JAUSEAU Baptiste P1509933

OLIVIE Maxime P1710179

Rapport de Projet

Table des matières

[I – Introduction et Choix 2](#_Toc28173424)

[A – Choix du sujet 2](#_Toc28173425)

[B – Présentation et support 2](#_Toc28173426)

[II – Développement 3](#_Toc28173427)

[A – Technologies 3](#_Toc28173428)

[B – Programmation asynchrone et fonctionnelle 3](#_Toc28173429)

# I – Introduction et Choix

## A – Choix du sujet

Nous avons choisi le sujet RC1 – Data-Mining. Les motivations de ce choix sont les suivantes :

* Actualité et omniprésence des données aujourd’hui. Ce projet pourrait être un point de départ, une introduction au traitement et à la représentation des données, domaine où nous n’avions pas ou peu de connaissances.
* Possibilité d’apprendre de nouvelles technologies du web tels que divers frameworks et également d’autres alternatives que celles abordées durant la licence en ce qui concerne la programmation côté serveur.
* Caractère « graphique » du sujet. C’est à notre sens beaucoup plus ludique et motivant, autant au niveau de la conception que du débogage, d’avoir une interface visuelle avec laquelle interagir lors de la réalisation d’un programme.

## B – Présentation et support

Nous avons d’abord sélectionné un jeu de données principal à représenter. Ce sera les quantités de pollution émises en tonnes de CO2, quantités totales par pays et per capita, pour diverses années allant de 1975 à 2014. Nous avons pris soin de sélectionner des données les plus neutres et sans conflits d’intérêts possibles, ce pourquoi nous les avons extraites du site de l’ONU, ayant admis sa « fiabilité ».

Le but est de représenter ces données d’un façon ludique, interactive, et orientée utilisateur avant tout. Le but est d’essayer de renouer avec les ordres de grandeurs et les quantités en général, l’amplitude des valeurs étant telle en matière de pollution que de simples chiffres sans autres repères ne permettent pas d’avoir une vision claire sur le sujet.

Le support de représentation sera un site Web. C’est un support adapté à l’utilisateur lambda car il n’y a pas d’étape de compilation/d’installation, il suffit à l’utilisateur de saisir une simple URL au sein de son navigateur pour tester et utiliser notre application.

Nous désirons donc produire un site web le moins « scolaire » possible, un site avant tout agréable à utiliser, totalement interactif et réactif. C’est pourquoi nous avons choisi une solution moderne : la SPA (Single Page Application) : C’est un nouveau standard dans l’industrie de site Web qui consiste à ne demander qu’une seule fois toute l’application web au serveur, aucun rechargement total de page n’est nécessaire et le potentiel chargement de données, ou bien les changements physiques de la page, sont tous réalisés de manière transparente pour l’utilisateur, côté client. Cette pratique « casse » un peu le modèle client-serveur scolaire que nous connaissons habituellement.

# II – Développement

## A – Technologies

Client : Nous utiliserons essentiellement du JavaScript pur, au dernières normes ES6 en vigueur. Les différents navigateurs étant tous à des stades différents de validation des normes JS, nous faisons le choix d’écrire un code très récent et actuel mais d’utiliser un transpiler (Babbel) afin de le convertir en code exécutable pour tous les navigateurs, même les plus anciens. Afin de représenter les données, nous utiliserons la librarie d3.js, souvent utilisée dans la représentation des données sur le web.

Serveur : Afin de posséder l’avantage d’utiliser le même langage côté client et serveur, le serveur sera écrit avec Node.js, standard émergent de la programmation côté serveur. Également, nous avons choisi MariaDB comme SGBD afin de stocker de manière efficace nos données de pollution, et d’exécuter des requêtes sur celles-ci.

## B – Programmation asynchrone et fonctionnelle

Une des particularités du projet, qui fut également une difficulté au début, est le langage JavaScript lui-même. Malgré le fais qu’il soit considéré comme multi-paradigme, les normes JavaScript récentes et surtout l’utilisation que nous en avons fait avec D3.js nous a refait penser totalement nos habitudes de développement.

En particulier, notre librairie graphique nous impose le paradigme fonctionnel : nous avons dû écrire en grande partie des fonction dites « pures »[[1]](#footnote-1), sans effets de bord, ainsi que des fonctions d’ordre supérieur[[2]](#footnote-2). Également, en général, l’affectation d’une variable n’est pas permise en dehors de sa déclaration, d’où l’utilisation massive de constantes dans notre code.

La programmation asynchrone est également au centre de ce projet et n’est pas triviale à mettre en place. En effet en JavaScript, lors de l’utilisation de *Promises*, spécialement dans le cas de requête en JS coté client, le code ne s’exécute plus naturellement « de haut en bas » et il nous faut prendre en compte les délais de réponse serveur/client lors d’échanges réseau.

Coté serveur, c’est également le cas, Node.js étant un langage mono-thread la programmation asynchrone est omni-présente, afin de pourvoir continuer l’exécution du code sans ralentissements lorsqu’une instruction prend trop de temps à l’exécution. (Chargements de données extérieur, requêtes vers une base de données).

1. *Une fonction dont le résultat ne dépend que de ces arguments. Une fonction pure doit renvoyer le même résultat pour les mêmes arguments.* [↑](#footnote-ref-1)
2. *Une fonction qui soit prend en paramètre une autre fonction, soit renvoi une fonction, au moins un des 2 cas.* [↑](#footnote-ref-2)