Parcigneau Victor

p1804093

16/02/2024

Rapport Big Data Web Application

Base de données sur les Pokémons

Université Claude Bernard Lyon 1

ISFA

Table des matières

I) Introduction

Dans le cadre de ce projet, j'ai développé une application web nommée "Mon Pokédex" qui permet aux utilisateurs de découvrir et d'explorer une vaste collection de Pokémon. Cette application est conçue pour offrir une expérience immersive aux amateurs de Pokémon en leur permettant de rechercher, de filtrer et d'obtenir des informations détaillées sur différents Pokémon.

L'objectif principal de cette application est de fournir une plateforme conviviale et interactive où les utilisateurs peuvent accéder à une variété de données sur les Pokémon, telles que leurs caractéristiques, leurs types, leurs attaques, etc. Cela leur permet d'approfondir leurs connaissances sur cet univers fantastique et de partager leur passion avec d'autres fans.

Dans ce rapport, je vais décrire en détail le processus de développement de l'application, en mettant en lumière les choix technologiques, les fonctionnalités implémentées et les défis rencontrés tout au long du projet. Je vais également expliquer comment l'application fonctionne, comment elle est déployée et quelles sont ses perspectives.

Ce projet offre une introduction significative à la gestion des big data en raison de sa nature complexe et variée. En examinant de près les données sur les Pokémon, il est possible de saisir les défis et les opportunités que présentent les ensembles de données volumineux, hétérogènes et en constante évolution.

En premier lieu, la quantité de données générée par ce projet est considérable. En recueillant des informations sur des centaines de Pokémon, notamment leurs caractéristiques, leurs statistiques de combat, leurs types et bien plus encore, il offre une expérience concrète de travail avec de grandes quantités d'informations.

De plus, la variété des données est un aspect essentiel à considérer. Les données sur les Pokémon sont riches et diversifiées, couvrant un large éventail d'attributs et de catégories. Cela met en lumière la nécessité de gérer des ensembles de données hétérogènes et de comprendre comment extraire des informations pertinentes à partir de cette variété de données.

La vitesse à laquelle les données peuvent être traitées ou mises à jour est également un point important à considérer. Bien que les données sur les Pokémon ne soient pas nécessairement en constante évolution, la possibilité de mettre à jour régulièrement les données ou d'ajouter de nouvelles fonctionnalités offre un aperçu de la gestion des flux de données et de la nécessité de maintenir les données à jour.

Enfin, la complexité des requêtes des utilisateurs met en lumière l'importance de concevoir des bases de données optimisées et des algorithmes de recherche efficaces. Les utilisateurs peuvent effectuer une variété de requêtes pour filtrer et rechercher des Pokémon en fonction de différents critères. Cela souligne la nécessité de répondre

rapidement aux requêtes des utilisateurs, même avec de grandes quantités de données, tout en maintenant des performances élevées du système.

Dans l'ensemble, ce projet offre une expérience immersive et pratique dans la gestion des big data, en abordant les aspects de volume, de variété, de vélocité et de complexité des données. Cela permet d'acquérir une compréhension approfondie des défis et des opportunités liés à la gestion des données à grande échelle dans divers contextes professionnels.

II) Choix des langages et technologies

Ce projet constitue une excellente introduction à la gestion de la big data, notamment en raison des choix technologiques qui ont été faits pour sa réalisation.

Tout d'abord, l'utilisation de Cassandra comme base de données principale est pertinente pour gérer efficacement de grandes quantités de données. Cassandra est une base de données NoSQL distribuée conçue pour fonctionner sur un grand nombre de serveurs, offrant ainsi une extensibilité horizontale pour répondre aux besoins croissants de stockage de données. De plus, son modèle de données flexible permet de modéliser les données de manière adaptable, ce qui est particulièrement avantageux dans un projet où les données peuvent être variées et évolutives.

Ensuite, l'utilisation de Node.js pour le backend de l'application apporte des avantages en termes de traitement asynchrone et non bloquant. Dans une application web où de nombreuses requêtes peuvent être traitées simultanément, Node.js permet de gérer efficacement ces opérations en parallèle, garantissant ainsi des performances optimales. De plus, son écosystème riche de modules et de bibliothèques simplifie le développement et l'intégration de fonctionnalités supplémentaires.

Ce projet a également exploré l'utilisation d'Apache, Heroku et AWS pour le déploiement de l'application. Cependant, malgré les tentatives, ces solutions n'ont pas été couronnées de succès pour diverses raisons.

En ce qui concerne Apache, bien que la configuration des directives ProxyPass et ProxyPassReverse ait été mise en place dans le fichier de configuration, des problèmes persistants ont été rencontrés lors de la redirection du trafic vers le serveur Node.js hébergeant l'application. Malgré plusieurs tentatives pour résoudre ces problèmes, l'application n'a pas pu être correctement déployée via Apache.

De même, l'utilisation de Heroku a également été explorée comme une alternative de déploiement dans le cloud. Cependant, en raison de certaines contraintes et difficultés techniques rencontrées lors de la configuration et du déploiement de l'application sur la plateforme Heroku, l'objectif de déploiement sur cette plateforme n'a pas été atteint.

Enfin j'ai exploré la possibilité d'utiliser les services d'AWS (Amazon Web Services) pour héberger mon application, mais j'ai rencontré des difficultés dans la configuration et le déploiement. Malgré mes efforts pour configurer l'environnement, le déploiement de l'application n'a pas abouti comme prévu. J'ai suivi les instructions fournies par AWS et

tenté différentes approches pour résoudre les problèmes rencontrés, mais je n'ai pas réussi à obtenir le résultat souhaité.

Malgré ces difficultés, ces tentatives de déploiement sur des plates-formes alternatives ont permis d'acquérir une expérience précieuse dans la gestion et la résolution des défis liés au déploiement d'applications Web. Bien que les résultats n'aient pas été aussi concluants que prévu, ces tentatives ont permis d'explorer différentes approches et solutions, contribuant ainsi à l'apprentissage et à l'amélioration des compétences en matière de développement et de déploiement d'applications.

Après avoir analysé les difficultés rencontrées lors du portage de mon application vers AWS, j'ai réalisé que l'absence d'utilisation d'une machine virtuelle (VM) ou d'un dual boot pour accéder à Linux pourrait être à l'origine des problèmes rencontrés. Cette prise de conscience est intervenue trop tard dans le processus, ce qui m'a empêché de rectifier la situation à temps. L'utilisation d'une VM ou d'un dual boot aurait pu fournir un environnement plus stable et mieux adapté pour le déploiement de mon application. Malheureusement, faute de temps, je n'ai pas pu revenir en arrière pour revoir cette approche et résoudre les problèmes rencontrés.

Enfin, du côté de l'interface utilisateur, l'utilisation d'HTML, CSS et JavaScript permet de créer une expérience utilisateur dynamique et réactive. Ces technologies offrent la flexibilité nécessaire pour concevoir une interface conviviale et interactive, tout en assurant une compatibilité élevée avec différents appareils et navigateurs. De plus, leur facilité de développement et de maintenance permet de créer rapidement et efficacement des composants frontend.

En résumé, ce projet combine des technologies adaptées à la gestion de grandes quantités de données, au traitement efficace des requêtes et à la création d'une interface utilisateur. Il offre ainsi une introduction pratique à la gestion de la big data dans le contexte d'une application web moderne.

III) Documentation

Partie 1 : Cassandra

Tout d’abord on se connecte à un cluster Cassandra, avec cqlsh par exemple.

-- Création d’un keyspace pokemon_keyspace s'il n'existe pas déjà

CREATE KEYSPACE IF NOT EXISTS pokemon_keyspace WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

-- Création de la table pokemon_table dans l'espace de clés pokemon_keyspace s'il n'existe pas déjà

CREATE TABLE IF NOT EXISTS pokemon_keyspace.pokemon_table (-- Définition des colonnes de la table

id INT, name TEXT, nom TEXT, type1 TEXT, type2 TEXT, total INT, hp INT, attack INT, defense INT, sp_atk INT, sp_def INT, speed INT, generation INT, legendary BOOLEAN, PRIMARY KEY(id)); -- Définition de la clé primaire sur la colonne id

-- Copie des données depuis le fichier CSV /home/snoopbob/pokemon.csv vers la table pokemon_table

-- Les valeurs sont séparées par ';' et la première ligne du fichier CSV est l'en-tête des colonnes

-- Les valeurs nulles sont représentées par une chaîne vide

COPY pokemon_keyspace.pokemon_table (id, name, nom, type1, type2, total, hp, attack, defense, sp_atk, sp_def, speed, generation, legendary) FROM '/home/snoopbob/pokemon.csv' WITH DELIMITER=';' AND HEADER=true AND NULL='';

-- Création d'un index sur les colonnes type1 et type2 de la table pokemon_table afin de pouvoir trier les pokémons par types

CREATE INDEX ON pokemon_keyspace.pokemon_table (type1);

CREATE INDEX ON pokemon_keyspace.pokemon_table (type2);

Partie 2 : Node.js

Maintenant que notre cluster Cassandra est en place nous pouvons lancer le serveur en utilisant la commande node app.js.

Le fichier app.js est un script Node.js qui crée un serveur backend pour une application web. Il configure un serveur HTTP avec Express, importe les modules nécessaires tels que Express, CORS, Cassandra, Multer, etc.

Le script définit plusieurs routes pour gérer les requêtes HTTP entrantes, telles que la récupération des données sur les Pokémon, l'insertion de nouveaux Pokémon, l'exportation des données au format CSV, etc. Il établit une connexion à une base de données Cassandra, gère les requêtes entrantes, effectue des opérations sur la base de données Cassandra en fonction des requêtes et renvoie les résultats au client.

De plus, il permet le téléchargement et l'importation de fichiers CSV pour mettre à jour la base de données. Enfin, il démarre le serveur Express pour écouter les requêtes entrantes sur un port spécifié.

En résumé, le fichier app.js agit comme un middleware entre le client et la base de données Cassandra, traitant les requêtes HTTP et fournissant des réponses en fonction des opérations demandées.

Une fois que le serveur est en place, on peut accéder à la table pokemon_table via localhost:$port/api/pokemon

On peut également accéder au fichier page.html qui contient l'interface utilisateur en via localhost:$port/page.html (il faut veiller à ce que le chemin soit bien configuré).

Partie 3 : HTML

Nous allons maintenant détailler le code de la page HTML, afin de mieux comprendre son fonctionnement.

Le fichier HTML contient un ensemble de fonctions JavaScript qui interagissent avec les données des Pokémon et fournissent une interface utilisateur interactive. La fonction capitalizeFirstLetter est utilisée pour formater les noms des Pokémon en majuscule, améliorant ainsi leur lisibilité lors de l'affichage.

Ensuite, la fonction getTypeIcon récupère les URL des icônes correspondant aux types de Pokémon à partir d'une source externe, facilitant ainsi l'affichage des types avec leurs icônes. Ces icônes sont ensuite intégrées dans la fonction getTypeHtml, qui génère du code HTML pour afficher les types de Pokémon avec leurs icônes respectives.

Lorsque les données des Pokémon sont récupérées à partir de l'API, la fonction fetchPokemonDetails formate ces données pour les afficher correctement sur la page HTML. La fonction displayPokemon utilise ces données formatées pour créer des cartes individuelles pour chaque Pokémon, permettant ainsi à l'utilisateur de visualiser facilement les détails de chaque créature.

Lorsque l'utilisateur clique sur une carte Pokémon, la fonction togglePokemonDetails affiche ou masque les détails de ce Pokémon, offrant ainsi une expérience utilisateur plus dynamique et interactive.

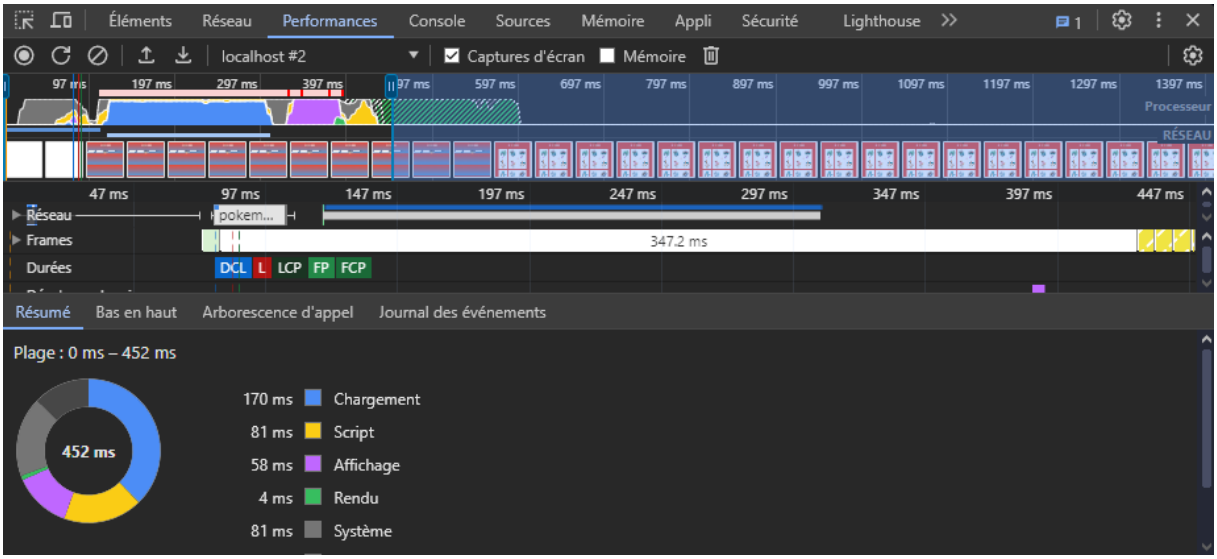
La fonction getPokemonName permet d'obtenir le nom d'un Pokémon dans la langue sélectionnée par l'utilisateur, garantissant ainsi une expérience multilingue. Lorsque l'utilisateur change la langue d'affichage, la fonction changeLanguage est appelée pour mettre à jour les noms des Pokémon en fonction de la langue sélectionnée.

Pour gérer les interactions avec la base de données, la fonction insertOrUpdatePokemon est utilisée pour insérer ou mettre à jour les données d'un Pokémon. De plus, la fonction getPokemonByType récupère les Pokémon correspondant à un type spécifique sélectionné par l'utilisateur, offrant ainsi une fonctionnalité de filtrage basée sur le type.

Enfin, les fonctions executeCommand, exportTable, et importTable fournissent des fonctionnalités supplémentaires telles que l'exécution de commandes spécifiques de la base de données Cassandra et l'exportation/importation de données au format CSV. Ces fonctionnalités enrichissent l'expérience utilisateur en offrant des options de gestion avancées des données Pokémon

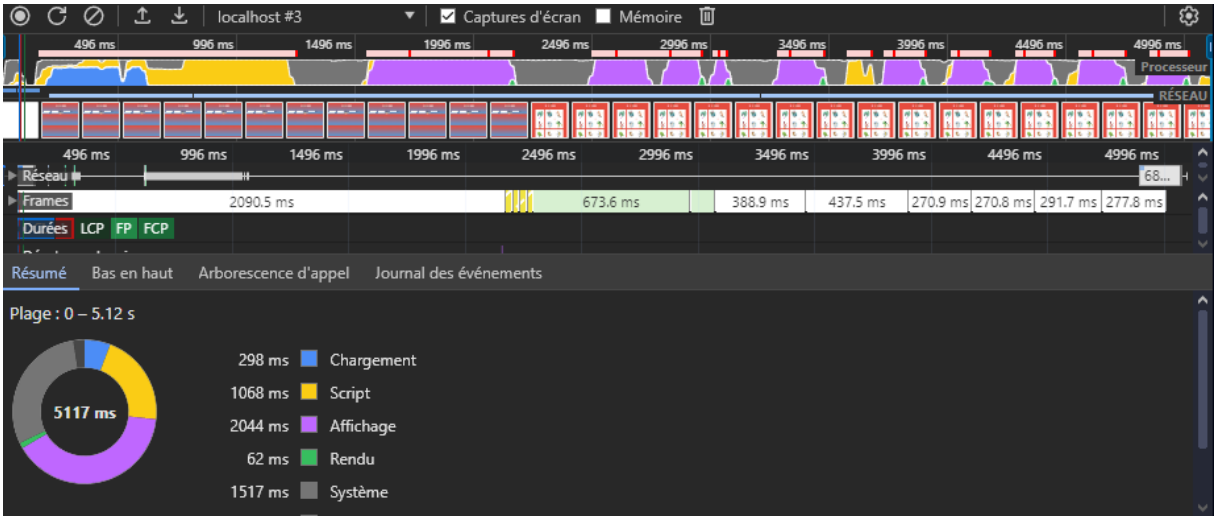
Sur cette page html, il y a donc la possibilité de traduire les noms des Pokémons en français, ajouter et modifier des Pokémons dans la base de données, importer et exporter les données sous forme de csv et entrer des commandes dans Cassandra.

IV) Analyse de performance



Avec les 721 Pokémons de base, la page met 170ms à charger les données et 143ms à les afficher.

Testons maintenant avec 10000 Pokémons pour voir l'évolution.



Cette fois le chargement prend un peu moins de 300ms mais l'affichage prend 3174 ms.

Au final on a fait x11,32 en temps d'exécution quand on a fait x13,86 en quantité de données à gérer, en passant de 721 à 10 000 Pokémons.

La fonction qui modélise l'efficacité du serveur est donc très vraisemblablement linéaire.

On peut donc estimer qu'il faudrait un peu moins d'une minute pour charger 100 000 Pokémons, 10 minutes pour 1 Millions etc...

C’est assez peu optimisé mais pour l’usage des Pokémons ce n’est pas problématique car il est peu probable qu’on arrive un jour, ne serait-ce qu’au 10 000 Pokémons.

V) Conclusion

Ce projet a été une expérience enrichissante qui m'a permis d'explorer un large éventail de langages et de technologies. À partir du choix initial des outils, j'ai opté pour Cassandra en tant que base de données, ce qui s'est avéré être une décision judicieuse pour stocker efficacement les données massives sur les Pokémon. L'utilisation de Node.js comme backend a également été un choix pertinent, offrant une flexibilité et une certaine facilité de développement.

Pendant le développement, j'ai rencontré divers défis, notamment lors de l'intégration avec Apache2 pour le déploiement, ainsi que lors de mes tentatives sur AWS et Heroku. Malgré ces difficultés, liées à des choix de matériel problématiques, j'ai persévéré et cherché des solutions alternatives pour faire avancer le projet.

La création de la page.html finale a été une étape cruciale, fournissant une interface utilisateur pour explorer les données sur les Pokémon. Les analyses de performance ont également été effectuées, révélant des temps de réponse satisfaisants malgré le volume élevé de données. Avec seulement 300 ms pour afficher 700 Pokémons et 3400 ms pour 10000, l'application a démontré son efficacité et sa capacité à gérer des charges importantes.

En conclusion, ce projet a été une véritable découverte technique, me permettant de développer mes compétences dans divers domaines, de la gestion de base de données à l'hébergement sur le cloud. Bien que des défis aient été rencontrés en cours de route, chaque obstacle a été une occasion d'apprentissage, renforçant ma compréhension des technologies impliquées et me préparant à relever de nouveaux défis dans le futur.