**Haute Ecole de la Province de Liège : Département Informatique**

**==================================================**

**Graduat en Informatique et Systèmes : finalité Réseaux**

**LABORATOIRE D'INFORMATIQUE ET SYSTEMES**

**===========================================**

##### SYSTEMES DISTRIBUES

3ème année

###### DOSSIER FINAL : version simplifiée d’une solution d'e-commerce

**======================**

Année Académique 2020-2021 ANCIAUX LOIC

Groupe : 2326

# Table des matières

Contents

[Table des matières 2](#_Toc59197676)

[1) Tableau de correspondance service – technologie employée 3](#_Toc59197677)

[Service User : 3](#_Toc59197678)

[Service Products : 3](#_Toc59197679)

[Service TVA : 3](#_Toc59197680)

[Service Stock : 3](#_Toc59197681)

[Service Cart : 3](#_Toc59197682)

[Service livraison : 3](#_Toc59197683)

[Service Order : 4](#_Toc59197684)

[Service Checkout : 4](#_Toc59197685)

[API de Fournisseurs : 4](#_Toc59197686)

[2) Diagramme global 5](#_Toc59197687)

[Zuul et Eureka : 5](#_Toc59197688)

[Diagramme global d’interactions entre services : 5](#_Toc59197689)

[Exemple pour l’ajout d’un article dans le panier d’un utilisateur connecté : 6](#_Toc59197690)

[3) Relevé des problèmes rencontrés 7](#_Toc59197691)

[Problème 1 : 7](#_Toc59197692)

[Problème 2 : 7](#_Toc59197693)

[Problème 3 : 8](#_Toc59197694)

[4) Scénario de démonstration 9](#_Toc59197695)

[Démonstrations : 9](#_Toc59197696)

[Vidéo : 9](#_Toc59197697)

[5) Solution à une situation exceptionnelle 10](#_Toc59197698)

[Une instance d’un service tombe down : 10](#_Toc59197699)

# 1) Tableau de correspondance service – technologie employée

## Service User :

Technologie : REST

Port : 8081

URIs : - /user : retourne l’utilisateur dont l’id et le mdp sont passés en paramètre

- /login : permet de log in un utilisateur par son mail et son mot de passe

## Service Products :

Technologie : REST

Port : 8082, 8092

URIs : - /getProducts : retourne tous les produits

- /getProduct : retourne le produit dont l’id est passé en paramètre

## Service TVA :

Technologie : REST

Port : 8083

URIs : - /getTvaByCategory : retourne la TVA pour une catégorie d’articles

- /getTvaByProduct : retourne la TVA pour un produit

## Service Stock :

Technologie : REST

Port : 8084

URIs : - /getStockByProduct : retourne le stock disponible d’un produit et recommande le produit auprès des fournisseurs si le stock est à 0

- /restock : recommande un produit auprès des fournisseurs

## Service Cart :

Technologie : REST et JMS

Port : 8085

URIs : - /addItem : ajoute un article au panier de l’utilisateur

- /removeItem : enleve un article au panier de l’utilisateur

- /getCart : retourne le panier complet de l’utilisateur

Utilisation de JMS : le service cart peut recevoir un article à enlever du panier via JMS (il faut donc lui envoyer l’user id, le mot de passe ainsi que le l’article à enlever).

## Service livraison :

Technologie : REST

Port : 8086

URIs : - /getShippingRates : retourne les forfaits de livraison

- /getShippingRateByName : retourne le prix du forfait de livraison lié au nom en paramètre

## Service Order :

Technologie : REST

Port : 8087

URIs : - /getOrders : retourne les commandes d’un utilisateur

- /getOrder : retourne la commande en cours de l’utilisateur

## Service Checkout :

Technologie : REST

Port : 8088

URIs : - /checkout : finalise la commande en cours d’un utilisateur

## API de Fournisseurs :

Technologie : REST

Port : 8100, 8101, 8102

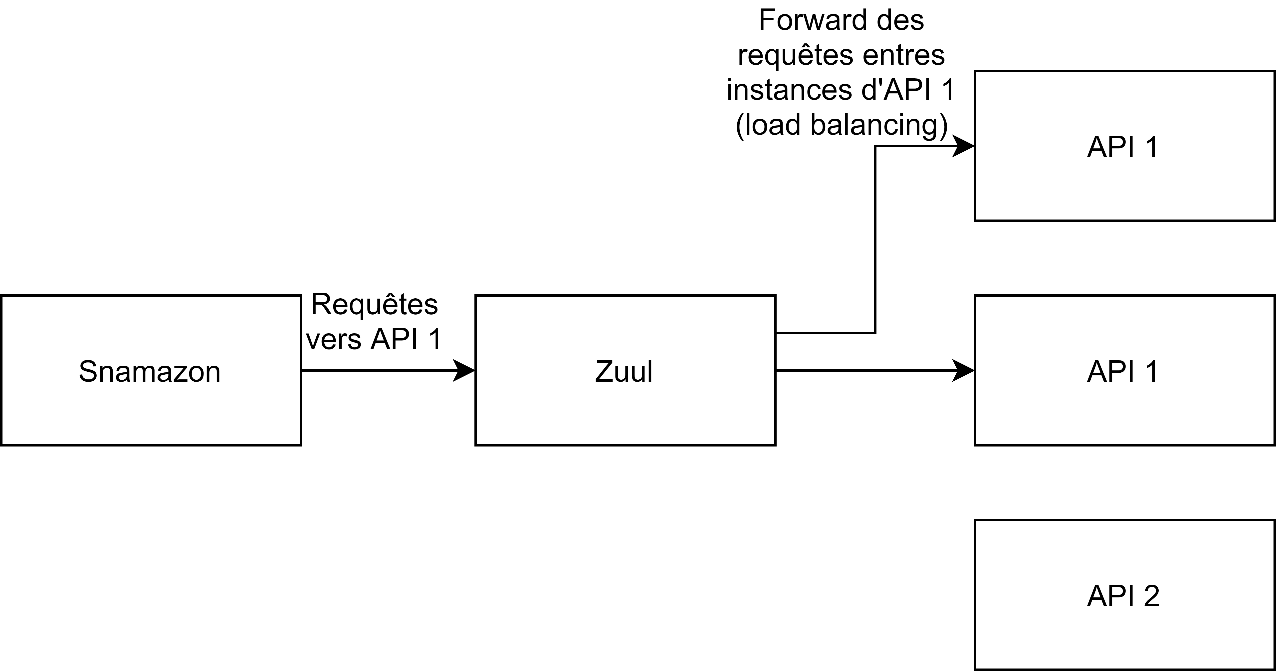
URIs : - /getPrice : retourne le prix d’un article

- /getQuantite : retourne la quantité disponible d’un article

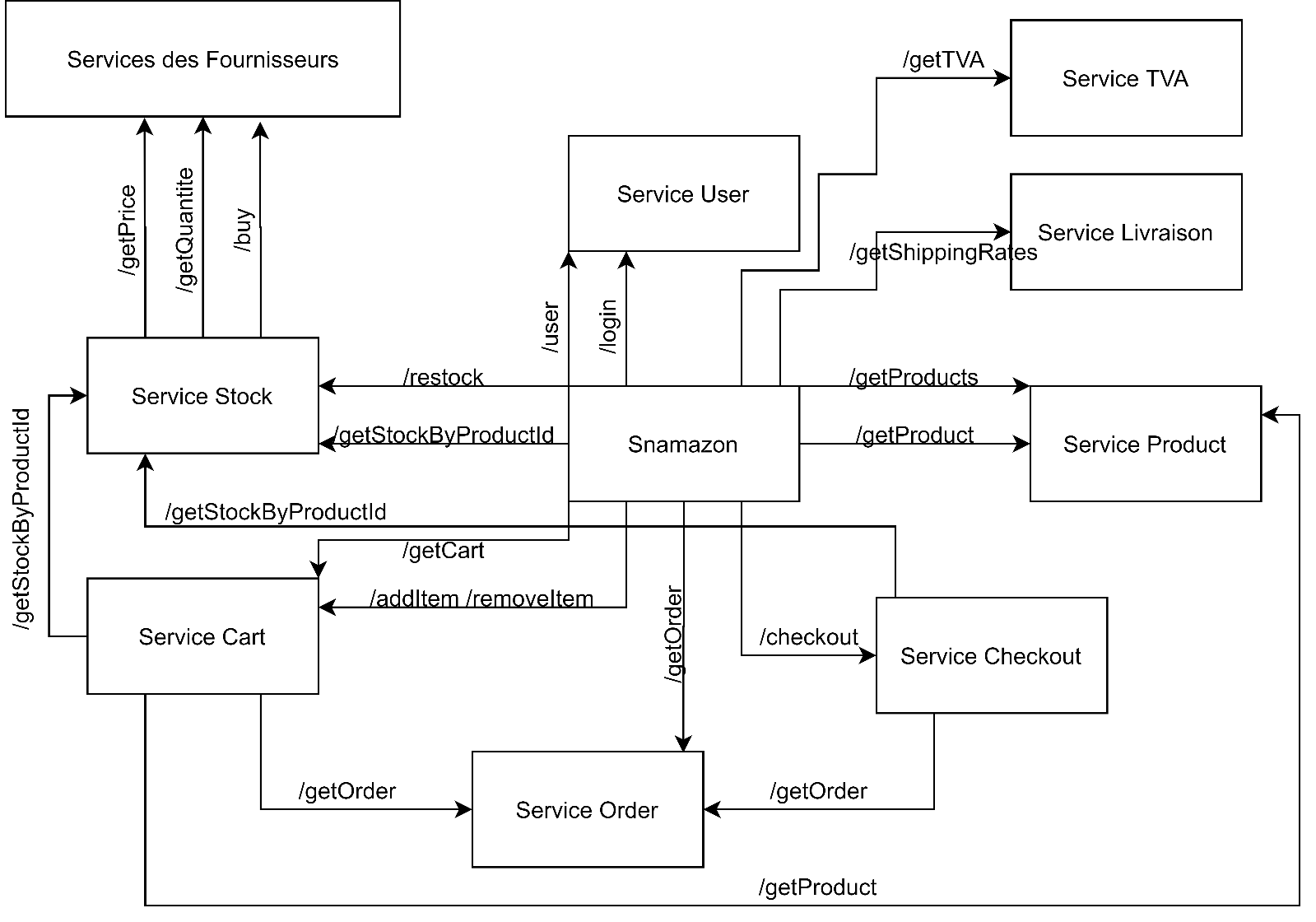
- /buy : achete la quantité voulue d’un article

# 2) Diagramme global

## Zuul et Eureka :

Tous les services sont connus de Eureka et sont accédés par l’intermédiaire de Zuul :

## Diagramme global d’interactions entre services :

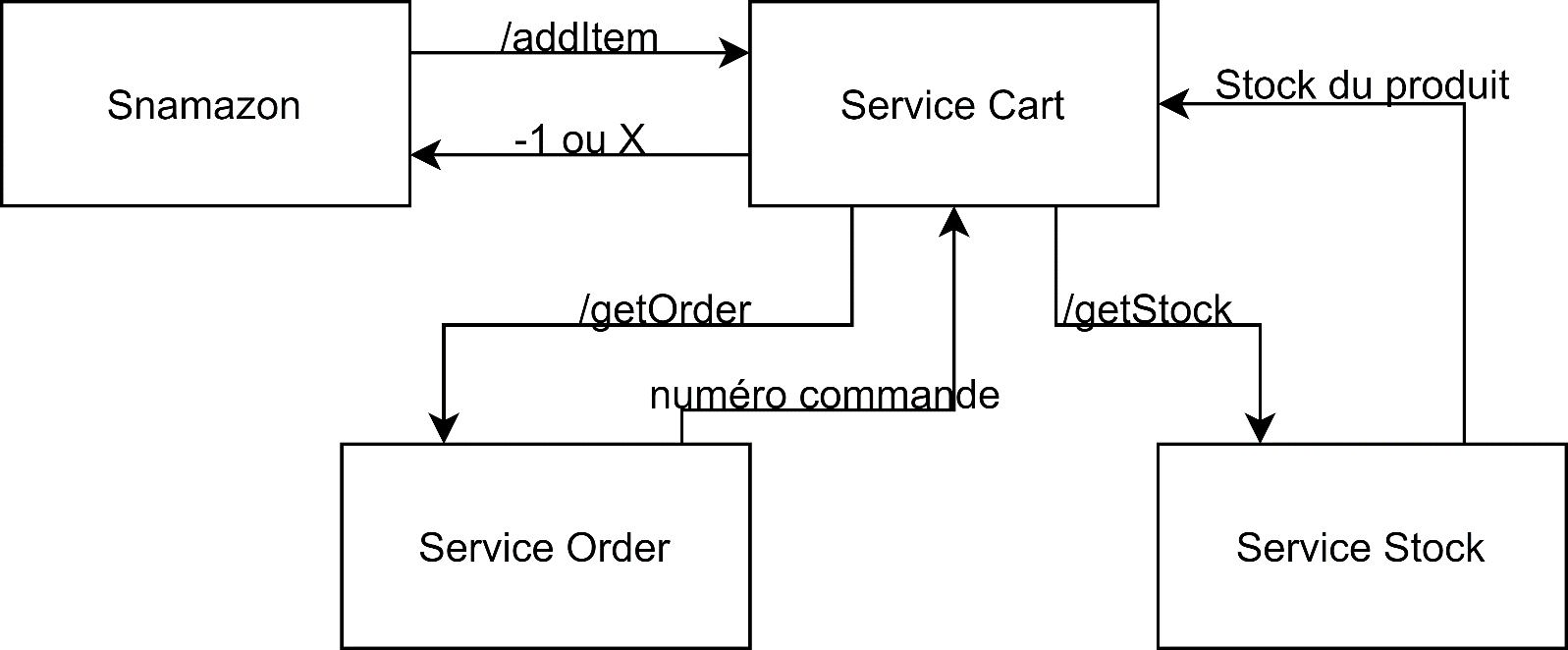


## Exemple pour l’ajout d’un article dans le panier d’un utilisateur connecté :

D’abord l’application récupère les informations sur le produit voulu via le Service Product puis contacte le Service Cart en lui donnant l’id de l’utilisateur, son mot de passe, le produit concerné et la quantité voulue.

De son côté le Service Cart contacte le Service Order pour connaître le numéro de commande sur lequel ajouter l’article (le Service Order renvoie le numéro de commande ayant un statut ‘non payé’ correspondant au client) puis contacte le Service Stock pour savoir si le produit a toujours assez de stock pour satisfaire le client.

Une fois cela fait, le Service Cart renvoie -1 à l’application de base si l’opération a réussi ou le stock restant du produit si elle a échoué.



# 3) Relevé des problèmes rencontrés

## Problème 1 :

Il me fallait passer des données depuis le code java vers la page HTML, par exemple pour afficher les produits, le panier, des erreurs custom, etc.

Ne sachant pas comment faire j’ai cherché sur le net et suis tombé sur plusieurs lien qui parlaient de « Thymeleaf »  [comme par exemple sur stackoverflow](https://stackoverflow.com/questions/50485083/passing-data-into-html-from-java-via-spring).

Après documentation sur notamment [baeldung](https://www.baeldung.com/thymeleaf-in-spring-mvc) et surtout [le site de thymeleaf](https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html) j’ai pu implémenter cette fonctionnalité à mon application :

J’utilise cette technologie dans plus ou moins toutes mes pages, voici comme exemple un extrait de la page du panier :

<th:block>  
<div th:if="${customError != null}">  
 <br><div class="alert alert-danger text-center" role="alert"> <strong>Erreur!</strong> <th:block th:text="${customError}"></th:block> </div></br>  
</div>  
</th:block>

Il suffit d’ajouter un objet « customError » à l’objet « ModelAndView » pour qu’elle s’affiche sur la page.

Exemple quand un client essaye de commander un produit dont il n’y a pas assez de stock :

mav.addObject("customError", "Plus que " + stock + " " + baseProduct.get("nom") + " de stock");

## Problème 2 :

Je ne comprenais pas comment fonctionnait les principes d’API gateway ou de load balancing en utilisant Zuul et Eureka mais [cette video](https://youtu.be/no-m5JK-gGc) à elle seule m’a permis de les comprendre et de les implémenter directement dans mon projet.

Maintenant, toutes les API sont connues par Eureka et accédées par Zuul.

Le path vers la gateway qu’est Zuul est renseigné dans application.properties :

zuul.path = http://localhost:8762/

Cette propriété est utilisée dans le code de la façon suivante :

@Value("${zuul.path}")  
private String zuulPath;

…

postRequest(zuulPath + "nom-du-service" + "/uriVoulue", jsonString);

Pour qu’un service soit connu de Zuul il doit donc s’enregistrer auprès de Eureka :

@SpringBootApplication  
@EnableDiscoveryClient  
public class MonService {…}

Et posséder un nom explicite dans application.properties pour pouvoir être contacté via Zuul :

spring.application.name=MonService

## Problème 3 :

Il me fallait pouvoir garder des informations notamment sur l’utilisateur connecté mais aussi par exemple sur le panier que remplirait un utilisateur non connecté.

Pour se faire je me suis rapidement penché sur le concept de session dans Spring Boot via, entre autres, ces documentations :

[https://docs.spring.io/spring-session/docs/2.2.x/reference/html/httpsession.html](https://docs.spring.io/spring-session/docs/2.2.x/reference/html/httpsession.html%20)

[https://docs.spring.io/spring-session/docs/current/reference/html5/](https://docs.spring.io/spring-session/docs/current/reference/html5/%20)

[https://docs.spring.io/spring-session/docs/current/reference/html5/guides/boot-jdbc.html](https://docs.spring.io/spring-session/docs/current/reference/html5/guides/boot-jdbc.html%20)

[https://examples.javacodegeeks.com/spring-boot-session-management/](https://examples.javacodegeeks.com/spring-boot-session-management/%20)

Les sessions ne se perde pas car elles sont stockées dans une base de donnée :

spring.datasource.url=jdbc:mysql://localhost:3306/snamazon  
spring.datasource.username=anciaux  
spring.datasource.password=f52db3e20  
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.session.store-type=*jdbc*spring.session.jdbc.initialize-schema=*always*

Il suffit de passer l’objet HttpSession la ou on veut l’utiliser et dans une requête il suffit de la mettre un paramètre pour pouvoir l’utiliser :

@GetMapping(value = "Cart")  
public ModelAndView cart(HttpSession session){…}

Il fût très simple de stocker le panier dans la session puisqu’on peut y ajouter n’importe quel Object :

ArrayList<Map<String, Object>> panier = (ArrayList<Map<String, Object>>)session.getAttribute("panier");

session.setAttribute("panier", panier);

# 4) Scénario de démonstration

## Démonstrations :

-Démonstration globale de l’application.

- Ajout et la suppression d’articles dans le panier d’un utilisateur connecté ou non.

Quand un utilisateur non connecté se connecte tous les articles du panier (qui étaient donc stockés dans la session) s’ajoutent à la base de données.

-Checkout : pour checkout il faut être connecté, avoir choisi un forfait de livraison, avoir un montant suffisant sur son compte et que tous les produits soient de stock.

-Si deux personnes ajoutent le dernier article de stock, c’est le premier à passer au checkout qui l’obtient, l’autre aura une erreur (la navigation privée permet de créer une nouvelle session).

-Quand un checkout est effectué, on vérifie que tous les produit sont toujours de stock, si ce n’est pas le cas, par défaut on en recommande 10. Dans la démonstration on voit que les fournisseurs ont été contacté pour le rachat 10 guéridons. Ikea en vendait 4 à 26€ et a donc été favorisé puisque moins cher que vandenborre qui en vendait 15 mais à 36€. Pour compléter les 10, on en commande quand même 6 chez vandenborre.

## Vidéo :

[Vidéo de démonstration](https://youtu.be/rB1F-FCKMbA)

# 5) Solution à une situation exceptionnelle

## Une instance d’un service tombe down :

Plusieurs instances d’un même service sont up, ce service est connu par Eureka et figure donc dans les routes connues par Zuul.

Par défaut Zuul fait du load balancing entre les deux instances du service, alors si à un moment l’une tombe down, Zuul forwardera simplement toutes les requêtes à l’autre jusqu’à ce que la première soit de nouveau up sans que le client ne remarque quoique ce soit (voir schéma pt.2).