Application web de formulaire **GLPI**

1. Contexte :
Pour faire de l'assistance avec GLPI, on utilise des formulaires qui guident pas à pas l'agent dans sa démarche pour ainsi créer un ticket sur GLPI où il pourra être traiter. Cependant certains agents peuvent créer directement un ticket, et n'expliquent pas vraiment leur problème comme il le faudrait.

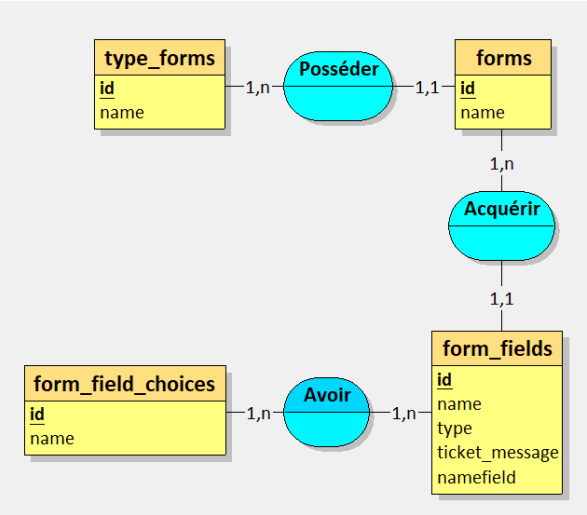
2. Nous avons utilisé le framework **Laravel** pour :

- Connexion LDAP avec le package « directorytree/ldaprecord » qui permet une connexion avec un serveur LDAP assez facilement (Overview – LdapRecord). Toute la configuration est dans le fichier .env à la racine.

- Gérer dynamiquement un formulaire avec **livewire** 3, ce package nous permet de manière dynamique de gérer étape par étape la suite de la demande, avec par exemple l'étape 1 → demander si c'est un incident ou demande, étape 2 → demander quelle formulaire voulu (en fonction de si c'est un incident ou demande), étape 3 → afficher les champs du formulaire sélectionné.

- Gérer la création de ticket, en créant un « model » Ticket qui a pour table « glpi_tickets » il permet la manipulation des tickets **GLPI** (création, supression, modification) ce qui est utile dans notre cas car dès qu'une requête est faite, alors un ticket doit être créé avec la bonne personne à l'origine de la demande.

- Pour gérer tous nos cas de figure, nous avons démarrer par un modèle conceptuel de données (MCD) pour pouvoir être clair et cohérent avec les demandes.



3. Comment fonctionne notre formulaire ? :

3.1 Chercher dans l'Active Directory un utilisateur en fonction de son nom et prénom

Pour ce faire, nous avons utilisé livewire avec « wire:model.lazy='lastName', wire:model.lazy='firstName' », livewire va nous permettre dès que le champ est modifié de déclencher automatiquement la méthode adéquate « updatedLastName » pour le nom de famille ainsi que « updatedFirstName », ces 2 méthodes vont appeler l'autre méthode « searchLdap » qui va lancer une recherche Ldap.

```php
    /**
     * Si le nom de famille a été modifié, alors on lance une recherche Ldap
     */
    public function updatedLastName()
    {
        $this->searchLdap();
    }

    /**
     * Si le prénom a été modifié, alors on lance une recherche Ldap
     */
    public function updatedFirstName()
    {
        $this->searchLdap();
    }
```

Une fois le prénom et nom de famille récupérés, alors dans la méthode « searchLdap » on débute la recherche seulement si le nom et prénom n'est pas vide à l'aide du modèle « LdapUser » qui se trouve dans le namespace « LdapRecord\Models\ActiveDirectory\User ». Avec ce modèle on fait simplement 2 conditions where avec givenname = $this→firstName et sn = $this→lastName. Le résultat de la requête est stocké dans la variable $user qui renvois :

- null si aucun résultat (utilisateur non trouvé), on affecte $this→ldapUserFound = false;
- si pas null (utilisateur trouvé), on affecte $this→ldapUserFound = true ;

Et donc si l'attribut $ldapUserFound = true à chaque mise à jour, alors dans notre vue avec @if($ldapUserFound), on affiche la suite c'est-à-dire on lui demande le type, le formulaire etc.

```php
    /**
     * Méthode permettant la rercherche Ldap en fonction du nom et prénom,
     * la recherche est faite avec "givenname" et "sn", puis on récupère
     * l'id GLPI de l'utilisateur dans l'active directory
     */
    public function searchLdap()
    {
        if (!empty($this->lastName) && !empty($this->firstName)) {
            $user = LdapUser::where('givenname', $this->firstName)
                            ->where('sn', $this->lastName)
                            ->first();

            if ($user) {
                $this->ldapUserFound = true;
                $this->glpiUserId = !is_null($user->glpi()) ? $user->glpi()->id : 0;
            } else {
                $this->ldapUserFound = false;
            }
        }
    }
```

Méthode pour la recherche Ldap

Une fois le nom et prénom fournis et que la recherche Ldap est concluante, alors le champ « type » apparaît parfaitement.

3.2 Afficher le formulaire sélectionné avec la base de données

Pour gérer tout cela, nous avons décidé de stocker le type, les formulaires, et le formulaire sélectionné respectivement dans des tables : « type_forms », « forms », « form_fields ». Les migrations sont donc disponible dans le répertoire : « database/migrations »

Donc, pour la première étape qui est de récupérer le type on va chercher dans la bonne table (type_forms) avec le modèle « TypeForm » qu'on stocke dans la variable $types que l'on injecte dans notre vue pour pouvoir l'utiliser plus tard.

```php
public function render()
{
    $types = TypeForm::all();

    if ($this->selectedType) {
        $this->forms = FormModel::where('type_form_id', $this->selectedType)->get();
    }

    if ($this->selectedForm) {
        $form = FormModel::with('formFields')->find($this->selectedForm);
        $this->formFields = $form->formFields;
    }

    return view('livewire.form', ['types' => $types]);
}
```

Puis, ensuite si un type a été sélectionné, alors on va chercher tous les formulaires du type sélectionné pour ensuite les stockés dans « $this→forms ».

Une fois un formulaire sélectionné alors on va chercher tous ses champs associé à ce formulaire (depuis la table « form_fields ») et les stockés dans « $this→formFields »

3.3 Valider le formulaire pour pouvoir créer un ticket GLPI

Pour pouvoir valider le formulaire, nous avons utilisé « wire:submit.prevent='submitForm' » cela va appeler la méthode « submitForm » dès que le formulaire est soumis pour ensuite faire le traitement nécessaire.

Ensuite, on parcours chaque champ stockés en base de données en fonction du formulaire sélectionné, puis on récupère le champ spécifique via « formFieldData.{$field->id} », on récupère aussi le ticket_message ce qui va nous permettre de le placer dans le contenu du ticket final.

Pour créer un ticket GLPI, nous avons créer un model « Ticket » qui fait référence à la table « glpi_tickets » grâce à un attribut dans le model « protected $table = 'glpi_tickets' ». Une fois notre model prêt, on utilise la méthode create en lui mettant les attributs nécessaires à la création.

```php
public function submitForm()
{
    $selectedFormName = FormModel::find($this->selectedForm)->name;
    $selectedTypeName = TypeForm::find($this->selectedType)->name;

    $rules = [];
    $content = $selectedFormName."\n\n";
    foreach ($this->formFields as $field) {

        $rules["formFieldData.{$field->id}"] = 'required';

        // Si le champ est présent (c'est-à-dire remplis) alors on récupère le messag
        if(isset($this->formFieldData[$field->id])) {
            $ticketMessage = FormField::where('id', $field->id)->value('ticket_messag
            $value = $this->formFieldData[$field->id];
            $content .= $ticketMessage. $value ."\n";
        }
    }

    $this->validate($rules);

    $dateNow = Carbon::now();
    $ticket = Ticket::create([
        'name' => $selectedTypeName . ' ' . $selectedFormName,
        'type' => $this->selectedType,
        'users_id_recipient' => $this->glpiUserId,
        'content' => $content,
        'date' => $dateNow,
        'date_creation' => $dateNow
    ]);

    $this->showSuccessModal = $ticket->id;
}
```