

WETHERGO

Table des matières

Introduction générale	1
1 Etat de l'Art	3
1.1 Big Data	3
1.1.1 Définition	3
1.1.2 Big Data Analytiques	4
1.2 Architecture Big Data	4
1.2.1 Datalake	4
1.2.2 Architecture lambda	6
1.2.3 Architecture Kappa	7
1.3 Frameworks	9
1.3.1 Stack hadoop	9
1.3.2 ZooKeeper	10
2 Etude préalable	11
2.1 Présentation de l'organisme d'accueil	11
2.1.1 Présentation de Dyno & Motiva Systems.....	11
2.1.2 Organisation de l'entreprise	12
2.1.3 Projets réalisés.....	12
2.2 Etude de l'existant	12
2.2.1 Analyse de l'existant	13
2.2.2 Critique de l'existant	13

2.2.3	Critique des applications similaires.....	14
2.3	Solution Proposée	15
2.4	Spécifications des besoins	19
2.4.1	Identification des acteurs	19
2.4.2	Spécification des besoins fonctionnels	19
2.4.3	Spécification des besoins non fonctionnels	19
2.5	Méthodologie du travail.....	21
2.5.1	Les principes de RAD.....	21
2.5.2	Les quatre phases.....	22
2.5.3	Les avantages de la méthodologie RAD.....	22
2.5.4	Les limites de la méthodologie RAD.....	23
2.5.5	Planning du stage.....	23
3	Etude conceptuelle	25
3.1	Diagramme du cas utilisation global	25
3.2	Diagramme de séquence	26
3.2.1	Scénario d'utilisation des filtres pour l'affichage des températures ..	27
3.3	Diagramme de classe.....	28
3.4	Architecture physique	29
3.4.1	Stockage de données : HDFS(Hadoop Distributed File System)	29
3.4.2	Traitement des données Map Reduce.....	31
3.4.3	Architecture trois tiers	32
4	Réalisation	34
4.1	Environnement de Travail	34
4.1.1	Environnement matériel.....	34
4.1.2	Environnement Logiciel	35
4.1.3	Installation et préparation de l'environnement.....	40
4.2	Conception de l'application avec Hadoop	43
4.2.1	Exploitement des données	43

4.2.2	Conception de la base de données HBase pour les températures	45
4.2.3	Analyse du fichier d'entrée dans un Mapper	47
4.2.4	Écriture dans HBase avec un Reducer	49
4.2.5	Mise en place d'une clé composite	50
4.2.6	Utilisation des clés composites.....	51
4.2.7	Lancement d'un modèle MapReduce d'import.....	52
4.3	Développement des modèles MapReduce.....	53
4.3.1	Lecture des données depuis HBase dans un Mapper.....	53
4.3.2	Agrégation des données dans un Reducer	56
4.3.3	Suivie des modèles MapReduce	57
4.3.4	Déboguage des modèles MapReduce	59
4.3.5	Exploration des sources de Hadoop	60
4.3.6	Réalisation des jointures de données.....	61
4.3.7	Résolution du problème de secondary sort	63
4.4	Industrialisation de l'application	65
4.4.1	Mise en place d'un workflow Oozie	65
4.4.2	Lancement d'un workflow Oozie	66
4.4.3	Filtrage des données de HBase.....	67
4.4.4	Exportation vers MySQL avec Sqoop.....	68
4.4.5	Lancement du workflow avec l'API HTTP REST	69
4.4.6	Couplage de l'application avec une interface web	70
4.5	Interfaces de l'application et amélioration.....	72
4.5.1	Interface d'accueil	72
4.5.2	Interface du résultat	74
4.5.3	Amélioration	75
Conclusion et perspectives		77
Netographie		78
Resumé		80

Table des figures

1.1	Architecture Data Lake	5
1.2	Architecture Lambda	7
1.3	Architecture Kappa	8
2.1	AccuWeather.com.....	14
2.2	fr.weather-forecast.com.....	15
2.3	Définition de l'application 1.....	16
2.4	Définition de l'application 2.....	17
2.5	Définition de l'application 3.....	17
2.6	Découpage de l'application 1	18
2.7	Découpage de l'application 2	18
2.8	Cycle de vie de la méthode RAD [8]	22
2.9	Diagramme de Gantt de modélisation du déroulement de stage	23
3.1	Diagramme du cas utilisation global	26
3.2	Diagramme de séquence de l'application	26
3.3	Diagramme de séquence d'utilisation des filtres pour l'affichage des températures	28
3.4	Diagramme de classe	29
3.5	Architecture HDFS	31
3.6	Architecture 3-tiers	32
3.7	Architecture 3-tiers	32

4.1 Logo PHP [9].....	35
4.2 Logo Java[10]	36
4.3 Logo MySQL [11].....	37
4.4 Logo Hadoop [12]	37
4.5 Logo Hbase [13]	38
4.6 Logo Eclipse [14]	39
4.7 Logo VSCode [15].....	39
4.8 Interface du site National Centers for Environmental Information.....	43
4.9 Dossier des données	44
4.10 Les données stockées dans HDFS.....	45
4.11 Structure hbase	46
4.12 Table ghcn dans HBASE.....	47
4.13 Class "GhcYearFileMapper".....	48
4.14 Class "HBaseReducer"	49
4.15 Class "StationDateWritable"	51
4.16 Class "TypeValueWritable"	52
4.17 Class "ImportDriver"	53
4.18 Class "QhcnMapper".....	54
4.19 Class "TwoDimensionsWritable"	55
4.20 Class " AggregatorReducer "	56
4.21 MapReduce Framework.....	58
4.22 MAP INPUT RECORDS	58
4.23 REDUCE INPUT GROUPS.....	59
4.24 Class " AggregatorReducer "	60
4.25 Import Hadoop sources.....	61
4.26 Les données de station (latitude et longitude).....	62
4.27 Class « TowDimensionsSortComparator »	64
4.28 Fichier "workflow.xml"	65
4.29 Fichier processor.properties	67

4.30 Job Run oozie.....	67
4.31 Class « GenerateScanString »	68
4.32 Index.PHP.....	71
4.33 Request.PHP.....	71
4.34 Result.PHP	72
4.35 Interface de choix de la dimension	73
4.36 Interface de choix de la station	73
4.37 Interface de choix de la mesure.....	74
4.38 Lancement du job en oozie.....	74
4.39 exécution du job dans oozie	75
4.40 Tableau "sparse".....	75
4.41 Carte d'affichage.....	76

Liste des tableaux

2.1 Tableau des besoins non fonctionnelles	20
--	----

Introduction générale

Le Big Data est devenu un enjeu majeur dans de nombreux domaines, offrant des opportunités sans précédent pour extraire des informations utiles à partir des ensembles de données vastes. Dans ce contexte, notre projet de fin d'études vise à exploiter le potentiel du Big Data pour observer et analyser les variations de température à travers une application intelligente.

Les températures sont des paramètres clés dans de nombreux domaines tels que la météorologie, la climatologie, l'agriculture, l'industrie, la santé, etc. Comprendre les variations de température à moyen et long terme, ainsi que leurs causes, est essentiel pour prendre des décisions éclairées, prévoir les conditions météorologiques, adapter les pratiques agricoles, et bien d'autres applications.

L'objectif principal de notre projet est de développer une application Big Data capable d'observer et d'analyser les variations de température en fonction de différents filtres. Ces filtres peuvent inclure des paramètres tels que la localisation géographique, l'heure de la journée, la saison, les conditions météorologiques actuelles, etc. Les utilisateurs auront la possibilité d'explorer les données de température et de visualiser les variations en fonction des filtres appliqués dans notre application.

Pour atteindre cet objectif, nous utiliserons des méthodes de collecte de données à grande échelle, telles que des capteurs de température répartis sur plusieurs sites, des données mé-

téorologiques en temps réel provenant de sources fiables, ainsi que des données historiques disponibles. Nous exploiterons ces données volumineuses en utilisant des technologies adaptées pour le traitement et l'analyse du Big Data.

L'importance de ce projet réside dans la possibilité d'obtenir des informations précieuses sur les variations de température à différentes échelles spatio-temporelles. Ces informations pourraient contribuer à améliorer la prise de décision dans divers domaines, notamment l'agriculture, le tourisme, la planification urbaine, la gestion des ressources naturelles, etc.

Ce rapport détaillera les différentes étapes de notre projet, de la collecte des données à la mise en œuvre de l'application, en mettant l'accent sur les méthodologies, les technologies utilisées et les résultats obtenus. Nous présenterons également des exemples concrets d'utilisation de notre application et discuterons des perspectives futures pour l'exploitation des données de température à travers le Big Data.

Chapitre 1

Etat de l'Art

introduction

Dans ce chapitre, nous aborderons les principes fondamentaux du Big Data, ses origines, les architectures couramment utilisées, ainsi que quelques exemples de frameworks. Nous soulignerons également leur importance pour les entreprises et passerons en revue les différentes étapes de déploiement d'un système informatique Big Data.[1]

1.1 Big Data

1.1.1 Définition

Le terme "big data" peut être traduit en français par "données massives" ou "mégadonnées". Il fait référence à des ensembles de données extrêmement volumineux et complexes, qui nécessitent des outils et des techniques spécifiques pour être analysés et exploités efficacement. Les big data sont caractérisées par les fameuses "3V" : volume (grande quantité de données), variété (diversité des types et des sources de données) et vitesse (vitesse à laquelle les données sont générées et doivent être traitées). Les big data ont un impact significatif dans de nombreux domaines, tels que la recherche scientifique, le marketing, la santé, la finance, et bien d'autres.

1.1.2 Big Data Analytiques

Le Big Data Analytiques permet d'explorer nos données pour découvrir des schémas, des tendances du marché et les préférences des clients afin d'aider les entreprises à prendre des décisions plus rapidement. Pour ce faire, il existe quatre types d'analyses distinctes :

1. L'analyse descriptive : elle examine rétrospectivement "où" un problème a pu survenir au sein de l'entreprise. Elle permet de comprendre les événements passés et d'identifier les causes des problèmes.
2. L'analyse diagnostique : plus approfondie que l'analyse descriptive, elle cherche à répondre à la question "pourquoi" un problème spécifique est survenu dans une entreprise. Elle vise à identifier les facteurs et les causes profondes des problèmes.
3. L'analyse prédictive : en utilisant l'intelligence artificielle et les techniques de machine Learning, cette analyse peut fournir à une organisation des modèles prédictifs de ce qui peut se produire ensuite. Elle vise à anticiper les événements futurs et à prendre des décisions éclairées en conséquence. Cependant, en raison de sa complexité, elle n'est pas encore largement adoptée.
4. L'analyse prescriptive : cette analyse permet de déterminer la meilleure solution parmi différents scénarios. Elle propose des recommandations précises pour prendre des décisions optimales et guider les actions de l'entreprise.[1]

1.2 Architecture Big Data

1.2.1 Datalake

Définition

Un Data Lake accepte et conserve toutes les données provenant de différentes sources et prend en charge tous les types de données. Il utilise une architecture plate pour le stockage des données. Chaque élément de données dans un Data Lake est attribué un identifiant unique et est marqué avec un ensemble de balises de métadonnées étendues. Ensuite, il

peut être interrogé pour obtenir des données pertinentes, et un sous-ensemble plus restreint de données peut être extrait pour être analysé afin de répondre à des problématiques spécifiques de l'entreprise.

Architecture

Les Data Lakes sont construits sur un modèle de données appelé "schema-on-read". Un schéma représente essentiellement la structure d'une base de données et la manière dont les données y sont organisées. Avec le modèle de données schema-on-read, vous pouvez charger vos données dans le Data Lake sans avoir à vous soucier de leur structure initiale. Cela rend la structure des données très flexible, car elle peut être définie lors de la lecture des données plutôt que lors de leur chargement initial.

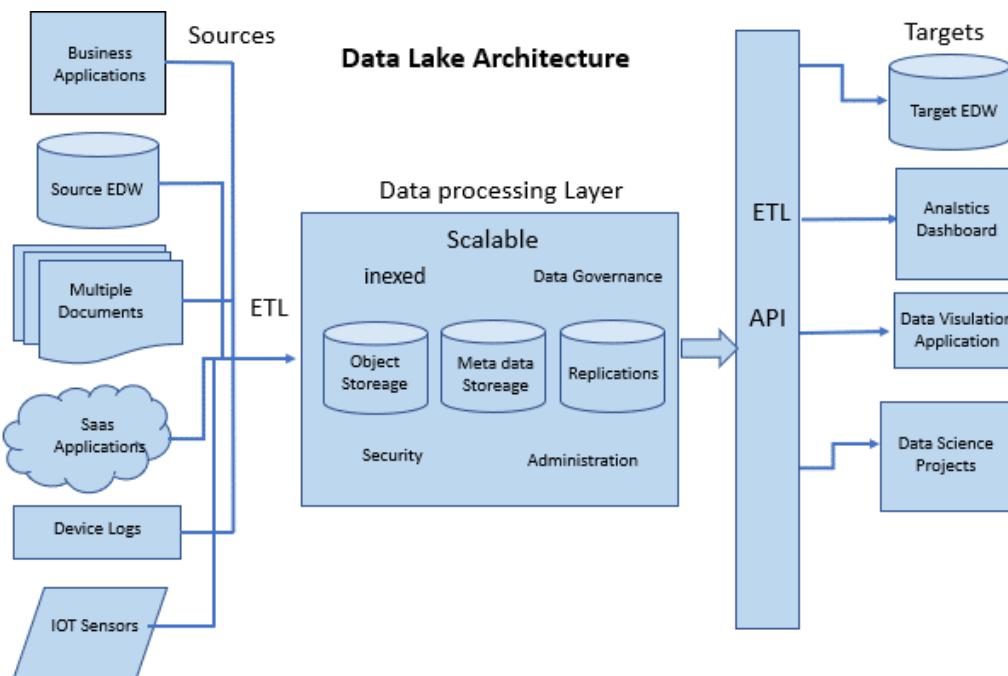


Figure 1.1 – Architecture Data Lake

Il existe deux types de traitement pour intégrer des données dans un Data Lake :

- Traitement par lots (Batch processing) : Il s'agit de traiter de grands ensembles de données sur de longues périodes. Ce type de traitement est moins sensible à la contrainte de temps et convient aux analyses volumineuses qui peuvent être exécutées de manière planifiée et périodique.
- Traitement des flux (Stream processing) : Il consiste à traiter de petits lots de données en temps réel, à mesure qu'elles arrivent. Le traitement des flux devient de plus en plus important pour les entreprises qui souhaitent exploiter l'analyse en temps réel, car il permet de prendre des décisions et de réagir rapidement aux événements en cours.[2]

1.2.2 Architecture lambda

Définition

L'architecture Lambda est une approche de traitement des données qui combine le traitement par lots et le traitement en temps réel pour gérer de grandes quantités de données. Elle vise à équilibrer la latence, le débit et la tolérance aux pannes en utilisant le traitement par lots pour des vues complètes et précises des données, et le traitement de flux en temps réel pour des vues en ligne. Les deux vues peuvent être fusionnées avant d'être présentées. L'architecture Lambda est largement utilisée avec la croissance du Big Data et de l'analyse en temps réel pour réduire les latences de MapReduce.

Architecture

Les données entrant dans le système suivent deux chemins distincts, ou couches :

1. La couche de traitement par lots (batch) : Elle stocke toutes les données brutes entrantes et les traite par lots. Le résultat de ce traitement est ensuite stocké sous la forme d'une vue basée sur le traitement par lots.
2. La couche temps réel (speed layer) : Elle analyse les données en temps réel, avec une faible latence mais potentiellement moins de précision.

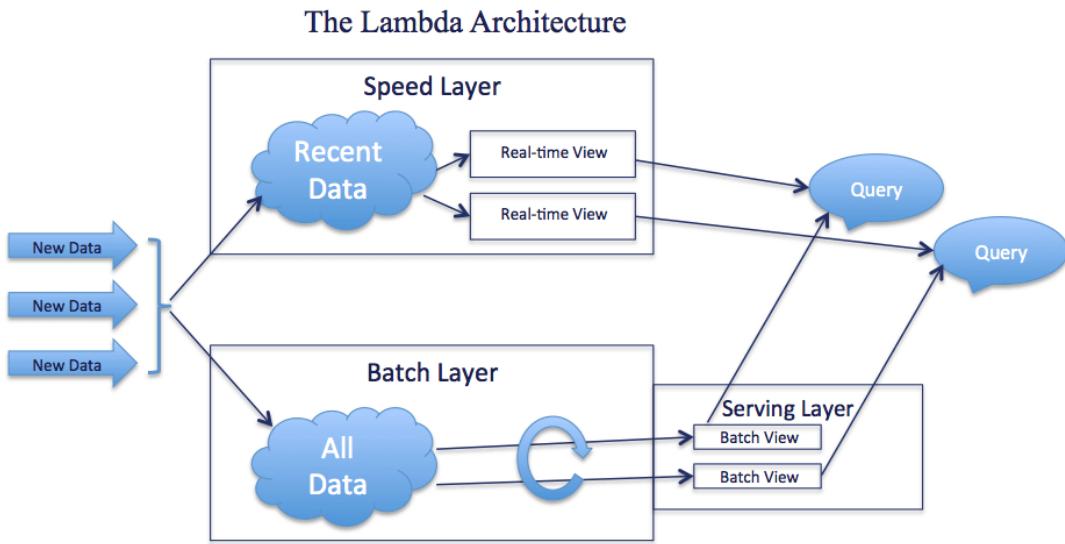


Figure 1.2 – Architecture Lambda

L’acheminement des données dans la voie réactive est limité par les contraintes de latence imposées par la couche vitesse, afin de garantir un traitement aussi rapide que possible. Cependant, cela peut entraîner une certaine perte de précision pour obtenir les données rapidement.[3]

1.2.3 Architecture Kappa

Définition

Le stockage de données dans l’architecture Kappa diffère de celui de l’architecture Batch Layer de Lambda. Dans l’architecture Kappa, nous utilisons un journal de données immuable, qui est similaire au concept d’ensemble de données brutes immuables de Lambda. Cependant, au lieu d’utiliser des technologies telles que Hadoop/HDFS, le journal de données immuables dans l’architecture Kappa est généralement basé sur Kafka.

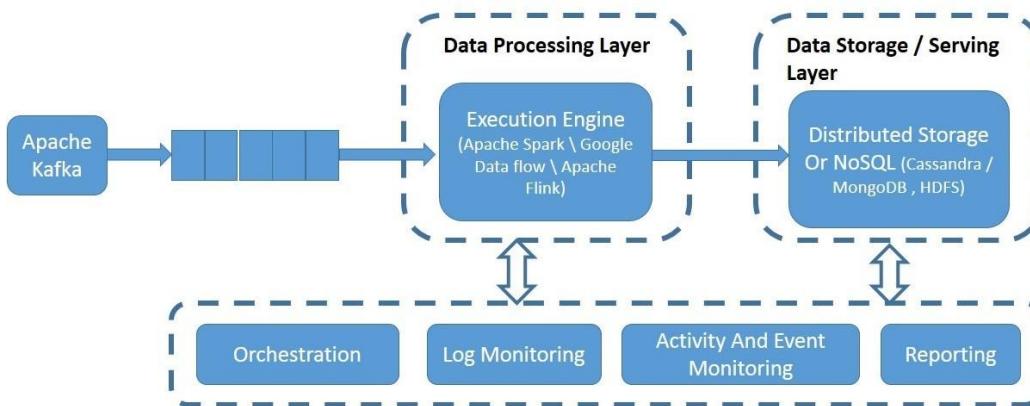
Architecture

Dans l'architecture Lambda, la couche temps réel (speed layer) est responsable de l'analyse des données en temps réel, ce qui garantit une faible latence dans le traitement des données. Cependant, cette couche peut sacrifier la précision des données. Les logs des transactions, qui contiennent nos données immuables, sont conservés dans cette couche.

La serving layer, quant à elle, est responsable de l'interrogation des données déjà analysées. Elle utilise la vue de traitement par lots provenant de la couche de traitement par lots. Cette couche de service indexe la vue de traitement par lots pour améliorer les performances d'interrogation.

La couche de traitement par lots alimente la couche de service en fournissant les données nécessaires. Les mises à jour incrémentales basées sur les données les plus récentes sont ensuite effectuées par la couche temps réel (speed layer) pour mettre à jour la couche de service.

Kappa Architecture



Siddharth Mittal

Figure 1.3 – Architecture Kappa

Lorsque les données circulent dans le chemin réactif de l'architecture Kappa, elles sont soumises à des conditions de latence imposées par la couche vitesse, ce qui permet un traitement rapide. Cependant, cela peut entraîner une perte de précision pour obtenir les données le plus rapidement possible.[4]

1.3 Frameworks

1.3.1 Stack hadoop

La stack Hadoop est un ensemble de logiciels open source utilisés pour le traitement et l'analyse des big data. Le principal composant de la stack Hadoop est le framework Hadoop lui-même, qui est conçu pour le stockage et le traitement distribué des données sur des clusters de serveurs.[5]

La stack Hadoop comprend plusieurs composants clés :

1. Hadoop Distributed File System (HDFS) : Il s'agit d'un système de fichiers distribué conçu pour stocker de grandes quantités de données sur des clusters de serveurs. Il répartit les données sur plusieurs nœuds du cluster pour assurer la redondance et la tolérance aux pannes.
2. MapReduce : C'est un modèle de programmation et un système de traitement distribué utilisé pour effectuer des calculs parallèles sur les données stockées dans le cluster Hadoop. Il divise les tâches en différentes étapes de mappage et de réduction pour traiter les données en parallèle.
3. YARN (Yet Another Resource Negotiator) : C'est un gestionnaire de ressources qui permet la planification et l'exécution des tâches sur un cluster Hadoop. YARN gère les ressources du cluster et alloue les ressources nécessaires pour exécuter les différentes tâches MapReduce ou d'autres applications.

1.3.2 ZooKeeper

ZooKeeper est un projet Apache open source qui offre un service centralisé pour la gestion des informations de configuration, des services de nommage, de synchronisation et de regroupement au sein de grands clusters dans des systèmes distribués. Son objectif est de faciliter la gestion de ces systèmes en assurant une propagation améliorée et fiable des changements.

Dans ZooKeeper, une application peut créer des "znodes", qui sont des fichiers persistants en mémoire sur les serveurs ZooKeeper. Les znodes peuvent être mis à jour par n'importe quel nœud du cluster, et tout nœud du cluster peut s'abonner pour être notifié des modifications apportées à un znode spécifique.[6]

Conclusion

Dans ce chapitre, nous avons présenté les concepts clés du Big Data ainsi que quelques frameworks importants. Nous avons également abordé l'architecture d'un projet Big Data. Le prochain chapitre sera consacré à l'étude préliminaire.

Chapitre 2

Etude préalable

Introduction

Le premier chapitre de notre étude met l'accent sur l'organisme d'accueil, l'étude de l'existant, la critique de l'existant, la solution proposée et la méthodologie choisie. Ce chapitre abordera les bases de notre travail en présentant le contexte dans lequel notre projet se déroule.

2.1 Présentation de l'organisme d'accueil

2.1.1 Présentation de Dyno & Motiva Systems

Dyno & Motiva Systems est une nouvelle société sur le marché tunisien. Elle a été fondée en 2022 par M. Mohamed Hedi JRIDI et M. Moez WADDEY. Dyno & Motiva Systems est certifiée en support client et en création de plateformes pour le bien-être des collaborateurs. Elle est experte des plateformes informatiques, des titres restaurant dématérialisés, des titres restaurant numériques, des plateformes de coaching professionnel, des courtiers en assurance, de la mesure de la satisfaction des salariés, de la gestion des comités d'entreprise et du team building [7]

Siège : Rue Béji Caïd Sebssi, Malek Centre, Bloc B, Bureau 3-1, Centre Urbain Nord,

Tunis 1082, TN.

2.1.2 Organisation de l'entreprise

L'équipe de Dyno & Motiva Systems se divise comme suit :

- CEO
- CTO & CO-Founder
- Équipe de développement web
- Équipe de développement mobile
- Équipe Big Data.

2.1.3 Projets réalisés

Depuis la création de Dyno & Motiva Systems, les projets ont été mis en œuvre sont :

- Une plateforme pour les titres restaurants digitales (pour les entreprises et les commerçants)
- Une plateforme pour organiser les Team Building
- Une plateforme de courtier d'assurance
- Une plateforme de Coaching professionnel

2.2 Etude de l'existant

L'étude de l'existant est une étape essentielle dans notre projet et constitue le cœur de la phase de l'étude préliminaire. Cela consiste à examiner et analyser attentivement le système actuel avant d'apporter des améliorations ou des modifications. L'objectif principal de l'étude de l'existant est de comprendre en détail les fonctions et les caractéristiques du système actuel, ensuite de dégager les différentes imperfections afin de les corriger.

2.2.1 Analyse de l'existant

Aujourd’hui, grâce aux avancées technologiques et à l'accès à des données météorologiques en temps réel, nous trouvons plusieurs applications qui permettent de fournir des prévisions météorologiques précises et fiables. Ces applications utilisent des modèles météorologiques sophistiqués, combinés à des algorithmes d'apprentissage automatique et à des techniques de traitement du signal, pour prédire les conditions météorologiques futures. Ces applications météorologiques offrent généralement une interface conviviale qui permet aux utilisateurs d'accéder facilement aux informations météorologiques pertinentes. Les données fournies peuvent inclure des informations sur la température, l'humidité, la vitesse et la direction du vent, les précipitations, les conditions de visibilité, etc.

Pour générer des prévisions météorologiques, ces applications combinent différentes sources de données, telles que des observations météorologiques en temps réel, des données satellitaires, des données provenant de stations météorologiques au sol, ainsi que des modèles numériques de prévision du temps. Les modèles météorologiques utilisent des équations mathématiques complexes pour simuler l'atmosphère et prévoir son évolution dans le temps.

2.2.2 Critique de l'existant

Les applications actuelles engendrent quelques lacunes :

- Manque d'historique (La plus ancienne prévision qu'on peut consulter est celle de l'année dernière),
- Prévisions limitées (On peut juste consulter la météo de l'année actuelle ou à la limite les prévisions des 12-30 prochains jours),
- Limitations des modèles de prévision : Les modèles météorologiques utilisés par les applications sont basés sur des hypothèses et des approximations de l'atmosphère. Bien que ces modèles soient continuellement améliorés, ils ne peuvent pas toujours capturer toutes les complexités et les interactions des phénomènes météorologiques.

Par conséquent, certaines conditions météorologiques spécifiques ou des événements météorologiques rares peuvent ne pas être correctement prédicts par ces modèles,

- Dépendance aux données en temps réel : Les applications météorologiques dépendent des données météorologiques en temps réel pour générer des prévisions précises. Si les données ne sont pas disponibles ou si elles sont incomplètes, cela peut affecter la qualité des prévisions. Des problèmes tels que des pannes de capteurs météorologiques, des interruptions de service ou des erreurs de transmission peuvent influencer la fiabilité des informations fournies par ces applications.

2.2.3 Critique des applications similaires

Cette section décrit les applications similaires à notre application afin d'avoir une idée sur les fonctionnalités à retenir et les problèmes à éviter.

*** AccuWeather.com

Dans la figure ci-dessous on distingue que l'utilisateur ne peut consulter que l'historique d'une année de moins.

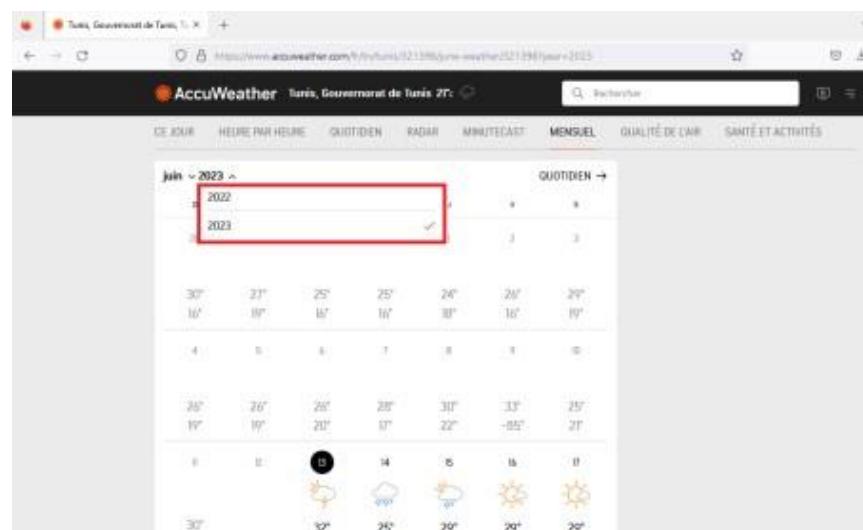


Figure 2.1 – AccuWeather.com

***** fr.weather-forecast.com**

Dans la figure ci-dessous nous distinguons que l'utilisateur ne peut consulter que les prévisions de 12 prochains jours au maximum.



Figure 2.2 – fr.weather-forecast.com

2.3 Solution Proposée

L'objectif de cette application est de permettre l'étude des températures observées sur toute la planète au cours des dernières centaines d'années. L'idée est de pouvoir analyser les variations de température en fonction de différents critères et d'afficher les résultats de manière simple et intuitive. L'application permettra aux utilisateurs de spécifier des paramètres tels que le type d'observation souhaité et d'obtenir un affichage tabulaire à deux dimensions.

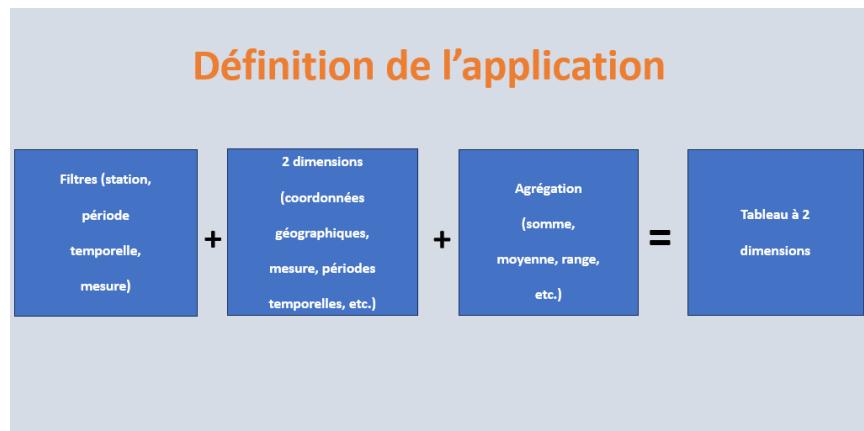


Figure 2.3 – Définition de l’application 1

Il y aura trois filtres disponibles, ce qui signifie que l’utilisateur pourra effectuer des filtrages en fonction d’une station spécifique, d’une période temporelle donnée ou d’une mesure précise. Par exemple, l’utilisateur pourra demander la température moyenne ou minimale pour chaque jour et spécifier deux dimensions d’observation. Ces dimensions serviront à regrouper les valeurs, telles que les coordonnées géographiques, les mesures et les périodes temporelles, et une opération d’agrégation pourra être appliquée, telle que la somme, la moyenne, l’écart type, etc.

Prenons un premier exemple : un utilisateur souhaite connaître, pour chaque mois, l’écart de température moyenne quotidienne, c’est-à-dire la différence entre la température maximale et la température minimale sur l’ensemble du mois. Pour ce faire, l’utilisateur filtrera en spécifiant uniquement la mesure de température moyenne (TAVG) et les deux dimensions d’observation seront le mois et une dimension vide (VOID), indiquant qu’il n’y a pas de deuxième dimension. L’opération d’agrégation utilisée sera le calcul de la plage (RANGE), afin de déterminer l’écart de température demandé. Le résultat sera un tableau affichant, par exemple, pour janvier 2016 : 10°C, pour février 2016 : 19°C, etc.

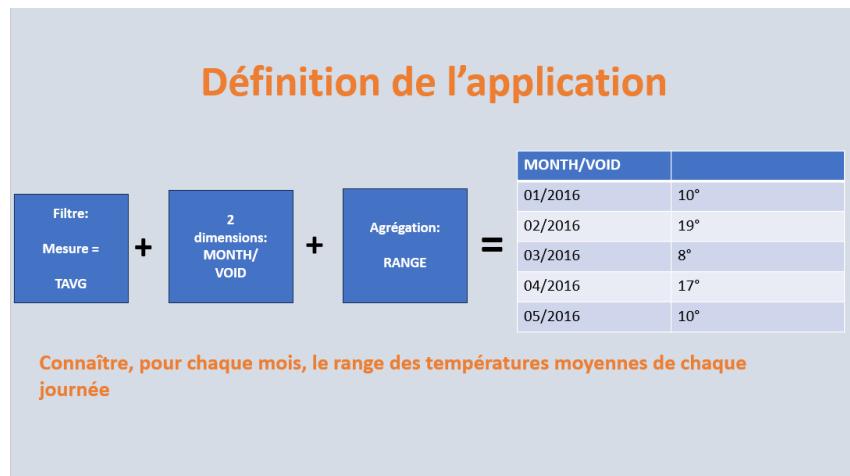


Figure 2.4 – Définition de l'application 2

Un autre exemple serait de vouloir connaître, pour chaque année, la moyenne de chaque mesure entre 2010 et 2011. Cette fois-ci, tous les types de mesures seront pris en compte, mais le filtrage se fera sur la période temporelle, c'est-à-dire de début 2010 à fin 2011. Les deux dimensions d'observation seront les mesures et l'année. Les données seront groupées en fonction de ces deux dimensions, et une moyenne sera calculée pour chaque mesure et chaque année. Le tableau résultant affichera, par exemple, une moyenne de 10°C pour les températures moyennes en 2010, une moyenne de 11°C pour les températures moyennes en 2011, une moyenne de 5°C pour les températures minimales de chaque jour en 2010, etc.

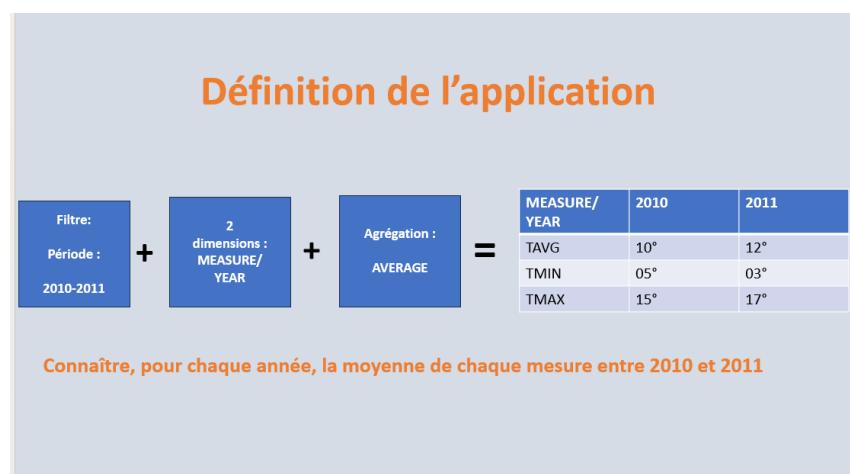


Figure 2.5 – Définition de l'application 3

L'objectif de cette application est de fournir un outil d'analyse adapté aux températures, en utilisant différentes étapes de traitement des données. Pour cela, l'application sera divisée en plusieurs parties. Tout d'abord, un utilisateur accédera à l'application via un navigateur web et remplira un formulaire pour exprimer son besoin. Ensuite, les paramètres du formulaire seront récupérés en PHP, et une première étape de filtrage des données sera effectuée. Les données agrégées seront ensuite enregistrées dans MySQL.

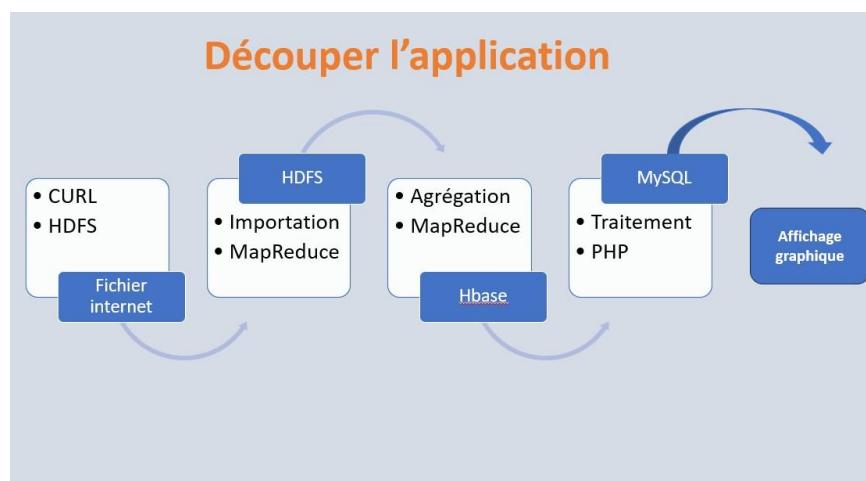


Figure 2.6 – Découpage de l'application 1

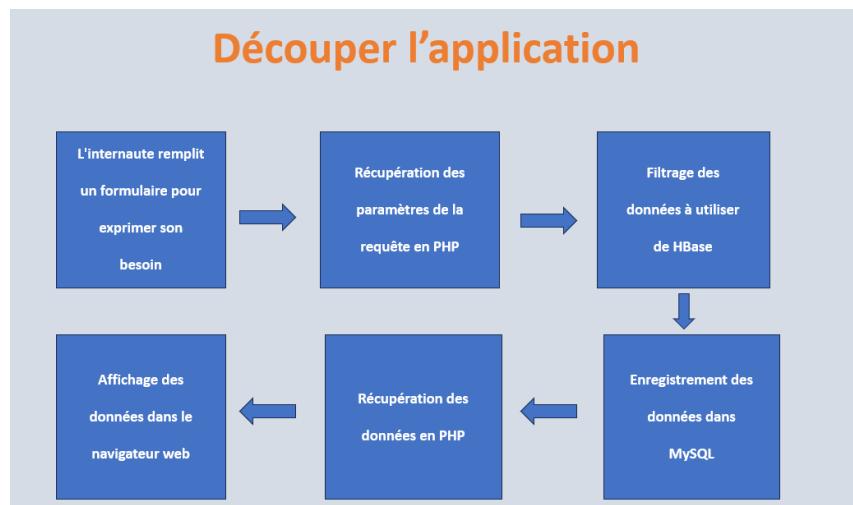


Figure 2.7 – Découpage de l'application 2

2.4 Spécifications des besoins

La phase de spécification des besoins implique la compréhension et la détermination des diverses fonctionnalités et exigences du système. Au cours de cette étape, nous présenterons les besoins fonctionnels et non fonctionnels de notre projet.

2.4.1 Identification des acteurs

Un acteur est une entité externe qui interagit avec le système. En réponse à l'action d'un acteur, le système fournit un service qui correspond à son besoin. Il existe deux types d'acteurs :

- Un visiteur : c'est l'internaute qui va accéder à l'application et qui peut consulter la météo d'une ville à une date déterminée et il peut consulter les prévisions météorologiques à long terme,
- Un admin : celui qui peut ajouter, modifier et supprimer des villes ou télécharger manuellement des fichiers de données.

2.4.2 Spécification des besoins fonctionnels

Les besoins fonctionnels sont les importantes fonctionnalités fournis à visiteur. Dans notre situation les besoins fonctionnels sont :

- Consulter la météo d'aujourd'hui.
- Consulter la météo les prévisions d'une date future.

2.4.3 Spécification des besoins non fonctionnels

L'étude de la qualité d'un logiciel permet de satisfaire les besoins non fonctionnels ou les besoins implicites des consommateurs. Cette partie est consacrée à la spécification de la qualité pour définir toutes les caractéristiques à respecter pour la réalisation de l'application.

Le tableau suivant décrit les sous caractéristiques tout en précisant leurs métriques et en attribuant un poids à chaque métrique.

Le poids est défini comme ceci :

- **E** : Élevé,
- **M** : Moyen,
- **F** : Faible.

Caractéristique	Sous caractéristique	Description	Métrique	Poids
Pertinence fonctionnelle	Précision	L'application fournit les fonctionnalités en adéquation avec ce que l'utilisateur souhaite voir	Nombre de fonctionnalités inutile et non utilisé	F
Performance	Comportement temporel	L'application fournit une réponse en un temps convenable.	Moyenne de temps en seconde pour afficher le résultat	E
Utilisabilité	Facilité d'utilisation	L'application permet à l'utilisateur de l'utiliser et de la contrôler facilement.	Moyenne des clics pour effectuer une recherche	M

TABLE 2.1 – Tableau des besoins non fonctionnelles

2.5 Méthodologie du travail

Dans le domaine du développement de projets, le choix de la bonne méthodologie est essentiel pour assurer une mise en œuvre réussie et efficace. Pour les projets Big Data avec moins d'équipes et de contraintes de temps, il est essentiel de choisir la bonne méthodologie pour permettre une progression rapide et la livraison de résultats tangibles.

Pour cette raison, nous avons décidé d'utiliser la méthodologie RAD [8]. En choisissant la méthodologie RAD, nous souhaitons maximiser l'efficacité de notre équipe et optimiser le temps qu'elle passe à atteindre les objectifs du projet. La méthodologie RAD permet des progrès rapides, une validation précoce des idées et une adaptation continue aux besoins changeants du projet.

Cette approche nous permet de développer de manière itérative notre application de surveillance de la température et de nous concentrer sur les fonctionnalités clés dès le départ. De cette façon, nous pouvons obtenir rapidement un produit fonctionnel minimum viable (MVP) qui peut être amélioré.

2.5.1 Les principes de RAD

Le développement doit être effectué par une petite équipe bien formée.
Le management RAD doit être actif et dynamique pour réduire les risques d'allongement des délais des cycles de développement.

Méthodologie :

- Liste de tâches seront utilisées pour vérifier qu'aucune action ne soit oubliée,
- Utilisation d'un prototype évolutif qui deviendra le produit final,
- Emploi de techniques d'aide à la définition des besoins comme JRP et JAD.

Les outils :

— Les outils doivent être fédérés par un référentiel et constituant un atelier puissant.

2.5.2 Les quatre phases

1. Phase de définition des besoins : Elle se matérialise par des sessions de travail appelées JRP.
2. Phase de conception utilisateur : Elle se caractérise par les sessions JAD, qui est un des signes distinctifs du RAD.
3. Phase de construction : Elle consiste à mêler les spécifications détaillées et le codage.
4. Phase de finalisation et mise en place

La figure 3 présente le cycle de vie de la méthodologie RAD

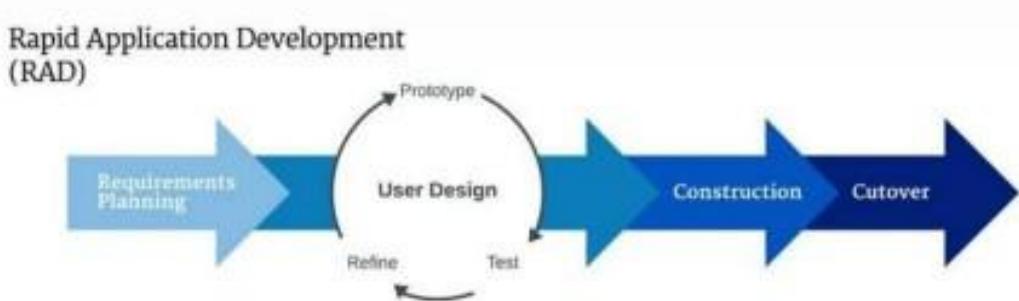


Figure 2.8 – Cycle de vie de la méthode RAD [8]

2.5.3 Les avantages de la méthodologie RAD

1. Le développeur est sûr du résultat final, car il l'aura réalisé avec son client.
2. L'implémentation du système est facile, car les prototypes sont testés tout au long du cycle de développement.
3. L'utilisateur est l'acteur du résultat final.
4. L'entreprise dispose d'un système de bonne qualité qui respecte les coûts et les délais.

2.5.4 Les limites de la méthodologie RAD

La méthodologie RAD (Rapid Application Development) présente plusieurs avantages en termes de rapidité, de flexibilité et d'implication des parties prenantes. Cependant, elle comporte également certaines limites qu'il est important de prendre en compte lors de son application. Voici quelques-unes de ces limites :

1. Complexité des projets.
2. Dépendance à l'égard des compétences techniques.
3. Risques de compromis sur la qualité.
4. Difficulté de rétrocompatibilité.

2.5.5 Planning du stage

La gestion de temps est un facteur nécessaire pour l'acheminement du projet, pour cela nous avons fixé un planning tout le long de stage et la figure 4 nous montre plus les détails.



Figure 2.9 – Diagramme de Gantt de modélisation du déroulement de stage

Conclusion

Ce premier chapitre marque le point de départ de notre rapport, offrant un aperçu global de la société, une analyse critique de son état actuel, une proposition de solution innovante et une méthodologie solide pour la mettre en œuvre. Les chapitres suivants approfondiront chacun de ces aspects, nous permettant ainsi de développer une vision complète et de fournir des recommandations concrètes pour façonner un avenir meilleur.

Chapitre 3

Etude conceptuelle

Introduction

Ce chapitre présente une étude conceptuelle du projet, mettant l'accent sur la phase de conception qui revêt une importance cruciale dans le développement informatique. La conception vise à décrire de manière non ambiguë le fonctionnement futur du système en utilisant un langage de modélisation. Dans ce contexte, nous utiliserons le langage de modélisation UML (Unified Modeling Language), qui offre un ensemble de schémas pour fournir différentes perspectives du projet à traiter.

3.1 Diagramme du cas utilisation global

Un diagramme de cas d'utilisation est une représentation visuelle des interactions entre un système donné et les acteurs, qu'ils soient des utilisateurs ou des systèmes externes. Il fournit une description des principales fonctionnalités ou actions d'un système du point de vue de l'utilisateur.

La figure ci-dessous représente le diagramme du cas d'utilisation global de notre système.

3.2. Diagramme de séquence

26

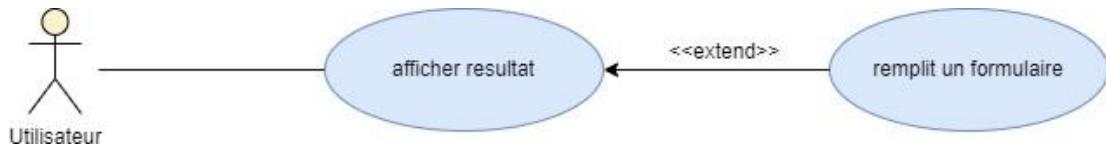


Figure 3.1 – Diagramme du cas utilisation global

3.2 Diagramme de séquence

Dans un cas d'utilisation, un diagramme de séquence est utilisé pour décrire l'interaction entre les éléments du système et les acteurs dans un ordre chronologique. Il montre de manière visuelle et séquentielle comment les différents composants du système travaillent ensemble pour accomplir une fonctionnalité particulière.

La figure 3.2 représente le diagramme de séquence de toute l'application.

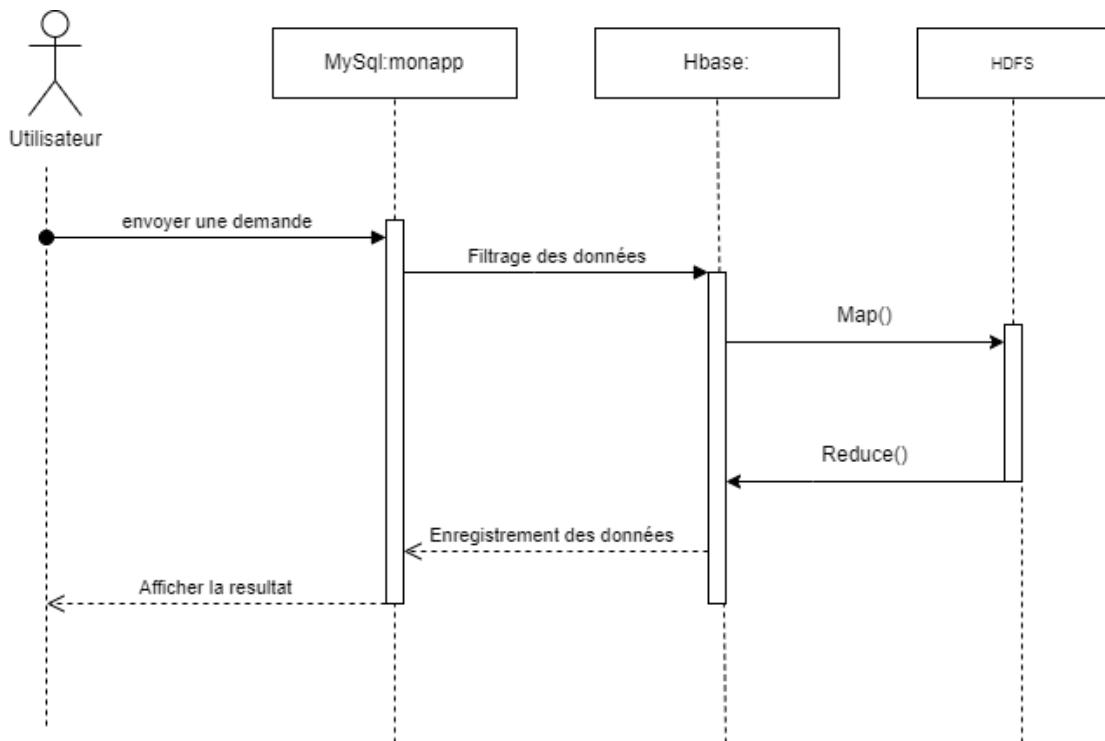


Figure 3.2 – Diagramme de séquence de l'application

3.2.1 Scénario d'utilisation des filtres pour l'affichage des températures

Le scénario d'utilisation des filtres pour l'affichage des températures implique l'exploration et l'analyse personnalisée des données de température dans une application dédiée. Voici une description détaillée de ce scénario :

1. L'utilisateur lance l'application web destinée à l'étude des variations de températures. Sur l'interface de l'application, l'utilisateur accède à une fonctionnalité de filtrage des températures pour affiner les résultats selon ses besoins.
2. L'utilisateur sélectionne les critères de filtrage, tels que la période de temps, la localisation géographique ou d'autres paramètres pertinents.
3. Après avoir choisi les filtres souhaités, l'utilisateur valide sa sélection et l'application traite les données en conséquence.
4. L'application affiche ensuite les températures filtrées dans un format clair et compréhensible, offrant une visualisation adaptée aux critères spécifiques choisis par l'utilisateur.
5. L'utilisateur peut explorer les données de température ainsi filtrées, en analysant les variations, les tendances et les corrélations entre différentes régions ou périodes.

Le diagramme présenté ci-dessous illustre le scénario d'utilisation du filtre par l'utilisateur.

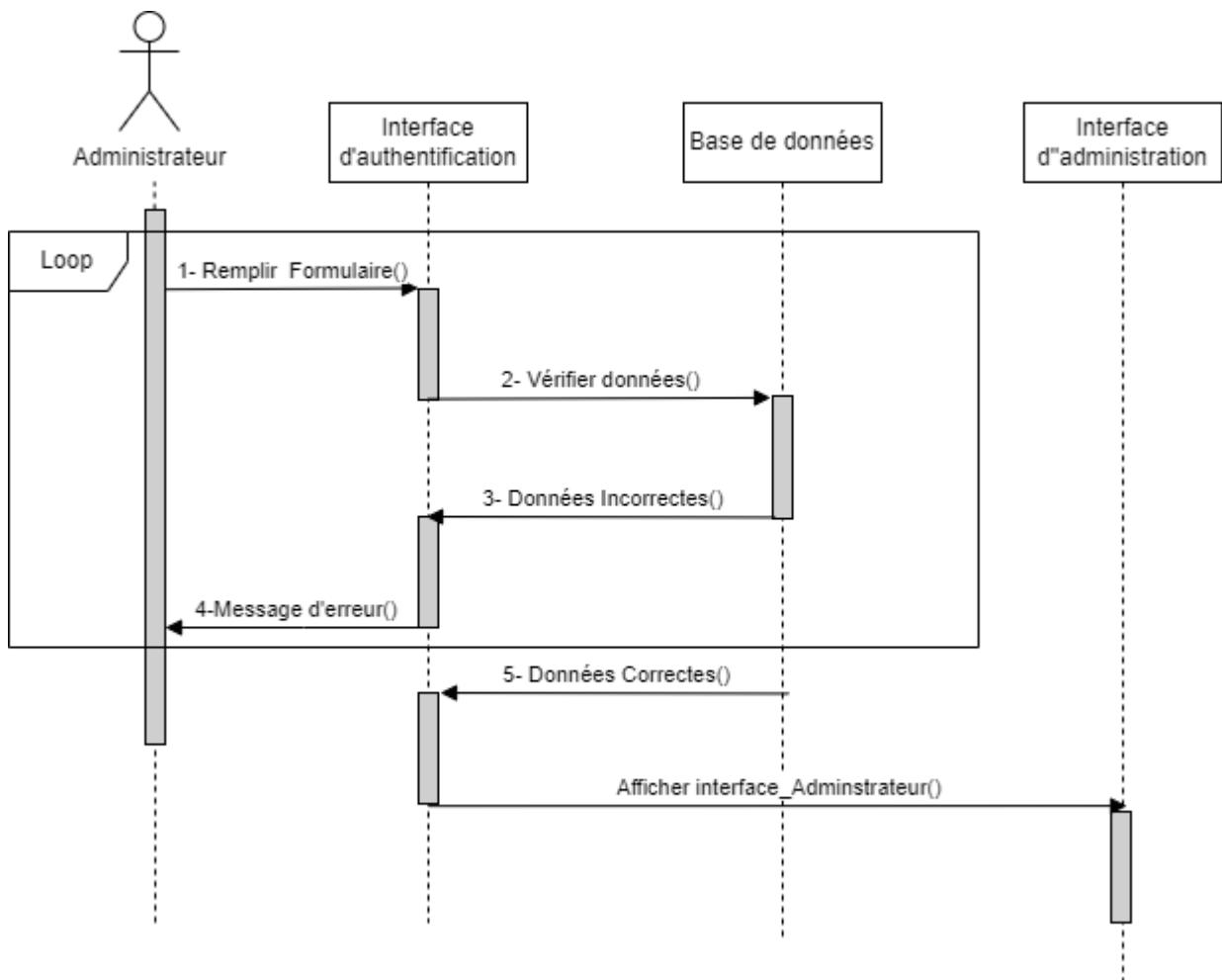


Figure 3.3 – Diagramme de séquence d'utilisation des filtres pour l'affichage des températures

3.3 Diagramme de classe

Le diagramme de classes d'analyse est largement reconnu comme l'un des diagrammes les plus utilisés dans le développement orienté objet et les spécifications UML. Il a pour objectif de décrire les classes, les attributs, les opérations et leurs relations que le système utilise.

Le diagramme de classes présenté dans la figure ci-dessous illustre l'implémentation

des différentes classes dans le système, ainsi que les relations et les cardinalités entre elles.

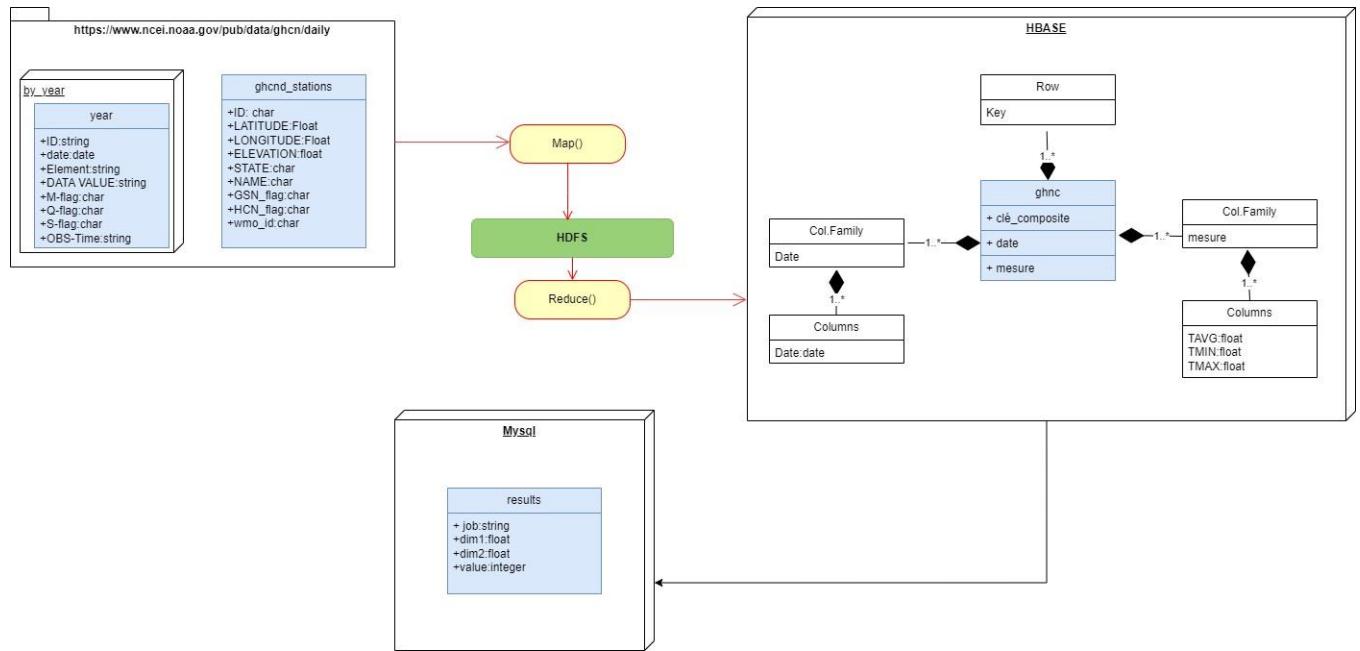


Figure 3.4 – Diagramme de classe

3.4 Architecture physique

3.4.1 Stockage de données : HDFS(Hadoop Distributed File System)

Pour stocker et gérer une quantité considérable de données à l'aide d'un ensemble de machines. Nous avons utilisé ce système de fichiers distribué qui est inclus dans le framework Apache Hadoop et spécialement conçu pour les environnements Big Data.

Plusieurs concepts fondamentaux sous-tendent l'utilisation de HDFS :

1. Architecture distribuée : HDFS fonctionne sur des clusters de machines. Le traitement parallèle des données est possible grâce à la répartition et au stockage des données sur plusieurs nœuds du cluster.

2. RéPLICATION DES DONNÉES : Pour assurer la tolérance aux pannes, HDFS réplique automatiquement les données. Chaque fichier est divisé en blocs de taille constante, qui sont initialement de 128 Mo, et ces blocs sont répliqués sur plusieurs nœuds du cluster.
3. ACCÈS À LA LECTURE ET À L'ÉCRITURE : HDFS prend en charge un modèle d'accès à la lecture et à l'écriture qui garantit que les fichiers ne subissent pas de modification une fois qu'ils sont écrits. Un fichier ne peut être lu ou supprimé qu'une fois qu'il est écrit dans HDFS, mais il ne peut pas être modifié. Deux processus peuvent être mis en valeur : Processus de lecture HDFS : Interrogation du NameNode pour localiser les adresses des nodes hébergeant les blocs sous-jacents les plus proches.(Voir figure ci-dessous). Processus d'écriture : Écriture sur le dataNode. DataNode communique ses blocs au NameNode.
4. SCALABILITÉ : HDFS peut facilement s'adapter au stockage de pétaoctets, voire d'exaoctets de données.
5. DÉBIT ÉLEVÉ : HDFS est conçu pour fournir un débit élevé lorsqu'il s'agit d'accéder aux données. Il est idéal pour les applications qui nécessitent l'analyse simultanée de grands ensembles de données.

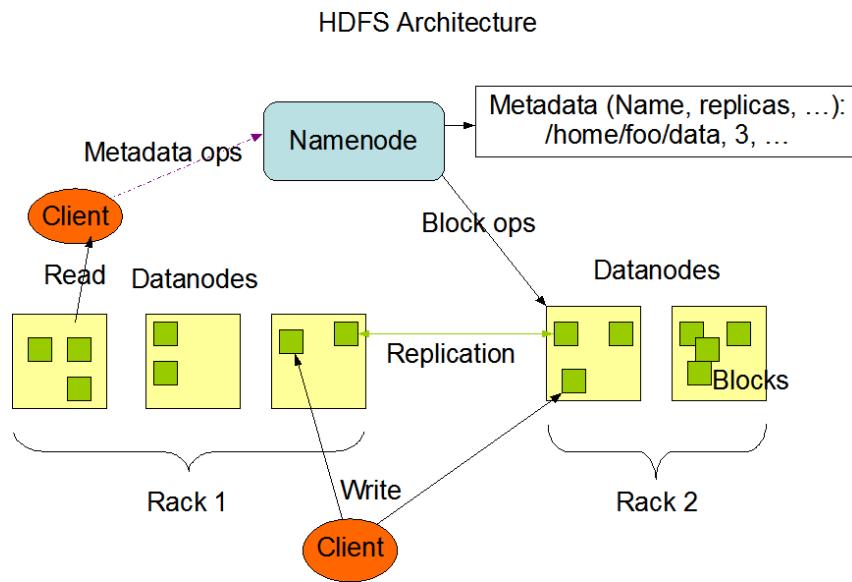


Figure 3.5 – Architecture HDFS

3.4.2 Traitement des données Map Reduce

Les environnements Big Data utilisent fréquemment la méthode de traitement parallèle et distribuée MapReduce. La phase Map et la phase Reduce sont les deux principales étapes du processus de traitement des données MapReduce.

La capacité de traiter simultanément une grande quantité de données en répartissant la charge de travail sur plusieurs nœuds du cluster est un avantage de l'approche MapReduce. Cela permet d'améliorer les performances globales du traitement des données tout en offrant une tolérance aux pannes grâce à la redondance des données.

Le processus de traitement des données MapReduce se déroule trois phases principaux :

- La première phase : (Map) est une étape d'ingestion et de transformation des données sous la forme de paires clé/valeur.
- La deuxième phase : Regroupement et tri par clé et chaque groupe est transmis à une fonction Reduce.

- La troisième phase (Reduce) est une étape de fusion des enregistrements par clé pour former le résultat final.

La figure ci-dessous représente les trois phases complètes du processus de traitement des données.

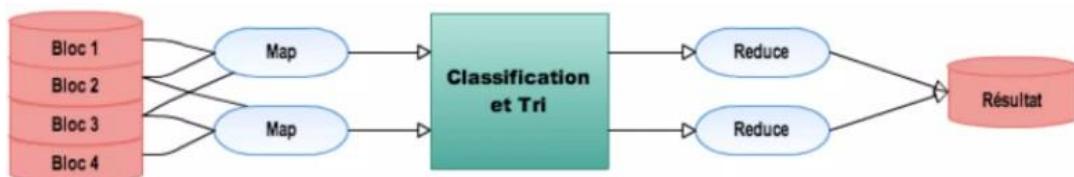


Figure 3.6 – Architecture HDFS

3.4.3 Architecture trois tiers

La figure ci-dessous représente l'architecture physique utilisée.

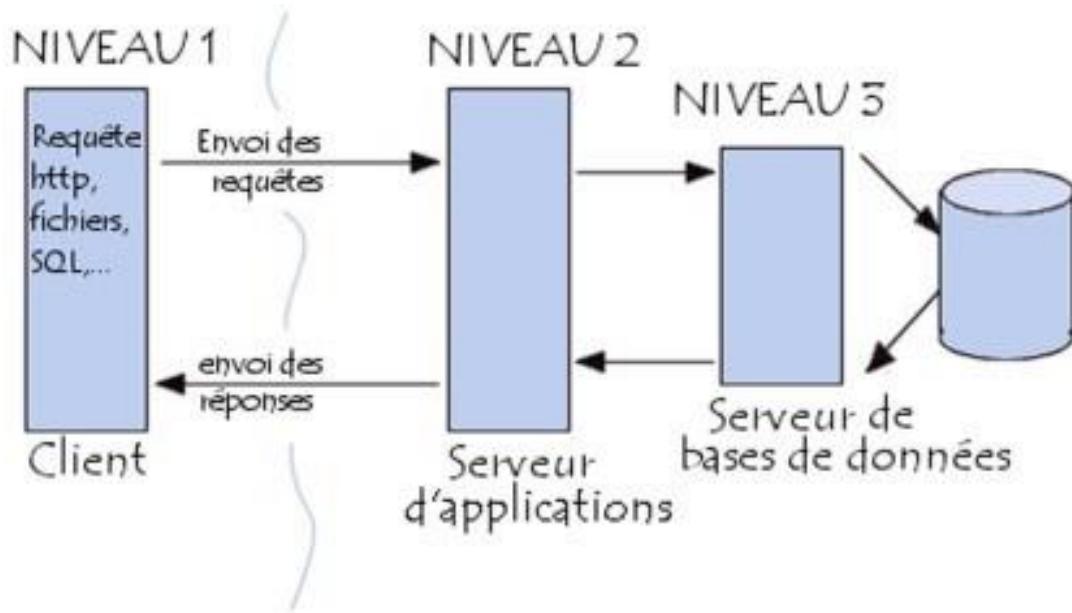


Figure 3.7 – Architecture 3-tiers

Les éléments suivants sont inclus dans l'architecture physique utilisée pour visualiser

la solution :

- Les serveurs de visualisation sont destinés à produire et à distribuer des visualisations. Ils peuvent utiliser des solutions de visualisation existantes telles que Tableau et Power BI ou héberger des applications de visualisation personnalisées. Les serveurs de visualisation sont chargés de traiter les requêtes de visualisation, de convertir les données en graphiques, tableaux, cartes ou autres formes visuelles et de fournir ces visualisations aux utilisateurs finaux.
- Les sources de données : sont les systèmes ou les bases de données qui contiennent des données. Les bases de données relationnelles, les entrepôts de données, les systèmes de fichiers, les services web, les API, etc. sont quelques exemples de sources de données.
- Réseau : Le réseau permet aux serveurs de visualisation et aux sources de données de communiquer. Il garantit le transfert des données vers les serveurs de visualisation depuis les sources de données. En fonction de l'architecture utilisée, le réseau peut être un réseau local (LAN) ou un réseau étendu (WAN).

Conclusion

Dans ce chapitre, nous avons présenté la conception de notre application en décrivant plusieurs éléments clés. Tout d'abord, nous avons exposé le diagramme de classe, qui illustre la structure des classes de notre système et les relations entre elles. Ensuite, nous avons abordé les diagrammes de séquence objet, qui décrivent les interactions entre les objets lors de l'exécution des scénarios spécifiques de notre application. Enfin, nous avons discuté de l'architecture logique et physique de notre système, en détaillant les différentes couches et composants qui le composent. Dans le chapitre suivant, nous allons nous concentrer sur la phase de réalisation de notre projet.

Chapitre 4

Réalisation

Introduction

Le chapitre de réalisation du projet marque une étape cruciale dans le processus de mise en œuvre de notre projet. Après avoir effectué une analyse approfondie, défini les objectifs et planifié les étapes à suivre, il est temps de passer à l'action et de concrétiser nos idées en une solution tangible. Ce chapitre met l'accent sur la mise en œuvre effective du projet, en utilisant les connaissances et les compétences acquises tout au long de notre parcours.

4.1 Environnement de Travail

Durant la phase d'implémentation, le PC portable utilisé porte les caractéristiques suivantes :

4.1.1 Environnement matériel

- PC de marque : Asus zenbook,
- Processeur : 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz,
- RAM : 32,0 Go,

- Système d’exploitation : Windows 11 Professionnel.

4.1.2 Environnement Logiciel

Citons ici les outils que nous avons utilisé tout le long du projet :

PHP

PHP est un langage de programmation du coté serveur conçu pour le développement web.

Les avantages de PHP :

- Facilité d’apprentissage,
- Large écosystème,
- Support des bases de données,

Les limites de PHP :

- Structure du code : PHP ne force pas une structure rigide pour l’organisation du code, ce qui peut rendre le code difficile à maintenir sur de longs projets ou lorsque plusieurs développeurs travaillent ensemble.



Figure 4.1 – Logo PHP [9]

Java

Java est largement utilisé dans le domaine du Big Data. Bien qu'il ne soit pas aussi couramment associé au Big Data que des langages tels que Python ou Scala, Java possède des bibliothèques et des outils puissants qui peuvent être utilisés pour gérer et analyser des

ensembles de données volumineux.

Les avantages de Java :

- Stabilité et performance,
- Vaste écosystème de bibliothèques,

Les Limites de Java :

- Consommation de mémoire élevée.



Figure 4.2 – Logo Java[10]

MySQL

MySQL est un système de gestion de base de données relationnelle (SGBDR) open source largement utilisé pour gérer et stocker des données structurées. Il s'agit de l'un des systèmes de base de données les plus populaires au monde et il est couramment utilisé dans les applications web pour la gestion et l'accès aux données.

Les avantages de MySQL :

- Compatibilité multiplateforme ,
- Scalabilité et performance,
- Open source ,
- Sécurité des données .

Les limites de MySQL :

- Complexité des requêtes,
- Fonctionnalités avancées limitées.



Figure 4.3 – Logo MySQL [11]

Hadoop

Hadoop est un framework open source qui permet de stocker et de traiter de grandes quantités de données de manière distribuée.

Les avantages de Hadoop :

- Évolutivité : Hadoop peut s'adapter à des quantités massives de données en distribuant les tâches de traitement sur de nombreux nœuds de calcul.
- Tolérance aux pannes : Hadoop offre une haute disponibilité et une tolérance aux pannes intégrées.

Les limites de Hadoop :

- Latence élevée : Hadoop est conçu pour traiter de gros volumes de données, ce qui peut entraîner une latence plus élevée dans le traitement des requêtes.

L'écosystème de Hadoop :

- Oozie : permet de planifier les tâches du framework.
- Sqoop : est un mécanisme de connexion et de transfert permettant de déplacer les données entre Hadoop et les bases de données relationnelles.
- HDFS (Hadoop Distributed File System) : est un système de fichiers distribué spécifiquement conçu pour le stockage et le traitement de données à grande échelle dans le cadre du framework Apache Hadoop. HDFS est un composant clé de l'écosystème Hadoop et est largement utilisé dans le domaine du Big Data.



Figure 4.4 – Logo Hadoop [12]

Hbase

HBase est une base de données NoSQL (Not Only SQL) distribuée et orientée colonnes, conçue pour le stockage et la gestion de données massives dans le cadre de l'écosystème Hadoop. Elle est construite au-dessus du système de fichiers distribué HDFS (Hadoop Distributed File System) et offre une évolutivité élevée et une tolérance aux pannes.

Les avantages de Hbase :

- Tolérance aux pannes,
- Évolutivité et consistance.

Les limites de Hbase :

- Opérations d'agrégation complexes,
- Complexité de configuration et d'administration.



Figure 4.5 – Logo Hbase [13]

Eclipse

Eclipse est un environnement de développement intégré (IDE) largement utilisé pour la programmation dans de nombreux langages, notamment Java. Il fournit des fonctionnalités avancées pour le développement, le débogage, la compilation et la gestion de projets.

Les avantages de Eclipse :

- Prise en charge de multiples langages,
- Interface utilisateur conviviale,
- Outils de test et d'intégration continue.

Les limites de Eclipse :

- Consommation de ressources,
- Complexité.



Figure 4.6 – Logo Eclipse [14]

VSCODE

C'est un éditeur de code source gratuit.

Les avantages de VSCode :

- Légereté et rapidité,
- Large prise en charge des langages et des plates-formes,
- Extensibilité,
- Communauté active et support.

Les limites de VSCode :

- Nécessite des extensions pour une fonctionnalité complète.



Figure 4.7 – Logo VSCode [15]

4.1.3 Installation et préparation de l'environnement

4.1.3.1 Téléchargement et installation Hadoop

Hadoop est aujourd’hui l’un des Framework Big Data les plus populaires, sinon le plus populaire. Bien que ses caractéristiques puissent paraître intimidantes, il peut s’installer en quelques étapes. Dans notre projet , nous avons réaliser l’installation sur Ubuntu 18.04.

Étapes de l’installation

1. Installation java

Hadoop (MapReduce) est écrit en Java et donc nous avons besoin d’avoir Java installé afin de pouvoir faire notre installation. Par ailleurs, Hadoop 3.x est compatible avec Java 8 donc c’est la version de java que nous allons utiliser. Pour installer java, depuis notre terminal Linux :

2. Crédation d’un utilisateur non root "hadoop" et un ssh pour cet utilisateur

C’est une bonne pratique communément acceptée de créer un utilisateur non root pour exécuter les daemons Hadoop. Ce faisant, nous prévoyons par exemple qu’un utilisateur malicieux soit en mesure de créer un job MapReduce qui supprime non seulement nos données hdfs mais aussi des données locales sensibles telles que le système d’exploitation.

Par ailleurs, Hadoop requiert ssh afin que le NameNode puisse communiquer avec les datanodes.

3. Téléchargement Hadoop

Nous téléchargeons ici la dernière version stable de Hadoop : 3.3.2 On la décomprime pour commencer la configuration.

```
$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.2/hadoop-3.3.2.tar.gz
```

Configuration Hadoop

Configurer Hadoop consiste à éditer des variables d'environnement (.bashrc) puis à modifier 4 principaux fichiers de configurations (hadoop-env.sh, core-site.xml, hdfs-site.xml, yarn-site.xml).

Nous avons configuré les variables d'environnement (.bashrc) qui seront définies à chaque démarrage de console interactive : \$ nano /.bashrc

Nous avons ajouté au fichier .bashrc les lignes suivantes :

```
* export JAVA_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")  
* export HADOOP_HOME=/nouveau/chemin/vers/dossier/hadoop  
* export HADOOP_INSTALL=$HADOOP_HOME  
* export HADOOP_MAPRED_HOME=$HADOOP_HOME  
* export HADOOP_COMMON_HOME=$HADOOP_HOME  
* export HADOOP_HDFS_HOME=$HADOOP_HOME  
* export YARN_HOME=$HADOOP_HOME  
* export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native  
* export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin  
* export HADOOP_OPTS=' -Djava.library.path=$HADOOP_HOME/lib/native'
```

Démarrage du cluster Hadoop

Pour démarrer le cluster Hadoop et lancer les différents daemons, nous nous positionnons dans le dossier suivant :

```
$ cd /home/hadoop/sbin/
```

Puis démarrez les NameNode et DataNode avec la commande :

```
$ ./start-dfs.sh
```

Pour démarrer YARN, la commande est :

```
$ ./start-yarn.sh
```

4.1.3.2 Téléchargement et Installation de Hbase

Après avoir installé l'environnement Hadoop sur votre VM, nous passons maintenant à l'installation d'Apache Hbase. La première étape consiste à télécharger Hbase sur le lien suivant : <https://hbase.apache.org/downloads.html>

Configuration

Elle consiste à g-éditer hbase-env.sh et ajouter au fichier la ligne suivante :

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Nous allons ajouter maintenant hbase aux variables d'environnement :

```
export HBASE_HOME=/home/hadoop/Downloads/hbase
export PATH=$PATH:$HBASE_HOME/bin
export CLASSPATH=$CLASSPATH:/home/hadoop/Downloads/hbase/lib/*::.
```

Nous pouvons accéder à apache Hbase via un navigateur comme montré dans la figure suivante : localhost :16010.

4.1.3.3 Téléchargement et Installation Oozie

Conditions préalables :

- Hadoop 3.x : installation expliquée
- Maven : 3.9.2
- Java 1.8 ou plus
- VM (Ubuntu)

Maintenant, nous passons à l'installation du Maven, pour cela, nous visitons la page de téléchargement de Maven : <https://maven.apache.org/download.cgi>

Nous avons téléchargé Apache Oozie 4.1.0 à partir du lien ci-dessous :

```
wget http://archive.apache.org/dist/oozie/4.1.0/oozie-4.1.0.tar.gz
```

Nous allons maintenant compiler Apache Oozie pour créer des fichiers binaires pour la distribution :

```
cd oozie-4.1.0
```

```
cd bin./mkdistro.sh -DskipTests -Dhadoopversion=3.3.1
```

Nous configurons à la fin oozie.

4.2 Conception de l'application avec Hadoop

4.2.1 Exploitement des données

Trouver les données était une étape essentielle dans le développement de l'application Big Data. En effet, la fiabilité de l'application reposait entièrement sur la qualité des données utilisées. Il était donc crucial d'examiner attentivement les données disponibles, de comprendre leur utilité et la manière dont elles pouvaient être exploitées. Dans notre cas, nous avons utilisé les données provenant du site <ftp.ncdc.noaa.gov>.

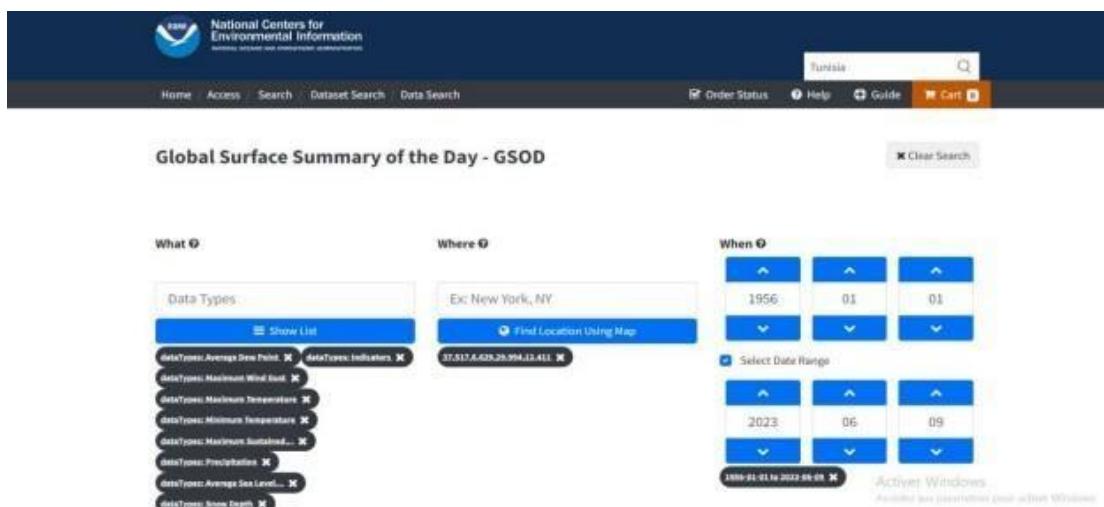


Figure 4.8 – Interface du site National Centers for Environmental Information

L'extension ".gov" du site suggérait qu'il s'agissait d'un site américain, et c'était effectivement le cas, car il était géré par le National Oceanic and Atmospheric Administration (NOAA), une agence du gouvernement américain chargée de collecter les données quotidiennes sur les températures, les précipitations et les pressions atmosphériques depuis 1763. À l'époque, le nombre de stations était beaucoup plus limité, ce qui signifiait que les données de cette période étaient moins abondantes. En explorant le répertoire "ftpghcn", nous

avons trouvé un fichier "readme" qui revêtait une grande importance, car il expliquait la méthodologie de collecte des données, la signification des différents indicateurs et constantes présents dans les fichiers, ainsi que le format des fichiers. Nous avons également trouvé un fichier "ghcnd" qui répertoriait chaque station avec ses coordonnées géographiques et des détails supplémentaires.

Nom	Taille	Date de modification
[répertoire parent]		
all/		08/03/2017 10:41:00
by_year/		12/01/2017 11:38:00
COOPDaily_announcement_042011.doc	33.5 kB	20/04/2011 00:00:00
COOPDaily_announcement_042011.pdf	122 kB	20/04/2011 00:00:00
COOPDaily_announcement_042011.rtf	66.5 kB	20/04/2011 00:00:00
figures/		06/02/2013 00:00:00
ghcnd_all.tar.gz	2.8 GB	08/03/2017 11:56:00
ghcnd-countries.txt	3.6 kB	23/06/2015 00:00:00
ghcnd_gen.tar.gz	140 MB	08/03/2017 11:56:00
ghcnd_hcn.tar.gz	278 MB	08/03/2017 11:56:00
ghcnd-inventory.txt	25.7 MB	07/03/2017 12:04:00
ghcnd-states.txt	1.1 kB	16/05/2011 00:00:00
ghcnd-stations.txt	8.3 MB	07/03/2017 12:04:00
ghcnd-version.txt	270 B	08/03/2017 11:56:00
grid/		07/03/2017 20:14:00
gsn/		08/03/2017 08:26:00
hcn/		08/03/2017 08:28:00
papers/		02/10/2012 00:00:00
readme.txt	23.6 kB	11/09/2015 00:00:00
status.txt	29.7 kB	14/04/2016 00:00:00
superghcnd/		07/03/2017 20:18:00

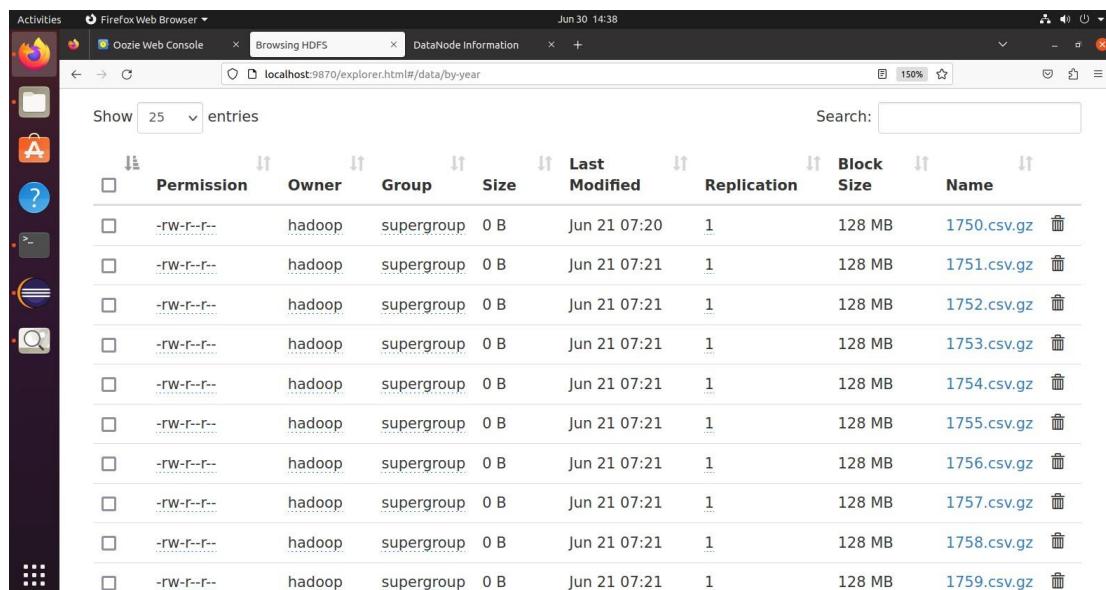
Figure 4.9 – Dossier des données

De plus, nous avons découvert plusieurs répertoires contenant les données dans différents formats. Dans notre cas, nous nous sommes concentrés sur le répertoire "by year", qui contenait des fichiers CSV compressés au format GZIP d'environ 190 Mo par année, à l'exception de l'année en cours, qui n'était pas encore complète. Nous avons pris soin de consulter le format des données dans le répertoire "by year" pour comprendre directement le format CSV. Ensuite, nous avons récupéré ces fichiers et les avons stockés dans le système de fichiers distribué HDFS.

Pour réaliser cela, nous avons copié le lien et utilisé la commande curl dans une console. Cependant, au lieu de les télécharger localement, nous les avons directement transférés vers HDFS. Au début du projet Big Data, il était essentiel de prendre en compte la structure des données que nous allions stocker dans HBase, une base de données NoSQL. Bien que

HBase soit une base de données NoSQL peu structurée, elle nécessitait un minimum d'informations sur la structure des données. Par conséquent, il était primordial de la définir dès le départ.

Nous avons déterminé le nombre de tables à utiliser, leurs noms et les familles de colonnes correspondantes. Dans notre cas, nous avons opté pour une seule table appelée "ghcn". Pour cette table, nous avons défini les familles de colonnes, leurs noms et leurs attributs. Nous avons également envisagé d'utiliser deux familles de colonnes. La première, "Date", contenait probablement une seule colonne sans identifiant spécifique. La deuxième famille de colonnes, "Measures",



The screenshot shows a Firefox browser window with the title bar "Activities Firefox Web Browser" and tabs for "Oozie Web Console", "Browsing HDFS", and "DataNode Information". The main content area displays a table of files in HDFS, with the URL "localhost:9870/explorer.html#/data/by-year" visible in the address bar. The table has columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. There are 10 entries, each representing a CSV file named from 1750.csv.gz to 1759.csv.gz, all owned by "hadoop" and grouped under "supergroup". The "Size" column shows 0 B for all files, and the "Last Modified" column shows Jun 21 07:21 for all. The "Replication" column shows 1 for all, and the "Block Size" column shows 128 MB for all. Each row has a trash icon in the last column.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:20	1	128 MB	1750.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1751.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1752.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1753.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1754.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1755.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1756.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1757.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1758.csv.gz
-rw-r--r--	hadoop	supergroup	0 B	Jun 21 07:21	1	128 MB	1759.csv.gz

Figure 4.10 – Les données stockées dans HDFS

4.2.2 Conception de la base de données HBase pour les températures

Dans notre application, nous stockons les données de température en tant que valeurs flottantes exprimées en degrés Celsius. Chaque enregistrement dans la table "ghcn" de HBase est identifié par une clé unique, composée du nom de la station et de la date cor-

respondante. Ainsi, si nous effectuons plusieurs opérations "put" pour la même station et date, les informations de cet enregistrement seront mises à jour.

Cependant, nous devons prendre en compte l'ordre de tri des clés dans HBase. Par défaut, HBase trie les clés par ordre croissant, ce qui signifie que les enregistrements les plus anciens apparaîtront en premier. Dans notre cas, nous souhaitons généralement consulter les enregistrements les plus récents en premier. Pour inverser l'ordre de tri, nous utilisons une méthode courante qui consiste à soustraire la date du nombre 99999999. Par exemple, au lieu d'utiliser l'année 2017 comme clé, nous utiliserions 7982. Cela garantit que les dates les plus récentes sont placées en tête de l'index de HBase, avec un ordre croissant vers les dates les plus anciennes. Cette optimisation améliore les performances de notre application dans les cas d'utilisation les plus fréquents.

Structure de HBASE				
Famely Column	Date	measurs		
Column		TAV	TMIN	TMAX
stationId-1-89855	20190101	03°	-4°	5°
StationId-2-89855	20190101	5°	-11°	16°

Clé composite : nom de la station, date (jour)sous forme de yyymmdd soustrait à 99999999

Famille de colonnes: measures, date (jour)sous forme de yyymmdd

Figure 4.11 – Structure hbase

Pour créer la table dans HBase, nous utilisons le Shell HBase. En lançant la commande "create 'ghcn', 'measures', 'date'", nous créons la table "ghcn" avec deux familles de colonnes : "measures" pour les différentes mesures de température et "date" pour la date

correspondante. Nous appliquons également la compression "SNAPPY" aux deux familles de colonnes afin de réduire l'espace requis pour les données et d'accélérer les transferts.

Regions										Description
Namespace	Name	State	OPEN	OPENING	CLOSED	CLOSING	OFFLINE	SPLIT	Other	
default	ghcn	ENABLED	1	0	0	0	0	0	0	'ghcn', [NAME -> 'date', COMPRESSION -> 'SNAPPY'], [NAME -> 'measures', COMPRESSION -> 'SNAPPY']

Figure 4.12 – Table ghcn dans HBASE

Maintenant, nous sommes prêts à commencer à travailler avec cette table dans HBase.

4.2.3 Analyse du fichier d'entrée dans un Mapper

Une fois la structure HBase mise en place, nous avons créé notre projet et notre MapReduce pour l'importation des données. Les données étaient déjà présentes dans HDFS, dans le répertoire "data/by-year".

Pour cela, nous avons configuré un projet Eclipse en ajoutant les bibliothèques correspondantes à Hadoop, trouvées dans "User lib Hadoop". Nous avons également ajouté les dépendances de HBase dont nous avions besoin. Ensuite, nous avons créé une classe Mapper appelée "GhcYearFileMapper" qui a étendu la classe Mapper de l'API v2 d'Apache Hadoop MapReduce.

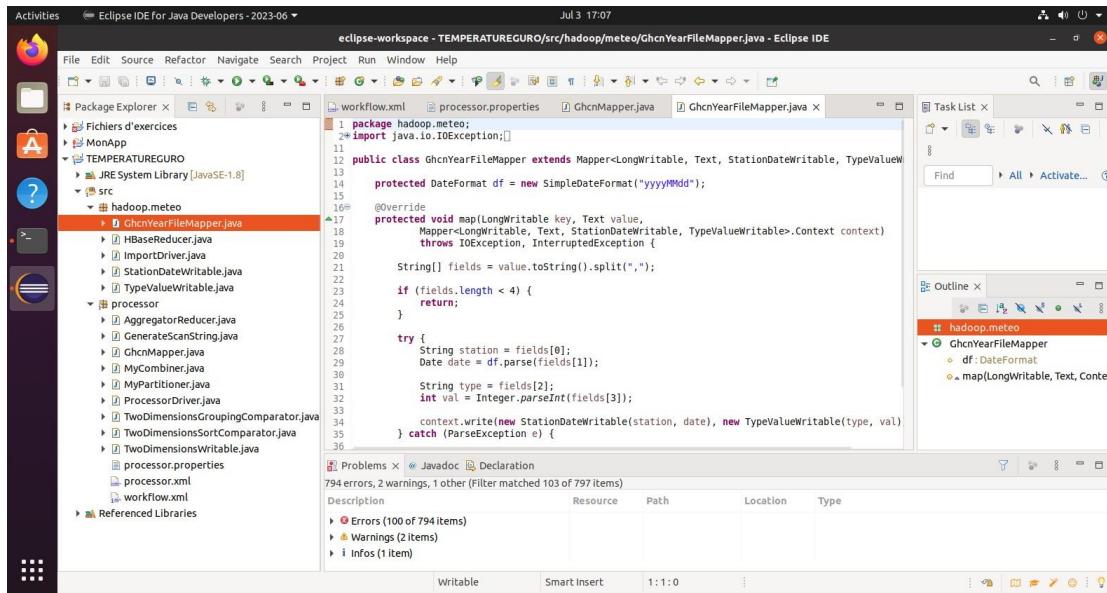


Figure 4.13 – Class "GhcnYearFileMapper"

Dans notre classe de Mapper, nous avons défini les types d'entrée (numéro de ligne et ligne) en tant que LongWritable et Text, respectivement. Pour la sortie, nous avons utilisé Text pour écrire les résultats.

La méthode Map a lu chaque ligne du fichier d'entrée et a extrait les données nécessaires, telles que le nom de la station, la date, le nom de la mesure et la valeur correspondante. Nous avons simplement divisé la ligne en utilisant la virgule comme séparateur et avons stocké les valeurs dans un tableau de chaînes (Fields). Ensuite, nous avons vérifié si la ligne contenait suffisamment de champs (au moins quatre) pour nos besoins.

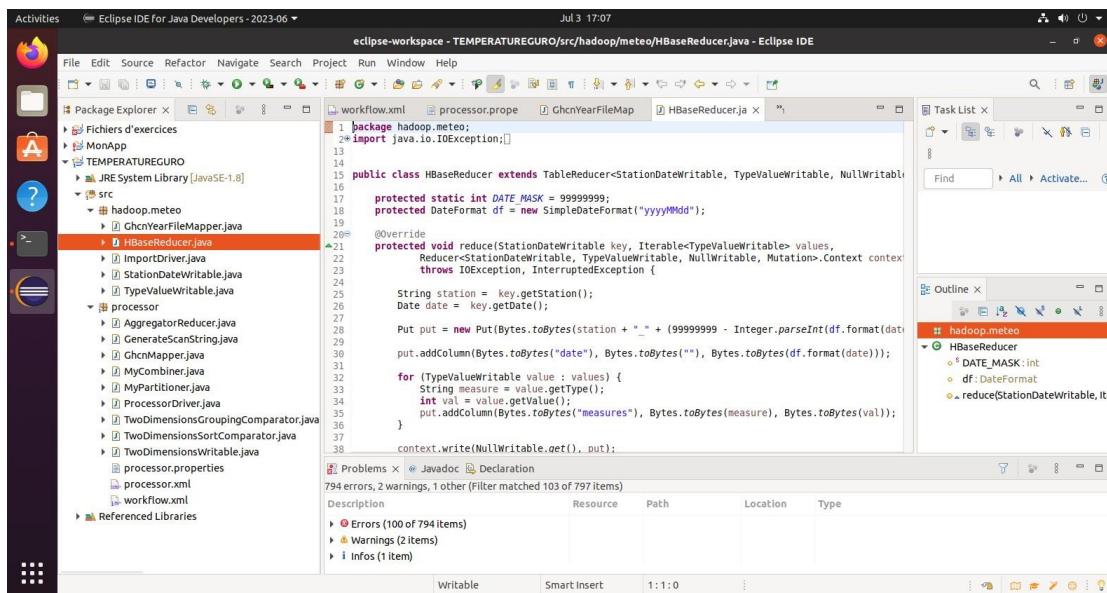
Ensuite, nous avons utilisé "context.write" pour écrire les résultats dans la sortie. Nous avons regroupé les enregistrements par station et date dans le premier champ de texte, et par nom de mesure et valeur dans le deuxième champ de texte. Par exemple, le premier champ contenait "station,date" et le deuxième champ contenait "nom_mesure,valeur". Nous avons utilisé les indices appropriés pour extraire les valeurs des champs.

Enfin, nous avons effectué l'écriture avec "context.write" sans nécessité d'un bloc try-catch.

4.2.4 Écriture dans HBase avec un Reducer

L'écriture dans HBase avec un Reducer a été réalisée en créant une classe appelée "HBaseReducer" qui étendait la classe spécifique à HBase, TableReducer. Dans cette classe, la méthode Reduce a été utilisée pour traiter les clés et les valeurs obtenues du Mapper.

Pour chaque paire clé-valeur, un objet Put a été créé, représentant une ligne à insérer dans HBase. La clé du Put a été formée en extrayant le nom de la station et la date de la clé d'entrée. Ensuite, la date a été ajoutée au Put en spécifiant le nom de la famille de colonnes, l'identifiant de colonne et la valeur correspondante. Ensuite, les valeurs ont été parcourues pour traiter les différentes mesures. Pour chaque valeur, la mesure et sa valeur ont été extraites et ajoutées au Put en spécifiant la famille de colonnes, l'identifiant de colonne et la valeur convertie.



```

Activities  Eclipse IDE for Java Developers - 2023-06  Jul 3 17:07
File Edit Source Refactor Navigate Search Project Run Window Help
eclipse-workspace - TEMPERATUREREGUO/src/hadoop/meteo/HBaseReducer.java - Eclipse IDE
Package Explorer  Fichiers d'exercices MonApp TEMPERATUREREGUO JRE System Library [JavaSE-1.8]
src hadoop.meteo GhcnYearMapper.java HBaseReducer.java ImportDriver.java StationDateWritable.java TypeValueWritable.java processor AggregatorReducer.java GenerateScanString.java GhcnMapper.java MyCombiner.java MyPartitioner.java ProcessorDrivenJava TwoDimensionsGroupingComparator.java TwoDimensionsSortComparator.java TwoDimensionsWritable.java processor.properties processor.xml workflow.xml Referenced Libraries
workFlow.xml processor.propre GhcnYearFileMap HBaseReducer.java
1 package hadoop.meteo;
2 import java.io.IOException;
3
4 public class HBaseReducer extends TableReducer<StationDateWritable, TypeValueWritable, NullWritable> {
5     protected static int DATE_MASK = 99999999;
6     protected DateFormat df = new SimpleDateFormat("yyyyMMdd");
7
8     @Override
9     protected void reduce(StationDateWritable key, Iterable<TypeValueWritable> values,
10                           Reducer<StationDateWritable, TypeValueWritable, NullWritable, Mutation>.Context context)
11     throws IOException, InterruptedException {
12         String station = key.getStation();
13         Date date = key.getDate();
14
15         Put put = new Put(Bytes.toBytes(station + "_" + (99999999 - Integer.parseInt(df.format(date)) * 1000000000)));
16         put.addColumn(Bytes.toBytes("date"), Bytes.toBytes("date"), Bytes.toBytes(df.format(date)));
17         for (TypeValueWritable value : values) {
18             String measure = value.getType();
19             int val = value.getValue();
20             put.addColumn(Bytes.toBytes("measures"), Bytes.toBytes(measure), Bytes.toBytes(val));
21         }
22         context.write(NullWritable.getInstance(), put);
23     }
24 }
25
26 
```

Figure 4.14 – Class "HBaseReducer"

Une fois le Put rempli, il a été envoyé à HBase en utilisant la méthode context.write

avec NullWritable comme clé (car elle n'était pas utilisée) et le Put comme valeur.

Cette approche a permis d'écrire efficacement les données dans HBase à partir des résultats du Mapper. Elle a offert une intégration transparente avec HBase grâce à l'utilisation de la classe TableReducer.

4.2.5 Mise en place d'une clé composite

Pour améliorer la qualité de notre programme d'import MapReduce dans HBase, nous avons décidé d'utiliser des clés composites plutôt que des textes concaténés. Une classe personnalisée appelée "StationDateWritable" a été créée pour représenter cette clé et implémente l'interface WritableComparable. Elle utilise un objet WritableComposite pour combiner le nom de la station et la date, en les séparant par une virgule. La classe "StationDateWritable" inclut une chaîne de caractères pour le nom de la station et une date au format JavaUtil.Date pour la date. Des méthodes getters et setters ont été ajoutées pour accéder aux valeurs de la station et de la date.

Pour la sérialisation, nous avons utilisé la méthode "writeFields" en utilisant la classe utilitaire "WritableUtils" pour écrire la station et la date dans le bon ordre. Pour la déserialisation, la méthode "readFields" a été implémentée en utilisant la méthode "readString" de "WritableUtils" pour lire la chaîne de caractères représentant la station. Ensuite, la classe "SimpleDateFormat" a été utilisée pour analyser la chaîne de caractères de la date et l'assigner à l'attribut de date correspondant.

De plus, la méthode "compareTo" a été mise en place pour comparer deux objets "StationDateWritable" en comparant d'abord les stations, puis en cas d'égalité, les dates. Les méthodes "equals" et "hashCode" ont également été implémentées pour assurer la cohérence des opérations dans Hadoop, en se basant sur les méthodes "equals" et "hashCode" des objets sous-jacents.

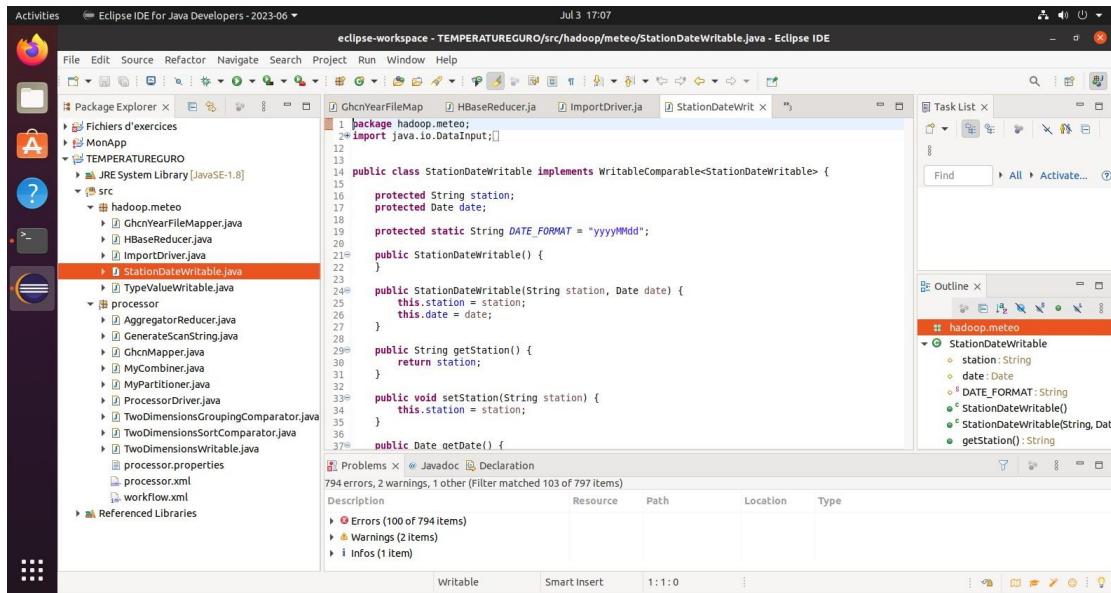


Figure 4.15 – Class "StationDateWritable"

En utilisant cette clé composite "StationDateWritable", nous avons amélioré la lisibilité et la qualité de notre programme MapReduce, tout en garantissant de bonnes performances et une fiabilité dans le traitement des données dans HBase.

4.2.6 Utilisation des clés composites

Nous avons décidé d'améliorer notre programme en utilisant des clés composites. Chaque clé était représentée par un objet "StationDateWritable" contenant une station et une date. Les valeurs des champs étaient extraites en utilisant l'index 0 pour la station et l'index 1 pour la date, après conversion de cette dernière au format Java et vérification de sa validité. Dans le mapper, nous avons émis des paires clé-valeur avec la clé en tant qu'objet "StationDateWritable" et la valeur en tant qu'objet "TypeValueWritable" qui prenait une chaîne de caractères pour le type et un entier pour la valeur. Dans le reducer, nous avons utilisé des objets "StationDateWritable" pour les clés et un itérable d'objets "TypeValueWritable" pour les valeurs. Nous pouvions extraire la station et la date de la clé pour créer une nouvelle clé composite selon le format requis. Les valeurs étaient directement extraites des objets "TypeValueWritable". Ces modifications ont utilisé des clés composites

pour améliorer la structure et la lisibilité de notre code, tout en maintenant de bonnes performances dans le traitement des données.

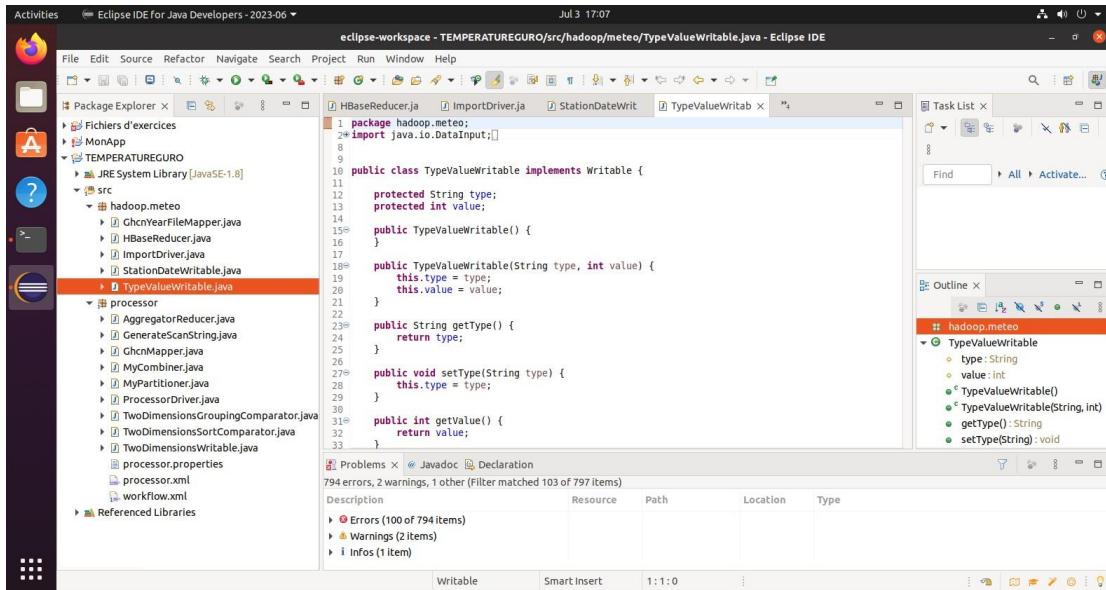


Figure 4.16 – Class "TypeValueWritable"

4.2.7 Lancement d'un modèle MapReduce d'import

Pour lancer le modèle MapReduce d'import, nous avons réalisé le mapper et le reducer correspondants. Ensuite, nous avons créé un driver Java avec une méthode Main. Ensuite, nous avons utilisé la classe Configured pour initialiser la configuration et avons implémenté l'interface Tool de Hadoop. En utilisant Toolrunner.run, nous avons lancé le MapReduce avec la configuration, le driver d'import et les arguments requis.

Dans la méthode Run, nous avons créé un nouveau job Hadoop en spécifiant le jar à déployer sur tous les serveurs à l'aide de setJarByClass. Nous avons donné un nom au job, avons défini les classes de sortie du mapper et du reducer, le format d'entrée du mapper, le chemin d'entrée des fichiers et les classes de mapper et de reducer à utiliser.

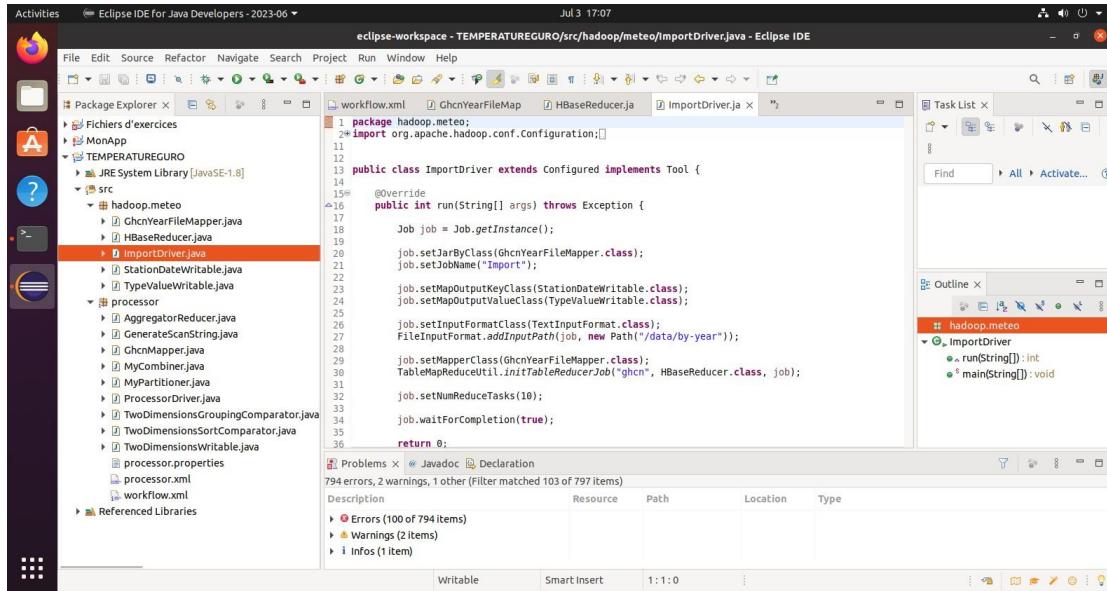


Figure 4.17 – Class "ImportDriver"

Ensuite, nous avons utilisé la méthode `initTableReducerJob` de la classe utilitaire `TableMapReduceUtil` pour spécifier la table, le reducer et le job. Nous avons également défini le nombre de reducers souhaité. Enfin, nous avons lancé le job avec `Job.waitForCompletion`.

Pour exécuter le modèle dans Hadoop, nous avons exporté le code en jar, ajouté le classpath de HBase et spécifié la classe à exécuter. Ensuite, nous avons pu suivre l'avancement du job en utilisant l'URL de suivi.

Après avoir corrigé d'éventuelles erreurs et relancé le job, l'import s'est effectué dans HBase.

4.3 Développement des modèles MapReduce

4.3.1 Lecture des données depuis HBase dans un Mapper

Après l'importation des données, nous avons procédé au calcul et à l'agrégation des données, regroupées selon les groupes définis. Le contenu a été déplacé dans les packages "importer" et "processor". Dans le package "processor", nous avons développé un mapper

appelé "QhcnMapper" qui a lu les données depuis la table Qhcn de HBase. Ce mapper héritait de TableMapper pour une pré-configuration avec HBase.

À ce stade, le mapper était configuré pour renvoyer un Text et un Int en sortie. Comme il s'agissait d'un IntWritable, nous avons implémenté la méthode Map. Dans cette méthode, la première étape consistait à récupérer les entrées. Les clés étaient de type "ImmutableBytesWritable" et les valeurs de type "Result". Nous avons utilisé "key.get" pour obtenir la clé sous forme de tableau d'octets, et la classe utilitaire "Bytes" de hadoop.hbase.util, notamment "Bytes.toString", pour extraire les informations nécessaires.

Nous avons extrait le nom de la station en utilisant "Bytes.toString(key.get, 0, 11)". Cette valeur a été stockée dans une variable "station". Ensuite, nous devions récupérer la date qui se trouvait dans la colonne "date :rien" de la ligne HBase. Nous avons utilisé "value.getValue(Bytes.toBytes("date"), Bytes.toBytes("rien"))" pour obtenir le tableau d'octets correspondant à la valeur de la colonne, puis nous avons converti le tableau en une chaîne de caractères à l'aide de "Bytes.toString". Nous avons également utilisé un "DateFormat" pour convertir la chaîne de caractères en un objet Date.

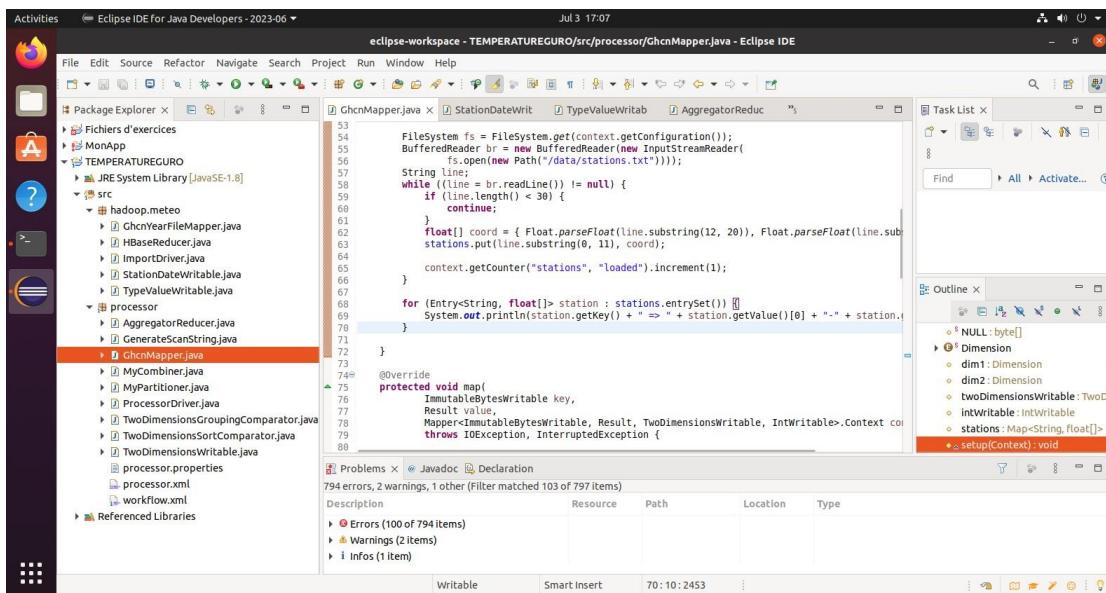


Figure 4.18 – Class "QhcnMapper"

Nous avons géré les exceptions en utilisant un bloc Try-Catch. À ce stade, nous avions

extrait le nom de la station et la date. Nous devions également récupérer les mesures. Pour cela, nous avons utilisé "value.ListCells" pour obtenir toutes les cellules de la valeur. Nous avons itéré sur ces cellules et vérifié si elles appartenaient à la famille "Date". Si c'était le cas, nous les avons ignorées. Nous avons utilisé la classe utilitaire CellUtil pour vérifier si la famille de colonnes correspondait à "Bytes.toBytes("date"))". Nous avons pu ignorer la cellule si c'était le cas, car nous avions déjà traité la date. Si la famille de colonnes était différente, nous avons extrait le nom de la mesure en utilisant "CellUtil.cloneQualifier" et la valeur de la mesure en utilisant "CellUtil.cloneValue". Nous avons converti la valeur en un entier à l'aide de "Bytes.toInt".

Ensuite, nous avons écrit les valeurs en utilisant "Context.write" avec un objet Writable appelé "TwoDimensionsWritable" comme clé. Nous avons modifié notre code pour utiliser cet objet comme clé et avons instancié cet objet directement. Nous avons récupéré les dimensions en utilisant un Enum et les avons stockées dans les variables Dim1 et Dim2.

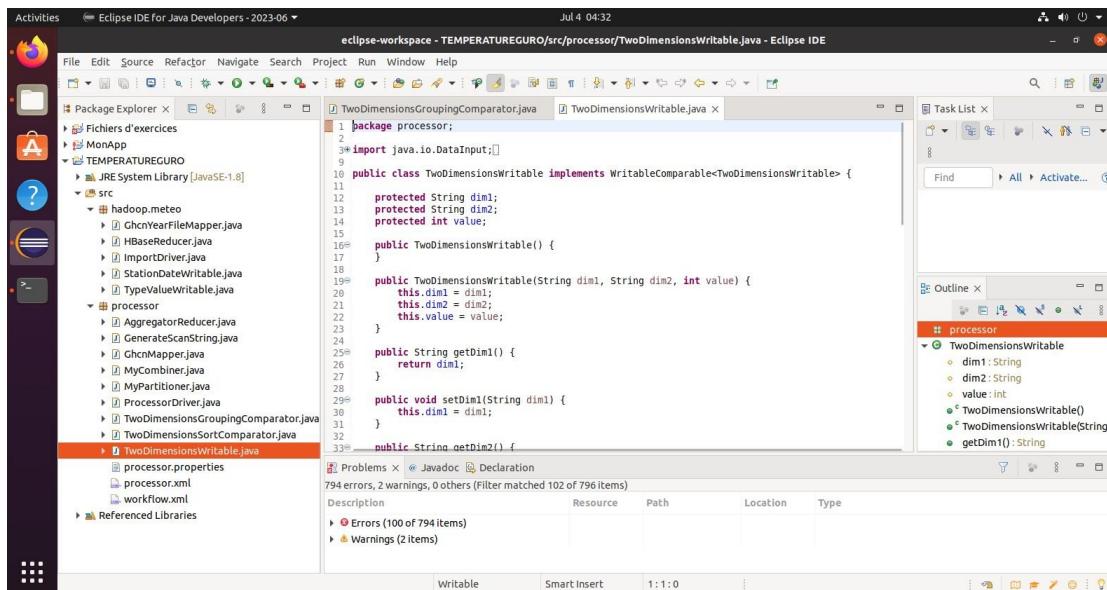


Figure 4.19 – Class "TwoDimensionsWritable"

Nous avions une méthode préparée appelée "getDimension" qui prenait les dimensions et les informations de la ligne HBase pour renvoyer la chaîne correspondante à cette dimen-

sion. Dans la boucle FOR, nous avons écrit le couple clé/valeur en utilisant "context.write" avec le TwoDimensionsWritable comme clé et l'IntWritable (mValue) comme valeur. Ainsi, nous avons terminé notre Mapper et avons pu passer à notre Reducer.

4.3.2 Agrégation des données dans un Reducer

Après avoir implémenté notre Mapper, nous sommes passés à la création du Reducer chargé d'effectuer l'agrégation des données reçues. Nous avons créé un nouveau Reducer, "AggregatorReducer", qui a directement étendu la classe Reducer. En entrée, nous avons pris les sorties du Mapper, c'est-à-dire les TwoDimensionsWritable. Comme valeur, nous avons utilisé un IntWritable, et en sortie, nous n'avons rien écrit dans la clé (NullWritable), mais un Text qui a été directement écrit dans le fichier en tant que valeur.

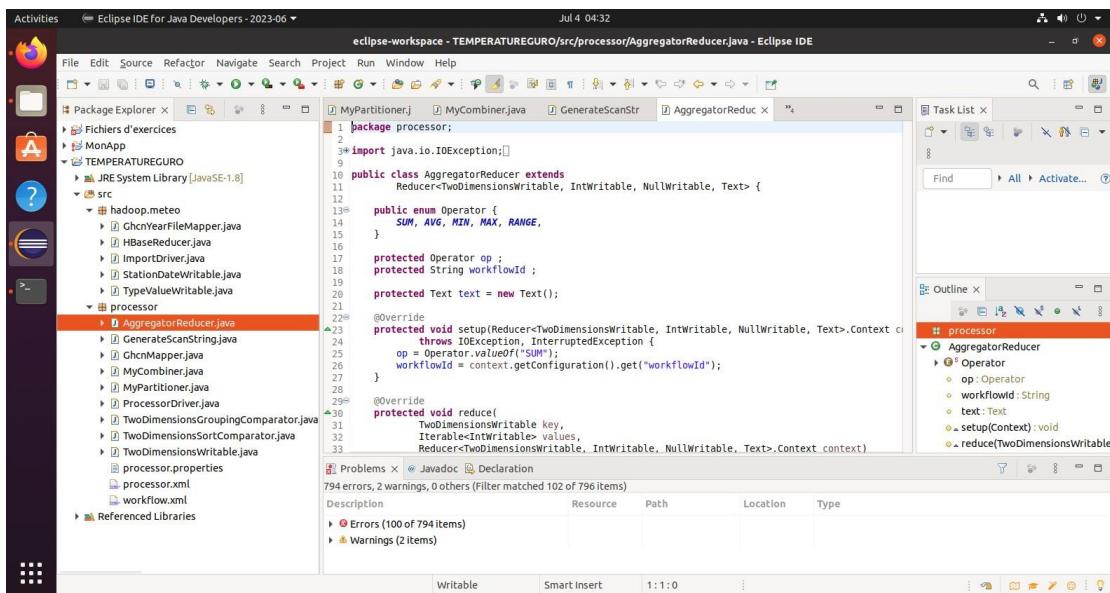


Figure 4.20 – Class " AggregatorReducer "

Nous avons initialisé une variable "op" de type Operator (enum), par exemple `op = Operator.valueOf("SUM")`, pour définir l'opérateur sur lequel nous travaillons. Dans la méthode Reduce, nous avons utilisé un switch sur l'opérateur et avons effectué différentes actions en fonction de l'opérateur sélectionné. Par exemple, pour l'opérateur "SUM", nous

avons parcouru toutes les valeurs et avons ajouté chaque valeur à la variable "result".

Ensuite, nous avons écrit la sortie en utilisant "context.write" avec NullWritable comme première clé et un nouvel objet Text comme deuxième paramètre. Nous avons écrit la Dim1, la Dim2 et le résultat en utilisant les méthodes "getDim1", "getDim2" et "result".

Nous avons également implémenté le cas pour l'opérateur "AVG". Dans ce cas, nous avons utilisé deux variables temporaires : "totalValue" initialisée à zéro et "countValues" initialisée à zéro. Nous avons parcouru toutes les valeurs et avons incrémenté "totalValue" avec la valeur de chaque valeur. Enfin, le résultat a été calculé en dehors de la boucle comme la somme totale de toutes les valeurs divisée par "countValues".

Nous avons ajouté un cas par défaut qui ne faisait rien et laissé le résultat vide. Nous avons également ajouté des "breaks" dans le switch-case. Maintenant, nous pouvons passer à la création d'un Driver pour exécuter notre Mapper et notre Reducer.

4.3.3 Suivie des modèles MapReduce

Lorsque notre processus MapReduce s'est exécuté sur la plateforme, nous avons pu observer les compteurs affichés dans la console à la fin. Parmi ces compteurs, certains étaient particulièrement intéressants, tels que le nombre de Mappers et de Reducers lancés, ainsi que le nombre total d'enregistrements en entrée et en sortie. Par exemple, pour notre Mapper, nous avons récupéré environ 1 483 608 enregistrements en entrée et généré environ 5 millions d'enregistrements en sortie.

Les Reducers ont traité ces 5 millions d'enregistrements en entrée et les ont regroupés en seulement 29 527 groupes. Ainsi, il y a eu 29 527 appels à la méthode Reduce, et le Reducer a renvoyé 29 527 paires clé-valeur. Ces compteurs étaient extrêmement utiles pour suivre l'évolution et le comportement du job MapReduce.

4.3. Développement des modèles MapReduce

58

		Combine input records	0	0	0
		Combine output records	0	0	0
		CPU time spent (ms)	70380	29890	100270
		Failed Shuffles	0	0	0
		GC time elapsed (ms)	536	536	1072
		Input split bytes	69	0	69
		Map input records	1483608	0	1483608
		Map output bytes	170845546	0	170845546
		Map output materialized bytes	180895344	0	180895344
		Map output records	5024869	0	5024869
		Merged Map outputs	0	10	10
		Physical memory (bytes) snapshot	640483328	3004309504	3644792832
		Reduce input groups	0	29527	29527
		Reduce input records	0	5024869	5024869
		Reduce output records	0	29527	29527
		Reduce shuffle bytes	0	180895344	180895344
		Shuffled Maps	0	10	10
		Spilled Records	10049738	5024869	15074607
		Total committed heap usage (bytes)	654311424	3558866944	4213178368
		Virtual memory (bytes) snapshot	1621008384	1604571952	17666727936
		Name	Map	Reduce	Total
		BAD_ID	0	0	0
		CONNECTION	0	0	0
		IO_ERROR	0	0	0
		WRONG_LENGTH	0	0	0

Figure 4.21 – MapReduce Framework

Nous avons pu les consulter à la fois dans la console lors de l'exécution et dans l'interface de suivi dédiée. Tout au long du processus MapReduce, ces compteurs étaient mis à jour en temps réel. Il est important de noter que si une tâche, qu'elle soit Mapper ou Reducer, échouait, le nombre d'enregistrements d'entrée du Map pouvait diminuer brusquement. Dans l'interface de suivi, nous pouvions obtenir des détails sur chaque tâche. Par exemple, si nous examinions le compteur MAP_INPUT_RECORDS, nous pouvions voir le nombre d'enregistrements traités par chaque Mapper. De même, le compteur REDUCE_INPUT_GROUPS montrait la répartition entre les Reducers.

The screenshot shows the Hadoop Job History interface. On the left, there's a sidebar with 'Application' and 'Job' sections, and a 'Tools' section containing 'Overview', 'Counters', 'Configuration', 'Map tasks', and 'Reduce tasks'. The main area displays a table titled 'org.apache.hadoop.mapreduce.TaskCounter' with a single row highlighted: 'MAP_INPUT_RECORDS for job_1488813889515_0023'. The table has columns 'Task' and 'Value'. The task listed is 'task_1488813889515_0023_m_000000' with a value of '1483608'. Below the table, it says 'Showing 1 to 11 of 11 entries'.

Task	Value
task_1488813889515_0023_m_000000	1483608
task_1488813889515_0023_r_000000	0
task_1488813889515_0023_r_000001	0
task_1488813889515_0023_r_000002	0
task_1488813889515_0023_r_000003	0
task_1488813889515_0023_r_000004	0
task_1488813889515_0023_r_000005	0
task_1488813889515_0023_r_000006	0
task_1488813889515_0023_r_000007	0
task_1488813889515_0023_r_000008	0
task_1488813889515_0023_r_000009	0

Figure 4.22 – MAP INPUT RECORDS

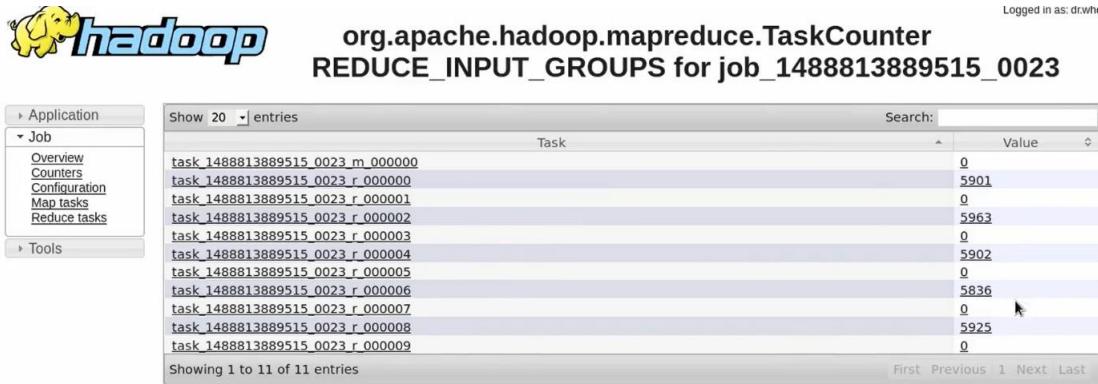


Figure 4.23 – REDUCE INPUT GROUPS

Il était également possible de créer nos propres compteurs en utilisant la méthode `getCounter()` sur le contexte, que ce soit dans le Mapper ou le Reducer. Cela nous permettait de suivre des métriques personnalisées, telles que le nombre de fois où nous entrions dans une clause `catch` ou dans chaque `case` d'un `switch`. Ces compteurs personnalisés nous aidaient à déboguer notre application et à surveiller précisément son fonctionnement.

4.3.4 Déboguage des modèles MapReduce

Pour faciliter le débogage de notre application, il était utile de journaliser les événements et de vérifier ce qui se passait. Nous avions accédé à l'interface Web pour consulter les journaux des Mappers et Reducers, y compris les journaux système tels que `stderr`, `stdout` et `syslog`. En plus des journaux système, nous avions pu ajouter nos propres entrées de journal. Par exemple, en ajoutant des logs dans la classe "AggregatorReducer", nous avions affiché l'évolution des valeurs "totalValues" et "countValues" à l'aide de "System.out.println".

4.3. Développement des modèles MapReduce

60

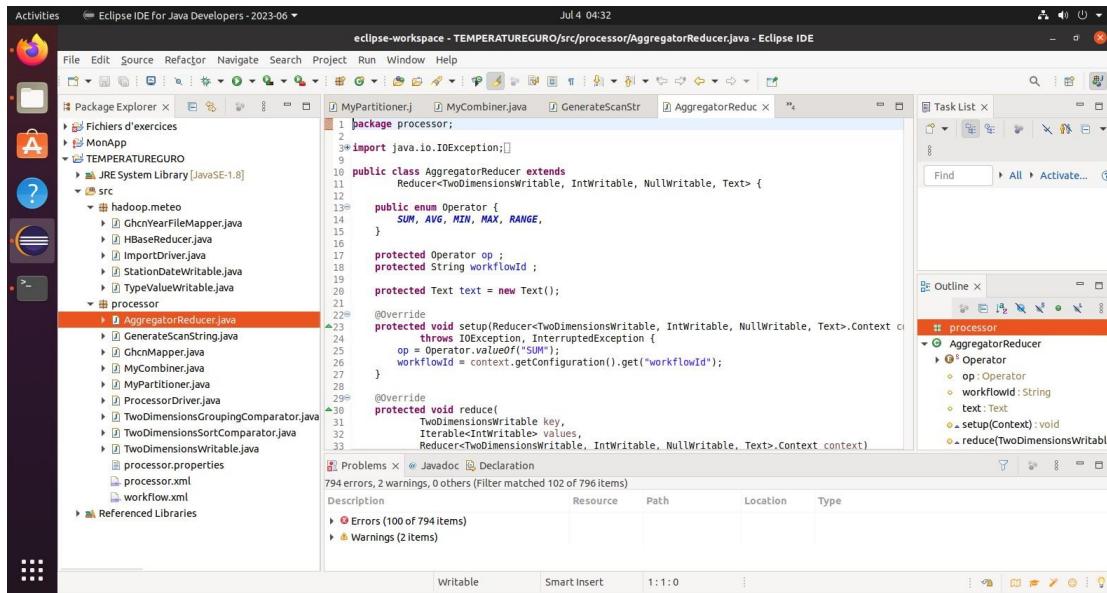


Figure 4.24 – Class " AggregatorReducer "

Une fois que nous avions modifié le code, nous avions recompilé le job et l'avions relancé. Les lignes de journal étaient alors affichées dans les journaux correspondants. En examinant les journaux des Reducers, nous avions pu vérifier les résultats. Il était important de supprimer le répertoire de sortie avant de relancer le job afin d'éviter des problèmes lors des enchaînements de Map-Reduce.

Dans l'interface de suivi, nous avions pu consulter les journaux complets pour observer l'évolution des valeurs et déboguer notre application. Les logs nous avaient permis de suivre les actions des Reducers pendant leur exécution.

Journaliser les événements avait été un moyen efficace de déboguer et de suivre le fonctionnement des Reducers.

4.3.5 Exploration des sources de Hadoop

L'accès aux sources d'Hadoop s'est avéré très utile dans le développement de MapReduce. Cela nous a permis de consulter les signatures des méthodes, les commentaires et

de mieux comprendre leur fonctionnement. Nous avons téléchargé les sources directement depuis le site d'Apache Hadoop, en utilisant les Tarballs correspondants à Apache Hadoop et éventuellement à Apache Hbase .

Après avoir téléchargé et extrait les fichiers sur notre poste, nous avons configuré Eclipse pour lui indiquer l'emplacement des sources. Cela nous a permis, lors de la consultation d'une méthode spécifique, comme la méthode "getInstance" de la classe "Job", d'accéder à la source, nous avons précisé à Eclipse où se trouvaient les sources, et il a automatiquement affiché la source correspondante à la classe.

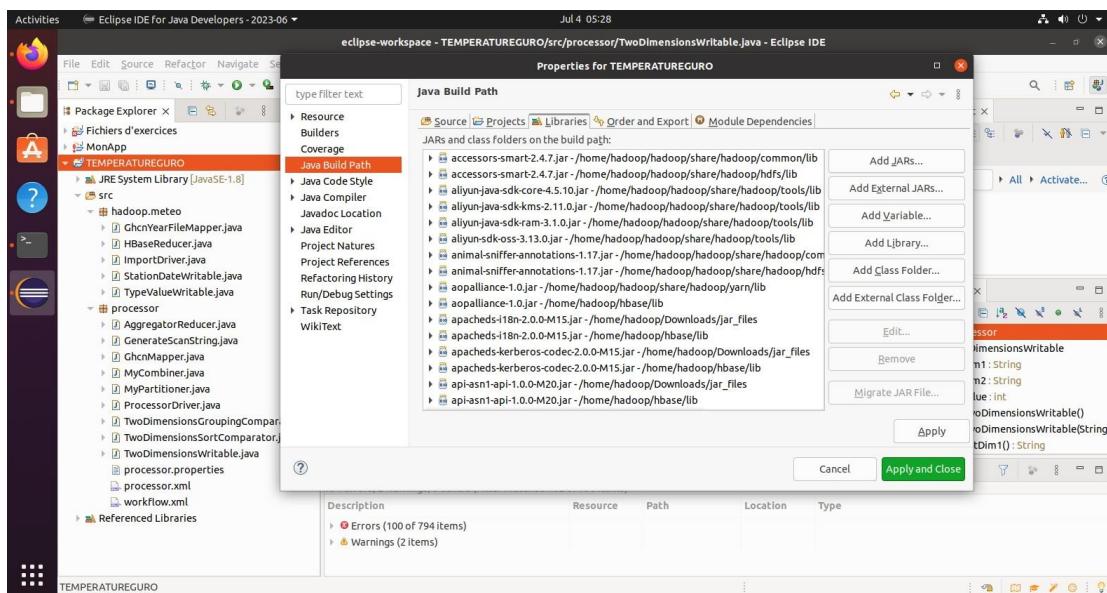


Figure 4.25 – Import Hadoop sources

L'accès aux sources d'Hadoop nous a offert une vision approfondie et précieuse pour le développement de MapReduce, en nous permettant de mieux comprendre le fonctionnement interne de la plateforme.

4.3.6 Réalisation des jointures de données

Dans notre Application, nous avons ajouté la prise en compte des dimensions de latitude et de longitude dans notre Mapper. Pour obtenir ces informations, nous avons téléchargé

le fichier Ghcnd-station.txt, qui contient les détails associés aux stations, telles que les longitudes et latitudes.

ACW00011604	11.1167	-61.7833	10.1	ST JOHNS COOLIDGE FLD		
ACW00011647	17.1333	-61.7833	19.2	ST JOHNS		
AE000041196	25.3330	55.5170	34.0	SHARJAH INTER. AIRP	GSN	41196
AEM00041194	25.2550	55.3640	10.4	DUBAI INTL		41194
AEM00041217	24.4330	54.6510	26.8	ABU DHABI INTL		41217
AEM00041218	24.2620	55.6090	264.9	AL AIN INTL		41218
AF000040934	35.3170	69.0170	3366.0	NORTH-SALANG	GSN	40934
AFM00040934	34.2100	62.2280	977.2	HERAT		40938
AFM00040934	34.5660	69.2130	1791.3	KABUL INTL		40948
AFM00040934	31.5000	65.8500	1000.0	KANDAHAR AIRPORT		40990
AG000060590	36.1167	3.3000	24.0	AF-GER-DAR EL BEIDA	GSN	60390
AG000060590	36.5667	2.8667	397.0	EL-GOLEA	GSN	60590
AG000060611	20.0500	9.6331	561.0	IN-AMENAS	GSN	60611
AG000060680	22.8000	5.4331	1362.0	TAMNIRASSET	GSN	60680
AGE00135030	35.7207	0.6500	50.0	ORAN-HOPITAL MILITAIRE		
AGE00147704	36.9700	7.7900	161.0	ANJABA-CAP DE GARDE		
AGE00147705	36.7800	3.0700	59.0	ALGIERS-VILLE/UNIVERSITE		
AGE00147706	36.8000	3.0300	344.0	ALGIERS-BOUZAREAH		
AGE00147707	36.8000	3.0400	38.0	ALGIERS-CAP CAXINE		
AGE00147708	36.7200	4.0500	222.0	TIZI OUZOU		60395
AGE00147709	36.6300	4.2000	942.0	FORT NATIONAL		
AGE00147710	36.7500	5.1000	9.0	BEJAIA-BOUGIE (PORT)		60401
AGE00147711	36.3697	6.6200	660.0	CONSTANTINE		
AGE00147712	36.1700	1.3400	112.0	ORLEANSVILLE (CHLEF)		
AGE00147713	36.1800	5.4000	1081.0	SETIF		
AGE00147714	35.7700	0.8000	78.0	ORAN-CAP FALCON		
AGE00147715	35.4200	8.1197	863.0	TEBESSA		
AGE00147716	35.1000	-1.8500	83.0	NEMOURS (GHAZAOUET)		60517
AGE00147717	35.2000	0.6300	476.0	SIDI-BEL-ABBES		
AGE00147718	34.8500	5.7200	125.0	BISKRA		60525
AGE00147719	33.7997	2.8900	767.0	LAGHOUAT		60545
AGE00147720	33.6800	1.0600	1320.0	GERVILLE (EL-BAYAOH)		
AGE00147780	37.0800	6.4700	195.0	SKIKDA-CAP BOUGROUNI		
AGE00147794	36.7800	5.1000	225.0	BEJAIA-CAP CARBON		

Figure 4.26 – Les données de station (latitude et longitude)

Après avoir envoyé le fichier Ghcnd-station.txt vers HDFS à l'aide des commandes "WGet" et "hdfs dfs -put", nous avons lu son contenu dans notre code Java. Les données ont été stockées dans une map, ce qui nous a permis de tenir la liste des stations avec leurs longitudes et latitudes en mémoire. Cette approche facilite la jointure des données avec d'autres sources provenant de HBase.

Lors de l'initialisation du Mapper, nous avons chargé les données du fichier dans la map "stations". Ensuite, nous avons réalisé une jointure entre les différentes sources de données, en effectuant des opérations spécifiques pour les dimensions de latitude et de longitude. Les valeurs ont été arrondies à 5 degrés près pour chaque enregistrement, ce qui nous a permis de traiter les groupes de données en prenant en compte ces dimensions.

Cette approche de jointure des données a considérablement amélioré notre compréhension des dimensions et nous a permis d'effectuer une analyse plus approfondie des données

4.3.7 Résolution du problème de secondary sort

Pour résoudre le problème du secondary sort dans notre projet MapReduce, nous avons utilisé le design pattern du même nom. Ce pattern tire parti de l'étape de tri entre le Mapper et le Reducer pour trier des données supplémentaires associées aux clés, en particulier lorsque la valeur est une clé composite.

Nous avons effectué plusieurs opérations pour résoudre ce problème. Tout d'abord, nous avons inclus la valeur supplémentaire à trier dans la clé en créant un comparateur spécifique. Ce comparateur permet à Hadoop de trier les valeurs en fonction de cette clé, y compris la valeur. Ensuite, nous avons indiqué à Hadoop d'utiliser un autre tri pour regrouper les valeurs, même si les clés sont différentes.

Pour cela, nous avons ajouté la notion de valeur (INT VALUE) à notre classe TwoDimensionsWritable. Nous avons effectué la sérialisation et la désérialisation de cette valeur dans les méthodes readFields et write de la classe. De plus, nous avons adapté les méthodes CompareTo, Equals et GetHashCode pour prendre en compte cette valeur lors des comparaisons.

Nous avons également créé des comparateurs spécifiques, tels que TowDimensionsSortComparator et TwoDimensionsGroupingComparator, basés sur WritableComparator. Ces comparateurs nous ont permis de trier les objets TwoDimensionsWritable et de regrouper les clés dans les Reducers en fonction des dimensions.

4.3. Développement des modèles MapReduce

64

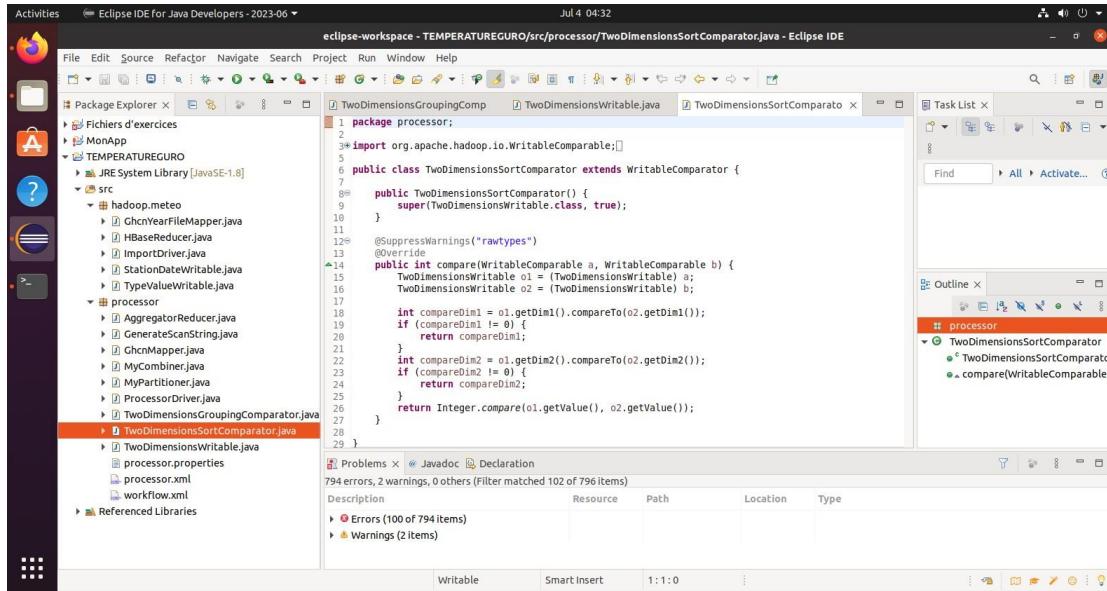


Figure 4.27 – Class « TowDimensionsSortComparator »

Enfin, nous avons mis en place un Partitioner appelé MyPartitioner, qui attribue un numéro de partition en fonction des dimensions de la clé. Cette étape nous a aidés à répartir efficacement les données entre les Reducers. Dans notre driver, nous avons configuré les différentes propriétés, telles que `setGroupingComparator`, `setPartitionerClass` et `setSortComparatorClass`, pour utiliser correctement nos classes personnalisées.

En utilisant ces différentes classes et configurations, nous avons réussi à trier les valeurs dans le Reducer et à les exploiter pour effectuer des opérations supplémentaires, telles que le calcul du minimum, du maximum ou de la plage entre ces valeurs.

Malgré la complexité de la résolution du problème du secondary sort et la nécessité de gérer plusieurs classes Java, cette approche s'est révélée très puissante en exploitant efficacement l'étape de tri du MapReduce.

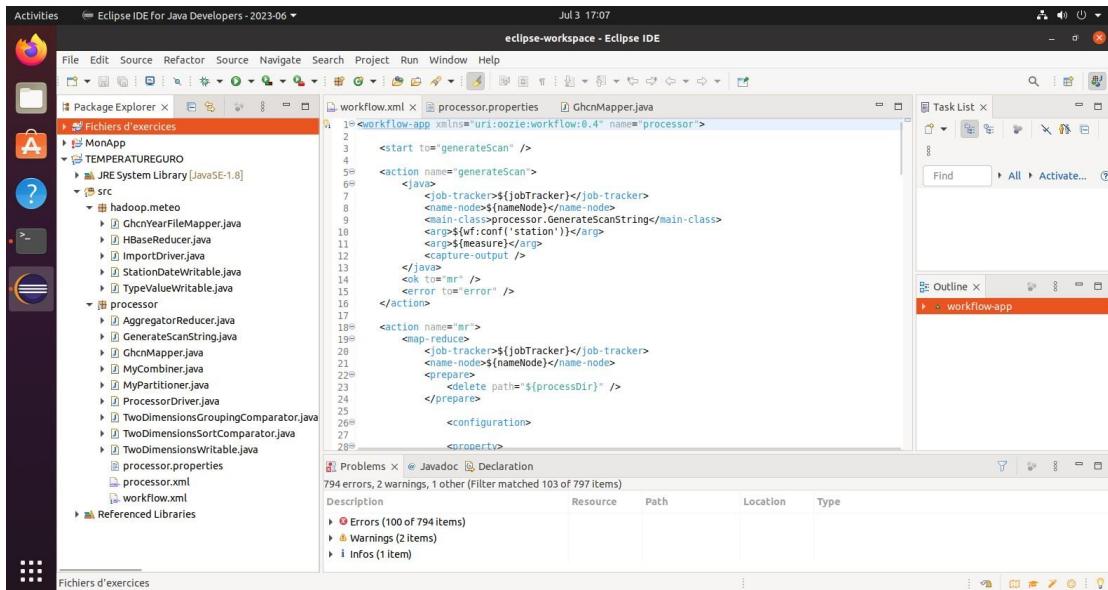
4.4 Industrialisation de l'application

4.4.1 Mise en place d'un workflow Oozie

Nous avons mis en place un workflow Oozie pour automatiser et exécuter séquentiellement les différentes actions de notre processus de traitement de données. Cela nous permet d'industrialiser notre code et d'éviter des tâches manuelles répétitives.

Tout d'abord, nous avons configuré Oozie pour mettre à disposition les librairies nécessaires sur HDFS. Ensuite, nous avons redémarré Oozie pour pouvoir utiliser ces librairies, notamment pour intégrer Sqoop avec Oozie.

Ensuite, nous avons créé un fichier XML appelé "workflow.xml" qui définit notre workflow. Ce fichier suit une structure spécifique, avec des nœuds tels que "start" pour indiquer quelle action est lancée en premier, des nœuds "action" pour chaque action du workflow, et des nœuds "end" et "error" pour gérer les cas de succès ou d'erreur.



```

<?xml version="1.0" encoding="UTF-8"?>
<workflow-app xmlns="uri:oozie:workflow:0.4" name="processor">
  <start to="generateScan" />
  <action name="generateScan">
    <javabinary>
      <job-tracker>$[jobTracker]</job-tracker>
      <name-node>$[nameNode]</name-node>
      <main-class>processor.GenerateScanString</main-class>
      <arg>$[conf:'station']</arg>
      <arg>$[measure]</arg>
      <capture-output />
    </javabinary>
    <ok to="mr" />
    <error to="error" />
  </action>
  <action name="mr">
    <map-reduce>
      <job-tracker>$[jobTracker]</job-tracker>
      <name-node>$[nameNode]</name-node>
      <prepare>
        <delete path="$[processDir]" />
      </prepare>
      <configuration>
        <property>
          <name>mapred.mapper.class</name>
          <value>GhcMapper</value>
        </property>
        <property>
          <name>mapred.reducer.class</name>
          <value>MyCombiner</value>
        </property>
        <property>
          <name>mapred.partitioner.class</name>
          <value>MyPartitioner</value>
        </property>
        <property>
          <name>mapred.input.dir</name>
          <value>${input}</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>${output}</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="copyFromFtp" />
    <error to="error" />
  </action>
  <action name="copyFromFtp">
    <copy-from-fuse>
      <job-tracker>$[jobTracker]</job-tracker>
      <name-node>$[nameNode]</name-node>
      <src>${output}</src>
      <dst>${ftp}</dst>
    </copy-from-fuse>
    <ok to="end" />
    <error to="error" />
  </action>
  <end name="end" />
</workflow-app>

```

Figure 4.28 – Fichier "workflow.xml"

Nous avons commencé par une action simple qui vide le répertoire de sortie, puis nous avons ajouté d'autres actions, telles que l'exécution d'un MapReduce. Les paramètres requis pour chaque action, tels que le job-tracker, le nameNode, etc., ont été définis dans un fichier de propriétés séparé.

Pour l'action MapReduce, nous avons spécifié les configurations nécessaires, comme le nom du job, les classes de Mapper et Reducer, les formats d'entrée et de sortie, les classes de Combiner, Partitioner, GroupingComparator et SortComparator, ainsi que le répertoire de sortie. Nous avons également utilisé l'astuce du nœud "prepare" pour supprimer le répertoire de sortie avant l'exécution.

En résumé, notre workflow Oozie nous permet de lancer automatiquement les actions nécessaires, en séquence, pour exécuter notre code métier, en évitant les tâches manuelles répétitives.

4.4.2 Lancement d'un workflow Oozie

Pour lancer notre workflow Oozie, nous avons effectué quelques étapes préliminaires. Tout d'abord, nous avons créé un répertoire de travail sur HDFS. Ensuite, nous avons placé le fichier workflow.xml dans ce répertoire.

Avant de lancer Oozie, nous avons également préparé les librairies nécessaires. Pour notre code, nous avons exporté un JAR contenant tous les composants requis, puis nous l'avons placé sur HDFS. En ce qui concerne les librairies HBase, nous avons copié tous les fichiers JAR depuis hbase et vers HDFS.

Ensuite, nous avons lancé le workflow en utilisant la commande Oozie, en spécifiant les paramètres nécessaires tels que l'URL d'Oozie, le fichier de configuration "processor.properties", etc. Après avoir redémarré Oozie, nous avons pu suivre l'exécution du

4.4. Industrialisation de l'application

67

job dans l'interface Oozie.

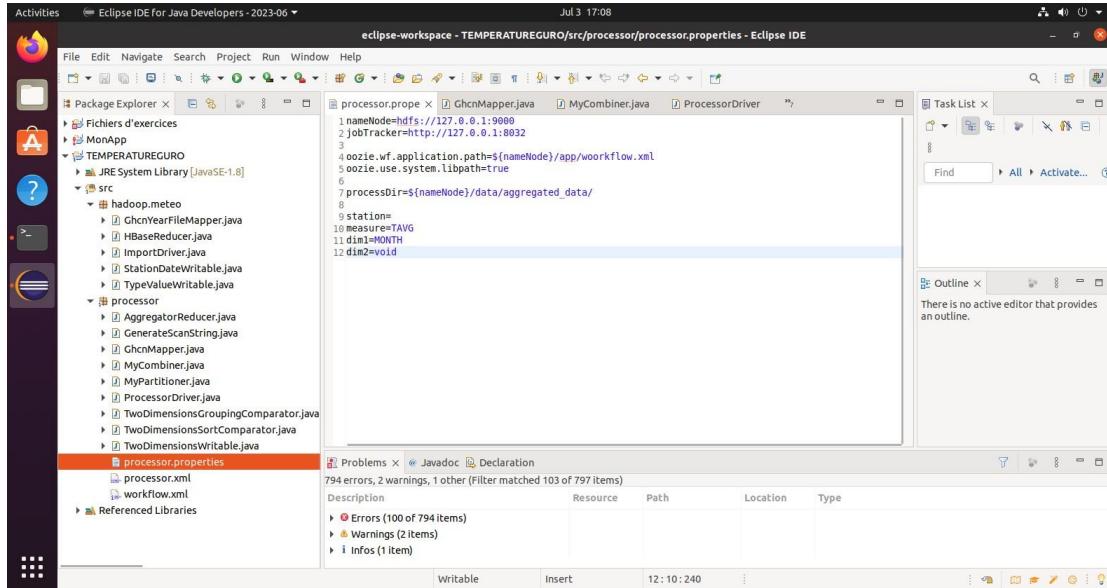


Figure 4.29 – Fichier processor.properties

```
0/oozie -Doozie.https.keystore.file=/home/hadoop/.keystore -Doozie.https.keystore.pass=password -Djava.library.path=
Using CATALINA_BASE: /home/hadoop/oozie/oozie-bin/oozie-server
Using CATALINA_HOME: /home/hadoop/oozie/oozie-bin/oozie-server
Using CATALINA_TMPDIR: /home/hadoop/oozie/oozie-bin/oozie-server/temp
Using JRE_HOME: /usr/lib/jvm/java-8-openjdk-amd64/jre/
Using CLASSPATH: /home/hadoop/oozie/oozie-bin/oozie-server/bin/bootstrap.jar
Using CATALINA_PID: /home/hadoop/oozie/oozie-bin/oozie-server/temp/oozie.pid
hadoop@ubuntu:~/oozie/oozie-bin/bin$ oozie job -oozie http://localhost:11000/oozie -config /home/hadoop/eclipse-workspace/TEMPERATUREREGUO/src/processor/processor.properties -run
```

Figure 4.30 – Job Run oozie

Ces étapes nous ont permis de mettre en place et de lancer notre workflow Oozie pour exécuter les actions séquentiellement selon nos besoins.

4.4.3 Filtrage des données de HBase

Pour filtrer les données qui nous intéressaient dans HBase, nous avons dû spécifier dans le workflow Oozie l'utilisation d'un scan spécifique. Nous avons créé une classe GenerateScanString qui générait un scan avec des filtres basés sur des préfixes ou des colonnes spécifiés en arguments. Ensuite, nous avons exporté le scan sous forme de chaîne en utili-

sant la méthode ProtobufUtil.toScan.

```

1 package processor;
2
3 import java.io.File;
4
5 public class GenerateScanString {
6
7     protected static byte[] MEASURES = Bytes.toBytes("measures");
8     protected static byte[] DATE = Bytes.toBytes("date");
9     protected static byte[] NULL = Bytes.toBytes("-");
10
11    public static void main(String[] args) throws IOException {
12
13        Scan scan = new Scan();
14
15        if (args.length >= 1 && args[0] != null && !"".equals(args[0])) {
16            Filter filter = new PrefixFilter(Bytes.toBytes(args[0]));
17            scan.setFilter(filter);
18        }
19
20        if (args.length >= 2) {
21            scan.addColumn(DATE, NULL);
22            scan.addColumn(MEASURES, Bytes.toBytes(args[1]));
23        }
24        ClientProtos.Scan proto = ProtobufUtil.toScan(scan);
25        String outputScan = Base64.encodeBytes(proto.toByteArray());
26    }
27}

```

Figure 4.31 – Class « GenerateScanString »

Nous avons utilisé la classe GenerateScanString comme une action préalable au MapReduce dans notre workflow Oozie. Cette action appelait la classe Java et passait les paramètres nécessaires. Le résultat était ensuite transmis au MapReduce en utilisant une expression d'élément de langage Oozie pour spécifier la sortie de GenerateScan.

Dans le MapReduce, nous avons récupéré la sortie de GenerateScan et l'avons utilisée comme valeur du paramètre "scan" dans notre configuration. Pour transmettre cette valeur en Java, nous avons écrit dans un fichier properties spécifié par la propriété "oozie.action.output.properties". Oozie s'est chargé de récupérer et lire ce fichier.

4.4.4 Exportation vers MySQL avec Sqoop

Nous avons souhaité exécuter notre "MapReduce" et récupérer les résultats dans MySQL pour les rendre accessibles à PHP. Pour commencer, nous avons créé une base de données dans MySQL et une table appelée "results" avec les colonnes nécessaires, telles

que le nom du job, "dim1" et "dim2".

Pour obtenir le nom du job, nous avons utilisé Oozie et l'avons passé en tant que propriété dans la configuration. De la même manière, nous avons pu passer d'autres propriétés, telles que "dim1" et "dim2", qui ont été utilisées dans le job. Nous avons mis à jour le Mapper et le Reducer pour utiliser ces valeurs à partir de la configuration.

Nous avons également souhaité stocker le "workflowId" et l'inclure dans chaque sortie de texte avant de les renvoyer. Ce format correspondait aux exigences de la table MySQL.

Pour exporter les données de notre fichier HDFS vers MySQL, nous avons utilisé une action Sqoop dans notre workflow Oozie. Nous avons défini les nœuds de succès et d'échec, et nous avons spécifié que l'action était de type "sqoop". Ensuite, nous avons passé les arguments nécessaires à l'exportation, tels que la chaîne de connexion MySQL, le nom d'utilisateur, le mot de passe, le répertoire à exporter et la table cible ("Results"). Nous avons également ajouté le connecteur "mysql jdbc" dans les sharelibs d'Oozie pour pouvoir l'utiliser.

4.4.5 Lancement du workflow avec l'API HTTP REST

Il était important de pouvoir lancer notre workflow Oozie depuis n'importe où, y compris depuis un serveur Web, sans avoir accès à la commande "oozie". Pour cela, nous avons utilisé l'API HTTP REST d'Oozie.

Nous avons ajouté une variable d'environnement appelée "oozie_debug" juste avant la commande "oozie" pour faciliter le processus de lancement. En réalité, cette variable utilisait une API HTTP qui effectuait une requête POST vers l'URL "oozie/v2/jobs" avec l'action "start" et les propriétés spécifiées dans le fichier "Processor.properties" au format XML.

Pour utiliser cette approche, nous avons converti notre fichier "Processor.properties" en un fichier XML("Processor.xml"), en transférant toutes les propriétés de l'autre côté. Une fois le fichier XML créé, nous avons effectué une requête curl pour appeler Oozie.

Nous avons utilisé curl en mode verbeux, spécifié le type de requête HTTP en POST et défini l'en-tête "Content-type" sur "application/xml" pour indiquer à Oozie que nous lui transmettions du XML en entrée. Avec l'option "-d", nous avons fourni le contenu du fichier "home/hadoop/workspace/MonApp/src/processor.xml".

Enfin, nous avons appelé directement l'URL fournie par Oozie, c'est-à-dire l'URL d'Oozie suivie de "/v2/jobs/action start". Oozie nous a renvoyé l'identifiant du job qui a été démarré avec succès. Grâce à cela, nous avons pu appeler Oozie pour lancer nos jobs et suivre leur état en utilisant l'API REST, par PHP.

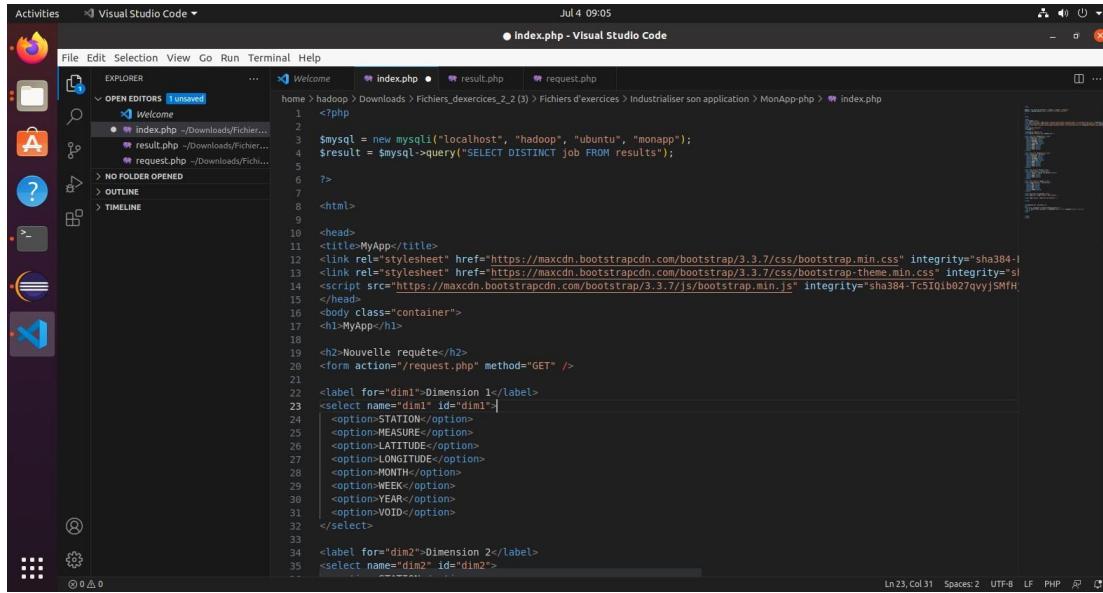
4.4.6 Couplage de l'application avec une interface web

A la suite du travail réalisé précédemment, nous avons développé un projet PHP qui prend en charge les requêtes des utilisateurs et affiche les résultats en les récupérant directement depuis MySQL.

La première page, "index.php", contenait un formulaire qui redirigeait vers "request.php" en transmettant les paramètres tels que "dim1", "dim2", "mesure", "opérateur" et éventuellement "station". Elle affichait également une liste des requêtes déjà exécutées avec leurs résultats stockés dans MySQL.

4.4. Industrialisation de l'application

71



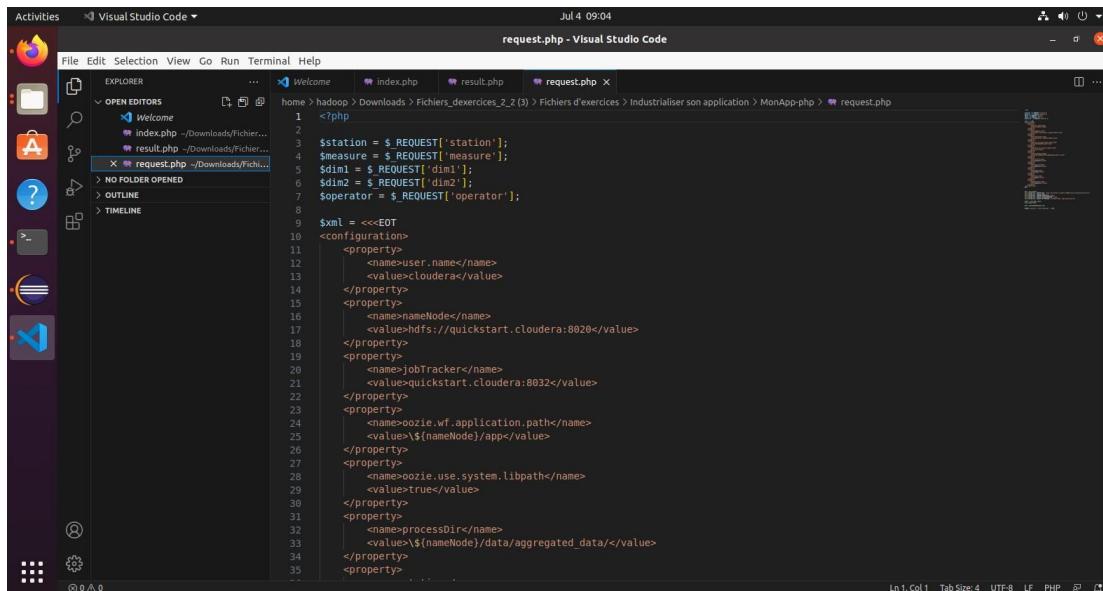
```

Activities ▾ Visual Studio Code ▾
File Edit Selection View Go Run Terminal Help
OPEN EDITORS Welcome index.php result.php request.php
EXPLORER ... Jul 4 09:05
Welcome
index.php ~/Downloads/Fichier...
request.php ~/Downloads/Fichier...
> NO FOLDER OPENED
> OUTLINE
> TIMELINE
home > hadoop > Downloads > Fichiers_d'exercices_2_2 (3) > Fichiers d'exercices > Industrialiser son application > MonApp-php > index.php
1 <?php
2
3 $mysql = new mysqli("localhost", "hadoop", "ubuntu", "monapp");
4 $result = $mysql->query("SELECT DISTINCT job FROM results");
5
6 ?>
7
8 <html>
9
10 <head>
11 <title>MyApp</title>
12 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-I
13 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="s
14 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQibGRC5
15 </head>
16 <body class="container">
17 <h1>MyApp</h1>
18
19 <h2>Nouvelle requête</h2>
20 <form action="/request.php" method="GET" />
21
22 <label for="dim1">Dimension 1</label>
23 <select name="dim1" id="dim1">
24   <option>STATION</option>
25   <option>MEASURE</option>
26   <option>LATITUDE</option>
27   <option>LONGITUDE</option>
28   <option>MONTH</option>
29   <option>WEEK</option>
30   <option>YEAR</option>
31   | <option>VOID</option>
32 </select>
33
34 <label for="dim2">Dimension 2</label>
35 <select name="dim2" id="dim2">

```

Figure 4.32 – Index.PHP

La page "request.php" récupérait ces variables et utilisait la bibliothèque Curl pour appeler Oozie et lancer le workflow avec les paramètres fournis.



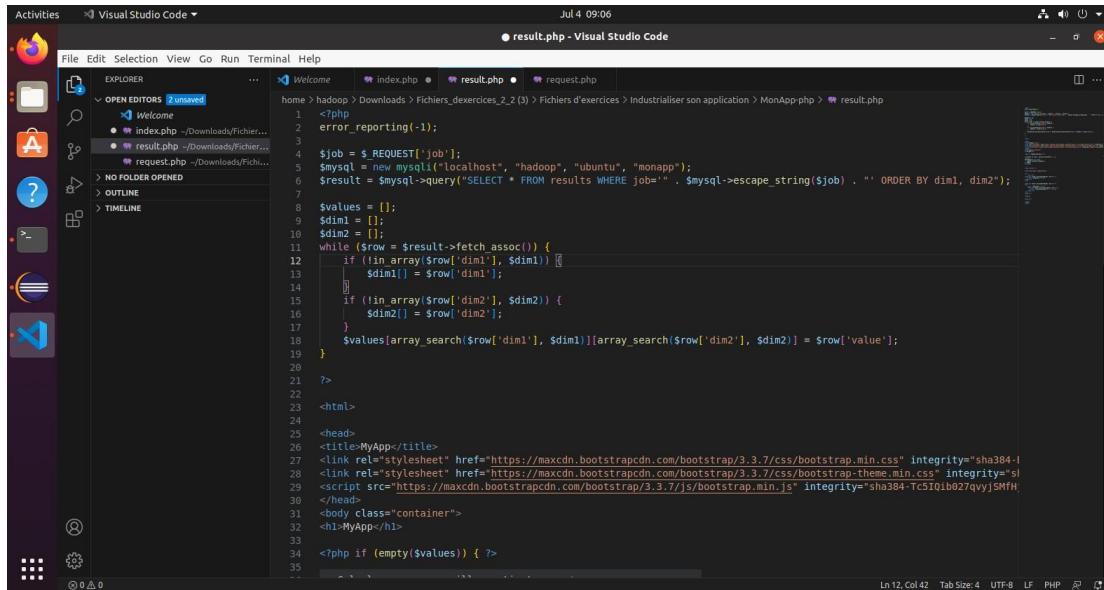
```

Activities ▾ Visual Studio Code ▾
File Edit Selection View Go Run Terminal Help
OPEN EDITORS Welcome index.php result.php request.php
EXPLORER ... Jul 4 09:04
request.php - Visual Studio Code
home > hadoop > Downloads > Fichiers_d'exercices_2_2 (3) > Fichiers d'exercices > Industrialiser son application > MonApp-php > request.php
1 <?php
2
3 $station = $_REQUEST['station'];
4 $measure = $_REQUEST['measure'];
5 $dim1 = $_REQUEST['dim1'];
6 $dim2 = $_REQUEST['dim2'];
7 $operator = $_REQUEST['operator'];
8
9 $xml = <<<EOT
10 <configuration>
11   <property>
12     <name>user.name</name>
13     <value>cloudera</value>
14   </property>
15   <property>
16     <name>nameNode</name>
17     <value>hdfs://quickstart.cloudera:8020</value>
18   </property>
19   <property>
20     <name>jobTracker</name>
21     <value>quickstart.cloudera:8032</value>
22   </property>
23   <property>
24     <name>oozie.wf.application.path</name>
25     <value>${nameNode}/app</value>
26   </property>
27   <property>
28     <name>oozie.use.system.libpath</name>
29     <value>true</value>
30   </property>
31   <property>
32     <name>processDir</name>
33     <value>${nameNode}/data/aggregated_data/</value>
34   </property>
35 </configuration>

```

Figure 4.33 – Request.PHP

La page "result.php" affichait les résultats en les récupérant depuis MySQL et les présentait sous la forme d'un tableau HTML à deux dimensions. Lorsqu'il n'y avait pas encore de résultats disponibles dans MySQL, la page affichait simplement un message indiquant que le calcul était en cours et se rafraîchissait automatiquement toutes les 5 secondes.



The screenshot shows the Visual Studio Code interface with the file 'result.php' open. The code is a PHP script that connects to a MySQL database named 'monapp' on 'localhost'. It retrieves rows from the 'results' table where the 'job' column matches the value from the \$_REQUEST['job'] superglobal. The script then loops through each row, extracting 'dim1' and 'dim2' values and their corresponding 'value' from the \$values array. Finally, it generates an HTML response with a title 'MyApp', a Bootstrap link, and an H1 header 'MyApp'. A conditional check at the bottom ensures the page only displays content if \$values is not empty.

```

    Jul 4 09:06
● result.php - Visual Studio Code

File Edit Selection View Go Run Terminal Help
OPEN EDITORS 3 unsaved
EXPLORER
    home > hadoop > Downloads > Fichiers_d'exercices_2_2 (3) > Fichiers d'exercices > Industrialiser son application > MonApp.php > result.php
    <?php
    error_reporting(-1);
    $job = $_REQUEST['job'];
    $mysql = new mysqli("localhost", "hadoop", "ubuntu", "monapp");
    $result = $mysql->query("SELECT * FROM results WHERE job='".$mysql->escape_string($job)."' ORDER BY dim1, dim2");
    $values = [];
    $dim1 = [];
    $dim2 = [];
    while ($row = $result->fetch_assoc()) {
        if (!in_array($row['dim1'], $dim1)) {
            $dim1[] = $row['dim1'];
        }
        if (!in_array($row['dim2'], $dim2)) {
            $dim2[] = $row['dim2'];
        }
        $values[array_search($row['dim1'], $dim1)][array_search($row['dim2'], $dim2)] = $row['value'];
    }
    ?>
<html>
<head>
<title>MyApp</title>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-IYSqzLwZlF7JzG8PegwAcqP+QWZyCfKXj+DkZJH007XqM8ZdU/Z0XqXhB+QVZo=" crossorigin="anonymous">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="sha384-SPqz7Z7h+gV9Ku/x+ZuOjZcDyEYQqzvWZIaDqOjNQZD8hVgqZDq0rD+uqDw=" crossorigin="anonymous">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5Ioib027qvj5SMFH" crossorigin="anonymous">
</head>
<body>
<div class="container">
<h1>MyApp</h1>
<?php if (empty($values)) { ?>

```

Figure 4.34 – Result.PHP

Enfin, en accédant à "localhost :8000" depuis un navigateur, on arrivait sur la page d'index où l'on pouvait sélectionner les dimensions et les mesures souhaitées pour effectuer une requête.

4.5 Interfaces de l'application et amélioration

4.5.1 Interface d'accueil

L'interface d'accueil se présente sous forme d'une nouvelle requête où l'utilisateur peut choisir les dimensions, les mesures et la station avant de lancer l'exécution.

Les figures ci-dessous montrent les différents choix possibles.

MyApp

Nouvelle requête

Dimension 1 Dimension 2 Mesure Mesure Station Submit Query

Requête exécutées

MEASURE
LATITUDE
LONGITUDE
MONTH
WEEK
YEAR
VOID



Figure 4.35 – Interface de choix de la dimension

MyApp

Nouvelle requête

Dimension 1 Dimension 2 Mesure Mesure Station Submit Query

Requêtes déjà exécutées

STATION
MEASURE
LATITUDE
LONGITUDE
MONTH
WEEK
YEAR
VOID



Figure 4.36 – Interface de choix de la station



Figure 4.37 – Interface de choix de la mesure

4.5.2 Interface du résultat

Après avoir envoyé la requête, on était automatiquement redirigé vers la page "result.php" qui affichait le nom du job et se rafraîchissait toutes les 5 secondes pour vérifier si le résultat du workflow Oozie était disponible comme l'indique les figures suivantes.

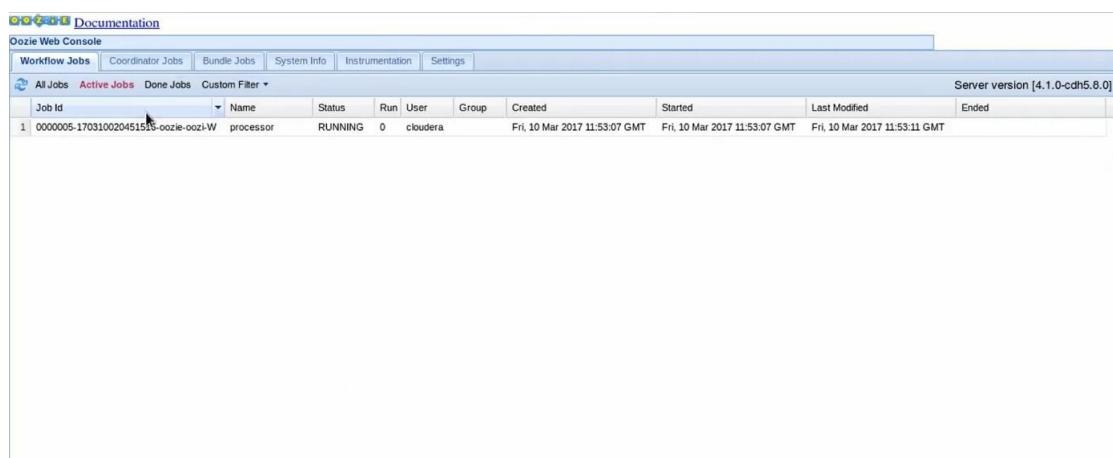


Figure 4.38 – Lancement du job en oozie

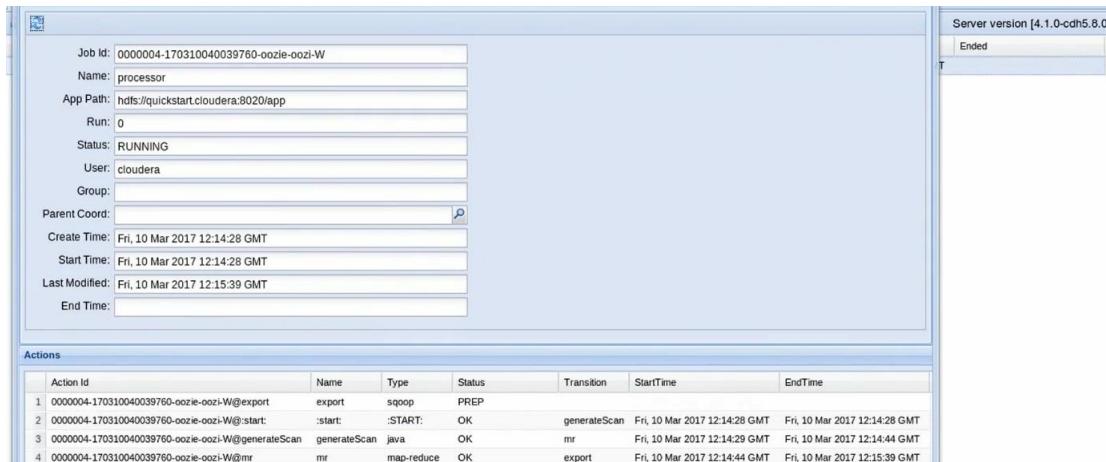


Figure 4.39 – exécution du job dans oozie

Les résultats s'affichent dans un tableau "sparse" représenté par la figure ci-dessous.

MyApp		-10	-15	-20	-25	-30	-35	-40	-45	-5	-50	-55	-60	-65	-70	-75	-80	-85	-90	0	10	15	20	25	30	35	40
-10	272																										
-35	266																260	42									
-40	331	233	293														294	35									
-50	296		256	242	266												272									266	
-55	279		291	270	257	240												-12								260	
-65	268	242	209	255	261	237	239	203									152	19	8						286	275	
-70	267	180	239	221	240	234	193	144	260	142	107						9								267	247	
-75	249	208				187	162										270								226	277	
-80	230																211									257	234
105	252																248									90	34
																		-378								264	273
																			245	190	117	99	-7	-46			

Figure 4.40 – Tableau "sparse"

4.5.3 Amélioration

les résultats n'étaient pas triés selon les dimensions, ce qui pouvait entraîner un affichage désordonné dans le tableau. Une amélioration possible serait d'afficher les résultats sur une carte pour une meilleure visualisation.

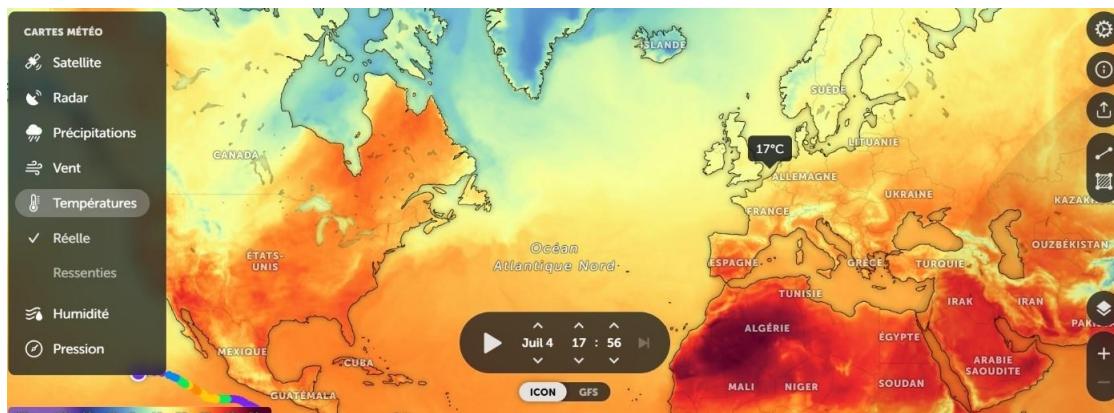


Figure 4.41 – Carte d'affichage

Conclusion

En résumé, ce chapitre de réalisation a permis de concrétiser les différentes étapes du projet, de la collecte des données à la prédition du climat. Les outils et modèles choisis ont été sélectionnés en fonction de leur pertinence, de leur précision et de leur efficacité. Le résultat est une application capable de fournir des prédictions météorologiques fiables pour les 3 prochaines années, aidant ainsi les utilisateurs à prendre des décisions éclairées en fonction des conditions climatiques prévues.

Conclusion et perspectives

Dans le cadre de notre projet de fin d'études, nous avons conçu et développé une Application de prévision météorologique intitulée Weather-app. Le présent rapport couvre l'étude, la conception et l'implémentation de la solution.

En appliquant la méthodologie RAD, nous avons commencé dans un premier lieu par l'étude du contexte général de notre application et l'analyse de l'existant. Puis, nous avons abordé la phase de spécification des besoins et la conception. Ensuite, nous avons rédigé les releases à faire et enfin, nous avons présenté nos tests réalisés.

Au terme de ce stage, nous avons pu achever pratiquement toutes les fonctionnalités, déjà spécifiées malgré de très fortes contraintes du temps et des difficultés techniques. Certes, cette application ne prendra pas fin avec l'achèvement du présent projet, mais elle doit être améliorée en ajoutant d'autres propositions. En effet, il est possible d'ajouter des nouvelles fonctionnalités à notre application.

Finalement, ce projet était une réelle occasion pour intégrer une équipe compétente et professionnelle.

Les situations de blocage, les problèmes techniques et la pression des délais nous ont appris à mieux gérer le temps, à adopter une méthodologie de travail et à respecter les spécifications. En outre, ce projet nous a permis d'approfondir nos connaissances dans les bonnes pratiques de la gestion de projet vu que nous avons eu l'opportunité d'organiser son déroulement dès le début.

Netographie

- [1] Big Data : <https://www.oracle.com/big-data/what-is-big-data/#:~:text=What%20exactly%20is%20big%20data,especially%20from%20new%20data%20sources.>, Consulté le 13 juin 2023.
- [2] DataLake : <https://bigdataanalyticsnews.com/how-does-data-lake-architecture-work/>, Consulté le 15 juin 2023.
- [3] Lambda : <https://www.voltactivedata.com/blog/2014/12/simplifying-complex-lambda-architecture/>, Consulté le 15 juin 2023.
- [4] Kappa : <https://jonboulineau.me/blog/architecture/kappa-architecture>, Consulté le 15 juin 2023.
- [5] Sack Hadoop : <https://www.educba.com/hadoop-stack/>, Consulté le 15 juin 2023.
- [6] ZooKeepe : <https://zookeeper.apache.org>, Consulté le 15 juin 2023.
- [7] Lien de LinkedIn de la société : <https://www.linkedin.com/company/dyno-motivasyystems/>, Consulté le 10 Mars 2023.
- [8] Lien pour la méthodologie RAD : <https://kissflow.com/application-development/rad/rapidapplication-development-methodology>, Consulté le 13 mars 2023.
- [9] Lien de PHP :<https://www.php.net/docs.php>, Consulté le 17 juin 2023.
- [10] Lien de site officiel de Java :<https://www.java.com/en/>, Consulté le 17 juin 2023.
- [11] Lien de site officiel de Mysql : <https://www.mysql.com/>, Consulté le 17 juin 2023.

[12] Lien de site officiel de Hadoop : <https://hadoop.apache.org/>, Consulté le 17 juin 2023.

[13] Lien de site officiel de Hbase : <https://hbase.apache.org>, Consulté le 17 juin 2023.

[14] Lien d'Eclipse : <https://eclipseide.org>, Consulté le 17 juin 2023.

[15] Lien de VS Code : <https://code.visualstudio.com>, Consulté le 17 juin 2023.

Resumé

Nous avons développé une application web conviviale appelée Weather-app qui permet aux utilisateurs d'explorer et d'analyser les variations de température à travers le monde. Grâce à notre interface intuitive, les utilisateurs peuvent sélectionner une période et une région géographique spécifiques, ce qui génère des tableaux et des graphiques informatifs. Notre objectif principal était de faciliter la compréhension des tendances climatiques passées, de différencier les variations régionales et saisonnières, et de fournir des indications pour les prévisions et les projections climatiques futures.

Ce projet a été réalisé dans le cadre de notre projet de fin d'études en Big Data & Analyse de Données à l'Université iTeam pour l'année universitaire 2022/2023.

Mots-clés : Big Data, Hadoop, MySQL, développement web, UML, Hbase...

Abstract

As part of our final year project, we developed a user-friendly web application called Weather-app that allows users to explore and analyze temperature variations worldwide. With our intuitive interface, users can select specific time periods and geographical regions, generating informative tables and graphs. Our main objective was to facilitate the understanding of past climate trends, differentiate regional and seasonal variations, and provide insights for climate forecasts and projections in the future.

This project was carried out as part of our final year project in Big Data & Data Analysis at iTeam University during the academic year 2022/2023.

Keywords : Big Data, Hadoop, MySQL, Web development, UML, Hbase...