

REPOBLIKAN'I MADAGASIKARA
« Fitiavana– Tanindrazana – Fandrosoana »
---00000---

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
---00000---



INSTITUT SUPERIEUR DE TECHNOLOGIE D'ANTSIRANANA
---00000---



TRAVAUX DE FIN D'ETUDES
POUR L'OBTENTION DU DIPLOME DE TECHNICIEN SUPERIEUR SPÉCIALISÉ

Mise en place d'une plateforme web dynamique pour le partage des tableaux de bord des programmes issu de DHIS2

Par BEZARA Kenn Keren

Travaux réalisés au Ministère de la Santé Publique
Lieu Antananarivo

ECOLE DU GENIE INDUSTRIEL
MENTION TELECOMMUNICATION
(PARCOURS Administration des Réseaux)

PROMOTION ' **MPANDRESY'23** '
Antsiranana, Décembre 2023

AVANT-PROPOS

Au cœur de la mission du Ministère de la Santé Publique, l'impératif de renforcer l'accès à l'information sanitaire et d'optimiser la gestion des programmes de santé s'est imposé comme un axe fondamental. La présente initiative vise à répondre à ce besoin crucial en mettant en place une plateforme web dynamique dédiée au partage des tableaux de bord issus du DHIS2.

REMERCIEMENTS

J'exprime, ici, toute ma gratitude pour l'achèvement de cette importante étape de ma vie en vue d'obtenir le diplôme de technicien supérieur spécialisé en Administration de Réseaux, aux personnes citées ci-après :

- Le Directeur Général de l'IST-D : Docteur TSIMITAMBY Briand,
- Le Directeur des Etudes, de la Planification et du Système d'Information : Docteur Fidelis Adolphe ANDRIAMIZARASOA,
- Le Responsable de Mention Technologie de Communication (MTC) : Docteur RAZAFINDRADINA Henri Bruno,
- Le Chef de Service SEMIDSI : Docteur ARISON Daniel
- Mon Encadreur Professionnel : Docteur RAKOTONDRABE Miora Harivony,
- Mon Encadreur Pédagogique : Monsieur VELOSON
- Le Responsable du parcours Administration de Réseaux (AdR): Monsieur RAONIZAFINANTENAINA Angelico

CAHIER DE CHARGES

Entreprise : Ministère de la Santé Publique (MSANP)

Direction des Études, de la Planification et du Système d'Information (DEPSI)

Service de l'Exploitation, de la Maintenance Informatique et du Développement du Système Informatique (SEMIDSI)

Thème : Mise en place d'une plateforme web dynamique pour le partage des tableaux de bord des programmes issu de DHIS2

Domaine :

Système d'informations géographiques

Durée :
12 semaines

Candidats \ Parcours :

BEZARA Kenn Keren \ Administration de Réseaux

Contexte :

L'information sanitaire et les bases factuelles sur la santé doivent jouer un rôle majeur dans l'orientation des flux de ressources et des programmes de santé. La production et la consolidation de l'information et des bases factuelles sur les questions de santé publique, y compris la publication de rapports comparatifs et analytiques et la conduite d'études de recherche sur les thèmes clés de santé publique, revêtent une importance cruciale.

C'est la raison pour laquelle il est nécessaire de mettre en place et d'entretenir un système solide, capable de produire l'information, les bases factuelles et les connaissances requises pour analyser, comprendre et faire fonctionner efficacement les systèmes de santé.

Problématique :

Un déficit notable est enregistré dans le pays concernant un déficit de connaissances en santé. L'un des goulots à cette carence est l'accès insuffisant à l'information sanitaire disponible à l'échelle nationale et internationale

C'est pourquoi, il est nécessaire de trouver une stratégie visant à améliorer la diffusion et le partage de l'information, des bases factuelles et des connaissances.

Objectif général :

Réaliser une plateforme web pour la diffusion des tableaux de bord des programmes au niveau du ministère de la santé publique.

Objectifs spécifiques :

- Comprendre le fonctionnement des différents programmes du DHIS2
- Identifier les éléments (indicateurs) du tableau de bord
- Réaliser la plateforme web

Travaux demandés :

- Étude de l'organisation de l'entreprise (1ère semaine) ;
- Étude et analyse du réseau et des services existants (2ème semaine) ;
- Analyse et modélisation du système de collecte de données et de partage de données
- Identification des forces et des faiblesses du système existant
- Description de l'organisation des données traitées par les différents programmes
- Formulation des différents indicateurs choisis
- Choix des outils et méthodes

- Modélisation de la solution
- Réalisation avec les outils choisis
- Test et validation

Encadreurs au sein de l'Entreprise :

Dr RAKOTONDRABE Miora Harivony (rakmiora@gmail.com)

TABLE DES MATIERES

| | |
|---|------------------------------|
| AVANT-PROPOS | I |
| REMERCIEMENTS | II |
| CAHIER DE CHARGES | III |
| TABLE DES MATIERES | V |
| LISTE DES ACRONYMES | VII |
| LISTE DES FIGURES | VIII |
| LISTE DES TABLEAUX..... | IX |
| INTRODUCTION | 1 |
| CHAPITRE 1. PRESENTATION DE L'ENTREPRISE..... | 2 |
| 1.1. HISTORIQUES | 2 |
| 1.2. ORGANISATION ET STRUCTURE..... | 3 |
| CHAPITRE 2. METHODOLOGIE | 5 |
| 2.1. PLANIFICATION..... | 5 |
| 2.2. ETUDES DE FAISABILITE..... | 6 |
| 2.2.1. Etudes de l'existant | 6 |
| 2.2.1.1. Plateforme DHIS2 | 7 |
| a) Fonctionnement de DHIS2..... | 8 |
| b) Points forts..... | 9 |
| c) Points faibles..... | 10 |
| 2.2.1.2. Les fonctionnalités des éléments du tableau de bord de DHIS2 | 10 |
| 2.2.1.3. WEB API de DHIS2..... | 11 |
| a) les APIs de DHIS2..... | 11 |
| b) Les types d'affichage d'objet du tableau de bord..... | 11 |
| 2.2.2. Etudes de besoins..... | 14 |
| 2.2.2.1. Application de modélisation..... | 14 |
| 2.2.2.2. Editeur de texte et langages | 14 |
| a) Editeur..... | 14 |
| b) Langages..... | 14 |
| 2.2.2.3. Modules externes (Frameworks) | 15 |
| 2.2.2.4. Hébergement | Error! Bookmark not defined. |
| a) Hébergement sur un serveur en ligne..... | 16 |
| b) Hébergement sur un serveur local..... | 16 |
| 2.3. MODELISATION..... | 17 |
| 2.3. 1. Diagramme UML : cas d'utilisation | 17 |

| | |
|---|-----------|
| 2.3. 2. Modélisation de la base de donnée (MERISE)..... | 18 |
| 2.3.2. 1. MCD : Modèle Conceptuel des Données | 19 |
| 2.3.2. 2. MLD : Modèle Logique des Données..... | 20 |
| 2.3.2. 3. MPD : Modèle Physique de Données | 20 |
| 2.3. 3. Composition et Maquettage | 21 |
| 2.3.3.1. Les utilisateurs (visiteurs):..... | 21 |
| 2.3.3.2. Les fonctionnalités : | 21 |
| 2.3.3.3. Les maquettes | 21 |
| CHAPITRE 3. REALISATION | 23 |
| 3.1. DEVELOPPEMENT DE L'APPLICATION..... | 23 |
| 3.1. 1. Partie Serveur (Django) : | 23 |
| 3.1.1. 1. Création de la base de données :..... | 23 |
| 3.1.1. 2. Communication entre Javascript et Django : | 24 |
| 3.1.1. 3. Formatage de données pour l'affichage de Graphique : | 26 |
| 3.1.1. 4. Formatage de données pour l'affichage de Carte Géographique : | 27 |
| 3.1.1. 5. Formatage des données de chaque élément d'un tableau de bord :..... | 29 |
| 3.1.1. 6. Récupération des données de chaque élément du tableau de bord..... | 30 |
| 3.1.1. 7. Récupération des données analytiques d'un élément du tableau de bord: | 31 |
| 3.1.1. 8. Formatage des données pour l'affichage d'un tableau :..... | 31 |
| 3.1.1. 9. Envoi de requêtes et récupération des données via l'API : | 32 |
| 3.1. 2. Partie HTML et Javascript : | 34 |
| 3.1.2.1. Validation des informations de l'utilisateur vers la base de données : | 34 |
| 3.1.2.2. Envoi et récupération de données avec le Serveur (Django) : | 35 |
| 3.1.2.3. Affichage de la graphique : | 37 |
| 3.1.2.4. Affichage du tableau..... | 40 |
| 3.1.2.5. Récupération des informations sur les éléments du tableau de bord : | 42 |
| 3.1.2.6. Affichage de la Carte Géographique : | 45 |
| 3.1.2.7. Création la barre de navigation:..... | 48 |
| 3.1.2.8. Mise en place des contenus de la page : | 49 |
| 3.1. 3. Résultats : | 50 |
| 3.2. TESTS | 52 |
| CONCLUSION..... | 56 |
| REFERENCES..... | 57 |
| ANNEXES | 58 |

LISTE DES ACRONYMES

CSB : Centre de Santé de Base

CSRF : Cross-Site Request Forgery

CSV : Comma-Separated Values

DEPSI : Direction des Etudes, de la Planification et du Système d'Information

DHIS : District Health Information System

HISP : Health Information System Programme

HTTP : HyperText Transfer Protocol

JSON : JavaScript Object Notation

ORM : Object-Relational Mapping

PNG : Portable Network Graphics

SEMIDSI : Service de l'Exploitation, de la Maintenance Informatique et du Développement du Système d'Information

SIS : Système d'Information Sanitaire

SVG : Scalable Vector Graphics

UML : Unified Modeling Language

USAID : United States Agency for International Development

XML : Extensible Markup Language

LISTE DES FIGURES

| | |
|---|----|
| Figure 1.1 : Organigramme du Ministère de la Santé Publique..... | 3 |
| Figure 1.3: Icône de Online Gantt..... | 6 |
| Figure 1.2 : Diagramme de Gantt de la planification avec Online Gantt..... | 6 |
| Figure 2.1: Icône de la plateforme DHIS2 | 7 |
| Figure 2.2 : Tableau de bord de la plateforme DHIS2 | 7 |
| Figure 2.3 : Graphique de l'application DHIS2..... | 12 |
| Figure 2.4 : Carte géographique de l'application DHIS2 | 12 |
| Figure 2.5 : Tableau croisé dynamique de DHIS2 | 13 |
| Figure 2.6 : Objet de type Texte de DHIS2 | 13 |
| Figure 2.7 : Diagramme de cas d'utilisations | 18 |
| Figure 2.8 : Modèle conceptuel de données du formulaire de renseignement personnel | 19 |
| Figure 2.9 : Modèle logique de données du formulaire de renseignement personnel..... | 20 |
| Figure 2.10 : Modèle physique de données du formulaire de renseignement personnel | 20 |
| Figure 2.11: Maquette du formulaire de renseignement personnel de la plateforme..... | 21 |
| Figure 2.12: Maquette du Tableau de Bord de la plateforme..... | 22 |
| Figure 3.1: Algorithme de la fonction <code>api_get_data()</code> | 33 |
| Figure 3.2: Formulaire de renseignement personnel sur l'utilisateur | 50 |
| Figure 3.3: Tableau de bord initial de la nouvelle plateforme WEB | 51 |
| Figure 3.4: Tableau de bord manipulé de la nouvelle plateforme WEB..... | 51 |
| Figure 3.5: Tableau de bord filtré de la nouvelle plateforme WEB..... | 52 |
| Figure 3.6: Page d'authentification de DHIS2 du Ministère de la Santé Publique..... | 52 |
| Figure 3.7: Application Tableau de bord de DHIS2 du Ministère de la Santé Publique | 53 |
| Figure 3.8: Page d'authentification de DHIS2 Demo | 53 |
| Figure 3.9: Application Tableau de bord de DHIS2 Demo | 54 |

LISTE DES TABLEAUX

| | |
|--|----|
| Tableau 1.1: Planification des tâches..... | 5 |
| Tableau 2.1: L'entité visiteur..... | 19 |
| Tableau 2.2: L'entité soumission..... | 19 |
| Tableau 3.1: Différence entre DHIS2 Demo et de MSANP..... | 54 |

INTRODUCTION

L'information sanitaire constitue la pierre angulaire de toute prise de décision stratégique dans le domaine de la santé. Cependant, il est devenu évident qu'un accès limité à cette information entrave la capacité du Ministère de la Santé Publique à répondre efficacement aux besoins de la population. Face à ce constat, notre projet s'engage à déployer une solution novatrice basée sur le DHIS2, visant à optimiser la diffusion et le partage des données de santé au sein du ministère.

Dans le cadre de ce projet, nous explorerons le fonctionnement détaillé des programmes du DHIS2, identifierons les éléments clés des tableaux de bord, et mettrons en œuvre une plateforme web intuitive pour faciliter l'accès aux informations cruciales. Notre démarche s'appuiera sur une analyse approfondie de l'organisation des données, la définition d'indicateurs pertinents et l'adoption d'outils et de méthodes adaptés.

Chapitre 1. PRESENTATION DE L'ENTREPRISE

1.1. HISTORIQUES

Le Ministère de la Santé Publique est une entité gouvernementale chargée de la supervision, de la réglementation, et de la promotion des politiques et des programmes liés à la santé publique au niveau d'un pays. Les principales responsabilités d'un Ministère de la Santé Publique comprennent la protection et l'amélioration de la santé de la population, la prévention des maladies, la promotion de modes de vie sains, la gestion des systèmes de soins de santé, et la coordination des interventions en cas d'épidémies ou de situations d'urgence sanitaire.

Pour partager plus d'informations sanitaires, le Ministère de la Santé Publique de Madagascar vise à disposer d'une plateforme WEB qui reproduira le tableau de bord de l'application DHIS2 qui est le SIS du Ministère de la Santé Publique de Madagascar actuellement.

En 2015, le Ministère de la Santé Publique a exprimé son intention de réformer le SIS. Un arrêté ministériel de juin 2015 a établi l'obligation pour toutes les formations sanitaires, publiques et privées, de produire des rapports périodiques et de les intégrer dans le SIS national.

En 2016, le Ministère de la Santé Publique, avec le soutien de l'USAID via le projet MEASURE Evaluation, a évalué la performance du SIS. Une recommandation clé était d'assurer l'accessibilité et l'intégration des données à tous les niveaux du système de santé. En réponse, le DHIS2 a été adopté comme entrepôt de données sanitaires dans le plan stratégique 2018-2022.

En 2017, création d'un sous-comité SIS pour une vision plus large du système d'information sanitaire, avec coordination et harmonisation des données de qualité.

En 2018, le Ministère de la Santé Publique à Madagascar a défini un plan stratégique (2018-2022) visant à renforcer le SIS. L'objectif était d'avoir un SIS performant, pérenne et intégré.

Le 23 avril 2018, DHIS2 a été officiellement lancé comme la plateforme unique de collecte, de rapportage, et d'entreposage des données de santé de routine au Ministère de la Santé Publique.

De juin 2018 à février 2019, 325 participants formés en DHIS2, ciblant les utilisateurs finaux et les décideurs.

À partir de 2019, mise à jour des outils de collecte RMA Communautaire et Hospitalier intégrant les besoins des décideurs et des programmes, exploités dans DHIS2.

Phase de test en décembre 2019 dans la Région de Menabe pour les modules DHIS2 hospitalier et communautaire.

Évaluation en juin 2020 concluant que les modules DHIS2 sont prêts à être mis à l'échelle nationale.

En 2020, les directeurs centraux sont formés sur le module d'analyse et d'utilisation de données DHIS2.

Entre 2021 et 2022, 145 responsables de base de données des directions centrales et des régions PARN sont formés sur le même module.

1.2. ORGANISATION ET STRUCTURE

Le DECRET N° 2020-1286 fixe les attributions du Ministère de la Santé Publique ainsi que l'organisation générale de son Ministère.

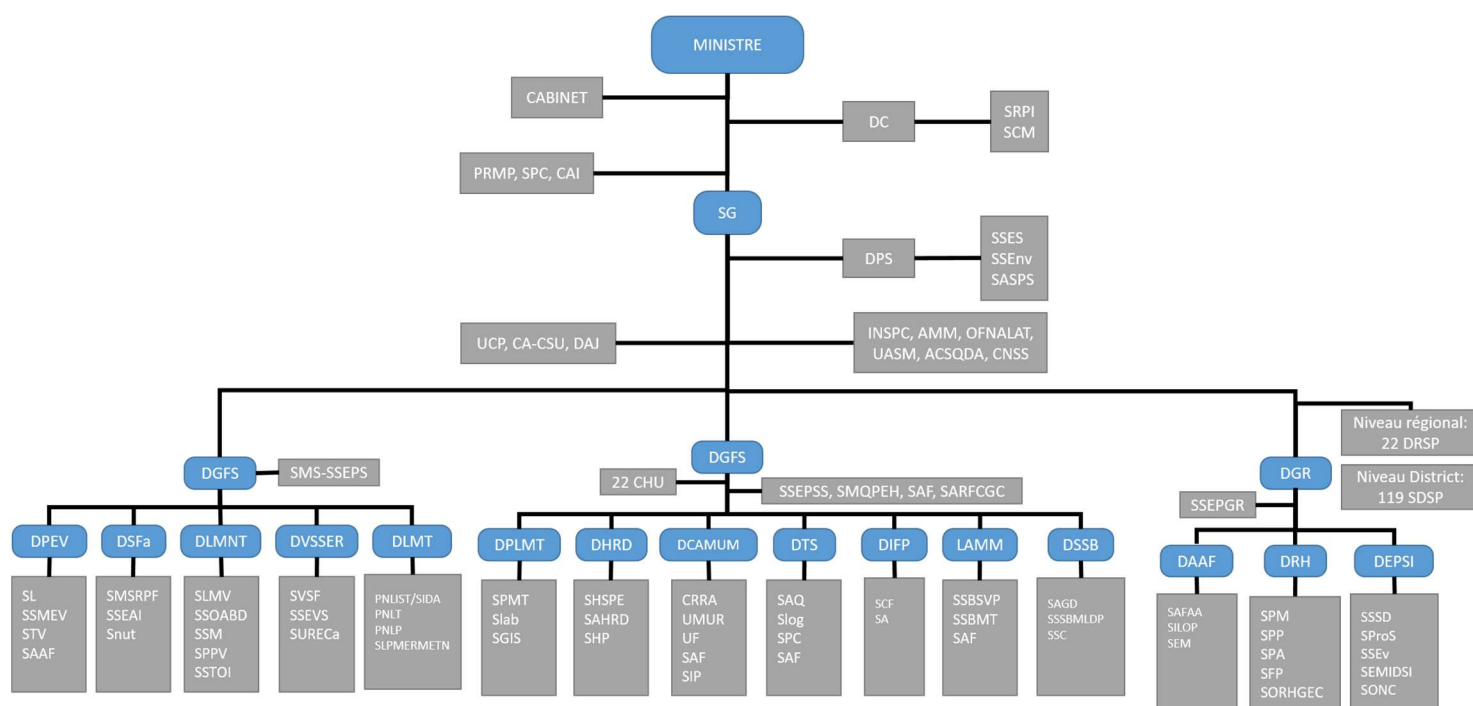


Figure 1.1 : Organigramme du Ministère de la Santé Publique

DEPSI joue un rôle central dans la gestion et l'évolution du système de santé de Madagascar. Elle est impliquée dans la conception, la planification, et le développement des systèmes d'information, en particulier le SIS. Ces principales responsabilités sont la conception et mise

en œuvre des systèmes d'information sanitaires, élaboration de stratégies de collecte, de traitement et d'analyse des données de santé et la coordination des différentes initiatives en cybersanté.

SEMIDSI est une entité essentielle qui se concentre sur l'exploitation, la maintenance et le développement continu des systèmes informatiques. Ces principales responsabilités sont l'exploitation des systèmes informatiques existants, maintenance préventive et corrective du matériel et des logiciels, développement continu des systèmes informatiques en réponse aux besoins évolutifs.

Chapitre 2. METHODOLOGIE

2.1. PLANIFICATION

Pour bien avancer dans un projet, il faut planifier et définir les tâches à réaliser. Chaque tâche aura un intervalle de temps correspondant.

Tableau 1.1: Planification des tâches

| Tâches | Intervalle de date |
|---|--------------------|
| Intégration | 19 Sep – 23 Sep |
| Etude de l'organisation | 25 Sep – 30 Sep |
| Etude du réseau et des services existants | 02 Oct – 07 Oct |
| Recherche et définition du projet | 09 Oct – 10 Oct |
| Etude détaillée du sujet de projet | 11 Oct – 01 Nov |
| Documentation sur DHIS2 | 11 Oct – 25 Oct |
| Tests avec l'API de DHIS2 | 26 Oct – 01 Nov |
| Codage et mise en place du plateforme WEB | 02 Nov – 15 Nov |
| Modélisation, Maquette, Algorithms | 02 Nov – 08 Nov |
| Codage des scripts | 08 Nov – 15 Nov |
| Vérification et validation | 16 Nov – 17 Nov |
| Hébergement | 17 Nov – 22 Nov |

Pour une visualisation plus graphique, je vais utiliser le diagramme de Gantt. J'ai trouvé une application facile à utiliser qui fait des rendus agréables et jolis. **Online GANTT** est un outil de conception d'un diagramme de Gantt en ligne et gratuit.



Figure 1.2: Icône de Online Gantt

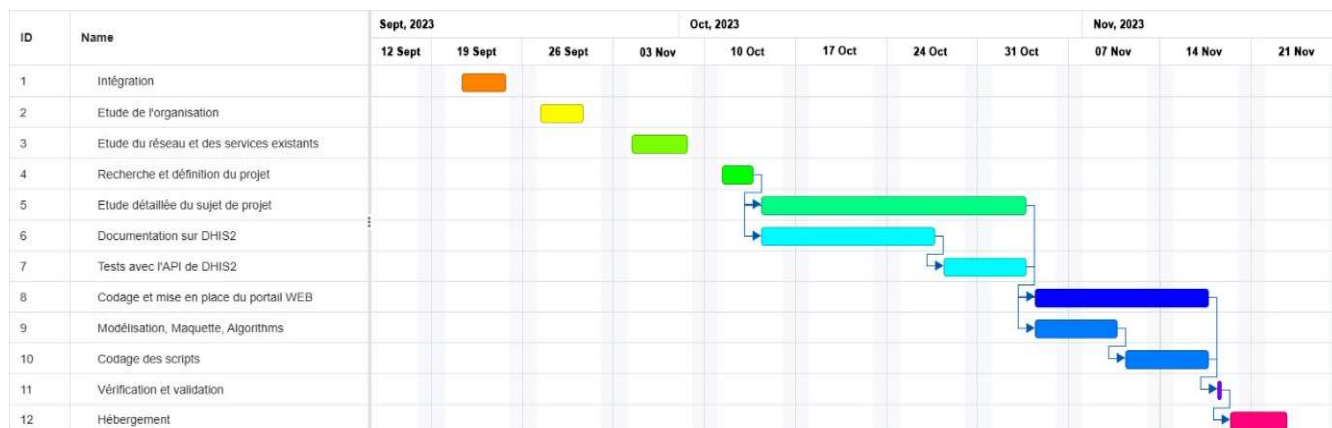


Figure 1.3 : Diagramme de Gantt de la planification avec Online Gantt

2.2. ETUDES DE FAISABILITE

2.2.1. Etudes de l'existant

Le Ministère de la Santé Publique à Madagascar dispose d'une infrastructure informatique robuste et performante, avec l'accès à Internet fourni par Telma, un opérateur de télécommunications renommé à Madagascar. Ce réseau joue un rôle crucial dans la gestion des opérations et des services au sein du ministère, offrant une connectivité rapide et fiable à l'ensemble de l'organisation.

La pièce maîtresse de l'infrastructure informatique est la salle des serveurs, dirigée par le Docteur ARISON Daniel, le chef de service SEMIDSI actuel. Cette salle héberge une multitude de serveurs qui alimentent les différentes applications et services du ministère. Ces serveurs sont essentiels pour stocker, traiter et fournir l'accès aux données vitales du secteur de la santé.

Révéléateur de l'engagement envers l'amélioration continue, le Ministère de la Santé Publique a récemment obtenu de nouveaux serveurs grâce au soutien de l'USAID. Cette initiative souligne la collaboration avec des partenaires internationaux pour renforcer l'infrastructure technologique du ministère. Ces nouveaux serveurs, venant de l'USAID, témoignent de l'importance accordée à l'innovation et à la modernisation des systèmes informatiques.

2.2.1.1. Plateforme DHIS2



Figure 2.1: Icône de la plateforme DHIS2

DHIS2 est une plateforme logicielle destinée à la collecte, la validation, la visualisation et l'analyse des données dans le domaine de la santé. Il est conçu pour répondre aux besoins des organisations et des gouvernements en matière de gestion des données de santé, notamment dans le contexte des systèmes de santé publique.

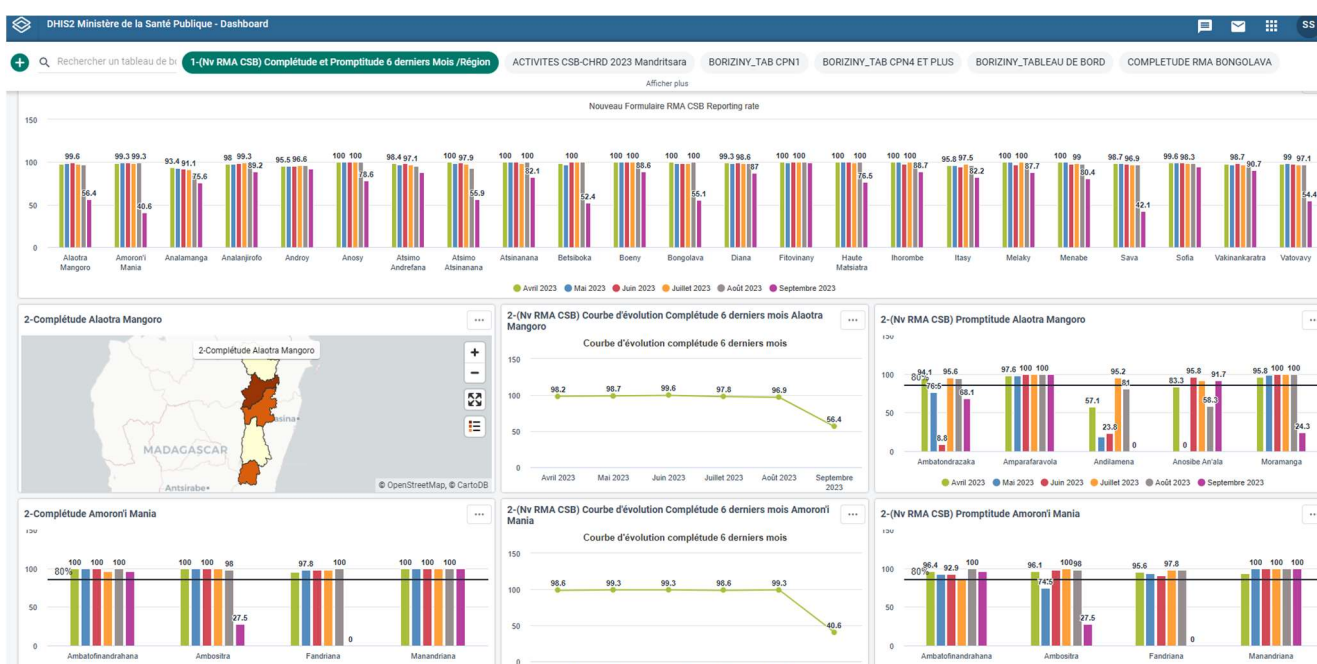


Figure 2.2 : Tableau de bord de la plateforme DHIS2

C'est un logiciel libre développé par un réseau de recherche, créé à l'initiative de l'Université d'Oslo (Norvège) en 1994. Il est utilisé pour la saisie, l'analyse, la diffusion et l'évaluation de données pour de nombreux programmes de santé.

Le DHIS2 est développé par le HISP comme un processus ouvert et distribué à l'échelle mondiale avec des développeurs en Inde, Vietnam, Tanzanie, Irlande, Norvège, etc. Le développement est coordonné par l'université d'Oslo.

Il fait actuellement ses preuves dans plus de 50 pays et se montre à la hauteur pour répondre aux besoins d'informations pour la prise de décision. Il est utilisé par de nombreuses

institutions, organisations non gouvernementales, agences gouvernementales et partenaires de santé à travers le monde.

Ce système offre une grande flexibilité en termes de personnalisation et de configuration pour s'adapter aux besoins spécifiques de chaque contexte. Il propose également des outils de visualisation de données, des tableaux de bord et des fonctionnalités de cartographie pour faciliter la prise de décision basée sur les données de santé.

En somme, DHIS2 joue un rôle fondamental dans la gestion des données de santé, tant au niveau national qu'international. Il contribue ainsi à améliorer les politiques et les programmes de santé publique en fournissant des informations fiables et exploitables.

a) Fonctionnement de DHIS2

Voici comment il fonctionne généralement :

Collecte de Données : DHIS2 permet de collecter des données de santé à partir de différentes sources. Cela peut inclure des établissements de santé, des cliniques, des hôpitaux, des programmes de santé communautaires, etc. Les données peuvent être collectées à l'aide de formulaires électroniques.

Stockage de Données : Une fois collectées, les données sont stockées dans une base de données centralisée. Cette base de données est conçue pour gérer de grandes quantités de données de santé et pour permettre une récupération rapide.

Analyse de Données : DHIS2 offre des fonctionnalités d'analyse puissantes. Il permet de générer des tableaux de bord, des graphiques et des rapports pour visualiser les tendances, les comparaisons et les indicateurs clés de performance.

Présentation des Données : Les résultats de l'analyse sont présentés sous forme de tableaux de bord interactifs. Ces tableaux de bord peuvent être personnalisés en fonction des besoins spécifiques de l'utilisateur.

Partage des Données : DHIS2 facilite le partage des données de santé avec différentes parties prenantes telles que les décideurs politiques, les professionnels de la santé, les chercheurs et le public. Cela peut se faire à travers des rapports automatisés, des visualisations et des exports de données.

Surveillance et Évaluation : DHIS2 prend en charge le suivi et l'évaluation des programmes de santé en permettant de suivre l'impact des interventions de santé au fil du temps.

Personnalisation et Extension : DHIS2 est hautement configurable et extensible. Il peut être adapté pour répondre aux besoins spécifiques d'un pays ou d'une organisation. Des modules complémentaires peuvent également être ajoutés pour étendre les fonctionnalités de base.

Sécurité et Confidentialité : DHIS2 intègre des fonctionnalités de sécurité pour garantir que les données de santé sont protégées et que l'accès est limité aux utilisateurs autorisés.

b) Points forts

Open Source : DHIS2 est un logiciel open source, ce qui signifie qu'il est accessible gratuitement. Cela favorise la collaboration, la transparence et permet aux utilisateurs de personnaliser le système selon leurs besoins.

Personnalisable : DHIS2 est conçu pour être configurable et adaptable. Les utilisateurs peuvent définir des ensembles de données, des indicateurs, des formulaires, etc., pour répondre aux besoins spécifiques de leur programme de santé.

Collecte de Données : DHIS2 facilite la collecte de données à partir de sources variées, y compris les formulaires en ligne, les importations de fichiers, les SMS, etc. Il offre une flexibilité pour s'adapter à différentes méthodes de collecte.

Analyse et Visualisation : DHIS2 propose des outils intégrés pour l'analyse et la visualisation des données, y compris des tableaux de bord, des graphiques et des cartes. Cela aide les utilisateurs à prendre des décisions éclairées.

Interopérabilité : DHIS2 prend en charge les normes internationales pour l'échange de données en santé, favorisant ainsi l'interopérabilité avec d'autres systèmes d'information.

Communauté Active : la communauté DHIS2 est active et mondiale. Les utilisateurs peuvent partager des expériences, poser des questions et contribuer au développement continu du logiciel.

c) Points faibles

Courbe d'Apprentissage : En raison de sa complexité et de sa richesse en fonctionnalités, DHIS2 peut avoir une courbe d'apprentissage abrupte pour les nouveaux utilisateurs.

Personnalisation Exigeante : Bien que la personnalisation soit une force, elle peut également être une faiblesse si elle n'est pas gérée correctement. Des niveaux avancés de personnalisation peuvent nécessiter des compétences techniques importantes.

Performance : Dans certaines instances, la performance de DHIS2 peut être un défi, en particulier lors de l'utilisation de grandes quantités de données. Cela peut nécessiter une infrastructure informatique robuste.

Maintenance : Comme tout logiciel, DHIS2 nécessite une maintenance régulière, y compris les mises à jour et la gestion des erreurs potentielles.

Documentation : Bien que la documentation de DHIS2 soit généralement bonne, des besoins spécifiques peuvent nécessiter une recherche approfondie et une compréhension approfondie de la documentation.

Flexibilité vs Structure : La balance entre la flexibilité et la structure peut être délicate. Trop de flexibilité peut rendre la gestion des données complexe, tandis que trop de structure peut limiter l'adaptabilité.

2.2.1.2. Les fonctionnalités des éléments du tableau de bord de DHIS2

L'objectif est de récupérer les données issues de l'application 'Tableau de bord' de DHIS2 du Ministère de la Santé Publique. Pour cela, il faut donc identifier les éléments du tableau de bord et leurs fonctionnalités.

Widgets (Objets) : sont des composants interactifs qui affichent différentes formes de données telles que des graphiques, des tableaux, des cartes, des jauges, etc.

Graphiques : visuels, tels que les diagrammes à barres, les camemberts, les graphiques en ligne, permettent de représenter les données de manière graphique.

Cartes : permettent de visualiser les données géographiques, souvent utilisées pour représenter des indicateurs par zone géographique.

Tableaux Croisés Dynamiques : affichent des informations sous forme tabulaire, offrant une vue détaillée des données.

Filtres : permettent aux utilisateurs de restreindre les données affichées en fonction de critères spécifiques, tels que la période de temps ou la zone géographique.

Alertes et Notifications : attirent l'attention sur des résultats inattendus ou des seuils critiques.

Options d'Exportation : permettent aux utilisateurs d'extraire les données du tableau de bord sous forme de rapports ou de fichiers.

Options de Partage : permettent aux utilisateurs de collaborer en partageant des vues spécifiques du tableau de bord avec d'autres.

Ces éléments et fonctionnalités combinés offrent aux utilisateurs une plateforme complète pour la surveillance, l'analyse et la présentation des données de santé dans le cadre de DHIS2.

2.2.1.3. WEB API de DHIS2

a) les APIs de DHIS2

Pour lister les différents APIs de DHIS2, nous avons la requête GET api/resources avec le lien URL « http://<domaine_serveur>/api/resources ».

Les APIs de DHIS2 peuvent renvoyer des données en format JSON, XML, PNG, SVG, CSV, MP4, et autres dépendant des requêtes envoyées. Chacun de ces formats a des utilisations spécifiques et est adapté à des types de données particuliers. JSON et XML sont utilisés pour la représentation de données structurées, PNG et SVG sont des formats d'image, CSV est utilisé pour stocker des données tabulaires, et MP4 est utilisé pour les fichiers multimédias.

Après avoir fait quelques recherches dans les documentations de DHIS2 [1] et quelques analyses sur DHIS2, j'ai compté 4 types d'objet dans le tableau de bord qui sont **VISUALISATION**, **TEXT**, **MESSAGE** et **MAP**.

b) Les types d'affichage d'objet du tableau de bord

Graphique : affiche graphiquement les données enregistrées dans l'application WEB DHIS2.

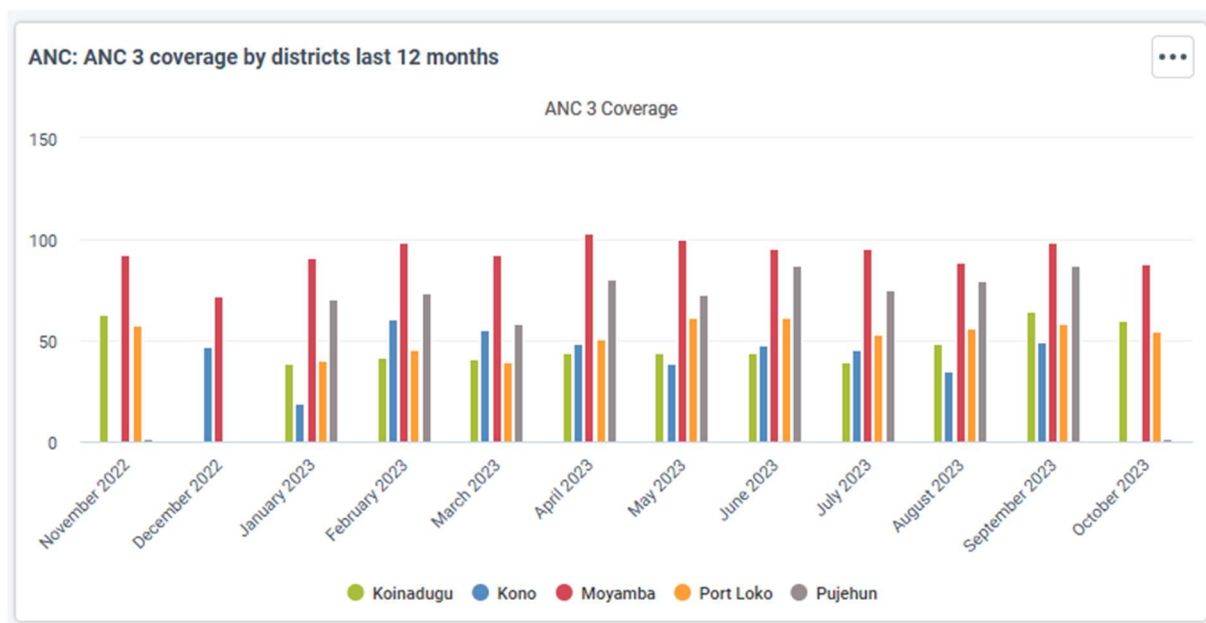


Figure 2.3 : Graphique de l'application DHIS2

Carte : affiche graphiquement sur un plan géographique les données enregistrées.



Figure 2.4 : Carte géographique de l'application DHIS2

Tableau : référence les données numériques et textuelles pour l’affichage du Graphique. Le Graphique se base sur le Tableau pour ces dimensions.

ANC: ANC 3 coverage by districts last 12 months

| ANC 3 Coverage | | | | | |
|----------------|-----------|------|---------|-----------|---------|
| | Koinadugu | Kono | Moyamba | Port Loko | Pujehun |
| November 2022 | 62.8 | | 92.4 | 57.8 | 1.2 |
| December 2022 | | 47.3 | 72.2 | | 0.4 |
| January 2023 | 38.8 | 18.7 | 91 | 40.5 | 70.5 |
| February 2023 | 42 | 60.7 | 98.8 | 45.4 | 73.2 |
| March 2023 | 40.7 | 55.4 | 92.3 | 39.1 | 58.6 |
| April 2023 | 44.1 | 48.5 | 102.7 | 50.4 | 80.3 |
| May 2023 | 44.2 | 39 | 100 | 61.7 | 72.4 |
| June 2023 | 43.7 | 47.4 | 95.7 | 61.2 | 87.2 |
| July 2023 | 39.4 | 45.6 | 95.6 | 53.3 | 74.8 |
| August 2023 | 48.3 | 34.6 | 88.4 | 56.2 | 79.4 |
| September 2023 | 64.7 | 49 | 98.8 | 58.6 | 87.2 |
| October 2023 | 59.5 | | 87.7 | 54.8 | 1.2 |

Figure 2.5 : Tableau croisé dynamique de DHIS2

Texte : affiche des informations comme des annonces.

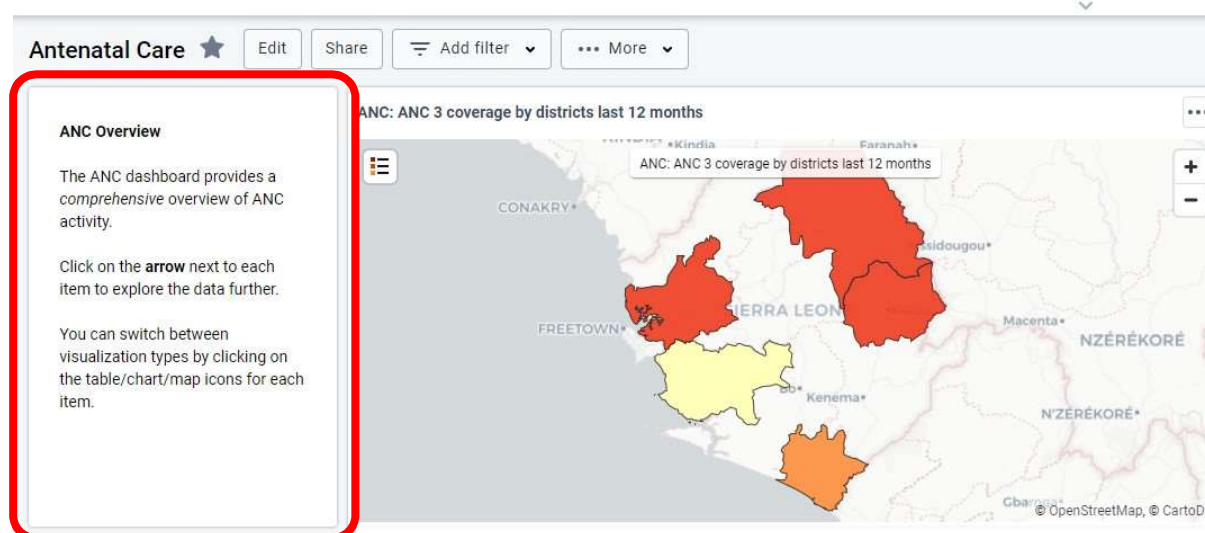


Figure 2.6 : Objet de type Texte de DHIS2

2.2.2. Etudes de besoins

2.2.2.1. Application de modélisation

Dia diagram : une application permettant de créer des diagrammes techniques. Son interface et ses caractéristiques s'inspirent du programme Windows Visio. Elle permet d'utiliser des formes personnalisées que l'utilisateur crée avec une simple description XML. Dia sert notamment à dessiner des diagrammes UML, des topologies de réseaux, et des diagrammes de flux de données.

2.2.2.2. Éditeur de texte et langages

a) *Editeur*

Visual Studio Code : un éditeur de code source léger, mais puissant, qui s'exécute sur votre bureau et est disponible pour Windows, macOS et Linux. Il est livré avec une prise en charge intégrée de Javascript, TypeScript et Node.js et dispose d'un riche écosystème d'extensions pour d'autres langages et environnements d'exécution (tels que C++, C#, Java, Python, PHP, Go, .NET).

b) *Langages*



Python : un langage de programmation populaire dans le domaine du développement web. Il est utilisé pour créer des applications web dynamiques, des API (Interfaces de Programmation d'Applications), et des scripts côté serveur. Le framework Django, écrit en Python, est largement utilisé pour le développement web.

HTML : le langage de balisage standard pour la création de la structure et du contenu des pages web. Il utilise des balises pour définir des éléments tels que les titres, les paragraphes, les liens, les images, les formulaires, etc.

CSS : est utilisé pour définir la présentation visuelle des pages web écrites en HTML. Il permet de styliser les éléments HTML en spécifiant les couleurs, les polices, les marges, les dispositions, et d'autres propriétés esthétiques.

Javascript : un langage de programmation côté client qui ajoute de l'interactivité aux pages web. Il est utilisé pour manipuler le DOM, réagir aux événements utilisateur, effectuer des validations côté client, et créer des applications web dynamiques.

2.2.2.3. Modules externes (Frameworks)

Leaflet (Carte) : une bibliothèque Javascript légère et open source utilisée pour créer des cartes interactives sur des sites web. Elle offre des fonctionnalités de base pour l'affichage de cartes, le marquage de points, l'ajout de calques, et la gestion d'événements liés à la carte

Chart.js (Graphique) : une bibliothèque Javascript simple et flexible qui permet de créer des graphiques et des tableaux de bord interactifs. Elle prend en charge différents types de graphiques tels que les graphiques en barres, les graphiques en ligne, les graphiques radar, etc.

DataTables (Tableau croisé dynamique): module CSS et Javascript dépendance JQuery qui permet d'améliorer la fonctionnalité des tableaux HTML. Il offre une variété de fonctionnalités utiles pour rendre les tableaux interactifs, dynamiques et plus conviviaux pour l'utilisateur.

Bootstrap (Responsivité et autres) : un framework open source qui facilite la conception et le développement de sites web responsives. Il fournit une collection d'outils, de styles CSS préconçus, de composants Javascript et d'autres éléments pour créer des interfaces utilisateur attrayantes et adaptées à différents appareils.

JQuery (Dynamicité) : une bibliothèque Javascript rapide, légère et riche en fonctionnalités. Elle simplifie la manipulation du DOM (Document Object Model), la gestion des événements, les animations, et les requêtes Ajax, facilitant ainsi le développement web interactif.

Django (Serveur) : un framework web open source écrit en Python. Il suit le modèle de conception MVC (Modèle-Vue-Contrôleur) et encourage une approche DRY (Don't Repeat Yourself) pour le développement web. Django simplifie la création de sites web robustes en fournissant des fonctionnalités intégrées telles que la gestion des bases de données, l'authentification, et la gestion des formulaires.

2.2.2.4. Hébergement

a) Hébergement sur un serveur en ligne

Voici quelques options gratuites pour héberger des applications Django :

Heroku offre un plan gratuit avec des limitations en termes de ressources, mais il peut être suffisant pour des petites applications ou des projets personnels.

PythonAnywhere propose un plan gratuit limité en ressources, mais il permet de déployer des applications Django.

Glitch est une plateforme qui permet de créer et d'héberger des applications web gratuitement. Elle prend en charge Python et Django.

Amazon Web Services offre un niveau gratuit avec certaines ressources gratuites. On peut utiliser Amazon EC2 pour déployer des applications Django.

Google Cloud Platform propose App Engine, qui offre un niveau gratuit pour les petites charges de travail. On peut déployer des applications Django sur App Engine.

Microsoft Azure propose App Service, qui dispose d'un niveau gratuit avec des limitations en ressources. On peut déployer des applications Django sur App Service.

Vercel offre un plan gratuit pour héberger des applications web statiques et dynamiques, y compris celles construites avec Django.

b) Hébergement sur un serveur local

Pour publier la plateforme WEB, on peut aussi utiliser un serveur local. Par contre, la mise en ligne ne serait pas aussi facile qu'avec les services cités précédemment.

Étapes principales :

Prérequis : s'assurer que Python et Django sont installés sur la machine, noter l'adresse IP locale de votre machine (on peut la trouver avec la commande `ipconfig` sur Windows ou `ifconfig` sur Linux/Mac).

Configurer Django : s'assurer que les paramètres de le fichier `settings.py` de Django sont configurés correctement pour la production (par exemple, **DEBUG** = **False**, **ALLOWED_HOSTS** contient l'adresse IP).

Collecter les fichiers statiques : exécuter la commande **python manage.py collectstatic** pour collecter les fichiers statiques.

Installer un serveur web : utiliser des serveurs web tels que Nginx ou Apache.

Configurer le serveur web : créer un fichier de configuration pour le site dans le répertoire de configuration.

Redémarrer le serveur web après avoir sauvegardé le fichier de configuration.

Exécuter Django avec Gunicorn : installer Gunicorn en utilisant **pip install gunicorn**.

Exécuter l'application Django avec Gunicorn :

Ouvrez le pare-feu : ajouter une règle dans le pare-feu pour permettre le trafic sur le port 80 (ou le port défini dans le fichier de configuration)..

2.3. MODELISATION

2.3. 1. Diagramme UML : cas d'utilisation

Au début, on a une page pour récupérer les informations (nom complet, domaine professionnel, lieu de travail, contacts) de l'utilisateur. L'utilisateur entre les informations à champs correspondants puis devrait les soumettre. Ces informations seront stockées dans une base de données et affichées dans la page d'administration pour permettre à l'administrateur de connaître le nombre de vues, d'interactions et d'utilisateurs sur la plateforme WEB.

Après, l'utilisateur peut explorer et profiter des fonctionnalités et des informations fournies par la plateforme WEB issues de l'application Tableau de Bord de DHIS2 du Ministère de la Santé Publique à Madagascar.

La page du tableau de bord affiche les informations et les objets du tableau de bord depuis le DHIS2. On peut consulter les informations partagées sur la plateforme WEB.

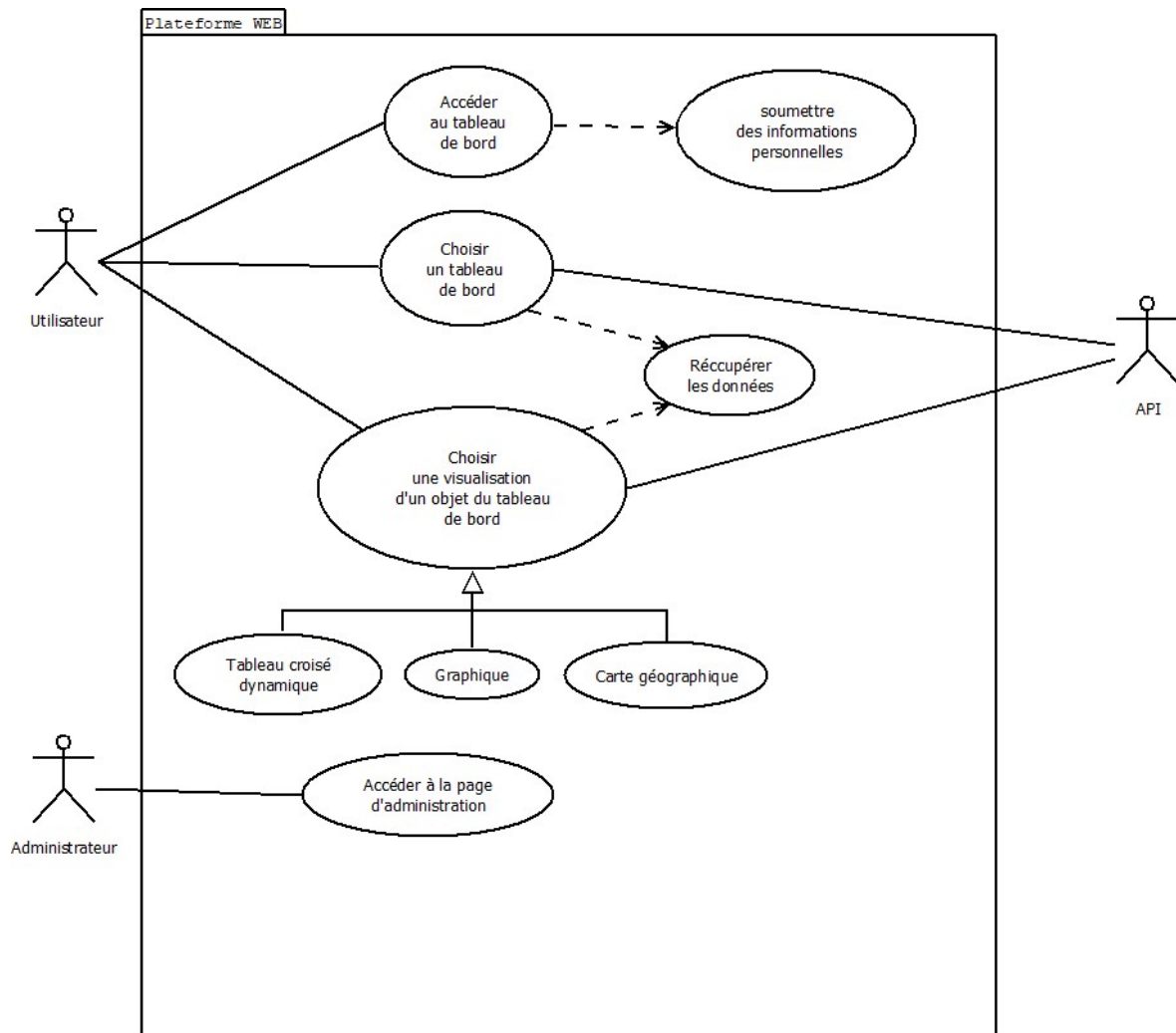


Figure 2.7 : Diagramme de cas d'utilisations

2.3. 2. Modélisation de la base de donnée (MERISE)

Ces données sont les informations sur les utilisateurs et le nombre de soumission sur la nouvelle plateforme WEB.

Pendant le définition de sujet, les demandes étaient de créer une plateforme WEB pour reproduire le tableau de bord issu de DHIS2 du Ministère de la Santé Publique, et ajouter à la place d'un formulaire de connexion, un formulaire de renseignement personnel dont les informations sont le nom complet, la structure, le lieu de travail et les contacts (téléphone, email).

Dictionnaire de données :

Tableau 2.1: L'entité visiteur

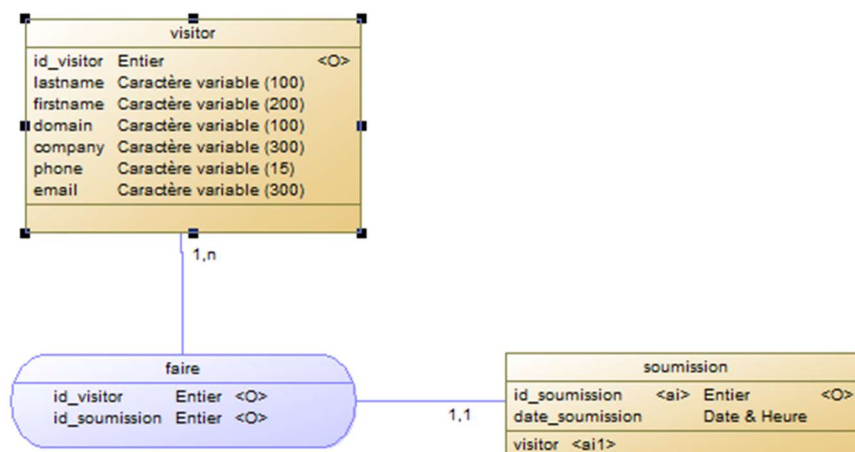
| attribut | type | nombre |
|-----------------|---------|--------|
| nom | varchar | 100 |
| prénom | varchar | 200 |
| structure | varchar | 100 |
| lieu de travail | varchar | 300 |
| téléphone | varchar | 15 |
| email | varchar | 300 |

Tableau 2.2: L'entité soumission

| attribut | type |
|--------------------|--------------|
| visiteur | Varchar(300) |
| date de soumission | datetime |

2.3.2. 1. MCD : Modèle Conceptuel des Données

Le modèle conceptuel de données est une représentation abstraite des concepts métier et des relations entre ces concepts, sans se soucier des détails de mise en œuvre physique des données. Son objectif principal est de capturer les concepts et les relations essentiels au sein d'un domaine métier spécifique.

**Figure 2.8 :** Modèle conceptuel de données du formulaire de renseignement personnel

2.3.2. 2. MLD : Modèle Logique des Données

Le modèle logique de données est une représentation intermédiaire entre le modèle conceptuel de données (qui capture les concepts métier) et le modèle physique de données (qui détaille la manière dont les données sont stockées physiquement).

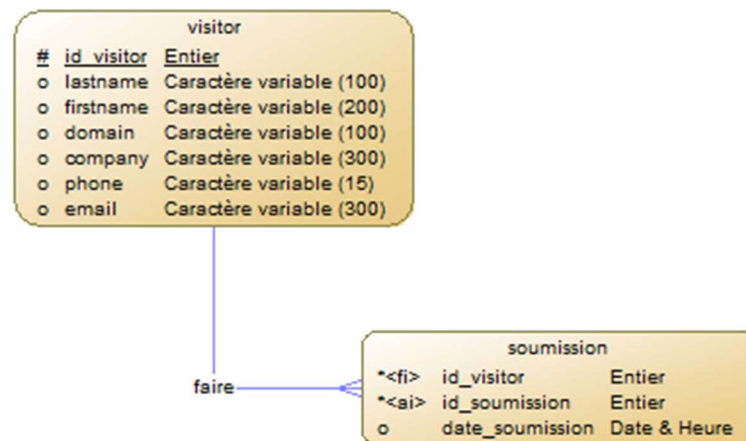


Figure 2.9 : Modèle logique de données du formulaire de renseignement personnel

2.3.2. 3. MPD : Modèle Physique de Données

Le modèle physique de données est une représentation concrète de la manière dont les données sont stockées dans un système de gestion de base de données (SGBD).

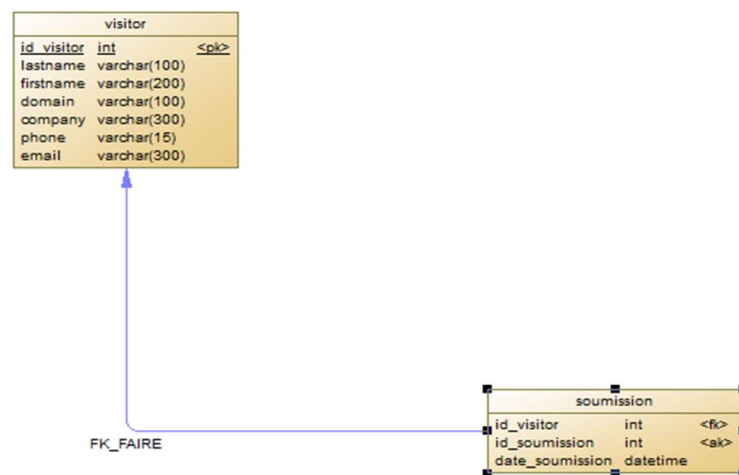


Figure 2.10 : Modèle physique de données du formulaire de renseignement personnel

2.3.3. Composition et Maquettage

Dans cette partie, nous allons offrir des solutions pour les besoins sur les outils à utiliser pour réaliser notre projet et certains composants importants de notre application.

2.3.3.1. Les utilisateurs (visiteurs):

Les utilisateurs, qu'on appelle 'visitor' dans la base de données sont les personnes qui utilisent la plateforme. Un utilisateur remplit le formulaire de renseignement personnel et profite des informations et des fonctionnalités accessibles sur la nouvelle plateforme WEB issu de l'application Tableau de Bord de DHIS2 du Ministère de la Santé Publique.

2.3.3.2. Les fonctionnalités :

La nouvelle plateforme doit avoir certaines fonctionnalités de DHIS2, comme de choisir un tableau de bord, changer de type d'affichage des objets (Graphique, Tableau et Carte) et faire du filtrage. Par contre, le filtrage est mensuel seulement.

Avec le module javascript **DataTables**, les tableaux ont des fonctionnalités supplémentaires comme la pagination, la recherche et le triage par colonne.

2.3.3.3. Les maquettes

La maquette se base sur l'interface du Tableau de Bord de DHIS2 mais en enlevant les éléments qui ne sont pas utiles à la nouvelle plateforme.

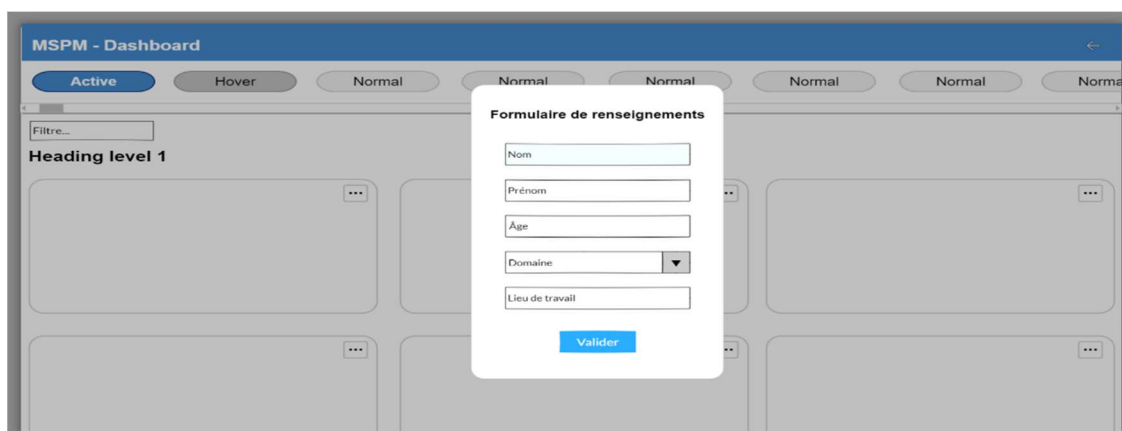
The image shows a web application mockup titled 'MSPM - Dashboard'. It features a top navigation bar with a blue header and several tabs labeled 'Active', 'Hover', and 'Normal'. Below the navigation bar, there is a 'Filtre...' input field and a 'Heading level 1' section. The main content area is a grid of placeholder boxes. A modal form titled 'Formulaire de renseignements' is overlaid on the grid. This form contains five input fields: 'Nom', 'Prénom', 'Âge', 'Domaine' (a dropdown menu), and 'Lieu de travail'. At the bottom of the modal is a blue 'Valider' button.

Figure 2.11: Maquette du formulaire de renseignement personnel depuis Pencil

Pour le tableau de bord, on a une représentation similaire à celle de DHIS2. La plateforme possède aussi un bouton qui permet d'afficher les informations sur l'utilisateur et sur la plateforme elle-même.

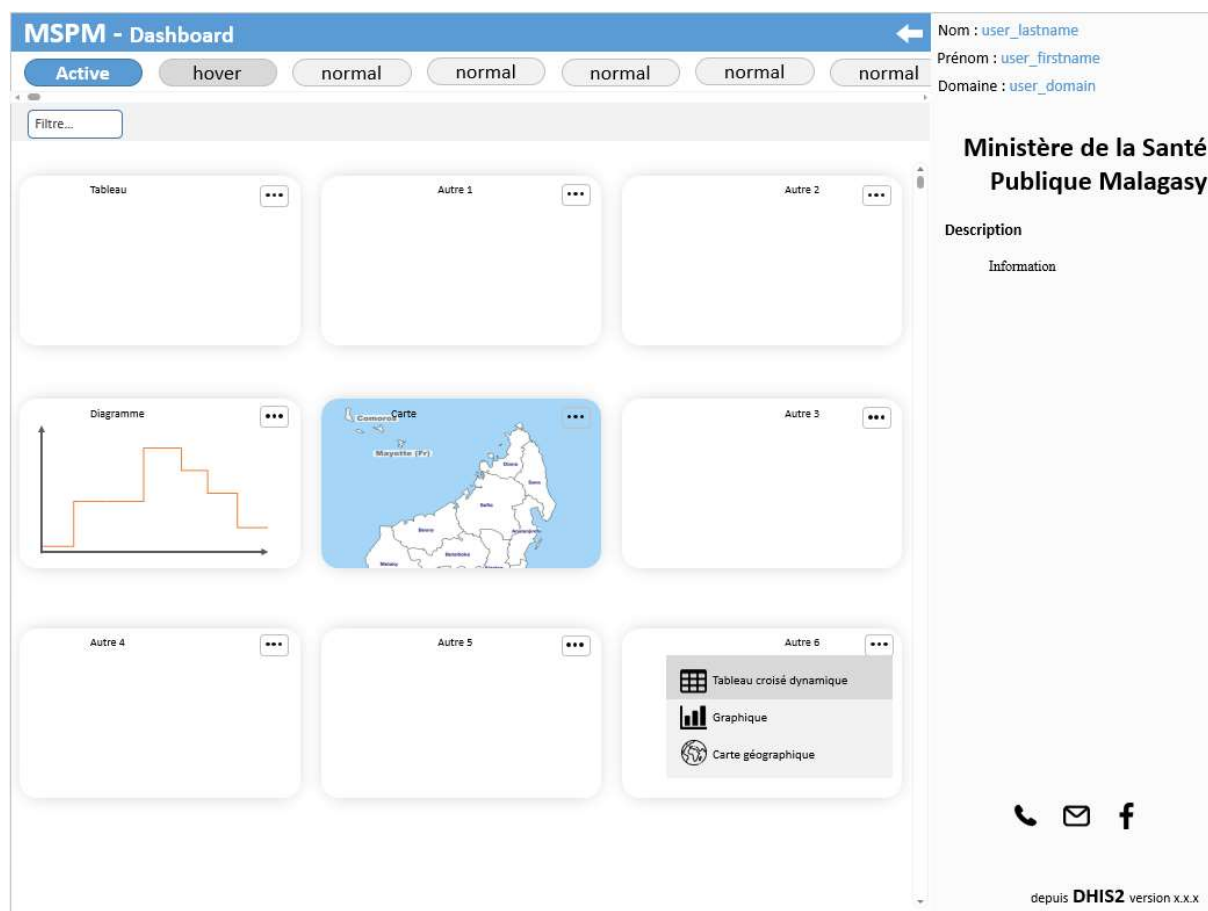


Figure 2.12: Maquette du Tableau de Bord de la plateforme depuis PowerPoint

Chapitre 3. REALISATION

La réalisation de cette plateforme web dynamique représente une avancée significative dans la diffusion et le partage de l'information sanitaire au sein du Ministère de la Santé Publique à Madagascar. Elle contribuera à renforcer les connaissances en santé et à améliorer la prise de décision en matière de santé publique.

3.1. DEVELOPPEMENT DE L'APPLICATION

Outils utilisés : Visual Studio Code (l'éditeur de texte), APIs de DHIS2

Langages : HTML, CSS, Javascript, Python

Modules : Django, Leaflet, Chart, DataTables, Bootstrap, JQuery

3.1. 1. Partie Serveur (Django) :

3.1.1. 1. Création de la base de données :

Django, par défaut, utilise SQLite pour sa base de données. Dans le fichier models.py :

```
from django.db import models

# création des tables.
class Visiteur(models.Model):
    nom = models.CharField(max_length=200)
    prenom = models.CharField(max_length=200)
    structure = models.CharField(max_length=100)
    lieu_travail = models.CharField(max_length=200)
    telephone = models.CharField(max_length=15)
    email = models.EmailField()

    def __str__(self):
        return f"{self.nom} {self.prenom}"

class Soumission(models.Model):
    information = models.ForeignKey(Visiteur, on_delete=models.CASCADE)
    date_soumission = models.DateTimeField(auto_now_add=True)
```

Après nous utilisons les commandes suivantes pour appliquer à la base de données les modifications :

console :

```
python manage.py makemigrations
python manage.py migrate
```

3.1.1. 2. Communication entre Javascript et Django :

def action(request): C'est une fonction vue Django qui prend une requête HTTP en paramètre.

if request.method == 'POST': Vérifie si la requête est de type POST.

data = dict(json.loads(request.body)) : Charge les données JSON à partir du corps de la requête POST et les convertit en un dictionnaire Python.

La suite du code utilise une structure conditionnelle (**if-elif-else**) pour déterminer l'action à effectuer en fonction de la valeur de la clé '**query**' dans le dictionnaire '**data**' dans les données JSON.

- Si la valeur de '**query**' est "**dashboardItem**", la fonction **get_dashboards_items** est appelée avec les données '**datas**'.
- Si la valeur de '**query**' est "**reportTable**", la fonction **reporttable_datas** est appelée avec les données '**datas**' après avoir filtré les valeurs analytiques si une clé '**filter**' est présente.
- Si la valeur de '**query**' est "**chart**", la fonction **get_item_chart** est appelée avec des données similaires à celles de "**reportTable**".
- Si la valeur de '**query**' est "**map**", la fonction **get_item_map** est appelée avec des données similaires à celles de "**reportTable**".
- Si la valeur de '**query**' est "**form**", la fonction **save_form** est appelée avec les données '**datas**'.

return JsonResponse(data): Renvoie une réponse HTTP JSON avec les données modifiées ou traitées.

Hors de la condition **if request.method == 'POST':**, si la méthode de requête n'est pas POST, renvoie une réponse JSON avec un message d'erreur.

Cette fonction est destinée à sauvegarder des données de formulaire dans une base de données Django.

def save_form(data): Définit une fonction **save_form** qui prend un argument **data**.

Extraction des données du dictionnaire **data** :

python

```
nom = data['nom']
prenom = data['prenom']
structure = data['structure']
lieu_travail = data['lieu_travail']
telephone = data['telephone']
email = data['email']
```

Ces lignes extraient différentes valeurs du dictionnaire **data** en fonction de leurs clés.

Vérification de l'existence de l'utilisateur dans la table **Visiteur** :

python

```
existing_user = Visiteur.objects.filter(email=email).first()
```

Cette ligne utilise l'**ORM** de Django pour vérifier si un utilisateur avec l'email donné existe déjà dans la table **Visiteur**.

Logique conditionnelle basée sur l'existence de l'utilisateur :

python

```
if existing_user:
    # Si l'utilisateur existe déjà, ajoutez simplement la soumission
    Soumission.objects.create(information=existing_user)
else:
    # Si l'utilisateur n'existe pas, créez-le dans la table Visiteur
    new_user = Visiteur.objects.create(
        nom=nom,
        prenom=prenom,
        structure=structure,
        lieu_travail=lieu_travail,
        telephone=telephone,
```

```

        email=email
    )

    # Créez la soumission pour le nouvel utilisateur
    Soumission.objects.create(information=new_user)

```

Cette section de code vérifie si l'utilisateur existe déjà dans la base de données (**existing_user**). Si c'est le cas, une nouvelle **Soumission** est simplement ajoutée. Sinon, un nouvel utilisateur est créé dans la table **Visiteur**, et une nouvelle **Soumission** est également créée pour cet utilisateur.

return data : Renvoie le dictionnaire de réponse, qui peut être utilisé pour informer l'utilisateur du succès de l'opération.

3.1.1. 3. Formatage de données pour l'affichage de Graphique :

get_item_chart() est une fonction Python qui prend deux paramètres, **data** et **info**, et retourne des données de graphique au format adapté pour Chart.js.

Voici une explication ligne par ligne :

chart_data = {'labels': [], 'datasets': []} : Initialise un dictionnaire **chart_data** qui sera utilisé pour stocker les données du graphique.

La première condition **if info['type'] == 'VISUALIZATION'**: vérifie si le type d'information (**info['type']**) est une visualisation (graphique). Si oui, elle récupère des informations supplémentaires sur le type de graphique utilisé en faisant une requête HTTP à l'URL spécifiée dans **info['href']**. Ces informations sont stockées dans le dictionnaire **chart_info**.

Si le type de l'axe est **'RANGE'**, des annotations sont ajoutées au dictionnaire **chart_data** en fonction des axes de la visualisation.

La condition **elif info['type'] == 'MAP'**: traite le cas où le type d'information est une carte. Elle fixe le type de graphique à **'COLUMN'**.

La section suivante récupère les labels du graphique à partir des données (**data**). Si une clé d'erreur (**KeyError**) est levée, elle récupère les labels à partir des colonnes.

La section suivante traite la récupération des **datasets** pour le graphique en fonction du type de visualisation. Elle itère sur les colonnes des données et crée des **datasets** avec des **labels** et des données associées.

Pour le type de graphique '**year_over_year_line**', elle récupère également des données spécifiques liées à une année.

La section suivante effectue des ajustements au type de graphique pour l'adapter à Chart.js. Par exemple, elle remplace le type '**COLUMN**' par '**bar**'.

La section suivante convertit les données en un format souhaité pour Chart.js et renvoie le dictionnaire **chart_data**.

3.1.1. 4. Formatage de données pour l'affichage de Carte Géographique :

Ce code est une fonction Python appelée **get_item_map()** qui récupère des données spécifiques et les formate de manière à créer une carte géographique basée sur des unités organisationnelles.

def get_item_map(analytic: dict, info): La fonction prend deux paramètres, **analytic** (qui contient des données analytiques) et **info** (qui est des informations sur la carte).

La section suivante utilise des conditions pour déterminer le type de carte, puis récupère les données géométriques de l'unité organisationnelle parente :

- Si le type est '**VISUALIZATION**', elle fait une requête pour récupérer les données géométriques à partir de l'URL spécifiée dans **info['href']**.
- Si le type est '**MAP**', elle fait une série de requêtes pour obtenir l'identifiant de l'unité organisationnelle parente, puis elle récupère les données géométriques de cette unité.

La section suivante détermine le type de géométrie de l'unité organisationnelle parente ('**Polygon**' ou '**MultiPolygon**') et calcule le centre.

Initialisation d'un dictionnaire **statesData** qui sera utilisé pour stocker les données nécessaires pour créer la carte, des données **GeoJSON**.

La boucle **for** itère sur les unités organisationnelles (**ou**) dans **analytic['metaData']['dimensions']['ou']** et fait des requêtes pour récupérer les données de nom et de géométrie. Elle ajoute ces informations au dictionnaire **statesData**.

Une autre boucle **for** itère sur les lignes de données analytiques (**analytic['rows']**). Elle associe les valeurs de densité (extraites des données analytiques) aux unités organisationnelles correspondantes dans **statesData**.

Une troisième boucle **for** calcule la moyenne des densités pour chaque unité organisationnelle.

La fonction renvoie le dictionnaire **statesData** qui contient toutes les informations nécessaires pour créer la carte géographique.

En résumé, cette fonction prend des données analytiques et des informations sur la carte en entrée, effectue des requêtes pour récupérer les données géométriques, puis formate ces données pour créer une carte basée sur les unités organisationnelles, avec des densités associées à chaque unité.

Récupération et formatage des données d'un ensemble d'éléments:

Ce code est une fonction Python appelée **get_dashboards_items()** qui récupère un ensemble d'éléments d'un tableau de bord de la plateforme DHIS2 du Ministère de la Santé Publique.

dashboard_items = {"dashboardItems": []} : Initialise un dictionnaire pour stocker les éléments du tableau de bord.

dashboards=api_get_data(f"dashboards/{id_dashboard}?fields=dashboardItems[type,href,id,name,visualization,map]") : Appelle une fonction **api_get_data** pour récupérer les données du tableau de bord, en spécifiant les champs à inclure (**type, href, id, name, visualization, map**).

La boucle **for** parcourt chaque élément du tableau de bord (**dashboard_item**) dans les données récupérées.

La condition **if dashboard_item['type'] == 'MAP'**: vérifie si le type de l'élément est une carte et récupère l'identifiant correspondant.

La condition **elif dashboard_item['type'] in ['TEXT', 'MESSAGES']**: ignore les éléments de type **TEXT** ou **MESSAGES** et passe à l'élément suivant dans la boucle.

La section **else** récupère l'identifiant pour les éléments de type **visualization** (par opposition à MAP).

item = requests.get(URL_api + dashboard_item['type'].lower() + 's' + '/' + item_uid + '/?fields=name,href', auth=config).json() : Effectue une requête pour récupérer les données spécifiques de l'élément (visualisation ou carte) en utilisant l'identifiant récupéré précédemment.

Création d'un dictionnaire (**data**) contenant les informations nécessaires pour chaque élément.

Ajout du dictionnaire à la liste **dashboard_items['dashboardItems']**.

return set_grid(dashboard_items, 3) : Appelle une fonction **set_grid** pour formater la disposition des éléments du tableau de bord en grilles à **3 colonnes**, puis retourne le résultat.

3.1.1. 5. Formatage des données de chaque élément d'un tableau de bord :

Cette fonction, nommée **set_grid()**, prend en entrée un dictionnaire **items** contenant des éléments de tableau de bord et un entier **nb_col** représentant le nombre de colonnes souhaité dans la grille. Elle retourne un nouveau dictionnaire **new_items** organisant les éléments en fonction du nombre de colonnes spécifié.

Explications détaillées :

new_items = {} : Initialise un nouveau dictionnaire pour stocker les éléments du tableau de bord organisés en grilles.

nb_constant = len(items['dashboardItems']) : Obtient le nombre total d'éléments dans le tableau de bord.

d_item = [] : Initialise une liste temporaire pour stocker les éléments de chaque ligne de la grille.

nb_counter = 0 : Initialise un compteur pour suivre le nombre de lignes ajoutées à **new_items**.

items['dashboardItems'].reverse() : Inverse l'ordre des éléments du tableau de bord. Cela est fait pour ordonner les éléments par date de création.

La boucle **for** itère sur chaque élément du tableau de bord, en utilisant **enumerate** pour obtenir à la fois l'index **nb** et l'élément **item**.

La première vérification **if** ajoute une nouvelle ligne à **new_items** lorsque le nombre d'éléments atteint un multiple de **nb_col**.

d_item += [item] : Ajoute l'élément actuel à la ligne actuelle de la grille.

La deuxième vérification **if** ajoute la dernière ligne à **new_items** une fois que tous les éléments ont été traités.

La fonction retourne le dictionnaire **new_items** organisé en grilles de 3 colonnes.

3.1.1. 6. Récupération des données de chaque élément du tableau de bord

Cette fonction, appelée **get_item_infos**, extrait des informations spécifiques à partir d'un ensemble d'éléments. Elle prend en entrée une structure items et utilise ses propriétés, telles que **type** et **href**, pour effectuer des requêtes et récupérer des informations détaillées. La fonction renvoie un dictionnaire **dimension_parameter** contenant des informations structurées sur les dimensions associées (**column**, **row**, **filtre**) aux éléments.

Explications détaillées :

La fonction commence par extraire le chemin (**path**) à partir de l'URL (**href**) de l'élément.

Elle effectue une requête pour obtenir les données (**response**) associées à l'élément en utilisant l'API.

Pour les éléments de type "**YEAR_OVER_YEAR_LINE**", elle extrait la série temporelle annuelle (**yearlySeries**) et l'ajoute à la structure items.

Pour les éléments de type "**VISUALIZATION**", la fonction itère sur les dimensions (**columns**, **rows**, **filters**) et extrait les informations associées.

Pour les éléments de type "**MAP**", la fonction extrait diverses informations telles que les vues de carte (**mapView**), les dimensions spatiales et les unités organisationnelles associées.

Le résultat final est un dictionnaire **dimension_parameter** contenant des informations structurées sur les dimensions associées à l'élément, en fonction de son type.

3.1.1. 7. Récupération des données analytiques d'un élément du tableau de bord:

Cette fonction, appelée **get_analytic_values**, est conçue pour construire une requête analytique en fonction des dimensions spécifiées dans **dimension_datas** et des périodes spécifiées dans la liste **periods**. Elle utilise ensuite cette requête pour obtenir des données analytiques à partir d'une API.

Explications détaillées :

La fonction commence par initialiser la variable **analytic_param** qui stockera la partie de l'URL utilisée pour la requête analytique.

Elle itère sur les différentes dimensions (**columns**, **rows**, **filters**) spécifiées dans **dimension_datas**.

Si la dimension est un filtre (**key == 'filters'**), la fonction itère sur les valeurs de ce filtre et construit la partie de l'URL correspondante.

Si la dimension n'est pas un filtre, la fonction itère sur les valeurs de cette dimension et construit également la partie de l'URL correspondante.

La fonction vérifie également si des périodes sont spécifiées dans la liste **periods**. Si c'est le cas et que la dimension est temporelle (**id_dimension == 'pe'**), la fonction utilise ces périodes pour filtrer la requête et n'utilise pas la période par défaut.

La fonction utilise l'API pour récupérer les données analytiques en utilisant la requête construite.

Finalement, les données analytiques sont renvoyées.

3.1.1. 8. Formatage des données pour l'affichage d'un tableau :

Cette fonction, nommée **reporttable_datas**, semble être destinée à transformer les données d'une table de rapport (**response**) en un format plus adapté pour l'affichage dans une table ou un tableau. Elle renvoie un dictionnaire structuré avec les colonnes et les données de la table.

Explications détaillées :

La fonction commence par initialiser des listes vides pour stocker les titres des lignes (**row_titles**), les colonnes (**columns**), et les données (**data**).

La fonction vérifie le nombre de colonnes dans la réponse. Si la réponse a plus de deux colonnes, elle suppose que la première colonne est constituée des titres des lignes.

Elle extrait le type des titres des lignes de la réponse en utilisant la deuxième colonne (**response['headers'][1]['name']**) et les stocke dans la liste **row_titles**.

Si la deuxième colonne représente la dimension 'ou', elle trie les titres des lignes.

La fonction construit ensuite les données associées aux titres des lignes en utilisant la liste **row_titles** et les lignes de la réponse.

Si la réponse a deux colonnes ou moins, chaque ligne est traitée séparément, et les données sont stockées dans la liste **data**.

La fonction extrait les informations sur les colonnes en utilisant la première colonne de la réponse (**response['headers'][0]['name']**) et les stocke dans la liste **columns**.

Finalement, la fonction retourne un dictionnaire structuré avec les colonnes et les données sous la forme **{'columns': columns, 'data': data}**.

3.1.1. 9. Envoi de requêtes et récupération des données via l'API :

Cette fonction, nommée **api_get_data**, est destinée à effectuer des requêtes vers une API en utilisant la bibliothèque **requests** en Python. Elle prend un chemin **path** en entrée, construit une URL en utilisant ce chemin, effectue une requête GET vers cette URL, et retourne les données de la réponse (ou un ensemble de données d'erreur si la requête échoue).

Explications détaillées :

La fonction commence par construire l'URL en concaténant le chemin **path** avec l'URL de l'API (**URL_api**).

Elle vérifie si le chemin ne contient pas déjà la chaîne **'analytics?'**. Si ce n'est pas le cas, elle ajoute **'?paging=false'** à l'URL pour désactiver la pagination.

La fonction initialise un ensemble de données d'erreur par défaut (**datas**) au cas où la requête échoue. Ces données d'erreur consistent en une liste de tableaux de bord fictifs.

La fonction tente d'envoyer une requête GET à l'API en utilisant la bibliothèque **requests**.

Si la requête réussit, elle récupère les données JSON de la réponse et les stocke dans la variable **datas**.

Finalement, la fonction retourne les données, qu'elles soient récupérées avec succès ou non.

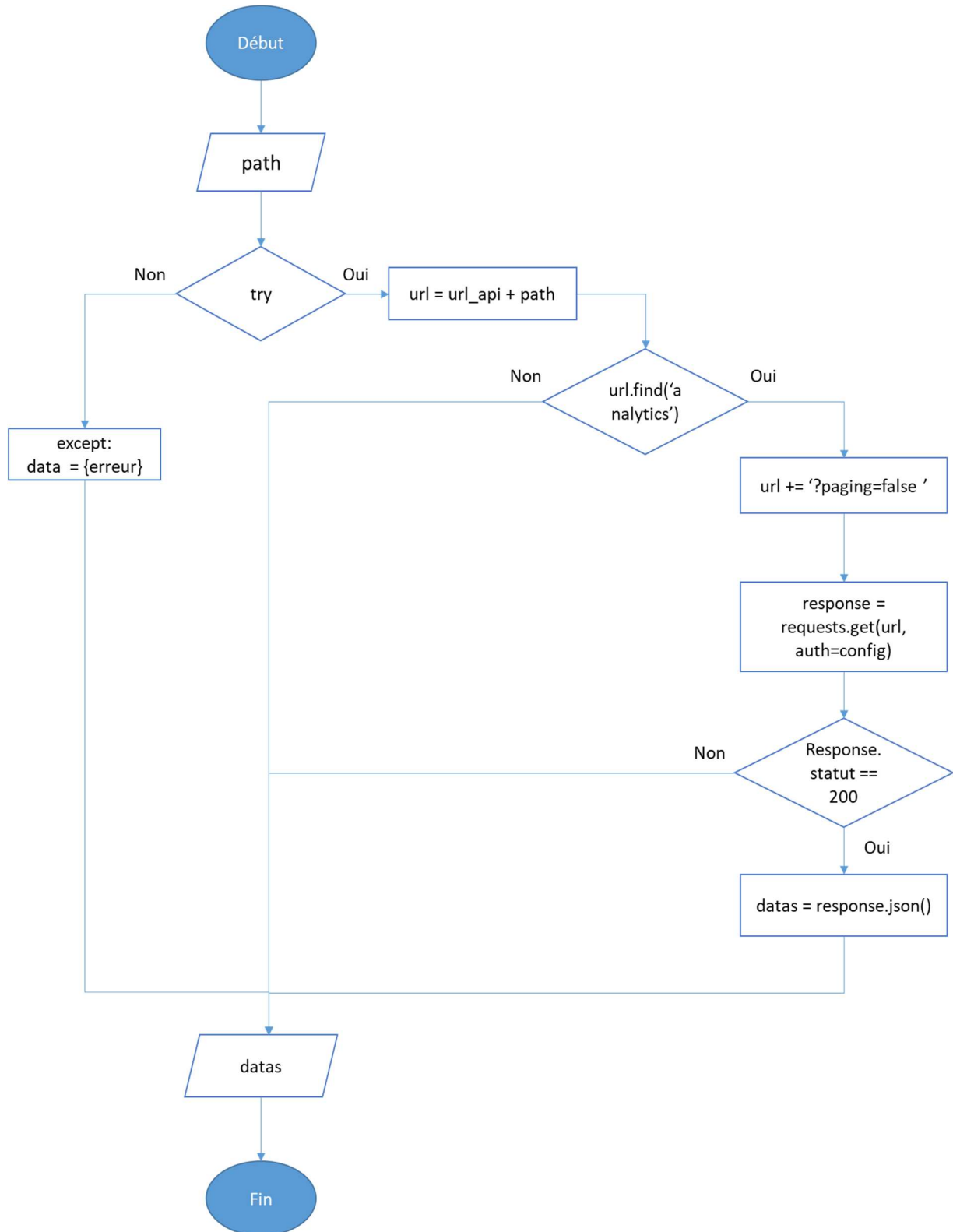


Figure 3.1: Algorithme de la fonction `api_get_data()`

3.1. 2. Partie HTML et Javascript :

3.1.2.1. Validation des informations de l'utilisateur vers la base de données :

Ce code est une fonction asynchrone Javascript nommée **validateAndSubmitForm()** qui est utilisée pour valider et soumettre un formulaire. Voici une explication détaillée de chaque partie du code :

Récupération des valeurs du formulaire :

Javascript

```
const nom = $('#lastname').val();
const prenom = $('#firstname').val();
const structure = $('#structure').val();
const lieu_travail = $('#jobplace').val();
const telephone = $('#phone').val();
const email = $('#email').val();
```

Ces lignes utilisent jQuery (\$) pour sélectionner les éléments du formulaire par leur ID (#lastname, #firstname, etc.) et récupérer les valeurs entrées par l'utilisateur.

Validation des champs du formulaire :

Javascript

```
if (!nom || !structure || !lieu_travail || !telephone || !email) {
    return;
}
```

Cette condition vérifie si certaines des variables récupérées du formulaire (nom, structure, lieu_travail, telephone, email) sont vides ou nulles. Si l'une d'entre elles est vide, la fonction retourne immédiatement, ce qui signifie que le formulaire n'est pas valide et la soumission ne se poursuit pas.

Construction des données du formulaire :

Javascript

```
const dataForm = {
    query: 'form',
```

```

    datas: {
        nom: nom,
        prenom: prenom,
        structure: structure,
        lieu_travail: lieu_travail,
        telephone: telephone,
        email: email,
    }
};

```

Cette partie crée un objet **dataForm** qui représente les données du formulaire. Cet objet contient une propriété **query** avec la valeur **'form'** et une propriété **datas** contenant les valeurs des champs du formulaire.

Envoi des données au serveur avec la fonction `getDataFromServer` :

Javascript

```
const responseForm = getDataFromServer(dataForm);
```

Cette ligne appelle une fonction **getDataFromServer** avec les données du formulaire (**dataForm**).

Suppression de la modale et du fond de modale :

Javascript

```

modal.remove();
$('.modal-backdrop').remove();

```

Ces lignes suppriment un élément modal et son fond associé. Cela est lié à la manipulation de l'interface utilisateur modale.

En résumé, cette fonction est destinée à être appelée lors de la validation du formulaire. Elle récupère les valeurs du formulaire, les valide, envoie les données au serveur (à travers une fonction **getDataFromServer**), puis supprime la modale associée au formulaire.

3.1.2.2. Envoi et récupération de données avec le Serveur (Django) :

Cette fonction Javascript, nommée **getDataFromServer**, est une fonction asynchrone destinée à envoyer des données au serveur via une requête POST. Voici une explication détaillée du code :

Paramètre **data** :

Javascript

```
async function getDataFromServer(data) {}
```

La fonction prend un seul paramètre **data**, qui représente les données à envoyer au serveur. Cela suppose que **data** est un objet Javascript contenant les informations nécessaires pour la requête POST.

Bloc **try-catch** pour la gestion des erreurs :

Javascript

```
try {  
    // ...  
} catch (error) {  
    console.error('Erreur lors de la récupération des données :', error);  
    throw error; // Rejeter l'erreur pour la gérer dans la fonction  
appelante  
}
```

La fonction utilise un bloc **try-catch** pour gérer les erreurs potentielles pendant l'exécution du code asynchrone. Si une erreur se produit, elle est capturée et une information d'erreur est affichée dans la console. Ensuite, l'erreur est rejetée (**throw error**), ce qui permet à la fonction appelante de gérer l'erreur si nécessaire.

Envoi de la requête POST avec **fetch** :

Javascript

```
const response = await fetch(URL, {  
    method: 'POST',  
    headers: {  
        'Content-Type': 'application/json',  
        'X-CSRFToken': csrf_token,  
    },  
    body: JSON.stringify(data),  
});
```

La fonction utilise **fetch** pour envoyer une requête POST au serveur. Les options de la requête incluent la méthode (**'POST'**), les en-têtes (spécifiant le type de contenu comme JSON

et incluant un jeton CSRF), et le corps de la requête qui est créé en convertissant l'objet data en une chaîne JSON à l'aide de `JSON.stringify(data)`.

CSRFToken, souvent appelé jeton anti-CSRF, est une mesure de sécurité qui garantit qu'une requête provient bien du site web légitime et non d'une source malveillante. L'objectif de ce jeton est de vérifier l'origine d'une requête HTTP et de s'assurer qu'elle est légitime.

Extraction des données JSON de la réponse :

Javascript

```
const dataJson = await response.json();  
return dataJson;
```

La fonction utilise **`await response.json()`** pour extraire les données JSON de la réponse. Cette étape est asynchrone, donc la fonction attend que la conversion de la réponse en JSON soit terminée avant de continuer. Les données JSON extraites sont ensuite retournées.

En résumé, cette fonction effectue une requête POST asynchrone vers le serveur (Django) en utilisant **`fetch`**. Elle envoie des données au serveur, récupère la réponse au format JSON, et retourne ces données JSON. En cas d'erreur lors de la requête, l'erreur est capturée, affichée dans la console, puis rejetée pour être gérée dans la fonction appelante.

3.1.2.3. Affichage de la graphique :

Cette fonction Javascript, nommée **`showChart()`**, est utilisée pour afficher un graphique sur la page web. Voici une explication détaillée du code :

Sélection de l'objet DOM :

Javascript

```
var objetct = $('#' + dataDashboard[keyRow][keyCol]['id']);
```

Cette ligne utilise jQuery (\$) pour sélectionner un élément du DOM avec l'ID spécifié. L'ID est construit à partir des données contenues dans **`dataDashboard`**.

Suppression du contenu existant :

Javascript

```
objetct.find('.overflow-auto').remove();
```

Cette ligne supprime tout contenu existant associé à l'élément sélectionné.

Ajout d'un indicateur de chargement :

Javascript

```
var loading = $('#spinner').clone();
loading.show();
objetct.append(loading);
```

Ces lignes créent un indicateur de chargement (clone d'un élément avec l'ID spinner) et l'ajoutent à l'élément sélectionné.

Création de l'élément Canvas pour le graphique :

Javascript

```
var chart = $('<canvas></canvas>');
chart.attr('id', 'Item-' + dataDashboard[keyRow][keyCol]['id']);
var ctx = chart[0].getContext('2d');
```

Ces lignes créent un élément canvas pour le graphique, définissent son ID et obtiennent son contexte de rendu en 2D.

Construction des données pour la requête au serveur :

Javascript

```
if (periods.length !== 0) {
    var data = { query: 'chart', datas: dataDashboard[keyRow][keyCol],
filter: periods };
    console.log('Filtered');
} else {
    var data = { query: 'chart', datas: dataDashboard[keyRow][keyCol] };
}
```

Ces lignes construisent l'objet **data** qui sera envoyé au serveur pour récupérer les données du graphique. La propriété filter est ajoutée si la variable periods n'est pas vide.

Récupération des données du serveur avec getDataFromServer :

Javascript

```
var chartData = await getDataFromServer(data);
```

Cette ligne utilise **await** pour attendre que la fonction **getDataFromServer** soit résolue, récupérant ainsi les données du graphique.

Ajout du graphique à l'élément DOM :

Javascript

```
objetct.append($('

Ces lignes ajoutent l'élément canvas (contenant le graphique) à l'élément sélectionné.



Création et configuration du graphique avec Chart.js :



Javascript



```
var chartItem = new Chart(ctx, { /* ... */ });
```



Ces lignes utilisent la bibliothèque Chart.js pour créer et configurer un graphique à partir des données récupérées.



Mise à jour du graphique :



Javascript



```
chartItem.update();
```



Cette ligne met à jour le graphique après sa création et configuration.



Suppression de l'indicateur de chargement et réactivation des éléments désactivés :



Javascript



```
objetct.find('.spinner-border').remove();
objetct.find('.dropdown-item').prop('disabled', false);
```



Ces lignes suppriment l'indicateur de chargement et réactivent les éléments de la liste déroulante.



Suppression de l'élément de chargement :



Javascript



```
$('#' + dataDashboard[keyRow][keyCol]['id']).parent('.col-md')
.remove(loading);
```



Cette ligne supprime l'élément indicateur de chargement.



En résumé, cette fonction effectue plusieurs opérations, notamment la construction d'un graphique à partir de données provenant du serveur, l'affichage de cet indicateur de chargement, la gestion des filtres de période, et l'utilisation de la bibliothèque Chart.js pour créer et afficher un graphique interactif sur la page web.



39


```

3.1.2.4. Affichage du tableau

Cette fonction Javascript, nommée **showTable**, est utilisée pour afficher une table sur une page web. Voici une explication détaillée du code :

Sélection de l'objet DOM :

Javascript

```
var objetct = $('#' + dataDashboard[keyRow][keyCol]['id']);
```

Cette ligne utilise jQuery (\$) pour sélectionner un élément du DOM avec l'ID spécifié. L'ID est construit à partir des données contenues dans dataDashboard.

Suppression du contenu existant :

Javascript

```
objetct.find('.overflow-auto').remove();
```

Cette ligne supprime tout contenu existant associé à l'élément sélectionné.

Ajout d'un indicateur de chargement :

Javascript

```
var loading = $('#spinner').clone();  
loading.show();  
objetct.append(loading);
```

Ces lignes créent un indicateur de chargement (clone d'un élément avec l'ID spinner) et l'ajoutent à l'élément sélectionné.

Création de l'élément Table pour la table :

Javascript

```
var table = $('#monTableau').clone();  
table.attr('id', 'Item-' + dataDashboard[keyRow][keyCol]['id']);  
table.show();
```

Ces lignes créent un élément de tableau (<table>) et lui attribuent un nouvel ID basé sur les données de dataDashboard.

Construction des données pour la requête au serveur :

Javascript

```
if (periods.length !== 0) {
```

```

    var data = { query: 'reportTable', datas: dataDashboard[keyRow][keyCol],
filter: periods };

    console.log('Filtered');
} else {
    var data = { query: 'reportTable', datas: dataDashboard[keyRow][keyCol]
};
    console.log('Normal');
}

```

Ces lignes construisent l'objet data qui sera envoyé au serveur pour récupérer les données de la table. La propriété filter est ajoutée si la variable periods n'est pas vide.

Récupération des données du serveur avec getDataFromServer :

Javascript

```
var dataItem = await getDataFromServer(data);
```

Cette ligne utilise **await** pour attendre que la fonction **getDataFromServer** soit résolue, récupérant ainsi les données de la table.

Affichage des données de la table dans la console :

Javascript

```

console.log('ShowTable : ' + dataDashboard[keyRow][keyCol] +
objetct.html());

console.log(data);
console.log(dataItem);

```

Ces lignes affichent des informations liées à la table dans la console à des fins de débogage.

Suppression de l'élément de chargement :

Javascript

```

objetct.find('.overflow-auto').remove();

objetct.append($('<div class="container overflow-auto justify-content-
center align-items-center" style="max-width: 580px; max-height: 300px;
margin: 0px"></div>').append(table));

```

Ces lignes suppriment l'indicateur de chargement et ajoutent l'élément de tableau à l'élément sélectionné.

Configuration et initialisation de la DataTable :

Javascript

```
table.DataTable({
  "data": dataItem['data'],
  "columns": dataItem['columns']
});
```

Ces lignes utilisent le plugin DataTable pour jQuery pour configurer et initialiser la table avec les données récupérées.

Suppression de l'indicateur de chargement et réactivation des éléments désactivés :

Javascript

```
objetct.find('.spinner-border').remove();
$('#'+dataDashboard[keyRow][keyCol]['id']).parent('.col-md')
.remove(loading);
```

Ces lignes suppriment l'indicateur de chargement et réactivent les éléments associés.

En résumé, cette fonction effectue plusieurs opérations, notamment la construction d'une table à partir de données provenant du serveur, l'affichage de cet indicateur de chargement, la gestion des filtres de période, et l'utilisation du plugin DataTable pour jQuery pour créer et afficher une table interactive sur la page web.

3.1.2.5. Récupération des informations sur les éléments du tableau de bord :

Cette fonction Javascript, nommée **getDashboardItem**, est utilisée pour récupérer et afficher les éléments d'un tableau de bord sur une page web. Voici une explication détaillée du code :

Création d'une surcouche (overlay) pendant le chargement :

Javascript

```
var overlay = $('<div class="d-flex justify-content-center align-items-center overlay position-absolute" style="z-index: 8;">\
  <div class="spinner-border text-primary" role="status" style="width: 7rem; height: 7rem;">\
    <span class="visually-hidden">Loading...</span>\
  </div>\
</div>');
```

Cette surcouche est une boîte de chargement qui apparaît au centre de la page pendant le chargement des données du tableau de bord.

Modification de l'apparence des boutons de menu et du titre du tableau de bord :

Javascript

```
$('.menu-btn').removeClass('bg-primary fw-bold text-white');  
$('#' + idDashboardItem).addClass('bg-primary fw-bold text-white');  
$("#titleDashboardItem").text($('#' + idDashboardItem).text());
```

Ces lignes ajustent l'apparence des boutons de menu en ajoutant ou supprimant des classes CSS.

Préparation des données pour la requête au serveur :

Javascript

```
if (periods.length !== 0) {  
    var data = { query: 'dashboardItem', datas: idDashboardItem, filter:  
periods };  
    console.log('Filtered');  
} else {  
    var data = { query: 'dashboardItem', datas: idDashboardItem };  
    console.log('Normal');  
}
```

Ces lignes préparent les données nécessaires pour la requête au serveur. Si des périodes sont spécifiées, un filtre est ajouté.

Récupération des données du tableau de bord à partir du serveur :

Javascript

```
dataDashboard = await getDataFromServer(data);
```

Cette ligne utilise await pour attendre la résolution de la promesse retournée par la fonction getDataFromServer.

Nettoyage du contenu existant :

Javascript

```
container.children().html("");
```

Cette ligne vide le contenu existant du conteneur.

Itération sur les données du tableau de bord et construction du HTML :

Javascript

```

for (rowKey in dataDashboard){
    var row = $('<div class="row"></div>');
    for (colKey in dataDashboard[rowKey]){
        var col = $('<div class="col-md bg-light text-dark m-1"><div id="' +
dataDashboard[rowKey][colKey]['id'] + '" class="container border rounded-3
bg-white justify-content-center align-items-center" style="max-width: 580px;
min-height: 350px; padding: 0px; position: relative"></div></div>');
        // ...
        row.append(col);
    }
    container.append(row);
}

```

Ces lignes créent une structure HTML pour afficher les éléments du tableau de bord. Chaque élément est placé dans une colonne du tableau.

Suppression de la surcouche après le chargement :

Javascript

```
overlay.remove();
```

Cette ligne supprime la surcouche de chargement après le chargement des données.

Affichage des éléments du tableau de bord :

Javascript

```

for (rowKey in dataDashboard){
    for (colKey in dataDashboard[rowKey]){
        if (dataDashboard[rowKey][colKey]['type'] === "VISUALIZATION"){
            showChart(rowKey, colKey);
        } else if (dataDashboard[rowKey][colKey]['type'] === "pivot_table") {
            showTable(rowKey, colKey);
        } else if (dataDashboard[rowKey][colKey]['type'] === "MAP") {
            showMap(rowKey, colKey);
        }
    }
}
}

```

Ces lignes appellent les fonctions **showChart**, **showTable**, et **showMap** pour afficher les éléments spécifiques du tableau de bord en fonction de leur type.

3.1.2.6. Affichage de la Carte Géographique :

Cette fonction Javascript, nommée **showMap**, est utilisée pour afficher une carte interactive sur une page web en utilisant la bibliothèque Leaflet. Voici une explication détaillée du code :

Initialisation des éléments de la carte :

Javascript

```
var objetct = $('#' + dataDashboard[keyRow][keyCol]['id']);
objetct.find('.overflow-auto').remove();
var loading = $('#spinner').clone();
loading.show();
objetct.append(loading);
var map_ = $('<div></div>');
map_.attr('id', 'Item-' + dataDashboard[keyRow][keyCol]['id']);
map_.css('width', '575px');
map_.css('height', '300px');
```

Ces lignes préparent l'élément HTML qui contiendra la carte. La carte est créée en utilisant Leaflet et est ajoutée à cet élément.

Préparation des données pour la requête au serveur :

Javascript

```
if (periods.length !== 0){
    var data = { query: 'map', datas: dataDashboard[keyRow][keyCol], filter:
periods };
    console.log('Filtered');
} else {
    var data = { query: 'map', datas: dataDashboard[keyRow][keyCol] };
    console.log('Normal');
}
var statesData = await getDataFromServer(data);
```

Ces lignes préparent les données nécessaires pour la requête au serveur et récupèrent les données géographiques (souvent au format GeoJSON) pour afficher sur la carte.

Création de la carte avec Leaflet :

Javascript

```
var map = L.map(map_.attr('id')).setView(statesData.center, 6);
var osm = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: '&copy; OpenStreetMap contributors',
  href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
});
osm.addTo(map);
```

Ces lignes créent une carte Leaflet avec une couche OpenStreetMap de base.

Ajout de différentes couches à la carte :

Javascript

```
var CartoDB_DarkMatter = L.tileLayer(...);
var googleStreets = L.tileLayer(...);
var googleSat = L.tileLayer(...);
var Stamen_Watercolor = L.tileLayer(...);
CartoDB_DarkMatter.addTo(map);
googleStreets.addTo(map);
googleSat.addTo(map);
Stamen_Watercolor.addTo(map);
```

Ces lignes ajoutent plusieurs couches de tuiles (tile layers) à la carte, y compris des options telles que des cartes Google, des cartes OpenStreetMap, etc.

Contrôle des couches de la carte :

Javascript

```
var baseLayers = {
  "Satellite":googleSat,
  "Google Map":googleStreets,
  "Water Color":Stamen_Watercolor,
  "OpenStreetMap": osm,
};
L.control.layers(baseLayers).addTo(map);
```

Ces lignes ajoutent un contrôle des couches qui permet à l'utilisateur de basculer entre différentes couches de carte.

Ajout des données géographiques à la carte :

Javascript

```
L.geoJSON(statesData).addTo(map);
```

Cette ligne ajoute les données géographiques (GeoJSON) à la carte Leaflet.

Personnalisation du style des zones géographiques :

Javascript

```
L.geoJson(statesData, {style: style}).addTo(map);  
function getColor(d) {...}  
function style(feature) {...}
```

Ces lignes définissent la couleur et le style des zones géographiques en fonction de certaines propriétés.

Gestion des interactions sur la carte :

Javascript

```
function highlightFeature(e) {...}  
function resetHighlight(e) {...}  
function zoomToFeature(e) {...}  
function onEachFeature(feature, layer) {...}
```

Ces fonctions gèrent différentes interactions telles que le survol d'une zone, le clic sur une zone, etc.

Ajout de légendes et d'informations sur la carte :

Javascript

```
var info = L.control();  
info.onAdd = function (map) {...}  
info.update = function (props) {...}  
info.addTo(map);  
var legend = L.control({position: 'bottomright'});  
legend.onAdd = function (map) {...}  
legend.addTo(map);
```

Ces lignes ajoutent une info-bulle (tooltip) et une légende à la carte Leaflet.

Suppression des éléments de chargement après l'affichage de la carte :

Javascript

```
objetct.find('.spinner-border').remove();  
$('#' + dataDashboard[keyRow][keyCol]['id']).parent('.col-md')  
.remove(loading);
```

Ces lignes suppriment les éléments de chargement une fois que la carte a été correctement affichée.

3.1.2.7. Création la barre de navigation:

Ce code représente une barre de navigation (navbar) dans une page web, généralement utilisée pour afficher le titre du site et d'autres éléments de navigation. Voici une explication détaillée :

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">: Cela crée une barre de navigation (<nav>) avec un style sombre (bg-dark) et des éléments de texte en couleur claire (navbar-dark). La barre de navigation est conçue pour s'étendre (navbar-expand-sm) sur les petits écrans (sm signifie small, c'est-à-dire petit écran).

<div class="container-fluid">: C'est une classe Bootstrap qui enveloppe le contenu de la barre de navigation pour le mettre en forme et le rendre réactif.

<ul class="navbar-nav">: C'est une liste non ordonnée () qui contient les éléments de la barre de navigation.

<li class="nav-item">: C'est un élément de liste () spécifique à la barre de navigation (nav-item).

: C'est un lien (<a>) qui agit comme un élément de navigation. La classe nav-link est spécifique à la barre de navigation, et text-white définit la couleur du texte. L'attribut href="#" indique que le lien pointe vers la même page (utilisé comme un espace réservé dans cet exemple). La classe active indique que le lien est actuellement sélectionné.

<h3>{% block menu_title %}{% endblock %}Ministère de la Santé Publique Madagascar</h3>: C'est un titre de niveau 3 (<h3>) qui peut être étendu (block menu_title). Le texte "Ministère de la Santé Publique Madagascar" est le titre principal de la barre de navigation.

<button class="btn btn-dark" type="button" data-bs-toggle="offcanvas" data-bs-target="#offcanvasRight" aria-controls="offcanvasRight">Info</button>: C'est un bouton qui, lorsqu'il est cliqué (data-bs-toggle="offcanvas"), active un élément offcanvas (une barre latérale ou un panneau qui s'affiche à côté de la page principale). Il a le texte "Info" et utilise le style sombre de Bootstrap (btn-dark).

3.1.2.8. Mise en place des contenus de la page :

Ce code représente une section dans une page web qui utilise une table (<table>) avec l'id "monTableau" pour afficher des données tabulaires. Cet élément sert de référence pour l'affichage des tableaux. Voici une explication détaillée :

<section class="section">: C'est une balise HTML5 de type section qui est utilisée pour définir une section dans la page. Elle peut contenir différents éléments et contenus.

<table id="monTableau" class="display table table-striped compact" style="display: none;">: Cette balise <table> crée une table HTML avec l'identifiant "monTableau". Les classes "display", "table", et "table-striped" sont des classes Bootstrap qui ajoutent des styles à la table pour la rendre plus lisible et attrayante. La classe "compact" est également probablement associée à un plugin ou à une fonctionnalité particulière pour rendre la table plus compacte. L'attribut style="display: none;" indique que la table doit être initialement cachée (display: none;).

<thead>: C'est l'en-tête de la table qui contient les lignes d'en-tête.

<th colspan="6">Tableau</th>: C'est une cellule d'en-tête (<th>) qui s'étend sur 6 colonnes (colspan="6") et contient le texte "Tableau". Cela crée une cellule qui fusionne horizontalement sur six colonnes.

{% block section %} <!-- Contenus --> {% endblock %}: C'est un bloc template Django. Cela permet d'étendre la section et d'insérer du contenu spécifique à cet endroit dans la page. Dans votre exemple, le contenu spécifique serait défini dans un autre fichier ou une autre partie du code.

En résumé, cette section de code crée une structure de tableau avec un identifiant spécifique ("monTableau") qui peut être utilisée pour afficher des données tabulaires, et elle utilise un bloc template pour permettre l'insertion de contenu spécifique. La table est initialement cachée (display: none;).

3.1. 3. Résultats :

Pour résumer, nous devons avoir un formulaire de renseignement et un tableau de bord. Dans le tableau de bord, on a une liste verticale de programmes, une grille de 3 colonnes d'objets, une fonctionnalité de filtrage mensuel et les fonctionnalités qui changent de type d'objet (Graphique, Tableau croisé dynamique et Carte Graphique).

Formulaire de renseignement personnel

The image shows a web application interface. In the foreground, a modal form titled "Renseignement Personnel" is displayed. It contains the following fields: "Nom" (with a red error icon), "Prénom" (with a green success icon), "Structure:" (a dropdown menu showing "Informatique" with a green success icon), "Lieu de Travail" (with a red error icon), "Téléphone" (with a red error icon), and "Email" (with a red error icon). A blue "Valider" button is at the bottom of the form. The background is a dashboard for the "Ministère de la Santé Publique Madagascar". It features a header with the ministry's name and an "Info" link. Below the header, there's a list of items, including "1-(Nv RMA CSB) Complétude et Promptitude 6 derniers Mois / Région". The dashboard also has a "Filtre..." button and "Ajouter" and "Supprimer" buttons.

Figure 3.2: Formulaire de renseignement personnel sur l'utilisateur

Tableau de bord initial

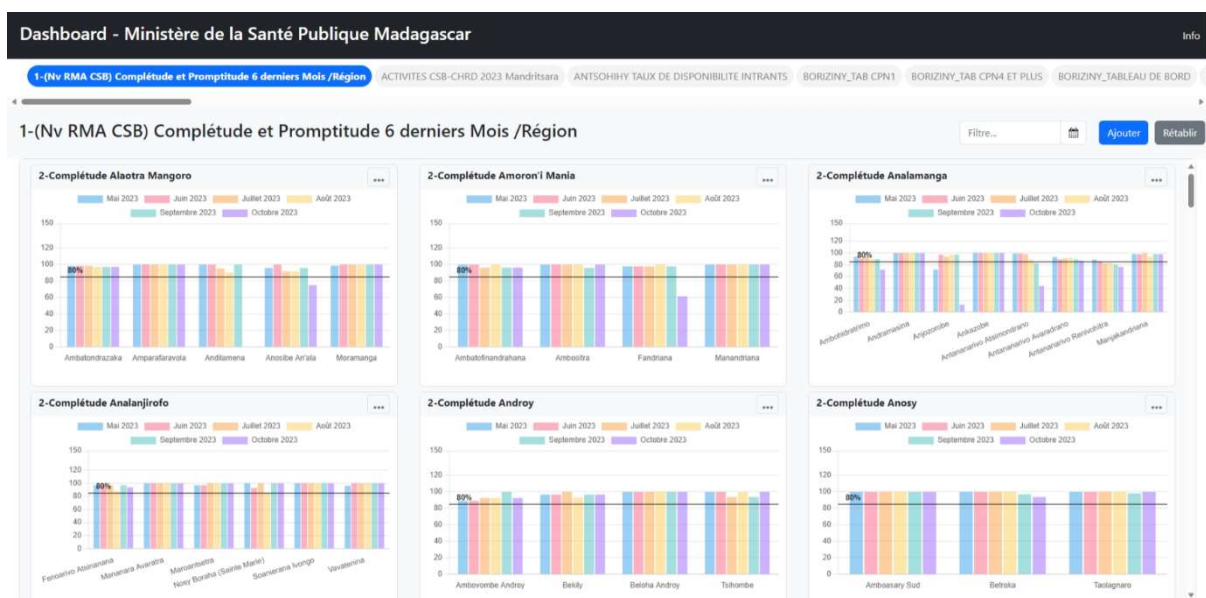


Figure 3.3: Tableau de bord initial de la nouvelle plateforme WEB

Tableau de bord avec Graphique, Tableau et Carte

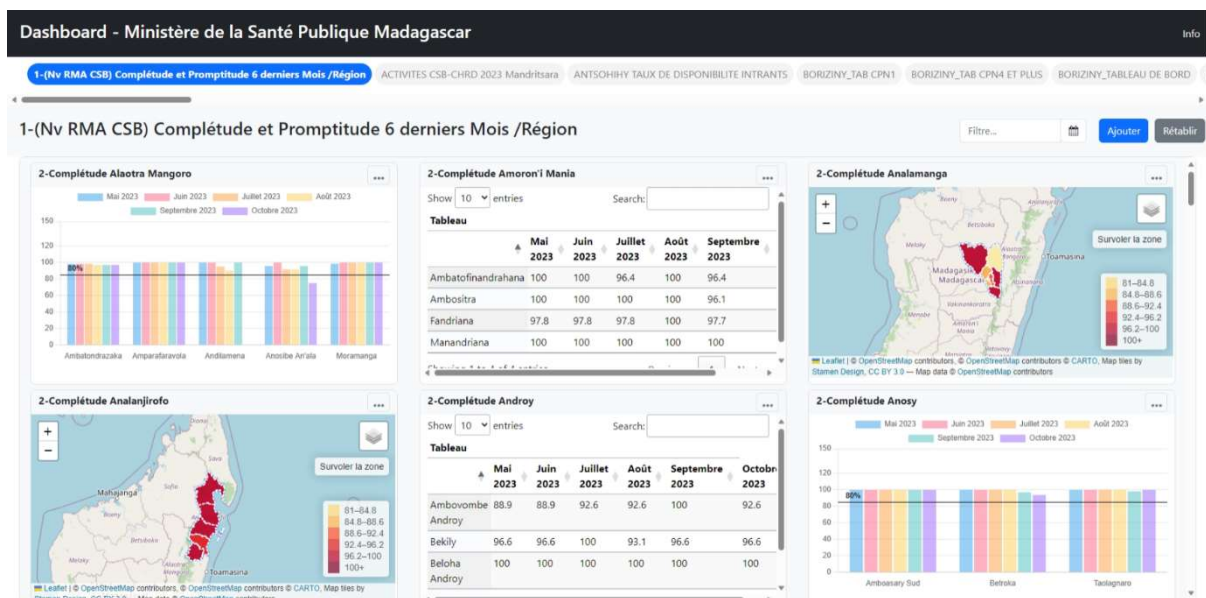


Figure 3.4: Tableau de bord manipulé de la nouvelle plateforme WEB

Tableau de bord avec Filtre

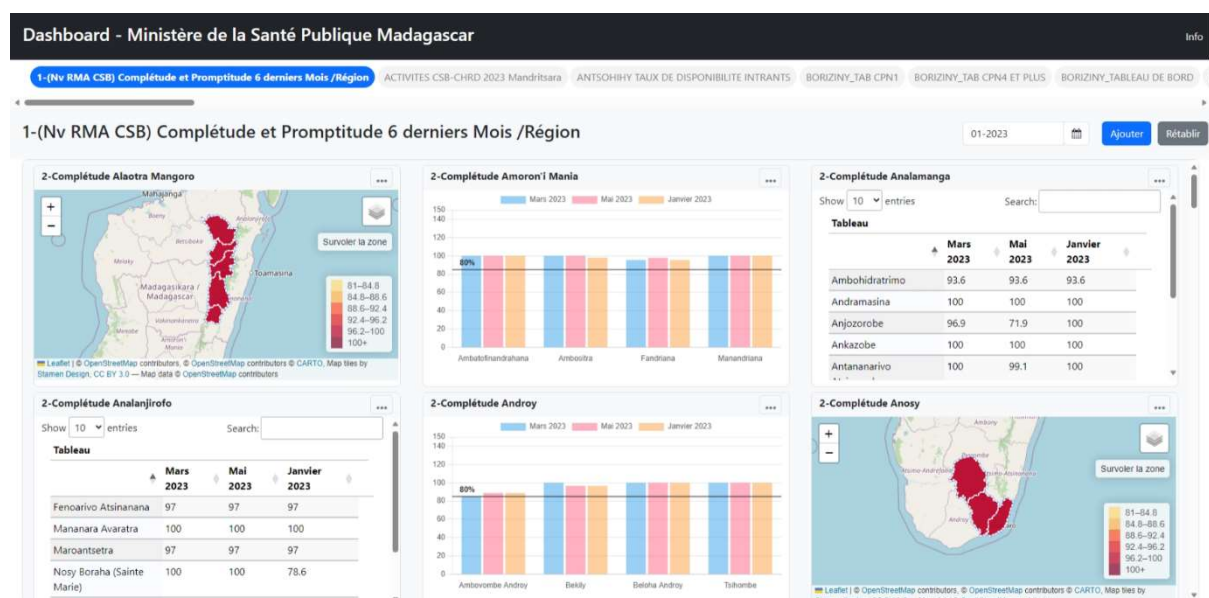


Figure 3.5: Tableau de bord filtré de la nouvelle plateforme WEB

3.2. TESTS

Sur DHIS2 du Ministère de la Santé Publique Madagascar :

Pour les tests, on m'a donné des identifiants d'authentification pour se connecter à la plateforme DHIS2 du Ministère de la Santé Publique, avec le lien « <https://ministere-sante.mg> ».

DHIS2 Ministère de la Santé Publique

dhis2

Enter

Nom d'utilisateur

Mot de passe

Enter

Produit par DHIS 2

French

Figure 3.6: Page d'authentification de DHIS2 du Ministère de la Santé Publique

Le tableau de bord

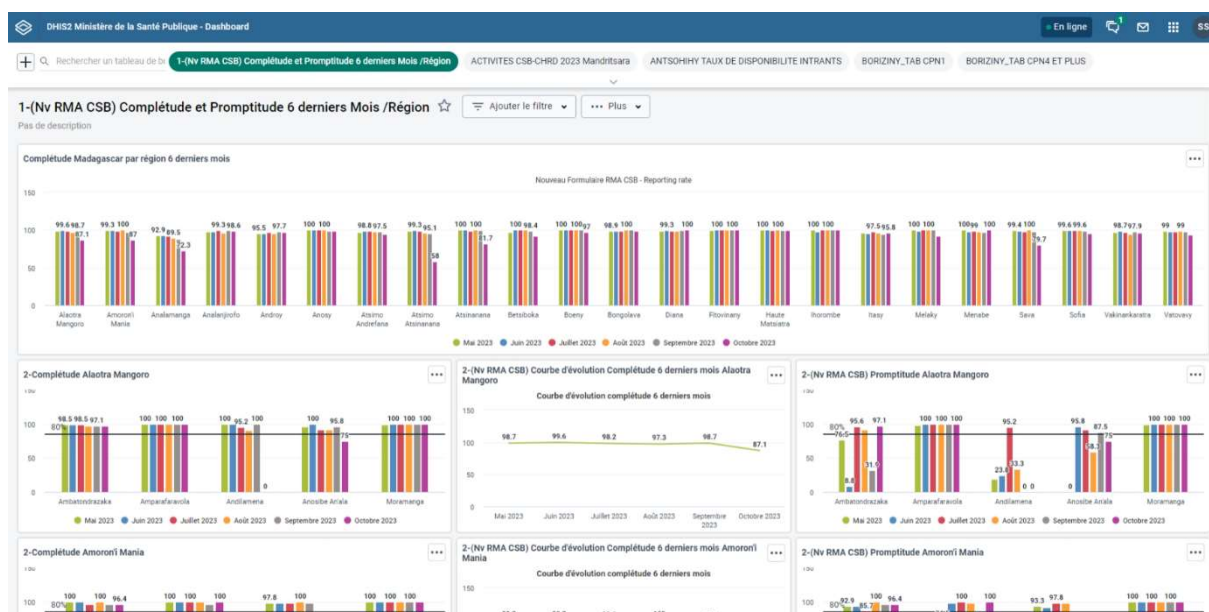


Figure 3.7: Application Tableau de bord de DHIS2 du Ministère de la Santé Publique

Sur le DHIS2 Demo :

Pendant la maintenance ou manifestation d'erreurs sur la plateforme DHIS2 du Ministère de la Santé Publique, je fais mes tests sur DHIS2 Demo. Ce dernier partage des identifiants publiques pour les utilisateurs qui veulent tester DHIS2 avant de le télécharger et l'installer, voici le lien « <https://play.dhis2.org/40.2.0> ».

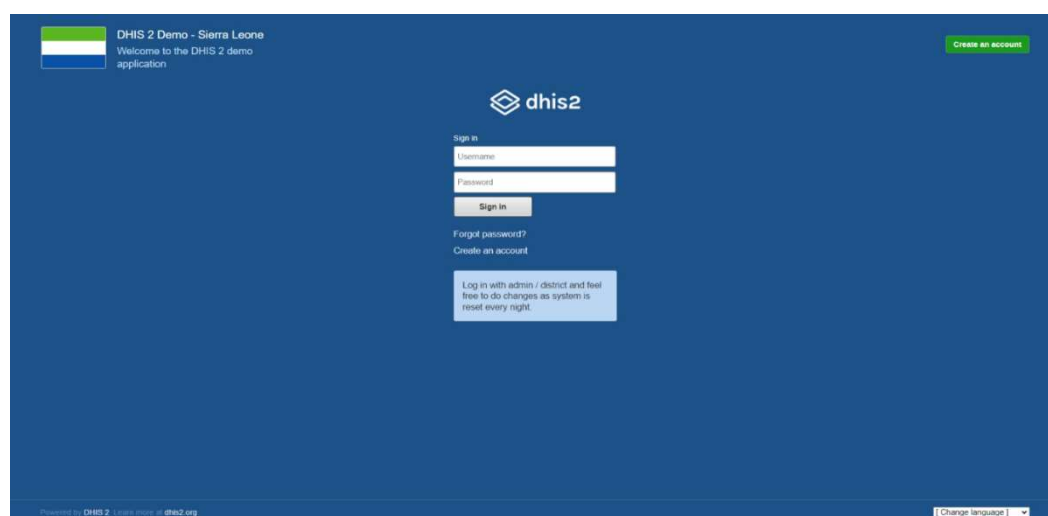


Figure 3.8: Page d'authentification de DHIS2 Demo

Il affiche des informations sanitaires sur Sierra Leone (un État d'Afrique de l'Ouest). On peut changer de version en changeant la dernière partie du lien « 40.2.0 » ou en allant sur la première page avec le lien « <https://play.dhis2.org> ».

Tableau de bord

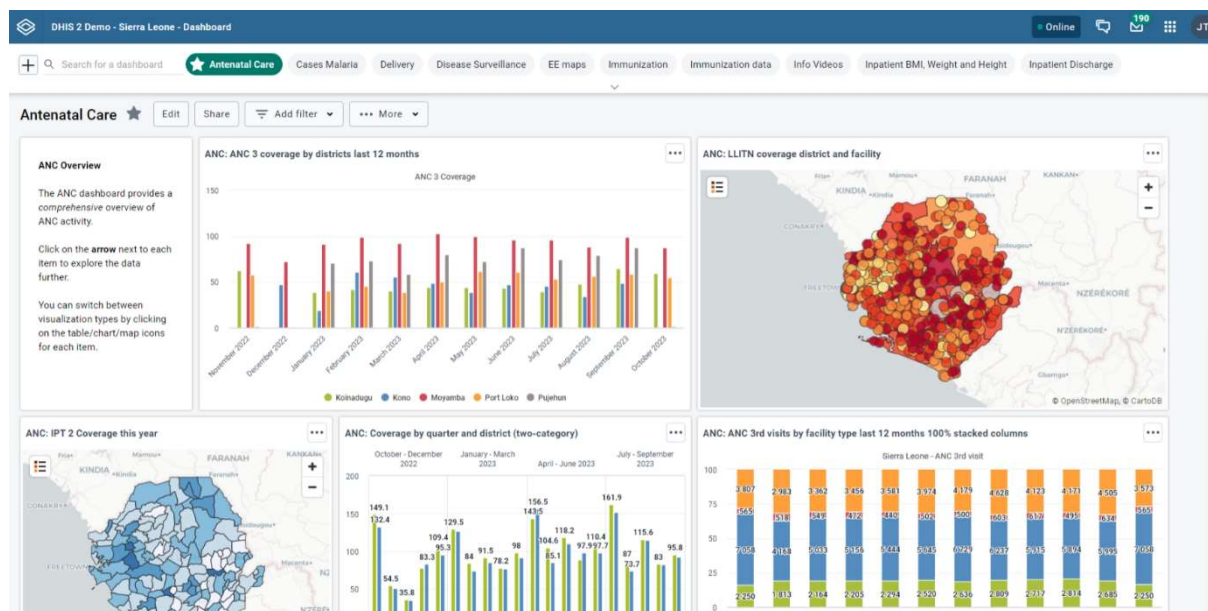


Figure 3.9: Application Tableau de bord de DHIS2 Demo

Différence entre DHIS2 du Ministère de la Santé Publique à Madagascar et DHIS2 Demo :

Pendant quelques tests faits avec les deux plateformes, j'ai constaté les contradictions dans le DHIS2 du Ministère de la Santé Publique au niveau des APIs.

Par exemple, une requête pour obtenir les données d'un objet du tableau de bord de type **VISUALISATION**.

Tableau 3.1: Différence entre DHIS2 Demo et de MSANP

| | DHIS2 Demo | DHIS2 du MSPM |
|------------|---|---|
| uid | DeRrc1gTMjn | pTtBROuQV4b |
| href | https://play.dhis2.org/40.2.0/api/visualizations/DeRrc1gTMjn | https://ministere-sante.mg/api/visualizations/pTtBROuQV4b |
| legendSets | fqs276KXCXi | Vide |

Les données de légendes sont récupérables avec les APIs de type **DATA_X** (Quoi : indicator, dataSet, dataElement, etc). Donc, en trouvant l'uid du data_x, on devrait trouver la légende appliquée à la Carte Géographique de l'objet.

Mais pour le cas de DHIS2 du Ministère de la Santé Publique, les legendSets sont vides pour tous les objets du Tableau de bord. En parcourant un peu dans les données de l'API **legendSets** avec le lien « <https://ministere-sante.mg/api/legendSets> », j'ai trouvé 34 configurations de légende dont y comprises les légendes « Par défaut » et « Completude ». Je précise aussi la légende « Completude » parce que dans le premier tableau de bord de DHIS2 du Ministère de la Santé Publique, il affiche des objets de Compétude.

CONCLUSION

La réalisation de cette plateforme web représente une avancée significative dans la modernisation et l'optimisation des pratiques de gestion de l'information sanitaire au sein du Ministère de la Santé Publique. À travers ce projet, nous avons concrétisé notre engagement envers l'amélioration de la diffusion des données de santé, contribuant ainsi à renforcer les fondations sur lesquelles reposent les décisions cruciales en matière de santé publique.

En conclusion, cette initiative ne marque pas seulement un jalon technologique, mais incarne également notre dévouement envers l'amélioration continue des services de santé. Nous sommes convaincus que cette plateforme deviendra un outil essentiel, facilitant la collaboration, renforçant la transparence et, surtout, contribuant à l'amélioration globale de la santé publique à travers notre nation.

REFERENCES

BIBLIOGRAPHIQUES

- [1] LOGICIEL DHIS2 Manuel d'utilisateur final de DHIS2, édition Octobre 2020
- [2] Learning Geospatial Analysis with Python, 2^{ème} Edition, Joel Lawhead

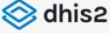
WEBOGRAPHIQUES

- [3] DHIS2 Documentation, docs.dhis2.org
- [4] DHIS2 Developer Portal, developers.dhis2.org
- [5] Stoplight, dhis2.stoplight.io/docs/dhis2/
- [6] Chart.js Documentation, <https://www.chartjs.org/docs>
- [7] DataTables Manual, <https://datatables.net/manual/>
- [8] Leaflet Documentation, <https://mourner.github.io/Leaflet/reference.html>
- [9] OpenStreetMap, <https://wiki.openstreetmap.org/>
- [10] W3schools, www.w3schools.com
- [11] Stack Overflow, www.stackoverflow.com

ANNEXES

Annexe 1. Les résultats de recherches














- Première de DHIS2 Demo

**DHIS 2 play**








Here you can find demo instances for the most recent versions of DHIS2! Each of the links below has a version of DHIS2 with our demo database.

Log in with:
- user: **admin**
- password: **district**

For links to additional demo databases for specific DHIS2 apps and use cases, please visit <https://www.dhis2.org/demo>

| Name | Description |
|--|---|
| Stable | |
|  40.2.0 | Demo of DHIS 2 version 40 latest stable patch |
|  2.39.3.1 | Demo of DHIS 2 version 2.39 latest stable patch |
|  2.38.5 | Demo of DHIS 2 version 2.38 latest stable patch |
|  2.37.10 | Demo of DHIS 2 version 2.37 latest stable patch |
| Canary | |
|  2.40nightly | Demo of DHIS 2 version 40 latest nightly build |
|  2.39nightly | Demo of DHIS 2 version 2.39 latest nightly build |
|  2.38nightly | Demo of DHIS 2 version 2.38 latest nightly build |
|  2.37nightly | Demo of DHIS 2 version 2.37 latest nightly build |
| Under Development | |
|  dev | Cutting edge DHIS 2 development snapshot |
|  2.40dev | Build of the DHIS 2 version 2.40 development snapshot |
|  2.39dev | Build of the DHIS 2 version 2.39 development snapshot |
|  2.38dev | Build of the DHIS 2 version 2.38 development snapshot |
|  2.37dev | Build of the DHIS 2 version 2.37 development snapshot |

Other links:

| | | |
|---|---------------------------|---|
|  | Documentation | DHIS 2 documentation |
|  | Online Training Academy | Free online academy for learning DHIS2 fundamentals |
|  | Academy Training Material | Moodle instance for academy training material and tests |
|  | Downloads | Links to the DHIS 2 war files for your own deployment |
|  | App Hub | App Hub for DHIS2 web apps |
|  | Global adoption | Maps showing global use of DHIS 2 by countries and NGOs |
|  | Jira issue tracking | Tracking for DHIS2 bug reports and feature requests |

Source : www.play.dhis2.org

- Modèle Physique de Données : script SQL

```

drop table if exists SOUMISSION;

drop table if exists VISITOR;

/*=====*/
/* Table : SOUMISSION */
/*=====*/
create table SOUMISSION
(
    ID_VISITOR          int not null,
    ID_SOUMISSION        int not null,
    DATE_SOUMISSION      datetime,
    key AK_VISITOR (ID_SOUMISSION)
);

/*=====*/
/* Table : VISITOR */
/*=====*/
create table VISITOR
(
    ID_VISITOR          int not null,
    LASTNAME             varchar(100),
    FIRSTNAME            varchar(200),
    DOMAIN               varchar(100),
    COMPANY              varchar(300),
    PHONE                varchar(15),
    EMAIL                varchar(300),
    primary key (ID_VISITOR)
);

alter table SOUMISSION add constraint FK_FAIRE foreign key (ID_VISITOR)
references VISITOR (ID_VISITOR) on delete restrict on update restrict;

```

Source : PowerAMC

- Extrait de l'API visualization de DHIS2 du Ministère de la Santé Publique

```

1 {
2   "href": "https://ministere-sante.mg/api/visualizations/pTtBROuQV4b",
3   "name": "2-Complétude Alaoatra Mangoro",
4   "created": "2020-01-22T07:22:46.996",
5   "lastUpdated": "2021-09-05T11:35:51.536",
6   "translations": [],
7   "externalAccess": false,
8   "publicAccess": "-----",
9   "createdBy": {
10    "id": "aUjgxf3o6o",
11    "code": null,
12    "name": "José RATOLOJANAHARY",
13    "displayName": "José RATOLOJANAHARY",
14    "username": "JoseDSI"
15  },
16  "userGroupAccesses": [],
17  "userAccesses": [],
18  "access": {
19    "manage": false,
20    "externalize": false,
21    "write": false,
22    "read": false,
23    "update": false,
24    "delete": false
25  },
26  "favorites": [],
27  "lastUpdatedBy": {
28    "id": "aUjgxf3o6o",
29    "code": null,
30    "name": "José RATOLOJANAHARY",
31    "displayName": "José RATOLOJANAHARY",
32    "username": "JoseDSI"
33  },
34  "sharing": {
35    "owner": "aUjgxf3o6o",
36    "external": false,
37    "users": {},
38    "userGroups": {},
39    "public": "-----"
40  },
41  "dataDimensionItems": [
42    {
43      "reportingRate": {
44        "name": "Nouveau Formulaire RMA CSB - Reporting rate",
45        "translations": [],
46        "externalAccess": false,
47        "userGroupAccesses": [],
48        "userAccesses": [],
49        "access": {
50          "manage": true,
51          "externalize": false,
52          "write": true,
53          "read": true,
54          "update": true,
55          "delete": true
56        },
57        "favorites": [],
58        "sharing": {
59          "external": false,
60          "users": {},
61          "userGroups": {}
62        },
63        "shortName": "Nouveau Formulaire RMA CSB - Reporting rate",
64        "dimensionItemType": "REPORTING_RATE",
65        "legendSets": [],
66        "dataSet": {
67          "id": "mQEM6PdZg8p"
68        },
69        "metric": "REPORTING_RATE",
70        "dimensionItem": "mQEM6PdZg8p.REPORTING_RATE",
71        "displayShortName": "Nouveau Formulaire RMA CSB - Reporting rate",
72        "favorite": false,

```

Source : <https://ministere-sante.mg/api/visualizations/pTtBROuQV4b>

- Extrait de l'API visualization de DHIS2 Demo

```

1 {
2   "href": "https://play.dhis2.org/40.2.0/api/visualizations/xiLNqnSaWP3",
3   "name": "ANC: Coverage by quarter and district (two-category)",
4   "created": "2021-08-11T20:26:40.055",
5   "lastUpdated": "2021-08-11T20:26:40.055",
6   "translations": [],
7   "externalAccess": false,
8   "publicAccess": "-----",
9   "createdBy": {
10     "id": "GOLswS44mh8",
11     "code": null,
12     "name": "Tom Wakiki",
13     "displayName": "Tom Wakiki",
14     "username": "system"
15   },
16   "userGroupAccesses": [],
17   "userAccesses": [],
18   "access": {
19     "manage": false,
20     "externalize": true,
21     "write": false,
22     "read": false,
23     "update": false,
24     "delete": false
25   },
26   "favorites": [],
27   "lastUpdatedBy": {
28     "id": "GOLswS44mh8",
29     "code": null,
30     "name": "Tom Wakiki",
31     "displayName": "Tom Wakiki",
32     "username": "system"
33   },
34   "sharing": {
35     "owner": "GOLswS44mh8",
36     "external": false,
37     "users": {},
38     "userGroups": {},
39     "public": "-----"
40   },
41   "dataDimensionItems": [
42     {
43       "indicator": {
44         "id": "Uvn6LCg7dVU"
45       },
46       "dataDimensionItemType": "INDICATOR"
47     },
48     {
49       "indicator": {
50         "id": "OdiHJayrsKo"
51       },
52       "dataDimensionItemType": "INDICATOR"
53     }
54   ],
55   "organisationUnits": [
56     {
57       "id": "O6uvpzGd5pu"
58     },
59     {
60       "id": "fdc6u0vgoji"
61     },
62     {
63       "id": "lc3eMKXaEfw"
64     },
65     {
66       "id": "iUhb8gEI0Ap1"
67     }
68   ]
69 }

```

Source : <https://play.dhis2.org/40.2.0/api/visualizations/xiLNqnSaWP3>