Devops

Group 1

2019-2020

Table des matières

[I. DevOps : a methodology 3](#_Toc30350745)

[1. Introduction 3](#_Toc30350746)

[2. CALMS 4](#_Toc30350747)

[1. CULTURE 4](#_Toc30350748)

[2. AUTOMATION 4](#_Toc30350749)

[3. LEAN 5](#_Toc30350750)

[4. MEASURE 5](#_Toc30350751)

[5. SHARING 5](#_Toc30350752)

[3. SLI, SLO, SLA: The Basis of DevOps 5](#_Toc30350753)

[II. Development 8](#_Toc30350754)

[1. The development pole 8](#_Toc30350755)

[2. The technologies used 8](#_Toc30350756)

[1. Back-end / Front-end 8](#_Toc30350757)

[3. Summary 9](#_Toc30350758)

[4. Data B ase 10](#_Toc30350759)

[5. The functionalities to be achieved 11](#_Toc30350760)

[6. The Project roads 12](#_Toc30350761)

[7. Task sharing 13](#_Toc30350762)

[8. Task sharing - Completed 13](#_Toc30350763)

[9. Road check 14](#_Toc30350764)

[10. API Configuration 15](#_Toc30350765)

[11. API Configuration 15](#_Toc30350766)

[III. Ops 16](#_Toc30350767)

[1. Docker CE installation 16](#_Toc30350768)

[2. NGINX Installation 18](#_Toc30350769)

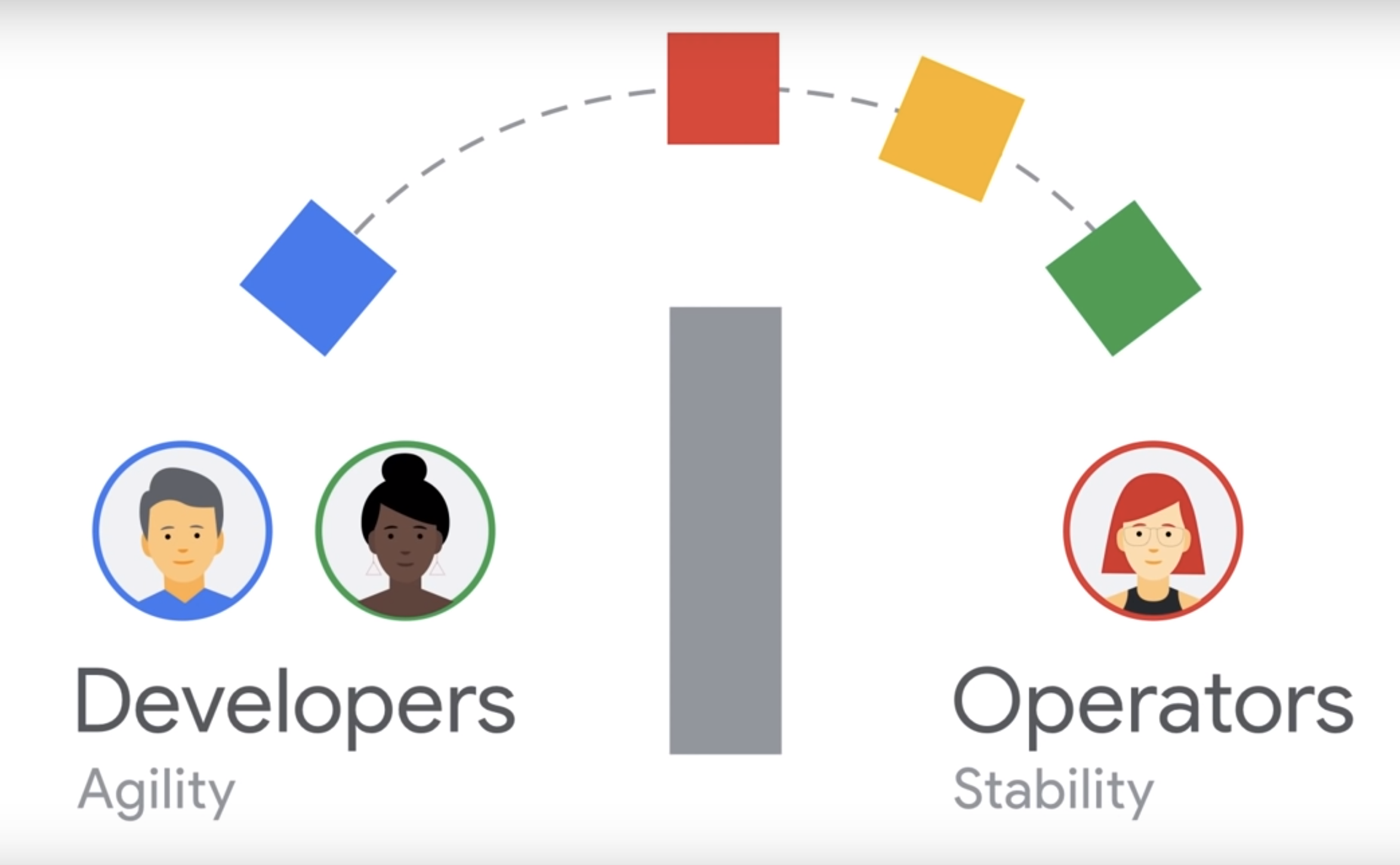
# DevOps : a methodology

## 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

Applications are deployed more often. This leads to reassurance about the deployment procedure. Typically, companies only deploy a few times a year, whereas companies such as Google, Amazon and Facebook, which are often taken as an example of automation, can deploy up to 17 times a minute in production.

Because of this famous wall of confusion, and the lack of interaction between Dev teams and OPS, configuration settings are often sent by email, or written in the operating document. It is then easy to make a mistake when deploying an application, having forgotten a step or executing a script, and end up with a system that doesn't work properly or at all. Moreover, the error being human, it is often difficult to set up the application properly by writing the right configuration parameters.

All these problems lead to the situation mentioned above, i.e. to deploy only a few times a year, for fear of not seeing the system working anymore.

Releases are also often done outside of business hours, or on weekends, to avoid impacting users. Thanks to automation, this deployment can take place during business hours, reducing stress for OPS teams.

### LEAN

Lean is used to qualify a waste-free management of resources.

In the context of DevOps, Lean will then focus on delivering added value to the end customer (in the context of a consumer application, the public), while minimizing long, costly, non-value-added processes. In a way, lean in DevOps is close to agility, with concepts of continuous improvement and acceptance of errors.

### MEASURE

As with any business transformation, it is necessary to have Key Performance Indicators (KPIs) to know if the continuous improvement transformation efforts are making a difference.

The different indicators used can be:

- How long did the new functionality take to go from development to production?

- How often does a recurring bug appear?

- How many people use the product in real time?

- How many users were gained or lost in a week?

### SHARING

The success of the DevOps culture lies in the fact that developers and operations talk to each other, communicate, and share successes as well as failures.

## SLI, SLO, SLA: The Basis of 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 are service level indicators or metrics that provide information on the status of a service.

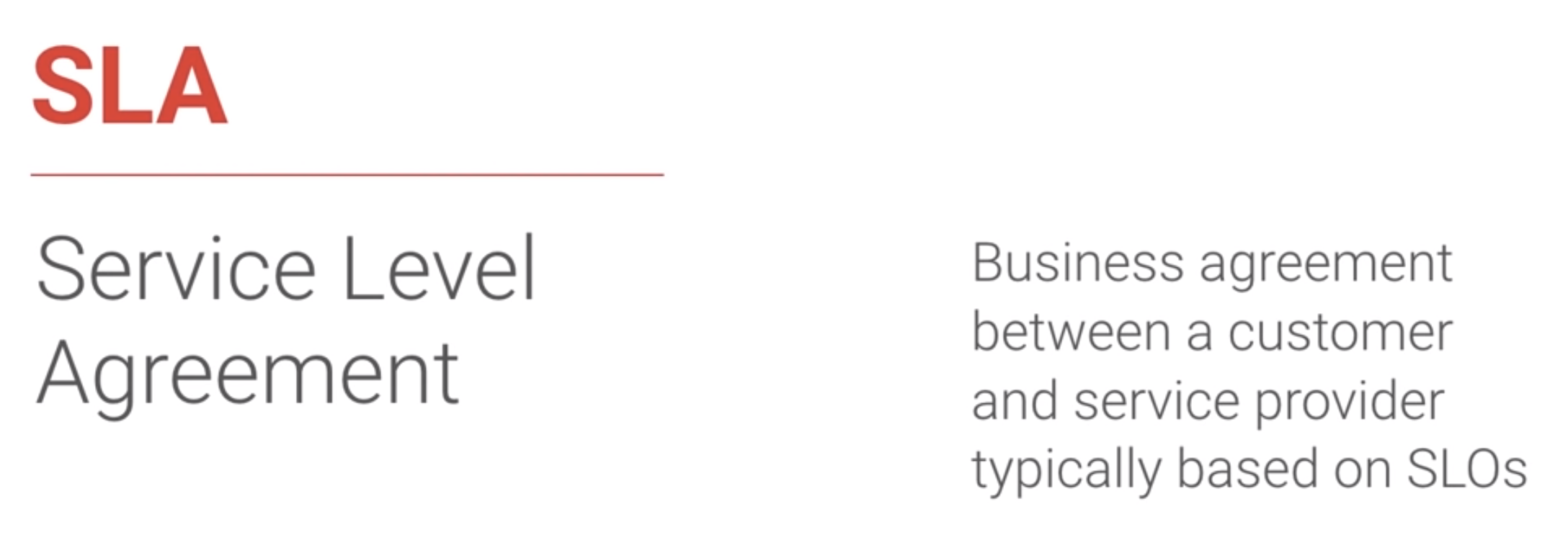
* If you want to have a reliable service, you must first define “reliability.” In most cases that actually translates to availability.
* If you want to know how reliable your service is, you must be able to measure the rates of successful and unsuccessful queries; these will form the basis of your SLIs.

Une image contenant capture d’écran

Description générée automatiquement

SLOs are service level objectives that are agreed-upon limits for the frequency with which these SLOs must be met.

* 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).



SLAs are company-level agreements that define the availability of a service to a customer and the penalties for failure to provide that availability.



SLI, SLO and SLA thus perfectly meet the very principle of DevOps that measurement is critical, the simplest way to break down organizational barriers is to have a common language about what it means to be available. SLOs therefore provide a very well-defined numerical value of what it means, with SLOs the idea is to collaborate between the product, the owners and the SREs to ensure that the service operates at an appropriate level of reliability for customers.

So we can say that the SRE class implements DevOps.

# Development

## The development pole

The development pole is made up of three people from the same background. We have two web developers and one software developer:

- Lorenzo LOGOS - WEB Developer

- Nolwenn MYRAN - SOFTWARE Developer

- Stacy PERALES - WEB Developer

Une image contenant photo, objet

Description générée automatiquement

Each of these people will have a very specific role in the project and will have to communicate with each other to limit conflicts and to carry out the project successfully. They will help each other when a person encounters a difficulty.

## The technologies used

### Back-end / Front-end

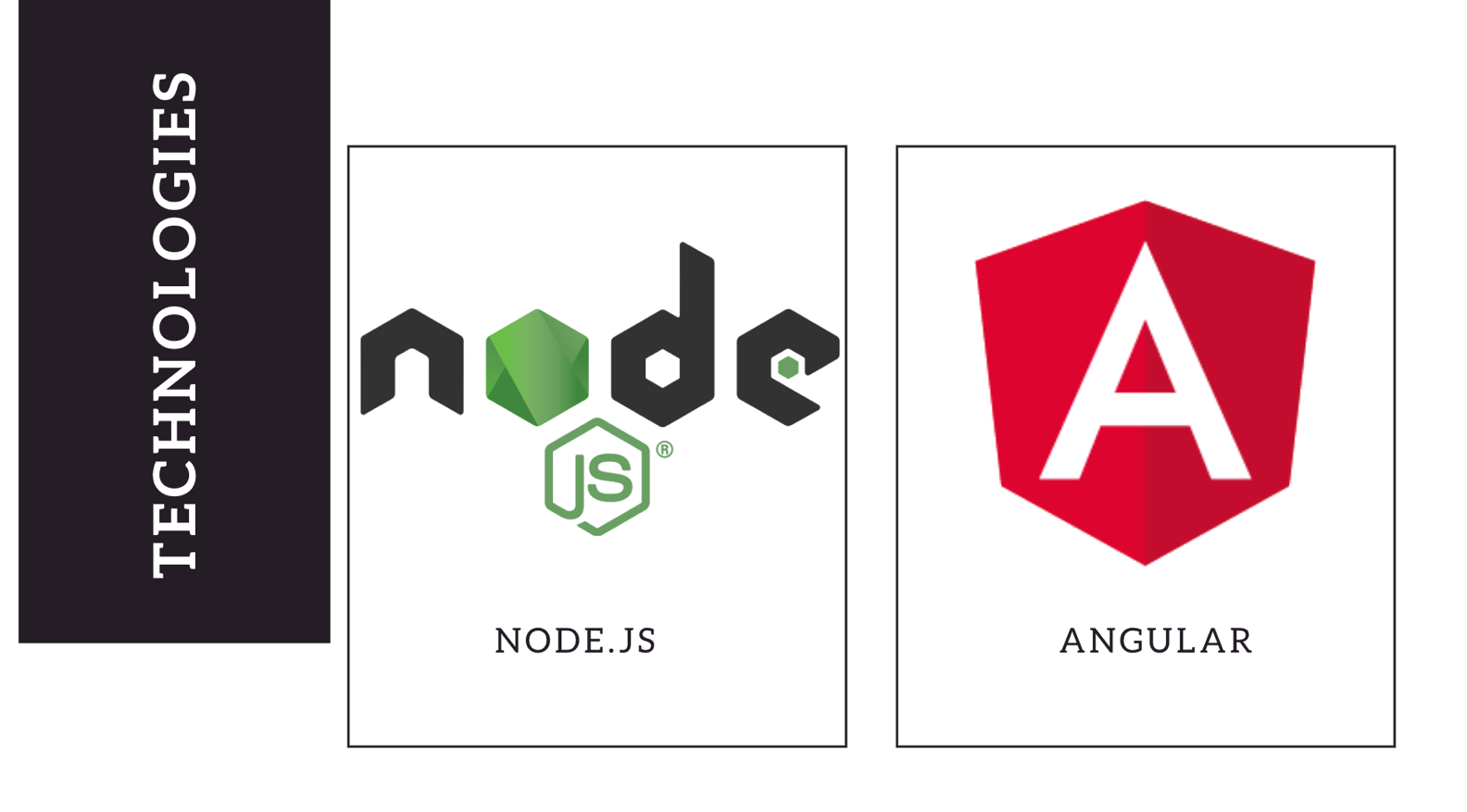
#### Back-end

To realize our API, we had to choose the technologies to use. After discussing the possible technologies for our API, we chose NodeJS and express which seemed very interesting to use for a project like this one. Moreover, using NodeJs in this project will allow us to learn more about this technology but also to see how powerful it can be.

#### Front-end

To allow a user to create his virtual armor we must therefore think about creating a man-machine interface. For the front-end of the project we decided to choose the Angular Framework which will allow us to set up a responsive and optimized interface. Moreover, the community of this technology being very important that will be able to help us in the realization of the project. To make the project even more presentable, we will just have to set up our own CSS.

## Summary



## Data B ase

It will be necessary to be able to store our parts to be able to display or modify them.

To store our armor and our pieces we had two possibilities:

- To store the elements in an array

- Store items in a database

We decided to choose the database option rather than the table option, as the database allows for clean organization and easier consultation. The choice of the database was made based on compatibility with our API. The MongoDB database seemed interesting to us since it associates very well with our API since it allows to retrieve our Json files without any problem.

Une image contenant capture d’écran

Description générée automatiquement

## The functionalities to be achieved

The instruction of the project being to realize a virtual armor we will have to set up functionalities allowing the realization, the modification or the suppression of the armor, but also of each of its components if necessary. That's why we thought about a way to be able to propose the following functionalities for each part of the armor:

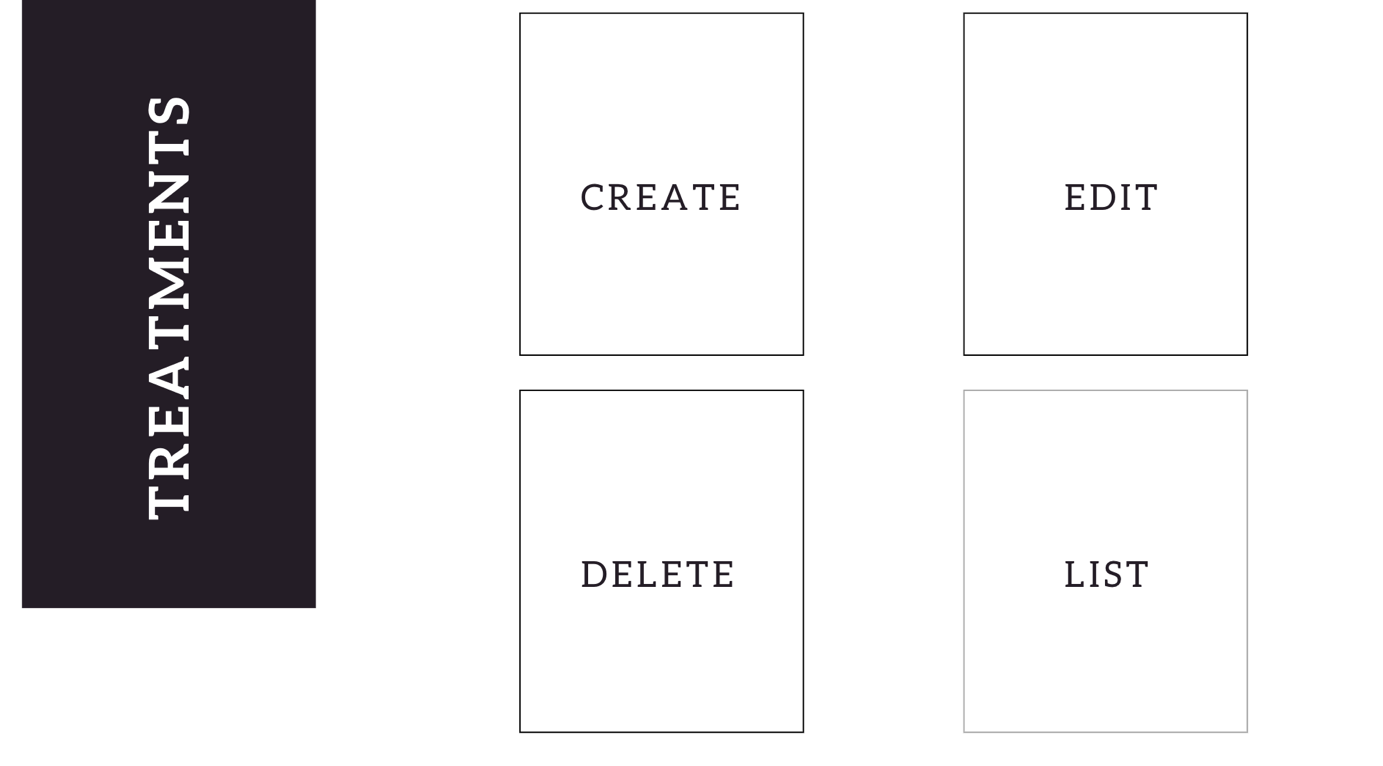
- Lister: if we want to see all our parts

- Create: To create new documents

- Modify: If the part does not suit us

- Delete: If you no longer want the part

All components must have all four functionalities.



## The Project roads

Each part of the armor will be managed in a different page, so you'll have to think about doing as many routes as you do games.

We will thus set up six distinct routes:

- /: which will group the complete armor with all the pieces...

- /helmet: which will contain the possibilities for the helmet

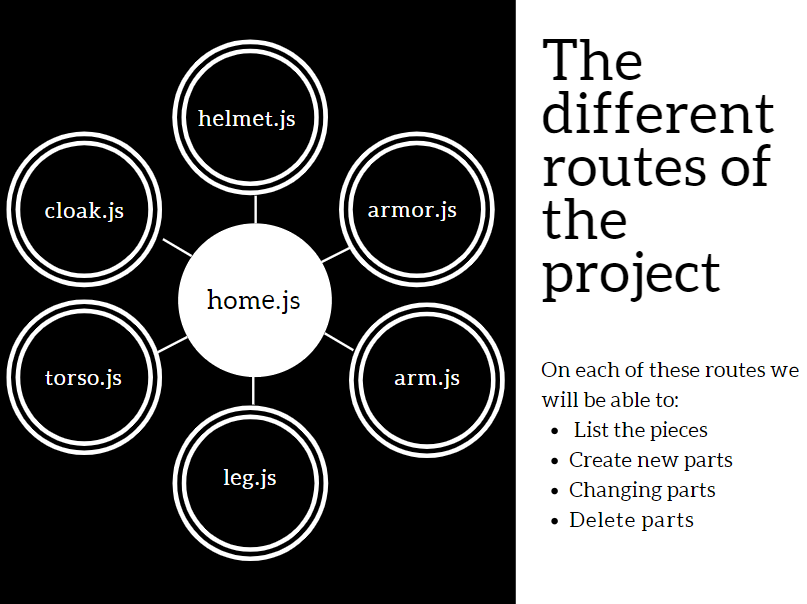
- /arm: which will contain the possibilities for the arms

- /torso: which will contain the possibilities for the torso

- /leg: which will contain the possibilities for the legs

- /cape: which will contain the possibilities for capes.

These routes will allow us to have pages for one and only one part, this will allow a better vision, a better organization and above all clear routes. Each of the pages will be used only for one element except the page containing the complete armor.

To be able to have well in mind our routes we realized a diagram which connects each of the routes concerning the "parts" of the armor to the page containing the complete armor: 

To avoid the project being coded in different ways we agreed on the naming. We have informed the other poles so that they are aware of it.

This will ensure that we don't end up with code differences. The goal is that when reading the code, one doesn't notice that several people are working on the project. We need to give harmony to the code so that an ordinary person taking over the project doesn't get lost in finding bits of code written differently but doing the same thing.

## Task sharing

As the project consists of 6 routes and has 3 developers, we opted for a division of tasks in this way:

- Each person in the project has to build 2 routes. This way we will have all six routes planned.

- The interface (front end) will be divided between two people while the third person will be in charge of checking the problems on the API side.

- The final touches concerning the API, or the interface will be done by everyone.

- The proofreading of the code will be done by the whole pole.

## Task sharing - Completed

The division of labor has undergone some changes. Indeed, after starting to work by dividing 2 routes each for the backend we changed our method and left one person on the backend who started again our backend to improve it.

We then followed the following distribution:

- One person is in charge of setting up the backend.

- The interface is divided between two people

o Each person has 3 components to realize.

o If a problem occurs one of the people can go and help the second one.

o The HSC will be carried out by the two persons each having different parts

o Tests are carried out by both persons

- The code will be proofread on the back side as well as on the front side by the entire development division.

- The final touch-ups can be done by the three people.

## Road check

Before the interface part is realized, each route realized in NodeJS will be checked.

To check if our routes are working just use POSTMAN. POSTMAN allows to execute our routes with http requests and to store them in a history. This way we can test delete, modify and insert routes without any problem. This software will allow us to move forward without any problem and specially to detect if one of our routes is not working correctly.

Une image contenant capture d’écran

Description générée automatiquement

## API Configuration

For the implementation of our API we decided to create a directory "routes" which contains as many files as the route set up for the project.

A "models" directory contains the database schemas.

At the root a constants.js file has been created. It contains the type of the parts. This will allow us to add the type to our equipment thanks to the route used. For example: We are on the route /torso, if we add an element this one will have the constant TORSO and thus the type torso.

At the root always we can find a. env. This file contains environment data and other sensitive information (like the URL to connect to the database) which must not be in our project code. This file will therefore be the only one not to be shared on the git.

## API Configuration

For our interface we use Angular technology as explained above. Angular works with components. To make our code much easier to use we have made one component per part. So, we will have the following seven components:

- Helmet

- Torso

- Arms

- Bequests

- Cloak

- Armor

- Home

Each component will have its own html code, its own CSS and its own component.ts file.

To be able to create or modify we need forms, that's why we will find several of them in this project.

We will send the information to our API through requests we will make.

# Ops

The solution on the OPS side will be presented according to the following plan: first we will have a detailed description of the different elements that make up the project, from their installation to their configuration (like docker or Nginx for example for the server). Afterwards, an automation script will be presented to you allowing you to set up the project on a blank machine under Debian.

Note that the docker files corresponding to the technologies used for the project will be present in the Duty Git.

|  |  |
| --- | --- |
| 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 |

## Docker CE installation

Docker-CE is an open source software that allows the handling of software containers. Docker does not embed operating systems; it uses the host kernel and only uses the bare minimum. This makes the management of infrastructure resources optimal.

We then come to install the packages that allow apt to use the HTTPS for resting:

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

We then add the GPG key of docker:

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

We can check the fingerprint of the key; this is not mandatory but it is better to do it to verify that we have the right 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

Adding repo edge to have the last updates + stable to have stable versions:

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

We install Docker CE with the latest version:

Apt-get install docker-ce

We then check that Docker is correctly installed with the command:

Sudo docker run hello-world

page7image13216112

Docker is therefore correctly installed!

## NGINX Installation

NGINX, pronounced as "engine-ex", is an open-source web server which, since its initial success as a web server, is now also used as a reverse proxy, HTTP cache, and load balancer.

So first we're going to come and get the Nginx image:

docker pull nginx

Then we launch our container with the containers with the necessary parameters.

docker run --detach \

--hostname devops-rp \

--publish 80:80 \

--name nginx-rp \

--restart always \

nginx:latest

Installing the GitLab CE container:

docker pull gitlab/gitlab-ce

Launching the container with the necessary parameters

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