# 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

Le Directeur Général de l’IST-D : Docteur TSIMITAMBY Briand

Le Directeur des Etudes, de la Planification et du Système d’Information :

Le Directeur de l’Ecole de Génie Industriel :

Le Responsable de Mention et mon Encadreur Pédagogique : Docteur RAZAFINDRADINA Henri Bruno

Le Chef de Service SEMIDSI :

Mon Encadreur Professionnel : Docteur RAKOTONDRABE Miora Harivony

Mon Encadreur Pédagogique : Monsieur VELOSON

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 plage d’une plateforme web dynamique pour le partage des tableaux de bord des programmes issu de DHIS2

**Domaine :**

**Système d’informations géographiquesDurée :**

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 :**

Une 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 un 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 (1ere semaine) ;
* Étude et analyse du réseau et des services existants (2eme 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 matières

[Avant-propos i](#_Toc152228152)

[Remerciements ii](#_Toc152228153)

[Cahier de charges iii](#_Toc152228154)

[Table des matières v](#_Toc152228155)

[Listes des symboles vii](#_Toc152228156)

[Liste des acronymes viii](#_Toc152228157)

[Liste des figures ix](#_Toc152228158)

[Liste des tableaux x](#_Toc152228159)

[Introduction générale 1](#_Toc152228160)

[Chapitre 1. Présentation de l’entreprise 2](#_Toc152228161)

[1.1. Historiques 2](#_Toc152228162)

[1.2. Organisation et structure 2](#_Toc152228163)

[Chapitre 2. Méthodologie 3](#_Toc152228164)

[2.1. Planification 3](#_Toc152228165)

[2.2. Etudes de faisabilité 4](#_Toc152228166)

[2.2.1. Etudes de l’existant 4](#_Toc152228167)

[2.2.1.1. Réseau et services existants : 4](#_Toc152228168)

[2.2.1.2. Plateforme DHIS2 4](#_Toc152228169)

[a) Fonctionnement de DHIS2 5](#_Toc152228170)

[b) Points forts 6](#_Toc152228171)

[c) Points faibles 7](#_Toc152228172)

[2.2.1.3. WEB API de DHIS2 7](#_Toc152228173)

[a) les APIs de DHIS2 7](#_Toc152228174)

[b) Les types d’objet du tableau de bord 8](#_Toc152228175)

[2.2.2. Etudes de besoins 9](#_Toc152228176)

[2.2.2.1. Application de modélisation 9](#_Toc152228177)

[2.2.2.2. Editeur de texte et langages 10](#_Toc152228178)

[a) Editeur 10](#_Toc152228179)

[b) Langages 10](#_Toc152228180)

[2.2.2.3. Modules externes (Frameworks) 11](#_Toc152228181)

[2.2.2.4. Hebergement 11](#_Toc152228182)

[a) Hébergement sur un serveur en ligne 11](#_Toc152228183)

[b) Hébergement sur un serveur local 12](#_Toc152228184)

[2.3. Modélisation 13](#_Toc152228185)

[2.3. 1. Diagramme UML : cas d’utilisation 13](#_Toc152228186)

[2.3. 2. Modélisation de la base de donnée (MERISE) 14](#_Toc152228187)

[2.3.2. 1. Modèle Conceptuel des Données 15](#_Toc152228188)

[2.3.2. 2. Modèle Logique des Données 15](#_Toc152228189)

[2.3.2. 3. Modèle Physique de Données 16](#_Toc152228190)

[Chapitre 3. Réalisation 17](#_Toc152228191)

[3.1. Développement de l’application 17](#_Toc152228192)

[3.1. 1. Partie Django : 17](#_Toc152228193)

[3.1.1. 1. Création de la base de données : 17](#_Toc152228194)

[3.1.1. 2. Communication entre Javascript et Django : 18](#_Toc152228195)

[3.1.1. 3. Formatage de données pour l’affichage de Graphique : 20](#_Toc152228196)

[3.1.1. 4. Formatage de données pour l’affichage de Carte Géographique : 21](#_Toc152228197)

[3.1.1. 5. Formatage des données de chaque élément d’un tableau de bord : 23](#_Toc152228198)

[3.1.1. 6. Récupération des données de chaque élément du tableau de bord 24](#_Toc152228199)

[3.1.1. 7. Récupération des données analytiques d’un élément du tableau de bord: 25](#_Toc152228200)

[3.1.1. 8. Formatage des données pour l’affichage d’un tableau : 25](#_Toc152228201)

[3.1.1. 9. Envoi de réquêtes et récupération des données via l’API : 29](#_Toc152228202)

[3.1. 2. Partie HTML et Javascript : 30](#_Toc152228203)

[3.1.2.1. Validation de données de l’utilisateur vers la base de données : 30](#_Toc152228204)

[3.1.2.2. Envoi et récupération de données avec le Serveur (Django) : 32](#_Toc152228205)

[3.1.2.3. Affichage de la graphique : 34](#_Toc152228206)

[3.1.2.4. Affichage du tableau 36](#_Toc152228207)

[3.1.2.5. Récupération des informations sur les éléments du tableau de bord : 39](#_Toc152228208)

[3.1.2.6. Affichage de la Carte Géographique : 42](#_Toc152228209)

[3.1.2.7. Création la barre de navigation: 45](#_Toc152228210)

[3.1.2.8. Mise en place des contenus de la page : 46](#_Toc152228211)

[3.2. Hébergement 48](#_Toc152228212)

[Conclusion générale 49](#_Toc152228213)

[Références 50](#_Toc152228214)

[Annexes 51](#_Toc152228215)

[Résumée 52](#_Toc152228216)

# Listes des symboles

# Liste des acronymes

CSRF : Cross-Site Request Forgery

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

ORM : Object-Relational Mapping

UML : Unified Modeling Language

# Liste des figures

[**Figure 1.1 :** Organigramme du Ministère de la Santé Publique 2](file:///D:\Kenn\AdR\Projet%20de%20mémoire\Documentation\rapport%20de%20mémoire.docx#_Toc152180431)

[**Figure 1.2 :** Diagramme de Gantt de la planification 3](file:///D:\Kenn\AdR\Projet%20de%20mémoire\Documentation\rapport%20de%20mémoire.docx#_Toc152180432)

[**Figure 2.1 :** Tableau de bord de la plateforme DHIS2 4](file:///D:\Kenn\AdR\Projet%20de%20mémoire\Documentation\rapport%20de%20mémoire.docx#_Toc152180437)

[**Figure 2.2 :** Graphique de l'application DHIS2 8](#_Toc152180438)

[**Figure 2.3 :** Carte géographique de l'application DHIS2 8](#_Toc152180439)

[**Figure 2.4 :** Tableau croisé dynamique de DHIS2 9](#_Toc152180440)

[**Figure 2.5 :** Objet de type Texte de DHIS2 9](#_Toc152180441)

[**Figure 2.6 :** Diagramme de cas d'utilisations 14](#_Toc152180442)

[**Figure 2.7 :** Modèle conceptuel de données 15](#_Toc152180443)

[**Figure 2.8 :** Modèle logique de données 15](#_Toc152180444)

[**Figure 2.9 :** Modèle physique de données 16](#_Toc152180445)

# Liste des tableaux

[**Tableau 1.1:** Planification des tâches 3](#_Toc152180408)

# Introduction générale

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.

1. Présentation de l’entreprise

## Historiques

Ministère de la Santé Publique Madagascar

Direction des Etudes, de la Planification et du Système d’Information ou DEPSI

Service de l’Exploitation, de la Maintenance Informatique et du Développement du Système Informatique ou SEMIDSI

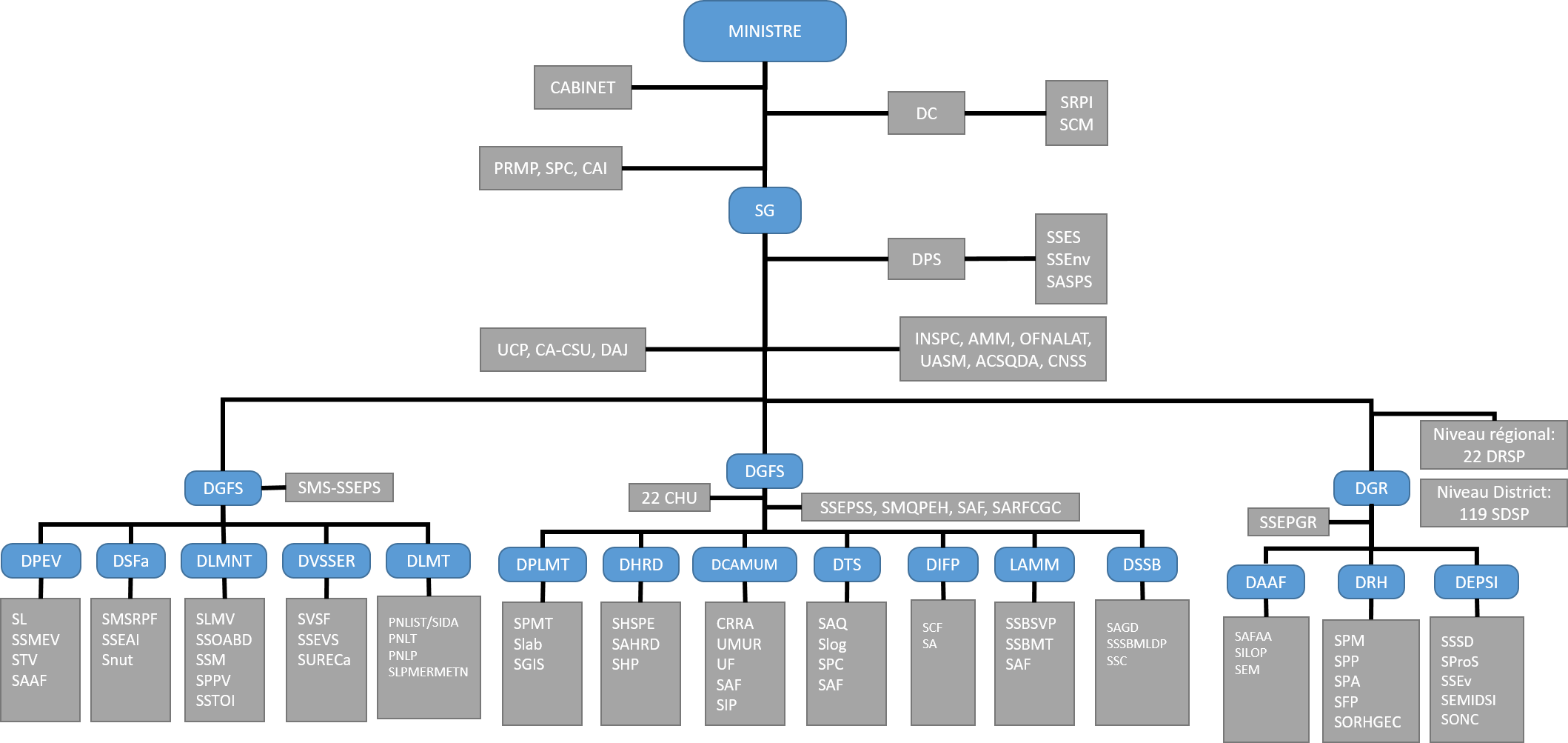
Le Ministère de la Santé Publique est un établissement

SEMIDSI, Ce service participe au bon développement des activités de la Santé Publique de Madagascar.

Organisation et structure

Organisation du Ministère de la Santé Publique

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

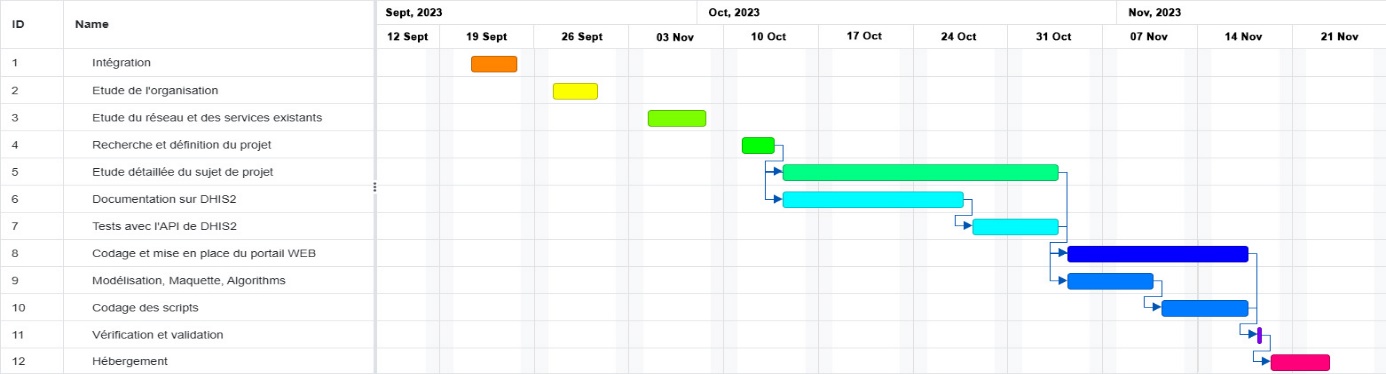
1. Méthodologie
2. Planification

**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 |

Online GANTT est un outil de conception d’un diagramme de Gantt en ligne et gratuit.





**Figure 1.2 :** Diagramme de Gantt de la planification

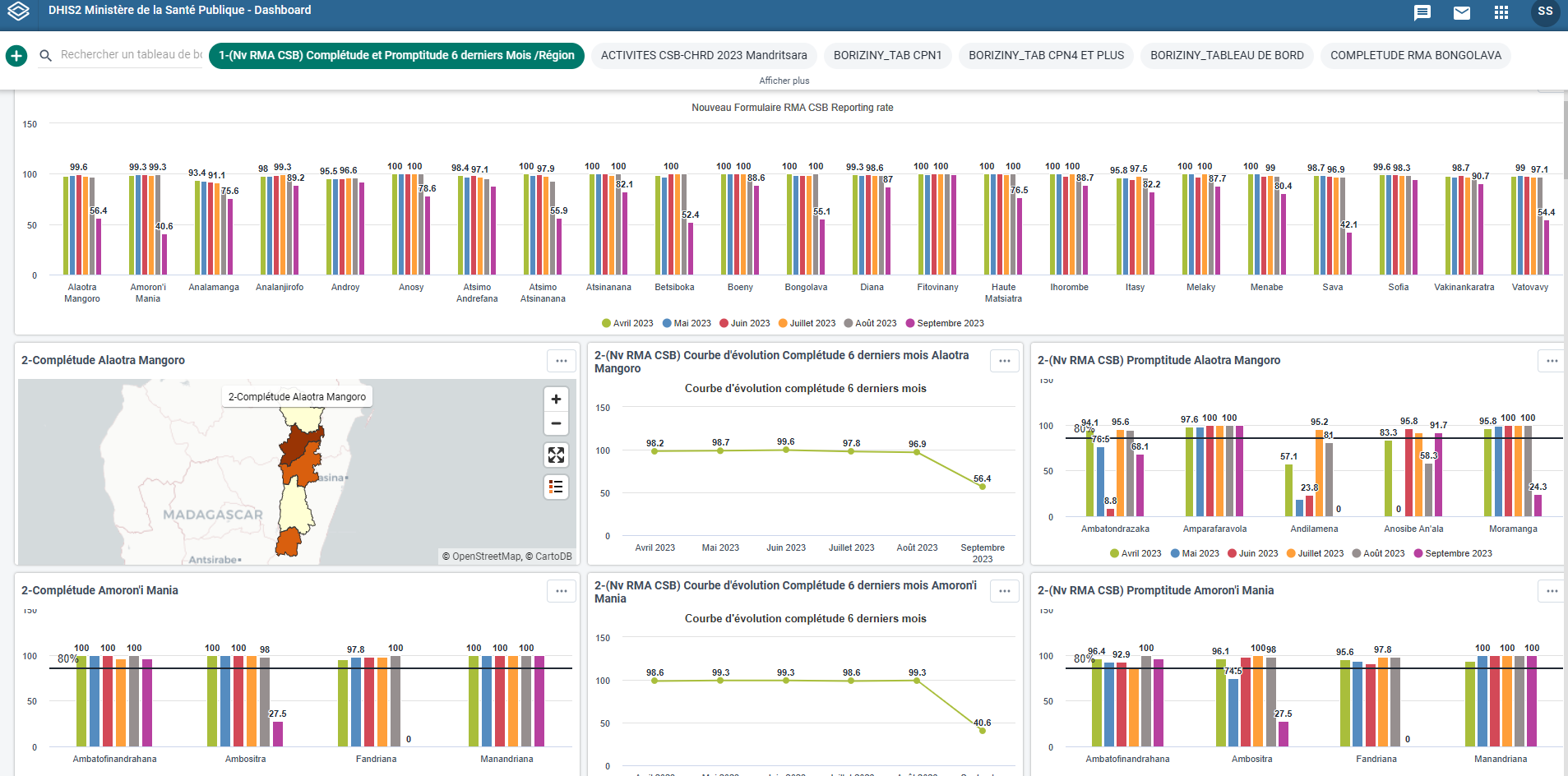
1. Etudes de faisabilité
2. Etudes de l’existant

Réseau et services existants :

Plateforme DHIS2



DHIS2 est une plateforme logicielle open source 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 2.1 :** Tableau de bord de la plateforme DHIS2

C’est un logiciel libre développé par un réseau de recherche – action 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 un entrepôt de données conçu pour stocker les données de santé essentielles et ainsi favoriser l’information pour l’action.

Le DHIS2 est développé par le Health Information System Programme (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 avec un appui du NORAD.

Le DHIS2 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 (ONG), 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.

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.

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.

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.

WEB API de DHIS2

1. les APIs de DHIS2

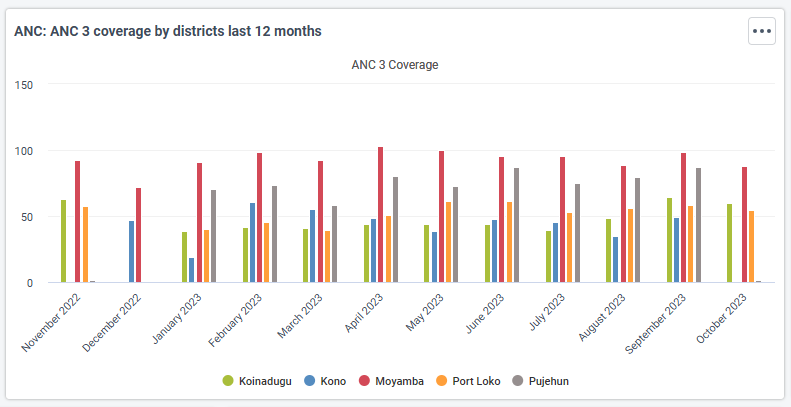
Pour lister les différents APIs de DHIS2, nous avons la requête GET api/resources avec le lien url http://<domaine\_serveur>/api/resources.

Le WEB API de DHIS2 peuvent renvoyer des données en format JSON, XML, PNG, SVG, CSV, MP4, et autres dépendant des requêtes envoyées. Et pour reproduire le tableau de bord de DHIS2 dans l’application WEB, une étude sur chaque objet qu’il contient.

Après avoir fait quelques recherches dans les documentations de DHIS2 [guide d’utilisation et documentation pour développeur] et quelques analyses sur l’application WEB DHIS2, j’ai compté 4 types d’objet dans le tableau de bord.

1. Les types d’objet du tableau de bord

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



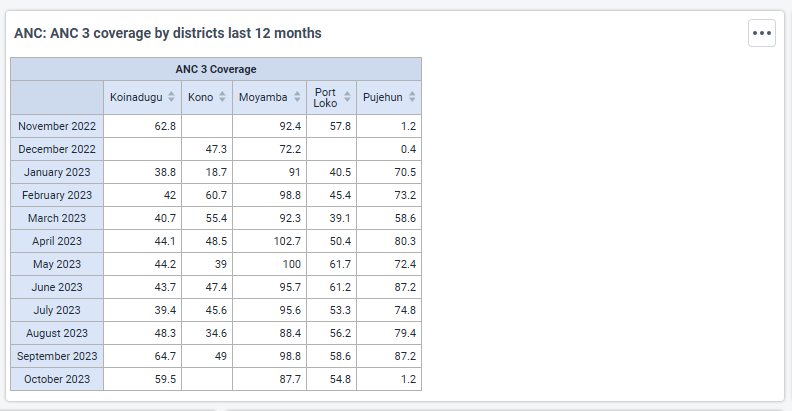
**Figure 2.2 :** Graphique de l'application DHIS2

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



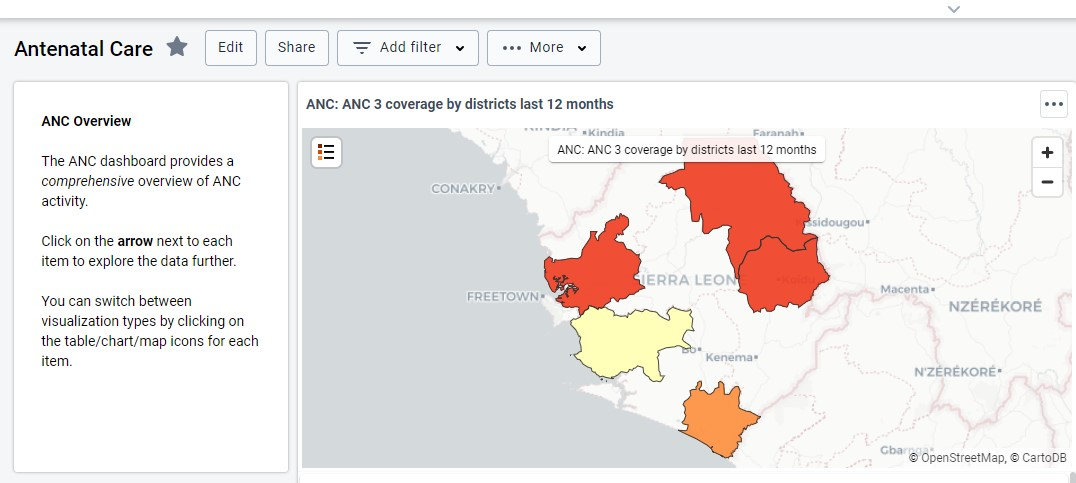
**Figure 2.3 :** Carte géographique de l'application DHIS2

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



**Figure 2.4 :** Tableau croisé dynamique de DHIS2

**Texte :** affiche des informations comme des annonces.



**Figure 2.5 :** Objet de type Texte de DHIS2

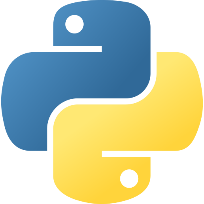
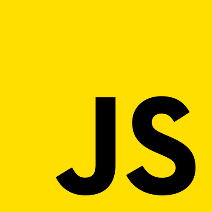
1. Etudes de besoins
2. Application de modélisation

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.

1. Editeur de texte et langages
2. 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).

1. 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.

1. 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 responsifs. 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.

1. Hebergement
2. 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.

1. 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 :

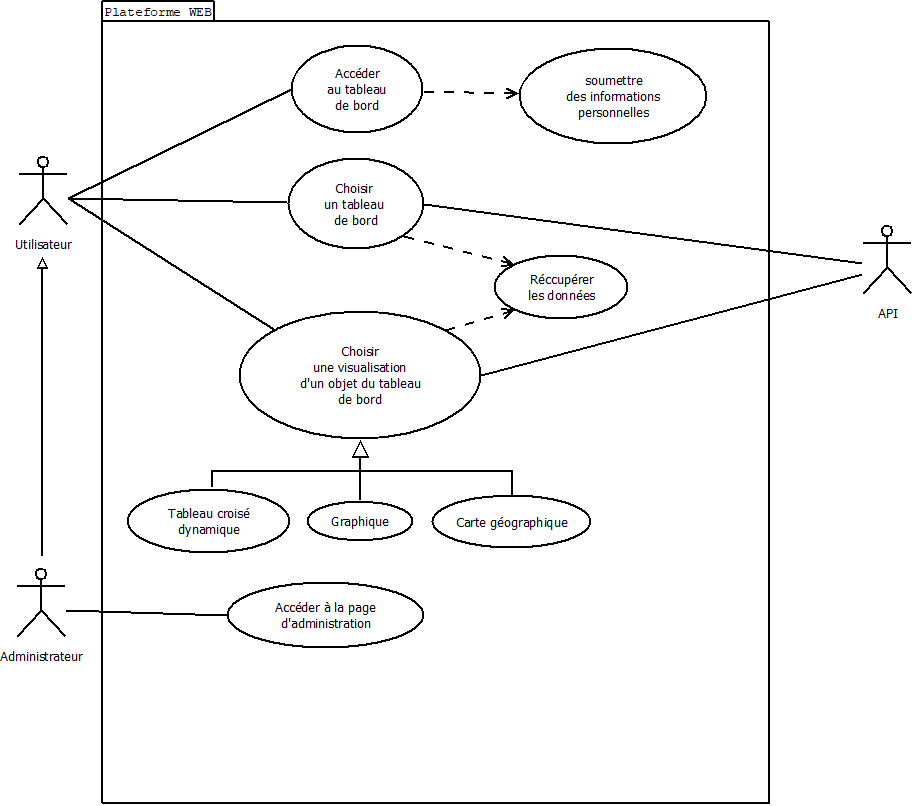
Ouvrez le pare-feu : ajouter une règle dans le pare-feu pour permettre le trafic sur le port 80 (ou le port défini dans le fichier de configuration)..

1. Modélisation
2. 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 de l’application.

Après, l’utilisateur peut explorer et profiter des fonctionnalités et des informations fournies par l’application WEB via l’API de DHIS2.

La page du tableau de bord affiche les informations et les objets du tableau de bord depuis le DHIS2. On peut voir consulter les informations partagées depuis la plateforme DHIS2 directement et publiquement sur l’application WEB.

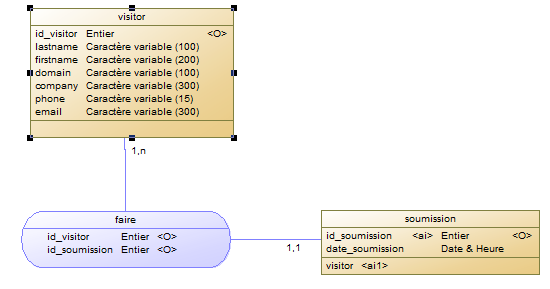


**Figure 2.6 :** Diagramme de cas d'utilisations

1. Modélisation de la base de donnée (MERISE)

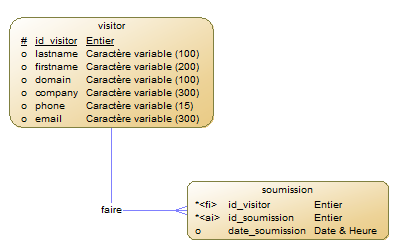
Ces données sont les informations sur les utilisateurs et le nombre de vues de l’application web.

1. Modèle Conceptuel des Données



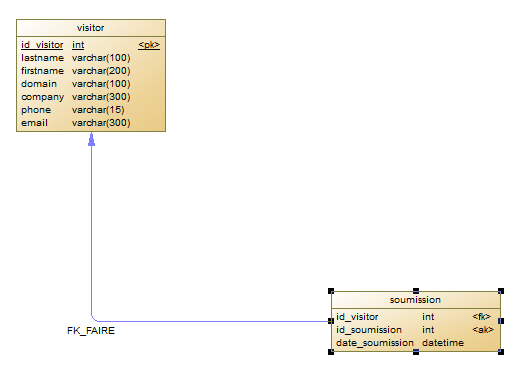
**Figure 2.7 :** Modèle conceptuel de données

1. Modèle Logique des Données



**Figure 2.8 :** Modèle logique de données

1. Modèle Physique de Données



**Figure 2.9 :** Modèle physique de données

1. Réalisation

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. Elle contribuera à renforcer les connaissances en santé et à améliorer la prise de décision en matière de santé publique.

1. Développement de l’application

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

Langages : HTML, CSS, Javascript, Python

Modules (Frameworks) : Django, Leaflet, Chart, DataTables, Bootstrap, Jquery

1. Partie Serveur (Django) :
2. 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

1. 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.

1. 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**.

1. 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 Pulique.

**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.

1. 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.

1. 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.

1. 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.

1. 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}**.

Algorigramme :

Début

|

|--> [Définir la fonction reporttable\_datas avec le paramètre response]

|

|--> [Initialiser les listes row\_titles, columns et data]

|

|--> [Si la longueur de response['headers'] est supérieure à 2]

| |

| |--> [Ajouter un dictionnaire à columns]

| |

| |--> [Pour chaque élément dans response['metaData']['dimensions'][response['headers'][1]['name']]]

| | |

| | |--> [Si l'élément contient '.', le diviser en utilisant '.' et prendre le premier élément]

| | |

| | |--> [Ajouter un dictionnaire à row\_titles]

| |

| |--> [Si response['headers'][1]['name'] est égal à 'ou', trier response['metaData']['dimensions'][response['headers'][1]['name']]]

| |

| |--> [Pour chaque info dans row\_titles]

| | |

| | |--> [Initialiser lieu\_resultat avec un dictionnaire]

| | |

| | |--> [Pour chaque ligne dans response['rows']]

| | | |

| | | |--> [Si la deuxième valeur de la ligne est égale à la première clé de info]

| | | | |

| | | | |--> [Si la première valeur de la ligne contient '.', la diviser en utilisant '.' et prendre le premier élément]

| | | | |

| | | | |--> [Ajouter une nouvelle entrée à lieu\_resultat avec la première valeur de la ligne comme clé et la dernière valeur de la ligne comme valeur]

| | |

| | |--> [Ajouter lieu\_resultat à data]

|

|--> [Sinon]

| |

| |--> [Pour chaque ligne dans response['rows']]

| | |

| | |--> [Si la première valeur de la ligne contient '.', la diviser en utilisant '.' et prendre le premier élément]

| | |

| | |--> [Ajouter un dictionnaire à data avec la première valeur de la ligne comme clé et la dernière valeur de la ligne comme valeur]

|

|--> [Pour chaque colonne dans response['metaData']['dimensions'][response['headers'][0]['name']]]

| |

| |--> [Si la colonne contient '.', la diviser en utilisant '.' et prendre le premier élément]

| |

| |--> [Ajouter un dictionnaire à columns avec 'data' comme clé et le nom de la colonne comme valeur]

|

|--> [Afficher columns]

|

|--> [Afficher data]

|

|--> [Retourner un dictionnaire avec 'columns' et 'data' comme clés et columns et data comme valeurs]

|

Fin

1. Envoi de réquêtes 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.

Algorigramme :

Début

|

|--> [Définir la fonction api\_get\_data avec le paramètre path]

|

|--> [Essayer]

| |

| |--> [Construire l'URL en ajoutant le chemin à l'URL de l'API]

| |

| |--> [Si le chemin ne contient pas 'analytics?', ajouter '?paging=false' à l'URL]

| |

| |--> [Afficher l'URL]

| |

| |--> [Envoyer une requête GET à l'URL]

| |

| |--> [Si le statut de la réponse n'est pas réussi, lever une exception]

| |

| |--> [Extraire les données JSON de la réponse]

|

|--> [Si une exception est levée]

| |

| |--> [Afficher un message d'erreur]

| |

| |--> [Définir datas comme un dictionnaire contenant un message d'erreur]

|

|--> [Retourner datas]

|

Fin

1. Partie HTML et Javascript :
2. Validation de données 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.

1. 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.

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.

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

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.

1. 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.

1. 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.

1. 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; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'

});

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.

1. 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 (<ul>) qui contient les éléments de la barre de navigation.

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

**<a class="nav-link active text-white" href="#">:** 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).

1. 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 element 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.

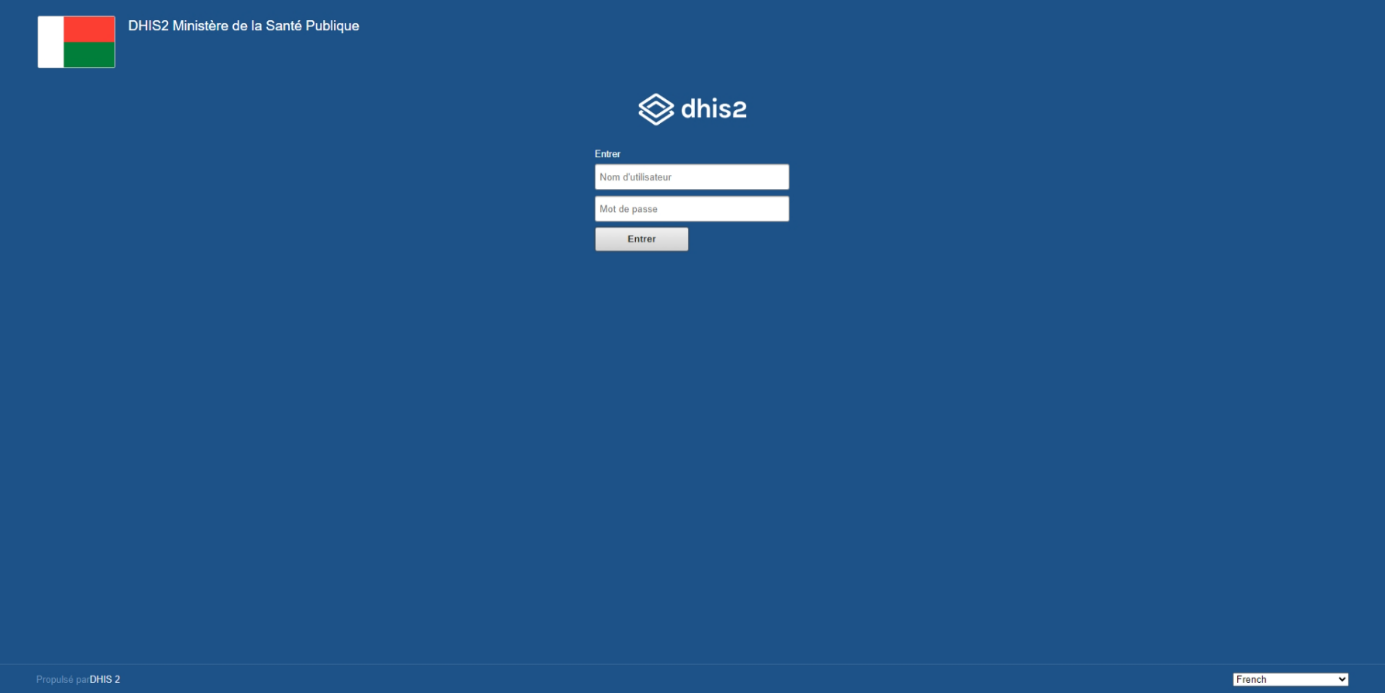
**{% block section %} <!-- Contenus --> {% endblock %}:** C'est un bloc template Django. Cela permet d'étendre la section et d'insérer du contenu spécifique à cet endroit dans la page. Dans votre exemple, le contenu spécifique serait défini dans un autre fichier ou une autre partie du code.

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;).

1. Tests

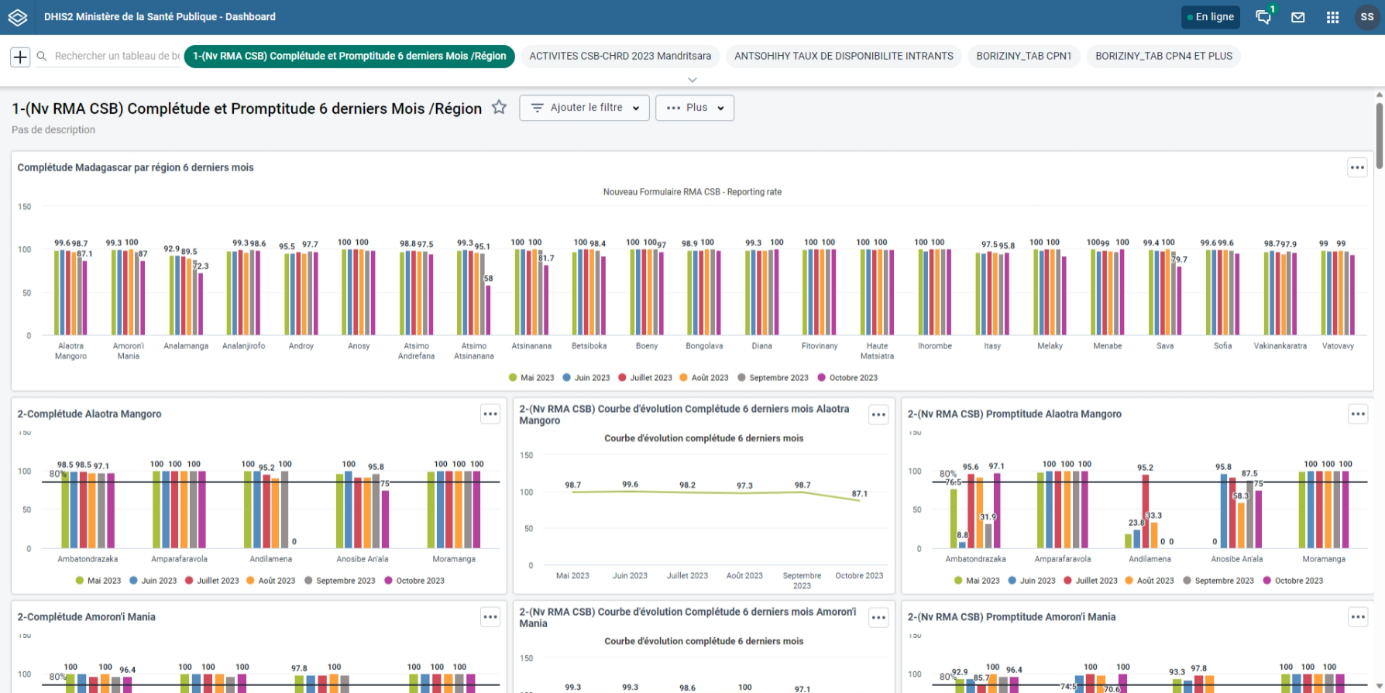
Sur DHIS2 du Ministère de la Santé Publique Madagascar :

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



**Figure 3.1:** Page d'authentication de DHIS2 du Ministère de la Santé Publique

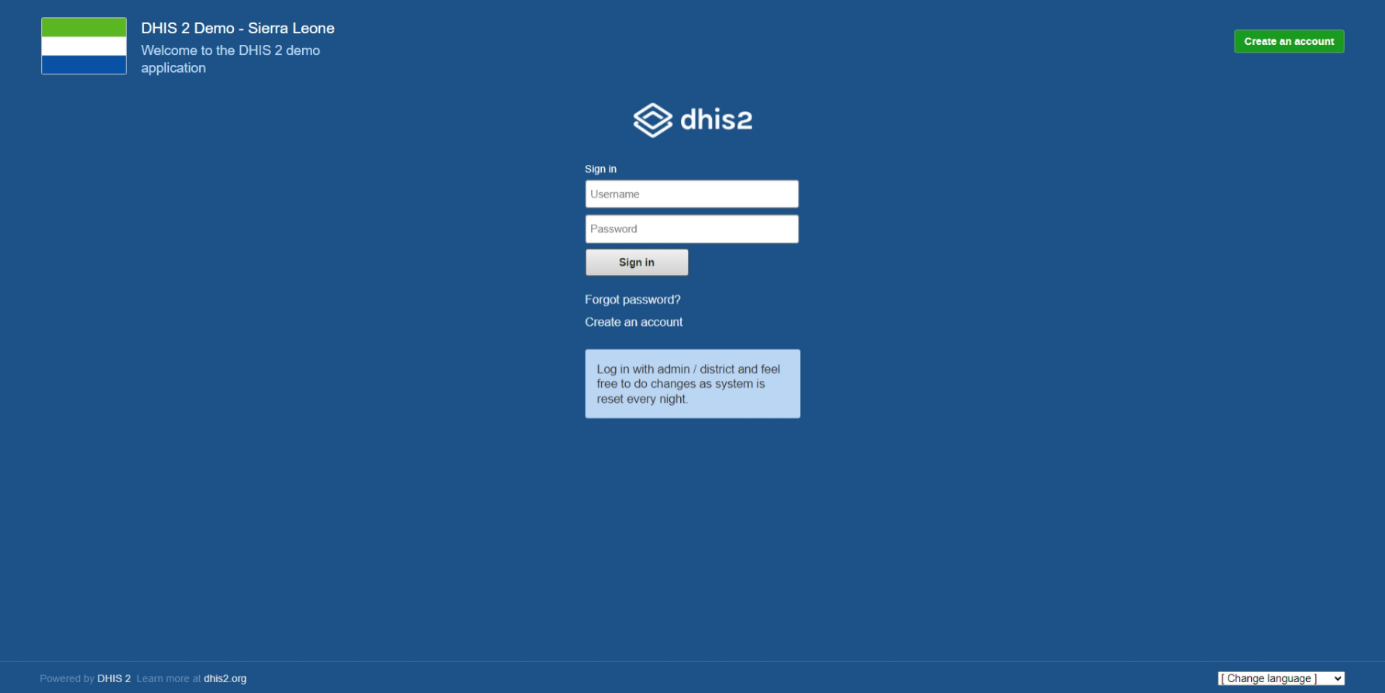
Le tableau de bord



**Figure 3.2:** Application Tableau de bord de DHIS2 du Ministère de la Santé Publique

Sur le DHIS2 Demo :

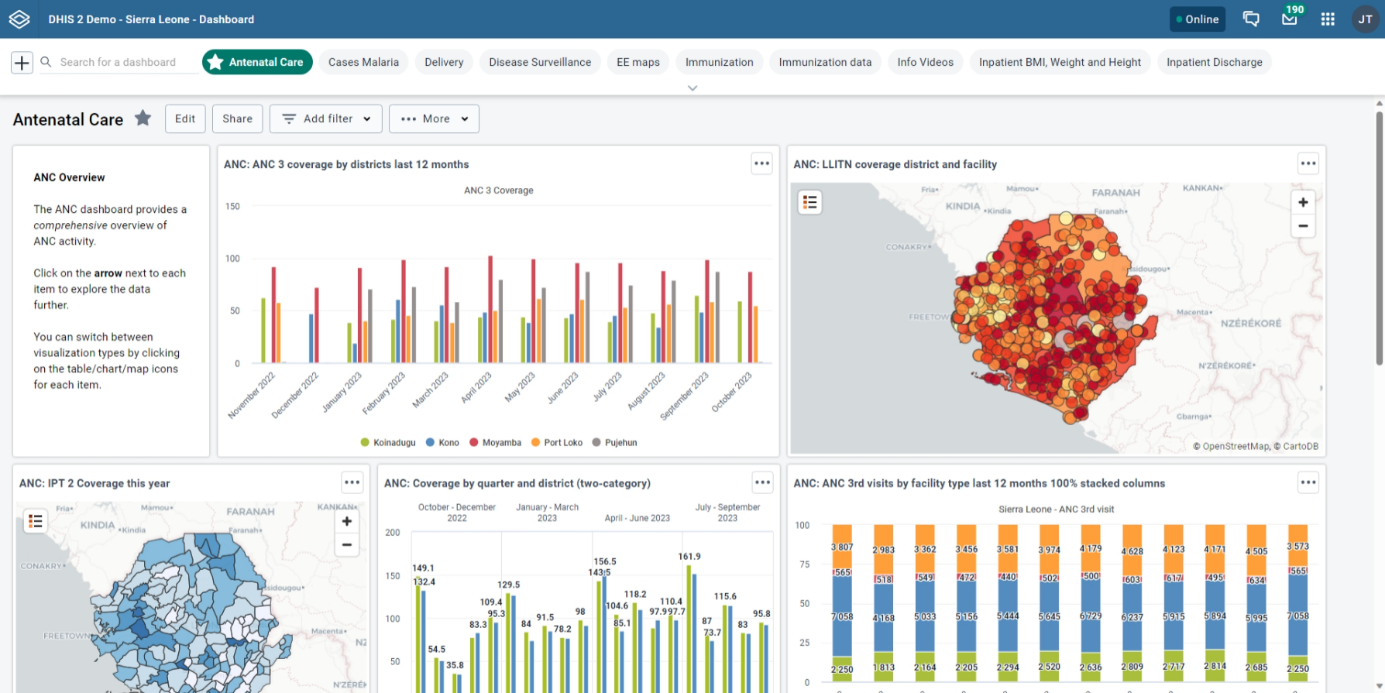
Pendant la maintenance ou manifestation d’erreurs sur la plateforme DHIS2 du Ministère de la Santé Publique, je fais mes tests sur DHIS2 Demo. Ce dernier partage des identifiants publiques pour les utilisateurs qui veulent tester DHIS2 avant de le télécharger et l’installer, voici le lien « https://play.dhis2.org/40.2.0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



**Figure 3.3:** Page d'authentication 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 3.5:** Application Tableau de bord de DHIS2 Demo

1. Hébergement

L’hébergement se fera dans le serveur local du Ministère de la Santé Publique.

# Conclusion générale

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.

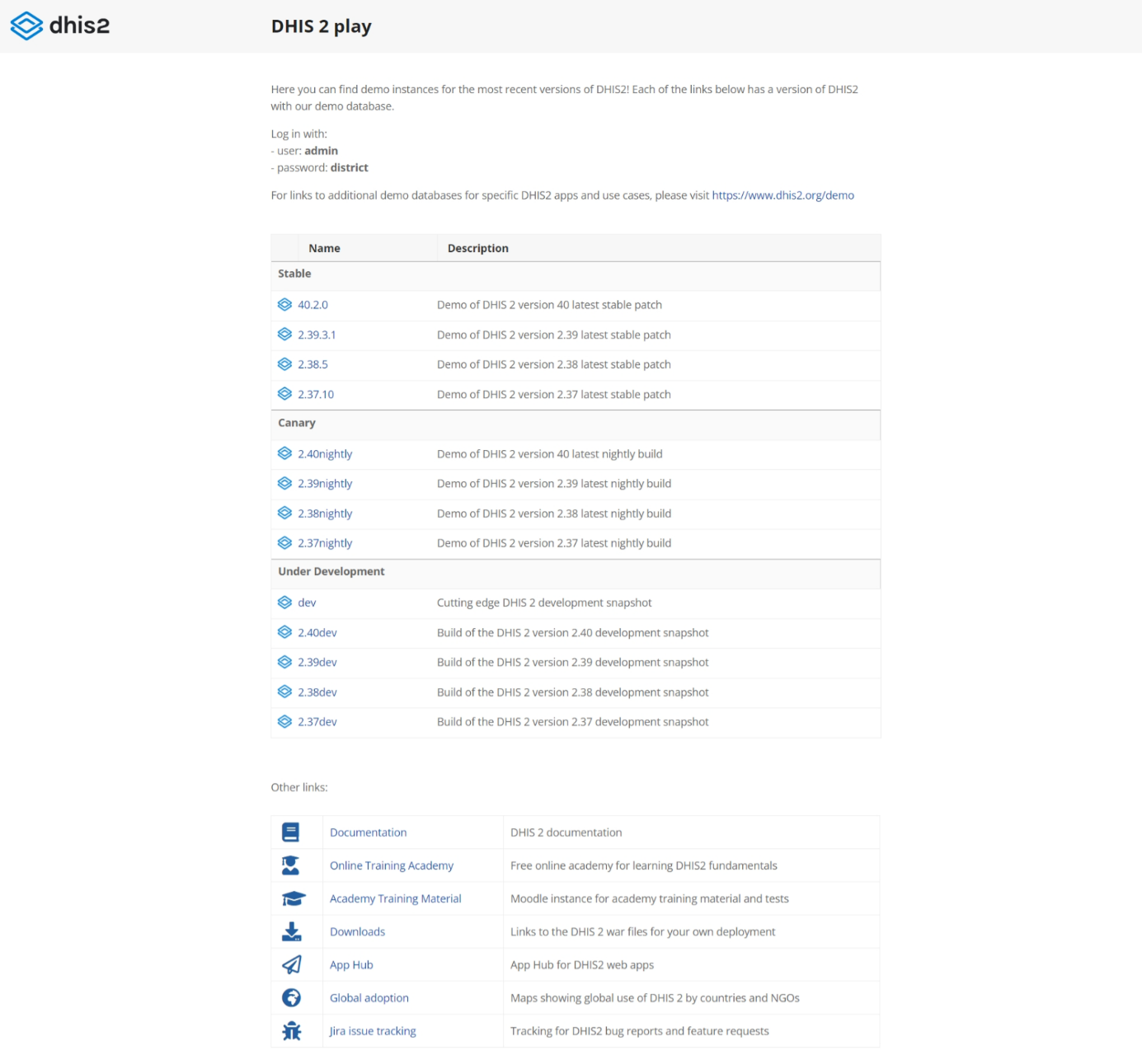
# Références

Bibliographiques

[1] LOGICIEL DHIS2 Manuel d’utilisateur final de DHIS2, édition Octobre 2020

webographiques

# Annexes



**Figure 3.4:** La première page de DHIS2 Demo

# Résumée