rouge",     "region": "Bordeaux",     "pays": "France",     ...   },   ... ]</pre>		

## Exercice 2.2 - Services pour la page ‘ma cave’

L’objectif de cet exercice est d’implémenter les services utilisés dans la ‘Ma cave’. On notera que ces services utilisent un token [JWT](#) pour identifier l’utilisateur et lui présenter sa cave. Ce token est passé dans le Header HTTP de la manière suivante :

Authorization:

Bearer

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0eXN0QGVraXRvLmZyIn0.-NCYxyQ06UC2\_W9VbSzfGVZ\_mmo2tLXXHRt6G90Nqc8

L’utilisation et la gestion de ce token sont fournis dans cet exercice dans la méthode :  
**devovx.microframeworks.exos.routes.CellarRoute#getUser**

### Cet exercice va m’apprendre à :

- Écrire le mapping des routes REST pour la page ‘Ma cave’ pour les opérations suivantes :
  - récupération du contenu de ma cave,
  - boire une bouteille,
  - mettre un vin comme favori.
- Implémenter les routes en utilisant les services Feign déjà développés.

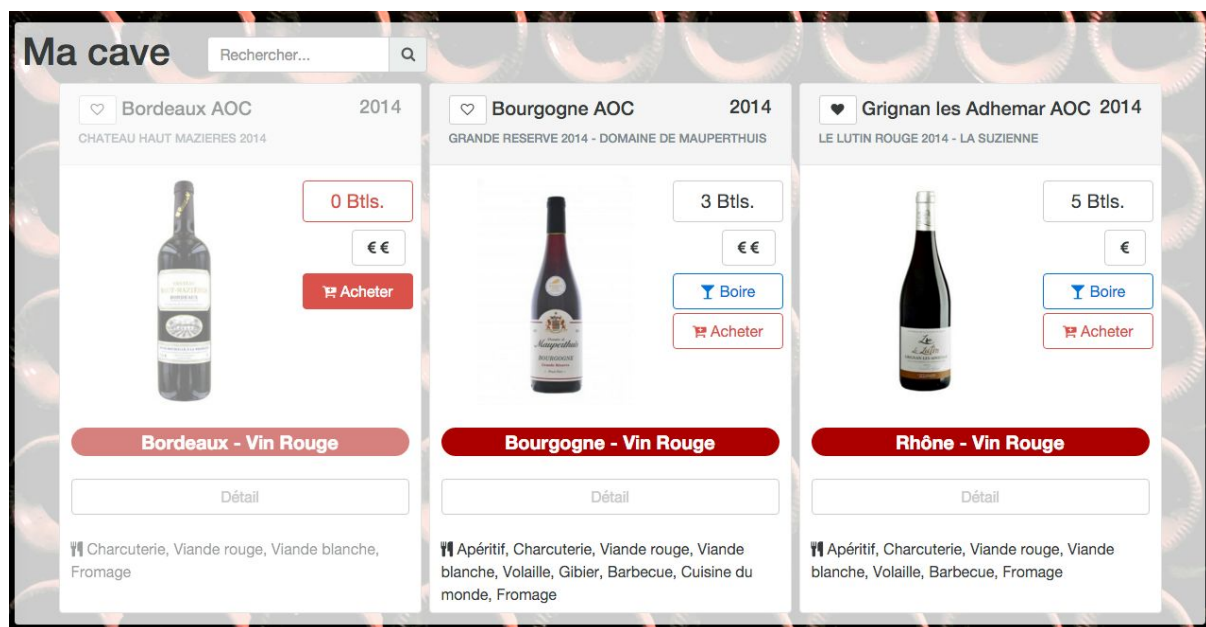


## Travail à réaliser

- Compléter la classe `devovx.microframeworks.exos.Main` afin d'ajouter les routes REST qui sont exposées sur le endpoint `Cellar` avec l'aide de la route `CellarRoute`.
- Éditer la classe `devovx.microframeworks.exos.routes.CellarRoute` afin d'implémenter les méthodes de la route `handleMyCellar`, `handleDrink` et `handleFavorite` avec l'aide du service `CellarService`.
- La route `handleMyCellar` doit construire un objet `List<CellarEntry>` en interrogeant le service local `CellarService`.
- La route `handleDrink` doit décrémenter de 1 le nombre de bouteilles du vin de notre cave.
- La route `handleFavorite` permet de mettre à jour le tag favori d'un vin. Ce service doit retourner l'état booléen qui a été positionné.

## Comment savoir si l'exercice est terminé ?

- Le test `devovx.microframeworks.exos.exo2.Exo2_2` doit passer.
- Regarder les logs dans la console pour voir les messages échangés en JSON
- Vous pouvez tester l'application dans votre navigateur. Au début de l'exercice la page d'authentification doit marcher pour tout adresse email valide avec le mot de passe **admin** et la page du catalogue de vin doit être fonctionnelle.
- La page 'Ma cave' doit fonctionner : on doit voir les vins de notre cave s'afficher, on doit pouvoir décrémenter le nombre de bouteilles en cliquant sur le bouton "Boire", et tagger en tant que favori les vins de votre cave. Pour ajouter des vins dans la cave, il faut passer par la page catalogue, puis acheter un ou des vins, enfin dans le panier il faut valider la commande.



## Support et documentation

- La [Documentation de Spark](#) est disponible. Pensez à l'utiliser.
- Si vous avez besoin d'explication sur le code Java8 : demandez nous !
- **Attention le frontend WEB a besoin des données au format JSON, pensez à mettre un encodeur JSON dans la déclaration de vos routes.**

## Services de gestion de la cave

Ce service est accessible aux URLs suivantes : <http://localhost:8080/>

Service	URL	Verbe HTTP	Paramètres du service
<b>Get Cellar</b>	<a href="http://localhost:8080/api/cellar">http://localhost:8080/api/cellar</a>	GET	Aucun paramètre.
	Le service retourne un message JSON qui correspond à la classe <b>List&lt;CellarEntry&gt;</b> avec un code de retour HTTP 200 OK : <pre>[   {     "wine": {       "id": "be147222-009d-446f-b4d2-ee0e80d3131c",       "produit": "CHATEAU HAUT MAZIERES 2014",       "millesime": 2014,       [...]       "auNez": "notes boisées",       "enBouche": "Tanins équilibrés, arômes du nez",       "aOeil": "Robe prune, reflets rubis"     },     "quantity": 2,     "favorite": true   },   {     "wine": {       "id": "2c61171e-8904-4131-a29b-4887b61bd44f",       "produit": "BOURGOGNE ALIGOTE 2014 - CAVE DE GENOUILLY",       "millesime": 2014,       [...]       "auNez": "Notes florales et d\u0027agrumes",       "enBouche": "Vif avec de la rondeur et du gras.",       "aOeil": "Limpide et clair."     },     "quantity": 3,     "favorite": false   } ]</pre>		
<b>Favorite</b>	<a href="http://localhost:8080/api/cellar/favorite/{id}">http://localhost:8080/api/cellar/favorite/ {id}</a>	POST	Dans l'URL de la requête HTTP l'identifiant du vin et dans le body la nouvelle valeur pour le tag "favorite" : true ou false

	Le service retourne la string true ou false correspondant à ce qui a été envoyé dans la requête avec un code de retour HTTP 200 OK		
Drink	http://localhost:8080/api/cellar/drink/ {id}	POST	Dans l'URL de la requête HTTP l'identifiant du vin et dans le body la valeur -1 (modification du stock)
	Le service retourne le nombre restant de bouteilles avec un code de retour HTTP 200 OK		

## Exercice 2.3 - Service pour l'ajout d'un commentaire

L'objectif de cet exercice est d'implémenter le service qui permet de poster un commentaire utilisés dans la page 'Détail'.

### Cet exercice va m'apprendre à :

- Créer la route REST pour l'ajout d'un commentaire sur la page détail d'un vin.
- Implémenter la route en utilisant le service Feign déjà développé.

### Travail à réaliser

- Compléter la classe `devoxx.microframeworks.exos.Main` afin d'ajouter la route REST qui est exposée sur le endpoint Comment avec l'aide du service CommentService. Cette route doit respecter la spécification suivante:

HTTP POST /api/wine/<id>/comments avec le message (String) dans le corps de la requête.

On doit retourner sous forme JSON le commentaire créé incluant la date, l'auteur, l'identifiant du commentaire, ...

- Éditer la classe `devoxx.microframeworks.exos.routes.CommentRoute` afin d'implémenter la méthode de la route `handleAddComment`.
- La route `handleAddComment` doit construire un objet Comment qui sera enrichie par le service d'ajout de commentaire. Elle devra retourner l'objet Comment qu'elle aura reçu en résultat de l'appel du service Feign.

### Comment savoir si l'exercice est terminé ?

- Le test `devoxx.microframeworks.exos.exo2.Exo2_3` doit passer.
- Regarder les logs dans la console pour voir les messages échangés en JSON
- Sur la page de détail d'un vin on peut ajouter un commentaire.

## Madiran AOC - 2012

MADIRAN ODE D'AYDIE 2012 - CHATEAU D'AYDIE

Sud-Ouest - Vin Rouge

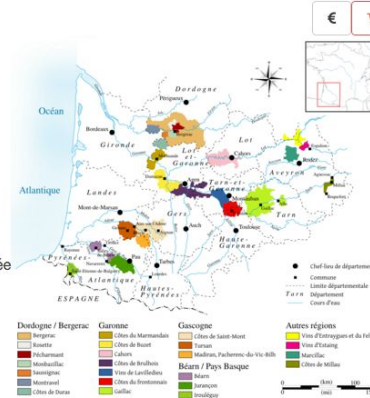


Cépage: 100% Tannat  
Alcool: 14,5% vol  
Température de service: 15-17°C

Conservation: A boire et à garder  
A boire à partir de: 2014  
Garder jusqu'en: 2021

A l'oeil: Robe rouge profond  
Au nez: Arômes de fruits noirs et d'épices  
En Bouche: Bouche pleine et ronde, avec une finale longue et vanillée

Accords mets-vin: Viande rouge, Viande blanche  
Accords recommandés: Gigot d'agneau, bœuf bourguignon



## Commentaires

Laudantium quo doloremque explicabo accusamus. Fuga explicabo dolor non. Laudantium consequatur eos.

— 29 févr. 2016, [Clément.Roy15](#)

Repudiandae consectetur quae aut facilis optio. Animi ratione deleniti suscipit et accusantium quibusdam blanditiis et excepturi. Earum aut deserunt quam architecto eum similique eligendi laborum aut.

— 28 févr. 2016, [Victor.Colin84](#)

Optio saepe suscipit. Voluptates sit vero est ex maiores delectus voluptatibus. Voluptates sed et incidunt assumenda quos quae fugit quisquam officia.

— 29 févr. 2016, [Nathan73](#)

Super avec les viandes en sauces.

— , [aze@aze](#)

J'adore

Ajouter un commentaire

## Support et documentation

- La [Documentation de Spark](#) est disponible. Pensez à l'utiliser.
- Si vous avez besoin d'explication sur le code Java8 : demandez nous !

Service	URL	Verbe HTTP	Paramètres du service
Add comment	<code>http://localhost:8080/api/wine/{id}/comments</code>	POST	Dans l'URL de la requête HTTP l'identifiant du vin et dans le body le commentaire (string) ; par exemple : "ceci est un commentaire"
	Le service retourne un message JSON qui correspond au commentaire enregistré avec un code de retour HTTP 200 OK : { "id": "64f48683-ec0b-46e0-bf55-99bd9ad7e520", "author": "test@ekito.fr", "email": "test@ekito.fr", }		

<pre>"date": "2016-03-12T20:33:25.731Z", "message": "ceci est un commentaire" }</pre>
---

## Exercice 2.4 - Gestion des exceptions

L'objectif de cet exercice est d'associer un code d'erreur HTTP et le message d'erreur qui seront retournés par SparkJava dans l'hypothèse où une exception vient à être levée dans une route.

### Cet exercice va m'apprendre à :

- Gérer les exceptions levées dans notre backend afin de retourner les codes d'erreurs classiques de HTTP.

### Travail à réaliser

- Compléter la classe `devvoxx.microframeworks.exos.Main` afin d'ajouter la gestion des exceptions suivantes :
  - `NoSuchElementException.class` => HTTP 404 Not Found
  - `IllegalArgumentException.class` => HTTP 400 Bad Request
  - `NumberFormatException.class` => HTTP 400 Bad Request
  - `SecurityException.class` => HTTP 403 Forbidden

### Comment savoir si l'exercice est terminé ?

- Ouvrir l'URL suivante dans votre navigateur :  
<http://localhost:8080/api/wine/be147222-009d-446f-b4d2-ee0e80d3131c>
  - Elle doit retourner un fichier JSON
- Ouvrir l'URL suivante dans votre navigateur:  
<http://localhost:8080/api/wine/be147222-009d-446f-b4d2-ee0e80d3131c-error>
  - Elle doit retourner une erreur 404 et le message "Wine not found: be147222-009d-446f-b4d2-ee0e80d3131c-error" car l'id du vin est inconnu.

```

▶ http :8080/api/wine/be147222-009d-446f-b4d2-ee0e80d3131c
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: Content-Type,Authorization,X-Requested-With,Content-Length,Accept,Origin,
Access-Control-Allow-Methods: GET,PUT,POST,DELETE,OPTIONS
Access-Control-Allow-Origin: *
Content-Type: text/html;charset=utf-8
Date: Sun, 13 Mar 2016 17:02:37 GMT
Server: Jetty(9.3.2.v20150730)
Transfer-Encoding: chunked

{
  "wine": {
    "id": "be147222-009d-446f-b4d2-ee0e80d3131c",
    "produit": "CHATEAU HAUT MAZIERES 2014",
    "millésime": 2014,
    "appellation": "Bordeaux AOC",
    "couleur": "Vin Rouge",
    "region": "Bordeaux",
    "pays": "France",
    "cepageDominant": "64% Merlot, 22% Cabernet-sauvignon, 14% Cabernet-franc",
  }
}

▶ http :8080/api/wine/be147222-009d-446f-b4d2-ee0e80d3131c-error
HTTP/1.1 404 Not Found
Content-Type: plain/text
Date: Sun, 13 Mar 2016 17:01:30 GMT
Server: Jetty(9.3.2.v20150730)
Transfer-Encoding: chunked

Wine not found: be147222-009d-446f-b4d2-ee0e80d3131c-error

```

## Support et documentation

- La [Documentation de Spark](#) est disponible. Pensez à l'utiliser.
- Si vous avez besoin d'explication sur le code Java8 : demandez nous !
- Utiliser les méthodes de `devoxx.microframeworks.exos.routes.ErrorRoute`.

## Bonus - Spark Java

### Bonus 2.5 - Asynchronisme

Une approche plus réaliste du monde réel serait de récupérer l'ensemble des résultats nécessaire pour la construction du `WineDetail` de manière parallèle et asynchrone tout en gérant le cas des timeouts et des valeurs par défaut qui doivent être utilisées dans le cas d'une non réponse d'un des micro-services.

La méthodes `WineRoute#handleFindById` et `OrderRoute#handleOrder` font plusieurs appels REST, on pourrait les traiter en parallèle avec les [CompletableFuture](#) ou les [Stream](#) de Java 8.

### Bonus 2.6 - Docker

Créer un [Dockerfile](#) pour exécuter votre service dans un docker.