|  |  |
| --- | --- |
| Systemes distibues | Résumé  Dossier écrit concernant le laboratoire de systèmes distribués.  Khamana bantu benedict & samuel hendrick  Professeur: Madani Mounawar |

Table des matières

[Tableau de correspondance service – technologie employée. 1](#_Toc61447267)

[Diagramme global représentant la solution d'un point de vue statique. 1](#_Toc61447268)

[Relevé de challenges rencontrés lors de l'implémentation de la solution. Décrivez chaque problématique et son contexte, mentionnez les ressources consultées et comment vous avez exploité ces ressources dans l'implémentation de l'application. 1](#_Toc61447269)

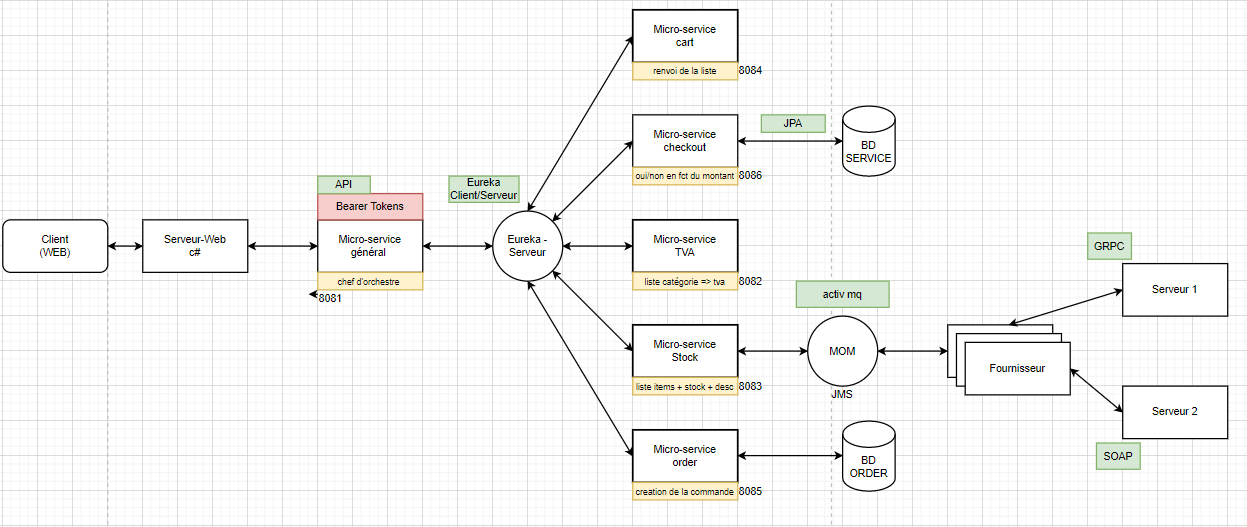
[Scénario de démonstration 2](#_Toc61447270)

[Solution apportée à une situation exceptionnelle que le système pourrait rencontrer. 2](#_Toc61447271)

## Tableau de correspondance service – technologie employée.

|  |  |
| --- | --- |
| Serveur-Web | ASP.Net |
| Service général | Spring boot, Web Api,  Spring security, Eureka Client |
| Eureka serveur | Spring boot, Eureka server |
| Service cart | Spring boot, Web Api, Eureka Client |
| Service checkout | Spring boot, Web Api, JPA, Eureka Client |
| Service tva | Spring boot, Web Api, Eureka Client |
| Service stock | Spring boot, Web Api, Eureka Client, JMS (topic) |
| Service order | Spring boot, Web Api, Eureka Client, JPA |
| Fournisseur | JMS (GRPC – bonus, SOAP – bonus) |

## Diagramme global représentant la solution d'un point de vue statique.



## Relevé de challenges rencontrés lors de l'implémentation de la solution. Décrivez chaque problématique et son contexte, mentionnez les ressources consultées et comment vous avez exploité ces ressources dans l'implémentation de l'application.

* Challenge 1 : sécurité spring boot
  + Sachant que notre serveur Web est fait en c# et non en spring boot avec ‘Thymeleaf’, nous ne pouvions pas nous permettre d’utiliser la fonction de login automatique prévue avec. Nous avons donc opté pour un token que l’application génère 1x au démarrage et utilise lors de ses requetés. C’est donc à lui de gérer les clients connectés ou les visiteurs de façon sécurisée.

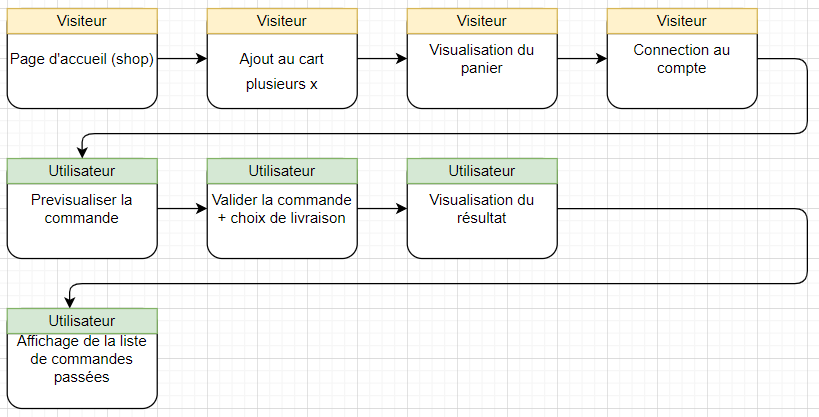
<https://www.freecodecamp.org/news/how-to-setup-jwt-authorization-and-authentication-in-spring/>

<https://auth0.com/blog/implementing-jwt-authentication-on-spring-boot/>

Les deux premiers articles ont été utiliser pour comparer la sécurité vue au cours avec JWT qui nous permet d’utiliser et d’envoyer les userDetails dans un token.

* Challenge 2 : Compréhension des termes @Component, @Config, @Bean, @Autowired
  + Afin d’utiliser l’injection de dépendance à son plein potentiel, en plus du cours nous avons voulu comprendre les termes précédents. Donc en plus de la documentation officielle etant assez vaste, nous avons cherchés des blogs/tutoriels plus détaillés.
* Component : <http://zetcode.com/springboot/component/>
* Config : <https://www.baeldung.com/spring-boot-annotations>
* AutoWired : <https://www.baeldung.com/spring-autowire>
* Challenge 3 : JMS : La spécification JMS fut assez simple à mettre en place dans un projet Spring. Néanmoins, cette facilité peut nuire à la compréhension. Pour remédier à cela, nous avons décidé de réaliser notre micro-service fournisseur en java classique supplanté de Maven pour une gestion plus aisée des dépendances. Cette prise de position nous a aidé à comprendre ce qui se cache derrière le jmsTemplate en Spring et donc percevoir une meilleure approche.

## Scénario de démonstration



## Solution apportée à une situation exceptionnelle que le système pourrait rencontrer.

* Load balancing et down detection grâce au serveur eureka sur les services stateless (tva)
* Verification pour empecher de valider une commande déjà validée