Devops

Groupe 1

2019-2020

Table des matières

[I. DevOps : une méthodologie 3](#_Toc30345972)

[1. Introduction 3](#_Toc30345973)

[2. CALMS 4](#_Toc30345974)

[1. CULTURE 4](#_Toc30345975)

[2. AUTOMATISATION 4](#_Toc30345976)

[3. LEAN 5](#_Toc30345977)

[4. MESURE 5](#_Toc30345978)

[5. SHARING 5](#_Toc30345979)

[3. SLI, SLO, SLA : la Base du DevOps 6](#_Toc30345980)

[II. Développement 8](#_Toc30345981)

[1. Le pôle développement 8](#_Toc30345982)

[2. Les technologies utilisées 8](#_Toc30345983)

[1. Le Back-end et le Front-end 8](#_Toc30345984)

[3. Récapitulatif 9](#_Toc30345985)

[4. La base de données 10](#_Toc30345986)

[5. Les fonctionnalités à réaliser 11](#_Toc30345987)

[6. Les routes du projet 12](#_Toc30345988)

[7. Répartitions des tâches 13](#_Toc30345989)

[8. Répartitions des tâches -Réalisée 13](#_Toc30345990)

[9. Vérification des routes 14](#_Toc30345991)

[10. Configuration de l’API 15](#_Toc30345992)

[11. Configuration de l’IHM 15](#_Toc30345993)

[III. Ops 16](#_Toc30345994)

[1. Installation de Docker CE 16](#_Toc30345995)

[2. Installation de NGINX 18](#_Toc30345996)

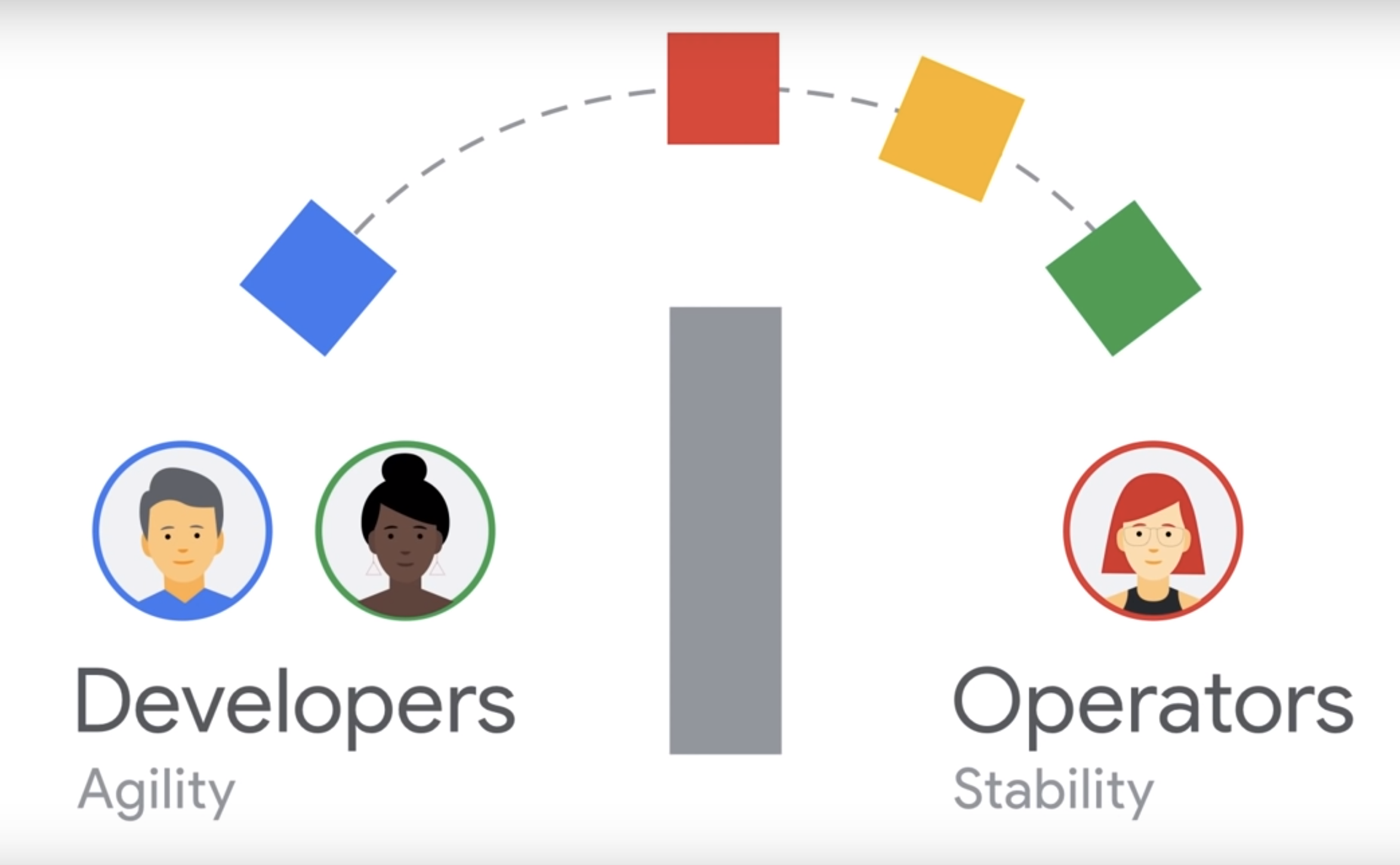
# DevOps : une méthodologie

## Introduction

Basically, Enterprise IT has "siloed" the Dev and OPS aspects of the applications, placing the respective responsibilities in separate teams. This created what is known as the wall of confusion. This wall arose because the two teams have conflicting goals.

The first one, the development team, has the main objective to make the application evolve, by adding new functionalities or fixing bugs, as quickly as possible and this, in a shorter time and at a lower cost, often to the detriment of quality when scheduling constraints arrive.

The second team, the OPS team, aims to keep the application in operational conditions, guaranteeing the stability and quality of the application, often sacrificing the cost and time needed to deploy a new version.



/

DevOps is therefore a set of practices that emphasizes the automation of processes between the development teams and the teams in charge of maintaining the developed application in operational conditions.

/

These practices then allow applications to be developed, tested and delivered more quickly and reliably.

The fundamental element of DevOps is speed: faster software delivery, faster problem resolution, faster innovation.

When we reach the speed, our business needs to seize new opportunities, we have reached digital velocity.

Data is the key to DevOps' success because it helps to :

- Measure and monitor DevOps performance

- Provide instant feedback that allows everyone to focus on the task at hand.

- Optimize software delivery, performance and business results

One of the main problems with the DevOps methodology is the lack of understanding of the work.

The list of advantages brought by DevOps is therefore long:

- Gaining trust between teams

- Acceleration of deliveries and deployments

- Faster ticket resolution

- More efficient management of unplanned tasks...

## CALMS

DevOps is often associated with the acronym CALMS, which stands for the 5 pillars of this methodology, which are :

- Culture

- Automation

- Lean

- Measure

- Share

We will therefore detail them before learning more about how they work:

### CULTURE

DevOps solves, first and foremost, human problems, communication problems and problems of responsibility between teams. In this sense, DevOps is close to agility, but includes other teams such as operations, testers, designers, developers, project managers, and generally anyone whose skills are required to deliver a quality product.

### AUTOMATION

Les applications sont déployées plus souvent. Cela amène à être rassuré sur la procédure de déploiement. Généralement, les entreprises sont amenées à ne déployer que quelques fois dans l'année, alors que des entreprises comme Google, Amazon et Facebook, qui sont souvent prises en exemple sur l'automatisation, peuvent déployer jusqu'à 17 fois par minute en production.

À cause de ce fameux mur de la confusion, et du manque d'interaction entre les équipes de Dev et les OPS, les paramètres de configuration sont souvent envoyés par mail, ou écrits dans le document d'exploitation. Il est alors facile de se tromper lors du déploiement d'une application, en ayant oublié une étape ou l'exécution d'un script, et de se retrouver avec un système fonctionnant mal ou ne fonctionnant pas du tout. De plus, l'erreur étant humaine, il est souvent difficile de bien paramétrer l'application en écrivant les bons paramètres de configuration.

Tous ces problèmes conduisent à la situation citée précédemment, c'est-à-dire à ne déployer que quelques fois dans l'année, de peur de ne plus voir le système fonctionner.

Les mises en production se font aussi, souvent, en dehors des plages d'heures ouvrées, ou le week-end, pour éviter d'impacter les utilisateurs. Grâce à l'automatisation, ce déploiement peut se faire durant les heures ouvrées, réduisant le stress des équipes d’OPS.

### LEAN

Le lean sert à qualifier une gestion des ressources sans gaspillage.

Dans le contexte du DevOps, Lean va alors s'intéresser à délivrer de la valeur ajoutée au client final (dans le cadre d'une application grand public, le public), tout en minimisant les processus longs, coûteux, sans valeur ajoutée. Dans un sens, le Lean de DevOps se rapproche de l'agilité, avec des concepts d'amélioration continue et d'acceptation des erreurs.

### MESURE

Comme toute transformation d’entreprise, il est nécessaire d’avoir des indicateurs de performance clés (KPI ou Key Performance Indicator) afin de savoir si les efforts de transformation d’amélioration continue transforment quelque chose.

Les différents indicateurs utilisés peuvent être :

* Combien de temps la nouvelle fonctionnalité a pris pour passer du développement à la production ?
* Combien de fois un bug récurrent apparaît ?
* Combien de personnes utilisent le produit en temps réel ?
* Combien d'utilisateurs a-t-on gagnés ou perdus en une semaine ?

### SHARING

La réussite de la culture DevOps réside dans le fait que les développeurs et les opérations se parlent, communiquent, et partagent les succès mais aussi les échecs.

## SLI, SLO, SLA : la Base du DevOps

SLIs, SLOs and SLAs aren’t just useful abstractions. Without them you cannot know if your system is reliable, available, or even useful. If they don’t tie explicitly back to your business objectives then you have no idea if the choices you make are helping or hurting your business. You also can’t make honest promises to your customers.

If you’re building a system from scratch, make sure that SLIs, SLOs and SLAs are part of your system requirements. If you already have a production system but don’t have them clearly defined then that’s your highest priority work.

Une image contenant capture d’écran

Description générée automatiquement

SLI sont des indicateurs ou des mesures de niveau de service qui informent sur l’état d’un service.

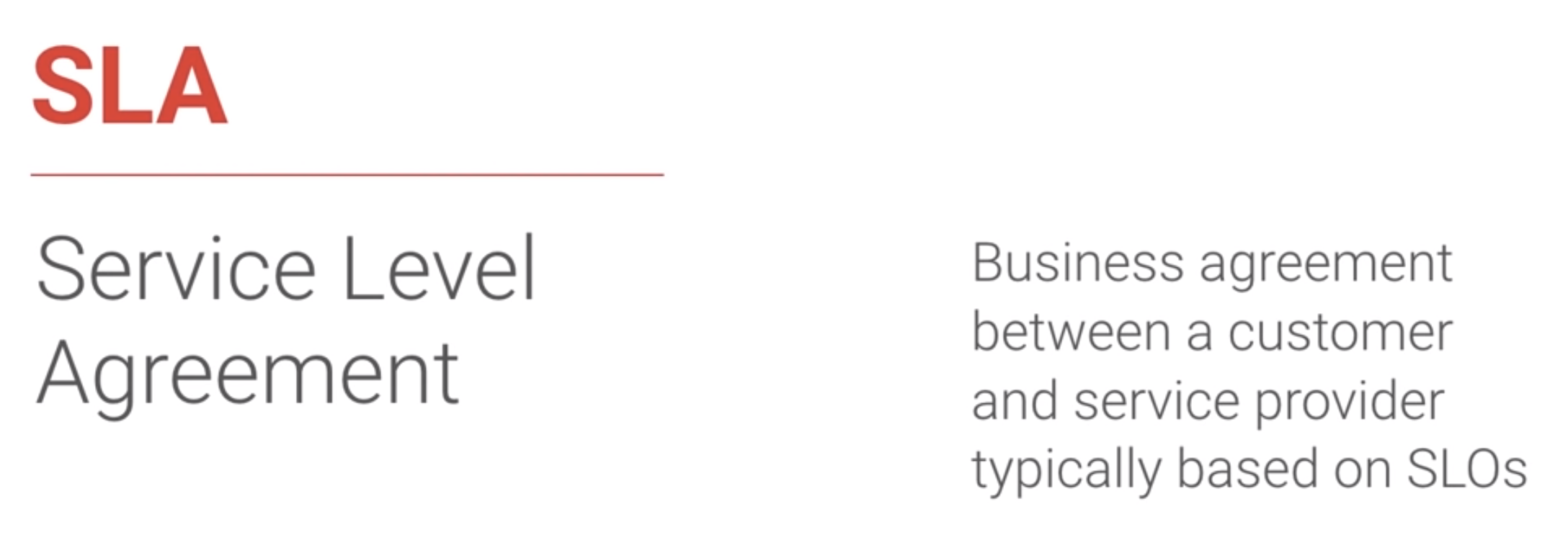
* If you want to have a reliable service, you must first define “reliability.” In most cases that actually translates to availability.
* If you want to know how reliable your service is, you must be able to measure the rates of successful and unsuccessful queries; these will form the basis of your SLIs.

Une image contenant capture d’écran

Description générée automatiquement

SLO sont des objectifs de niveau de service qui sont des limites convenues pour la fréquence à laquelle ces SLI doivent être respectées.

* The more reliable the service, the more it costs to operate. Define the lowest level of reliability that you can get away with, and state that as your Service Level Objective (SLO).
* Without an SLO, your team and your stakeholders cannot make principled judgements about whether your service needs to be made more reliable (increasing cost and slowing development) or less reliable (allowing greater velocity of development).



SLA sont des accords signés au niveau des entreprises qui définissent la disponibilité d’un service pour un client et les sanctions en cas d’échec pour fournir cette disponibilité.



SLI, SLO and SLA répondent donc parfaitement au principe même du DevOps selon lequel la mesure est critique, la façon la plus simple de décomposer les barrières organisationnelles est d’avoir un langage commun sur ce que signifie être disponible. Les SLI permettent donc de donner une valeur numérique très bien définie de ce à quoi cela correspond, avec les SLO l’idée est de collaborer entre le produit, les propriétaires et les SRE afin de s’assurer que le service fonctionne à un niveau approprié de fiabilité pour les clients.

On peut donc dire que la classe SRE implémente DevOps.

# Développement

## Le pôle développement

Le pôle de développement se compose de trois personnes issues de la même formation. Nous disposons de deux développeurs web et d’un développeur logiciel :

* Lorenzo LOGOS – Développeur WEB
* Nolwenn MYRAN – Développeur LOGICIEL
* Stacy PERALES – Développeur WEB

Une image contenant photo, objet

Description générée automatiquement

Chacune de ces personnes aura un rôle très précis dans la réalisation du projet et devront communiquer entre eux pour limiter les conflits et mener à bien le projet. Ils s’entraideront lorsqu’une personne rencontre une difficulté.

## Les technologies utilisées

### Le Back-end et le Front-end

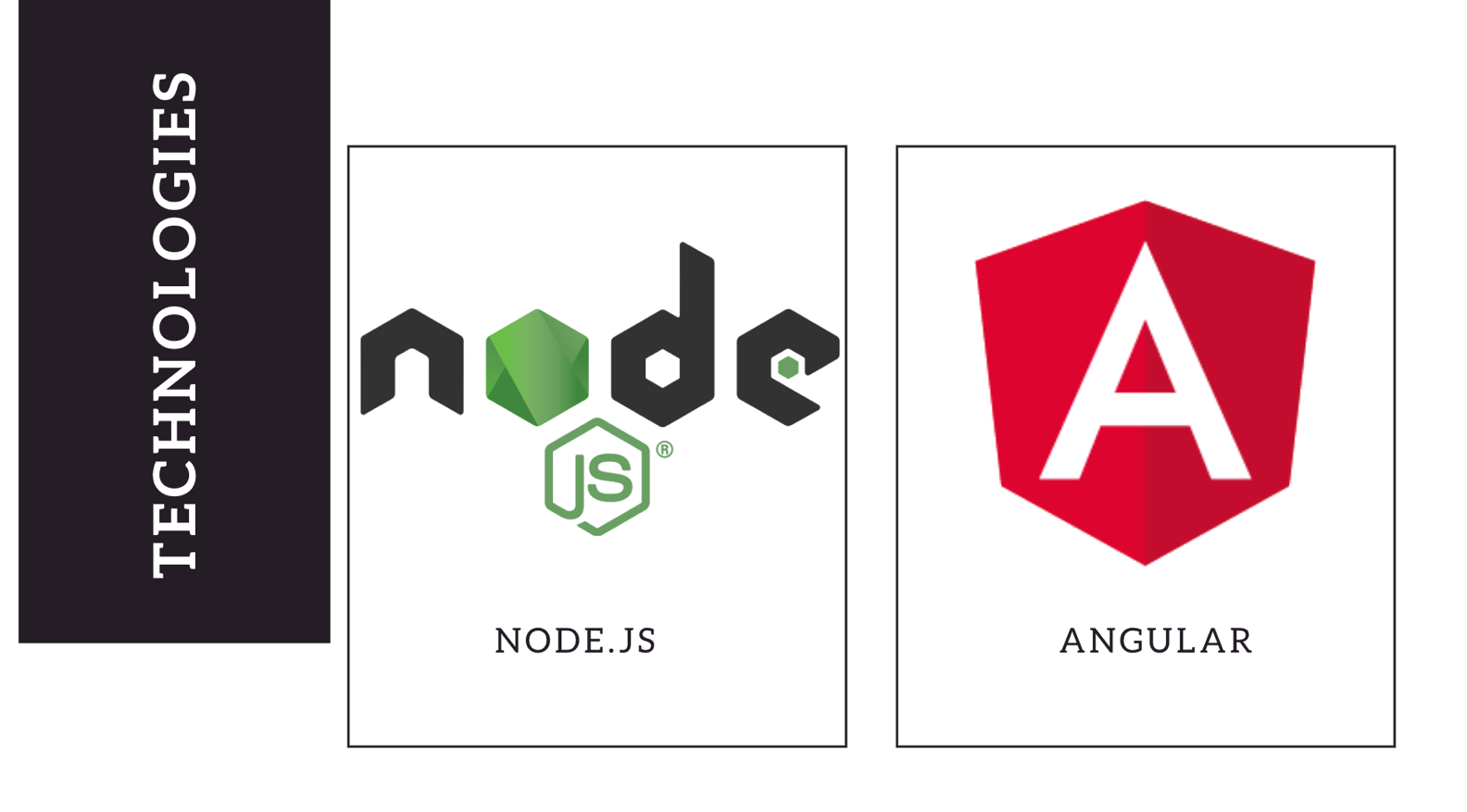
#### Le Back-end

Pour réaliser notre API nous avons dû choisir les technologies à utiliser. Après avoir discuté des technologies possibles pour notre API nous avons opté pour NodeJS et express qui nous paraissaient très intéressant à utiliser pour un projet comme celui-ci. De plus utilisé le NodeJs dans ce projet nous permettra d’en apprendre davantage sur cette technologie mais aussi de voir à quel point elle peut être performante.

#### Le Front-end

Pour permettre à un utilisateur de pouvoir créer son armure virtuelle il nous faut donc penser à réaliser une interface homme-machine. Pour le Front-end du projet nous avons décidé de choisir le Framework Angular qui nous permettra de mettre en place une interface responsive et optimisée. De plus la communauté de cette technologie étant très importante cela pourra nous aider dans la réalisation du projet. Pour rendre le projet encore plus présentable il nous suffira simplement de mettre en place notre propre CSS.

## Récapitulatif



## La base de données

Il faudra pouvoir stocker nos pièces pour pouvoir les afficher ou bien les modifier.

Pour stocker nos armures et nos pièces nous avions deux possibilités :

* Stocker les éléments dans un tableau
* Stocker les éléments dans une base de données

Nous avons décidé de choisir l’option de la base de données plutôt que celle du tableau, la base de données permettant d’organiser proprement et de rendre la consultation plus facile. Le choix de la base de données a été choisi en fonction de la compatibilité avec notre API. La base de données MongoDB nous paraissait intéressante puisque celle-ci s’associe très bien avec notre API puisqu’elle permet de récupérer nos fichiers Json sans problème.

Une image contenant capture d’écran

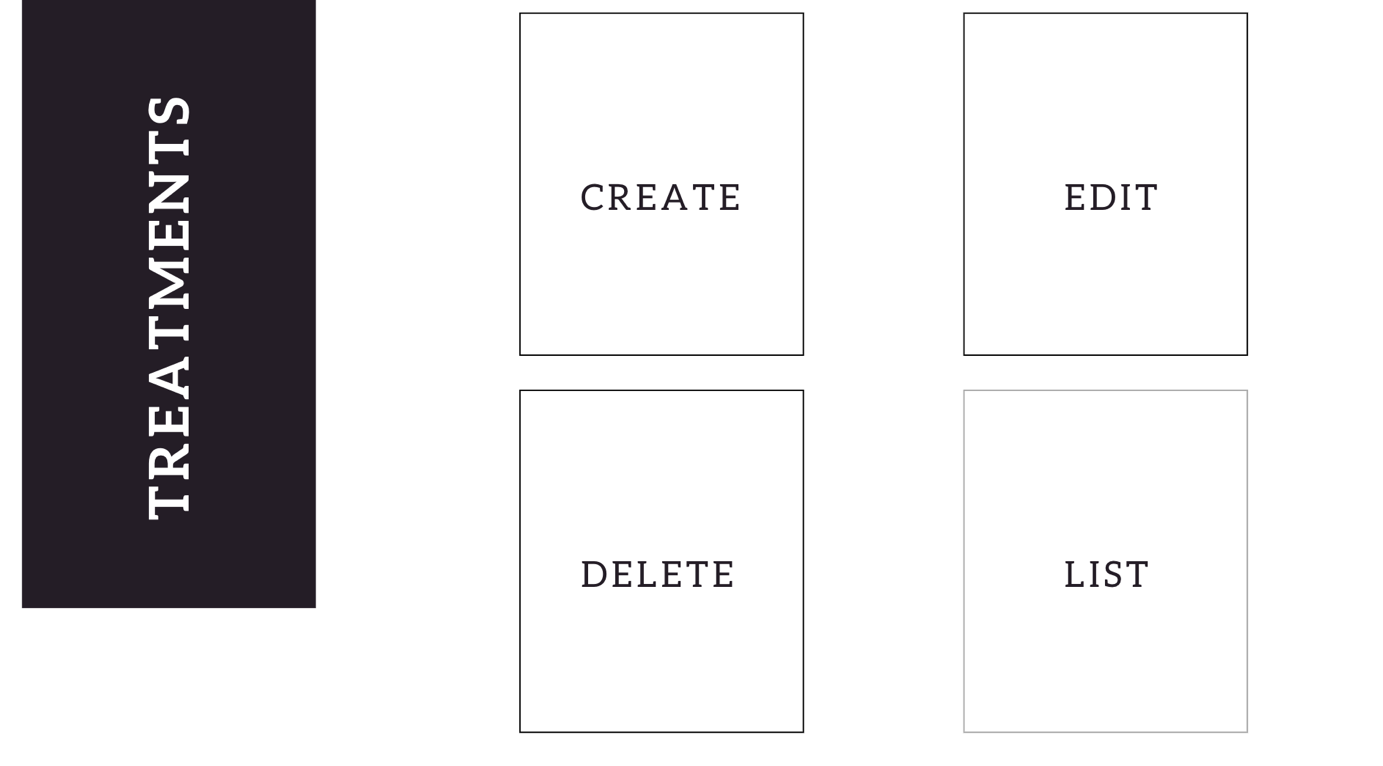
Description générée automatiquement

## Les fonctionnalités à réaliser

La consigne du projet étant de réaliser une armure virtuelle nous allons devoir mettre en place des fonctionnalités permettant la réalisation, la modification ou la suppression de l’armure, mais aussi de chacun de ses composants si nécessaire. C’est pour cela que nous avons réfléchis à un moyen de pouvoir, pour chaque partie de l’armure proposer les fonctionnalités suivantes :

* Lister : si l’on veut voir toute nos pièces
* Créer : Pour créer de nouvelles pièces
* Modifier : Si la pièce ne nous convient pas
* Supprimer : Si l’on ne veut plus de la pièce

Tous les composants devront avoir les quatre fonctionnalités.



## Les routes du projet

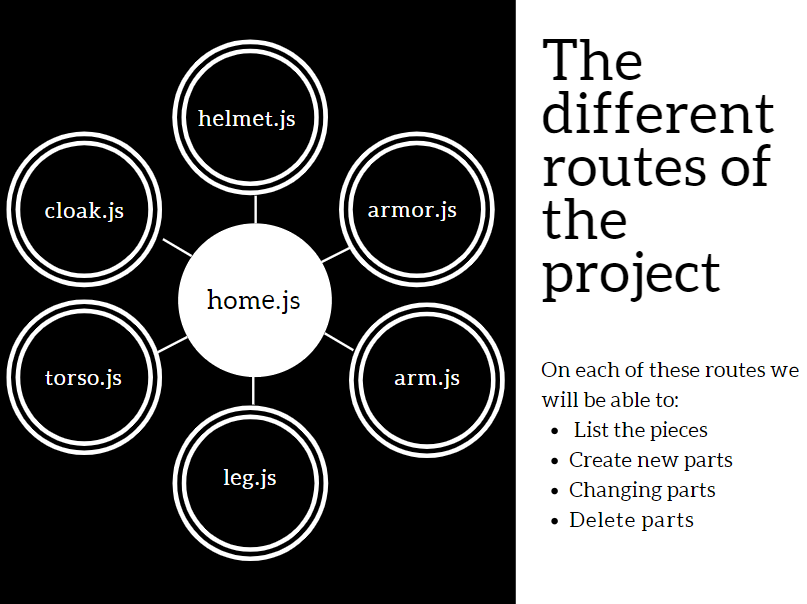
Chaque partie de l’armure sera gérer dans une page différente, de ce fait il va falloir penser à réaliser autant de route que de partie.

Nous allons donc mettre en place six routes distinctes :

* /  : qui regroupera l’armure complète avec toutes les pièces
* /casque : qui contiendra les possibilités pour le casque
* /bras : qui contiendra les possibilités pour les bras
* /torse : qui contiendra les possibilités pour le torse
* /jambe : qui contiendra les possibilités pour les jambes
* /cape : qui contiendra les possibilités pour les capes.

Ces routes nous permettrons d’avoir des pages pour une et une seule partie, cela permettra une meilleure vision, une meilleure organisation et surtout des routes claires. Chacune des pages ne servira uniquement à un élément excepté la page contenant l’armure complète.

Pour pouvoir bien avoir en tête nos routes nous avons réalisé un schéma qui relie chacune des routes concernant les « parties » de l’armure à la page contenant l’armure complète :



Pour éviter que le projet soit codé de différente manière nous nous sommes mis d’accord sur le nommage. Nous en avons informé les autres pôles pour qu’ils soient au courant.

Cela permettra ainsi de ne pas se retrouver avec des différences de codes. Le but étant qu’à la lecture du code on ne remarque pas que plusieurs personnes travaillent sur le projet. Nous devons donner une harmonie au code pour qu’une personne lambda reprenant le projet ne se retrouve perdu en retrouvant des bouts de codes écrit différemment mais faisant la même chose.

## Répartitions des tâches

Le projet se composant de 6 routes et ayant 3 développeurs, nous avons opté pour une répartition des tâches de cette manière :

* Chaque personne du projet doit réaliser 2 routes. Ainsi nous aurons les six routes prévues.
* L’interface (Front-end) sera répartie entre deux personnes pendant que la troisième se chargera de vérifier les problèmes du côté de l’API.
* Les dernières retouches concernant l’API ou l’interface seront réalisées par tout le monde.
* La relecture du code sera faite par tout le pôle.

## Répartitions des tâches -Réalisée

La répartition des tâches a subi quelques modifications. En effet, après avoir commencé à travailler en se répartissant 2 routes chacun pour le backend nous avons changé de méthode et laissé une personne sur le backend qui a recommencé notre backend pour l’améliorer.

Nous avons donc par la suite suivi la répartition suivante :

* Une personne s’occupe de mettre ne place le Backend
* L’interface est répartie entre deux personnes
  + Chacune des personnes possède 3 composants à réaliser.
  + Si un problème survient une des personnes peut aller aider la seconde
  + Le CSS sera effectué par les deux personnes chacun ayant des parties différentes
  + Les tests sont effectués par les deux personnes
* La relecture du code du côté back ainsi que du côté front sera revu par l’ensemble du pôle développement
* Les dernières retouches pourront être réalisé par les trois personnes.

## Vérification des routes

Avant de réaliser la partie interface chaque route réalisée en NodeJS sera vérifiée.

Pour vérifier si nos routes fonctionnent il suffit d’utiliser POSTMAN. POSTMAN permet d’exécuter nos routes avec les requêtes http et de les stocker dans un historique. Ainsi nous pourrons tester les routes de suppression, de modification et d’insertion sans le moindre problème. Ce logiciel nous permettra donc d’avancer sans problème et surtout de détecter si une de nos routes ne fonctionnent pas correctement.

Une image contenant capture d’écran

Description générée automatiquement

## Configuration de l’API

Pour la mise en place de notre API nous avons décidé de créer un répertoire « routes » qui contient autant de fichier que de route mise en place pour le projet.

Un répertoire « models » lui contient les schémas de la base de données.

A la racine un fichier constants.js a été créé. Celui-ci contient le type des parties. Cela permettra d’ajouter le type à notre équipement grâce à la route utilisée. Par exemple : Nous sommes sur la route /torso, si on ajoute un élément celui-ci aura la constante TORSO et donc le type torse.

A la racine toujours nous pouvons trouver un.env. Ce fichier contient des données d'environnement et autres informations sensibles (comme l'URL de connexion à la base de données) qui ne doivent en aucun cas se trouver dans le code de notre projet. Ce fichier sera par conséquent le seul à ne pas être partagé sur le git.

## Configuration de l’IHM

Pour notre interface nous utilisons la technologie Angular comme expliqué précédemment. Angular fonctionne avec des composants. Pour que notre code soit bien plus facile à exploiter nous avons donc réalisé un composant par partie. Nous aurons donc les sept composants suivants :

* Helmet
* Torso
* Arms
* Legs
* Cloak
* Armor
* Home

Chaque composant possédera son propre code html, son propre CSS ainsi que son propre fichier component.ts

Pour pouvoir créer ou modifier il nous faut des formulaires, c’est pourquoi nous en retrouverons plusieurs dans ce projet.

Nous enverrons à notre API les informations grâce à des requêtes que nous effectuerons.

# Ops

La solution côté OPS sera présenté selon le plan suivant : dans un premier temps nous aurons la description détaillée des différents éléments qui composent le projet, de leur installation jusqu’à leur configuration (à l’image de docker ou encore Nginx par exemple pour le serveur). Par la suite, un script d’automatisation vous sera présenté permettant de mettre en place le projet sur une machine vierge sous Debian.

A noter que les docker files correspondants aux technologies utilisées pour le projet seront présent dans le Git relatif au Devoir.

|  |  |
| --- | --- |
| Outil | Explication |
| Docker-ce 18.09.0 | Permet la manipulation des conteneurs |
| Git 12.6 | Le but de ce projet est de fournir de la manière la plus simple possible des images dockers |
| Nginx | Serveur Web |
| Angular 8.0 | Environnement de développement |
| Node.JS 10. | Environnement de développement |

## Installation de Docker CE

Docker-CE est un logiciel libre qui permet la manipulation des conteneurs de logiciel. Docker n’embarque pas de systèmes d’exploitation, il utilise le noyau de l’hôte et ne fait qu’utiliser le strict minimum. Rendant ainsi la gestion des ressources de l’infrastructure optimale.

On vient alors installer les packages qui permettent à apt d’utiliser le HTTPS pour les repos :

Sudo apt-get install \  
Apt-transport-https \  
ca-certificates \  
curl \  
gnupg2 \  
software-properties-common

On vient alors ajouter la clé GPG de docker :

Curl -fsSL <https://download.docker.com/linux/debian/gpg> | sudo apt-key add -

On peut vérifier l’empreinte de la clé, cela n’est pas obligatoire mais il est préférable de le faire pour vérifier que nous avons le bon package.

$ sudo apt-key fingerprint 0EBFCD88  
  
pub 4096R/0EBFCD88 2017-02-22  
 Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88  
 Docker Release (CE deb) [docker@docker.com](mailto:docker@docker.com)  
sub 4096R/F273FCD8 2017-02-22

Ajout du repo edge afin d’avoir les dernières MAJ + stable pour avoir les versions stables :

$ sudo add-apt-repository \   
 “deb [arch=amd64] <https://download.docker.com/linux/debian> \  
 $(lsb\_release -cs) \  
 stable”

On installe Docker CE avec la dernière version :

Apt-get install docker-ce

Nous vérifions ensuite que Docker est correctement installé avec la commande :

Sudo docker run hello-world

page7image13216112

Docker est donc correctement installé !

## Installation de NGINX

[NGINX](https://www.nginx.com/), prononcé comme « engine-ex », est un serveur web open-source qui, depuis son succès initial en tant que serveur web, est maintenant aussi utilisé comme reverse proxy, cache HTTP, et load balancer.

On va donc dans un premier temps venir récupérer l’image Nginx :

docker pull nginx

Ensuite on lance notre container avec les containers avec les paramètres nécessaires.

docker run --detach \

--hostname devops-rp \

--publish 80:80 \

--name nginx-rp \

--restart always \

nginx:latest

Installation du container GitLab CE

docker pull gitlab/gitlab-ce

Lancement du container avec les paramètres nécessaires

docker run --detach \

--hostname gitlab.devops.com \

--name gitlab-ce \

--restart always \

--volume /srv/gitlab/config:/etc/gitlab \

--volume /srv/gitlab/logs:/var/log/gitlab \

--volume /srv/gitlab/data:/var/opt/gitlab \

gitlab/gitlab-ce:latest