Documentation GELEPI

Annexe

Contents

General

Contexte

La mission locale a besoin d'un système capable de répertorier le matériel de manière très précise. Il est essentiel de connaître l'état de chaque équipement (bon, neuf, ou non déballé), de savoir s'il est encore sous garantie et pour quelle durée. Cette fonctionnalité améliore l'organisation globale et facilite la disponibilité du matériel, permettant une gestion plus efficace et une utilisation optimisée des ressources.

Pour répondre à ces besoins, j'ai proposé la création de notre propre système de gestion libre de parc informatique. C'est le terme exact que l'on utilise pour désigner un logiciel conçu pour répondre précisément à ce type de demande. Bien qu'il existe divers GLPI (Gestion Libre de Parc Informatique), j'ai décidé de créer le mien afin que le logiciel soit parfaitement adapté aux besoins spécifiques de l'entreprise. Cela me permet d'offrir une solution sur mesure, optimisée pour les exigences de gestion du matériel et de l'infrastructure.

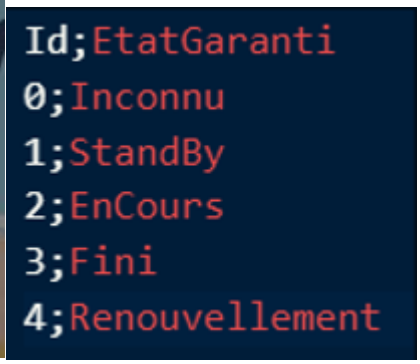
Cette documentation détaille le fonctionnement de l'application Symfony que j'ai développé. Elle couvre les différentes fonctionnalités, les concepts techniques, ainsi que les modules et composants intégrés dans le système. Vous y trouverez des informations sur la structure du projet, l'architecture utilisée, et les étapes de mise en œuvre pour adapter le logiciel aux besoins de gestion du parc informatique.

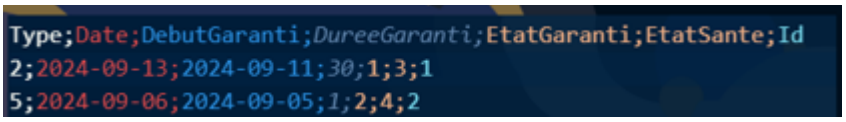
Data

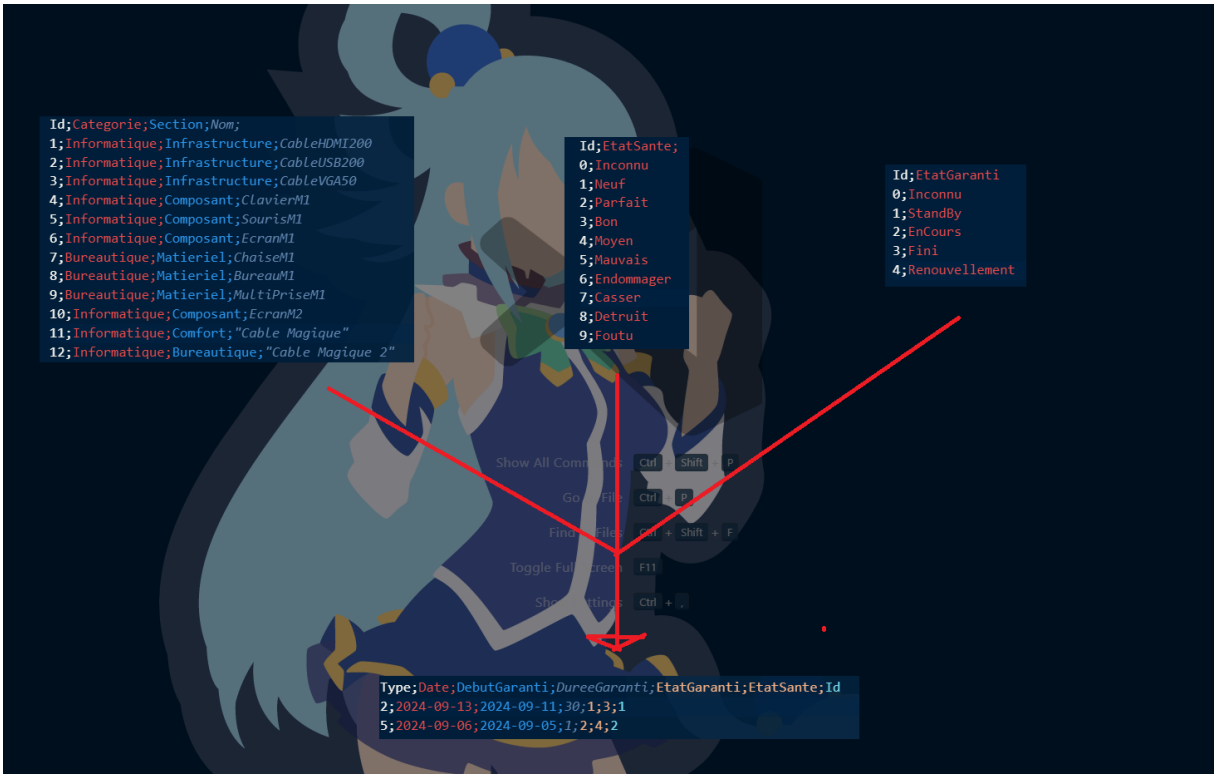
Stockage

Le stockage des données se repose sur un système de gestion de données basé sur des fichiers que j'ai développés. Il utilise plusieurs fichiers CSV comme des tables, similaires à ce que l'on trouve dans une base de données classique. Pour manipuler ces données de manière flexible et rapide, des classes PHP dédiées ont été mises en place. L'une des principales classes est CSVManager.php, située dans le dossier Service, qui permet la gestion efficace des fichiers CSV en centralisant les opérations de lecture, écriture et traitement des données. Ce système offre une approche légère et adaptée aux besoins spécifiques du projet, tout en évitant les complexités d'une base de données traditionnelle. (Même si Symfony aide beaucoup)

On retrouve des tables avec un minimum de donnée qui font le lien avec d'autre table de cette manière







L'intérêt est de pouvoir réduire les répétitions quand c'est possible et de stocker seulement le nécessaire.

Ses 4 tables ont chacun leurs propre fichier csv avec un nom pour les reconnaitre :

- MaterielListe
- MaterielType
- MetaDataGaranti
- MetaDataSante

Il existe aussi un dossier archive qui permettra de déplacer de copier les plus anciennes informations comme les appareils qui ne sont plus utiliser ou simplement jeter, se dossier n'est pas utiliser dans le projet.

CSVManager et Filter

Ses deux Services sont emmenés à être appeler dans les contrôleurs pour manipuler les données.

Manager possèdent des fonctions LoadNomFichier pour la lecture et des fonction getNomFichier séparément, l'intérêt est de pouvoir récupérer sois toutes les données d'une table, sois l'ID et le nom uniquement, ça réduit la charge et le code à écrire au moment de créer des formulaires à partir de ses données par exemple. Au niveau du nommage, aucune restriction, l'encodage est en UTF-8.

La classe Filter viens en complément de Manager, il possède une collection de fonction qui permettent de créer des conditions et donc de lancer du code celons l'information des fichiers, ce qui ajoute une certaine souplesse dans le code, ont peu par exemple créer une fonction Lowercase qui met en minuscule l'information au moment même de la recevoir. Le but de la séparation entre Manager est Filter est simplement de rendre le code plus lisible.

Le Filter peut aussi décider d'une langue par défaut pour pouvoir traduire l'affichage

Symfony

Trois commande Symfony on suffit à créer le projet

- symfony new gelepi –webapp
- php bin/console make:controller GelepiBase
    - Route:
    - Racine. (/)
- php bin/console make:controller GelepiListing
    - Route:
    - Affichage des élément (/lister_element)
    - Ajout de type d'éléments (/ajouter_variable) (Aucun rapport avec les variables)
    - Ajout des éléments (/ajouter_element)

Aucune base de données n'a été implémenter et le projet a été créer en Symfony 7.1.3.

Il existe deux formulaires dans le dossier Form qui permet d'ajouter des types d'élément dans les fichiers méta et d'ajouter des nouveaux éléments dans le parc informatique ses formulaire vont être appeler dans le contrôleur et afficher via les Twigs.

Plus sur les Routes

Lister_element (app_gelepi_listing)

Des conditions ont été mises en place sur la route d'affichage afin de garantir le bon fonctionnement du système de fichiers. En cas de dysfonctionnement, un message d'erreur suivant le format [GLPI : Erreur] s'affichera pour aider au diagnostic et à la maintenance. Cela permet d'identifier rapidement les problèmes et de les résoudre efficacement.

Ajouter_element (app_gelepi_element)

Après avoir récupéré les éléments dans des tableaux (arrays) manipulables, différents traitements peuvent être appliqués aux informations. Cependant, l'ordre d'apparition des données est fixe : d'abord l'ID de l'élément, suivi de la date d'arrivée de l'élément dans l'entreprise, la date de début de la garantie, le temps restant en années, l'état de la garantie, et enfin l'état de santé de l'élément. Cette structure garantit une organisation cohérente des données, facilitant ainsi leur traitement et leur affichage.

Ajouter_variable (app_gelepi_variable)

Avant d'ajouter un élément, on commence par ajouter un type d'élément qui permet de fournir des informations générales sur les futurs éléments. Ces informations, que j'appelle ici des "variables", ne sont en réalité que des données complémentaires qui viennent enrichir les éléments finaux. Elles servent à structurer et organiser les détails, afin de faciliter l'ajout et la gestion des éléments tout en maintenant une cohérence dans le système.

Twig

À la racine du dossier Template, il existe un fichier Twig principal qui permet d'afficher jusqu'à trois éléments de Template par défaut. J'ai choisi d'y intégrer directement les informations CSS pour simplifier la gestion du style. En général, le premier Template inclus est la barre de navigation, suivie du corps de la page. Cette structure permet de maintenir une organisation claire et cohérente, tout en facilitant l'intégration des différentes sections de la page à travers des inclusions Twig, rendant ainsi la gestion des composants visuels plus flexible.