COMPTE-RENDU DE STAGE

Fin de première année de Brevet de Technicien Supérieur

Service Informatique aux Organisation

Option : Solution Logicielles et Applications Métiers

Par

Amine DERBEL

Encadrant professionnel	Monsieur Benoit Bénard
Encadrant académique	Madame Carine Autret

Réalisé au sein de Raccords et Plastiques Nicoll

Sommaire

Introduction

1.2 Introduction

Du 21 mai au 28 juin, j'ai eu l'opportunité d'effectuer un stage au sein du service informatique de l'entreprise Raccords et Plastiques Nicoll. Cette immersion m'a offert un avant-goût du métier de développeur et m'a permis d'enrichir mes compétences. J'ai eu l'occasion d'acquérir de nouvelles connaissances et d'apprendre des techniques novatrices. Ce stage a été une expérience précieuse pour appliquer mes connaissances théoriques dans un cadre professionnel et développer mes compétences dans un environnement stimulant.

1.2 Informations rudimentaires relative au stage

- Nom de l'entreprise :
- Adresse :
- Nom de l'encadrant professionnel :
- Mail :
- Numéro de téléphone :

Raccords et Plastiques Nicoll
37 rue Pierre et Marie Curie – 49300 Cholet
Benoit Bénard
bbenard@aliaxis.com
06 07 17 60 55

1.3 Contexte de l'entreprise

Depuis 2003, Nicoll, GIRPI et AUI font partie du groupe Aliaxis, leader mondial des solutions pour la gestion des fluides (eau et énergie) utilisées dans les bâtiments, les travaux publics et les applications industrielles.

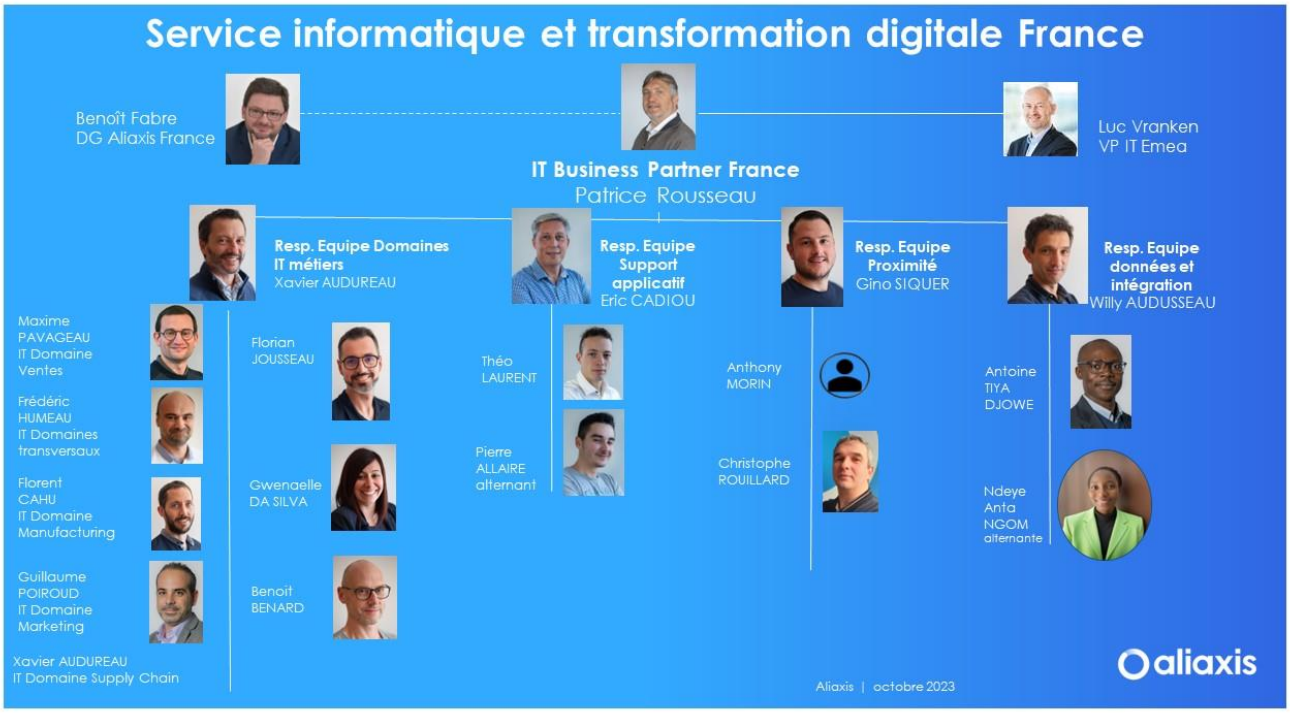
Le groupe basé à Bruxelles, qui emploie 14 000 personnes dans le monde et génère près de 3.7 Milliards d'euros de chiffre d'affaires, s'est développé en s'appuyant sur un réseau de marques fortes et reconnues (plus de 100 entités industrielles et commerciales dans 40 pays), expertes de la fabrication et de la commercialisation de produits en matériaux de synthèse.

Pour proposer des solutions innovantes et durables, le groupe investit fortement en Recherche & Développement et s'appuie sur la complémentarité de ses différentes entités pour développer des projets internationaux.

Cette capacité d'innovation et les opportunités internationales nourrissent les ambitions d'Aliaxis France.

Contractuellement, je suis pris en stage par Nicoll et c'est à Cholet qu'est basée l'antenne la plus importante de la France. C'est donc aussi la raison pour laquelle le département informatique de Cholet gère la majorité de l'informatique du sol français (Pour Nicoll comme pour GIRPI), comme le support applicatif, la gestion de l'ERP (Enterprise Resource Planning), les logiciels internes : logiciel de ressources humaines, de commande

d’échantillon pour les commerciaux, de paye, de demande d’achat… Je travaille donc dans ce service, composé d’une vingtaine de personnes.



Cette infographie n’est pas à jour mais on peut y voir mon Tuteur de stage : Benoit Benard, ainsi que le deuxième collègue

avec qui je partageais mon bureau. Et plus globalement toutes les personnes avec qui j'ai pu échanger et collaborer occasionnellement.
Pourtant ils ne s’occupent pas des sites web vitrine tel que www.nicoll.fr, www.aliaxis.fr. Cela est le fruit du travail de leur sous-traitant à Nantes : Mazedia (Qui eux utilisent Drupal, un CMS proche de Wordpress mais bien plus stable et robuste). Ils s’occupent plus d’interfaces entre AX et les besoins des utilisateurs, d’où les solutions qu’ils produisent.

La mission

2.1 La problématique des fournisseurs

Aujourd'hui, toute personne au sein de Aliaxis en France (Nicoll, GIRPI, AUI) voulant faire un achat passe par une interface web (« la demande d'achat ») , pour tous type d'achat : la contraction des prestaires ou l'achat de produits non stockés ou fournitures (pas les matières premières), la contraction d'un prestataire, ou même des stylos. Pour cela ils choisissent un fournisseur parmi un liste définie, lorsqu'il est nécessaire d'en ajouter un nouveau cela est réalisé par un échange ce mail avec la comptabilité, cela prend un temps précieux et ne permet pas une traçabilité suffisante. Pour répondre à cette problématique, la comptabilité a sollicité le département informatique. Un collègue et moi-même avons pu nous entretenir avec deux personnes de la comptabilité afin de récolter les premières informations et dresser un cahier des charges. Nous avons tiré cette analyse et défini ce process :

- Lorsqu'un fournisseur doit être ajouté, l'employé va envoyer une fiche (au format Excel) au fournisseur pour qu'il renseigne plusieurs champs (Dénomination sociale, adresse, N° de TVA intercommunication, email, téléphone, N° de Siret).
- Une fois reçu l'employé va reporter ces informations dans un formulaire (Il sera possible d'importer un fichier Excel si j'ai le temps de le faire).
- Une première approbation sera donnée par le GCC
- Ensuite le service achat il devra ajouter plusieurs informations comme :
    - Le nom qui apparaitra sur la plateforme d'achat (s'il souhaite le changer),
    - Le mode de paiement via une liste déroulante (alimenté en fonction du pays, Union Européenne, Hors UE, France)
    - Les conditions de paiement (exemple : sous 30 jours) avec une liste déroulante alimenté par leur ERP (car susceptible de changer),
    - La famille d'achat (un code interne utilisé pour indiquer la nature du produit (aussi une liste déroulante alimenté par leur ERP).
- Ces informations seront vérifiées et validées ou non par le Directeur des achats
- Pour finir, les informations sont transmises à la comptabilité. Ce service va vérifier les informations et compléter les données comptables.
- Une fois toutes ces données remplies, une interface SSIS permettra de créer automatiquement le fournisseur dans l'ERP.

2.2 Définition du besoin

Pour mener à bien ce projet, il me faudra donc réaliser plusieurs choses :

- Définir les champs, nécessitant des listes déroulantes

- Réaliser un workflow grâce aux indications des clients (la comptabilité)
- Une base de données, où la grande majorité des tables servent à alimenter des listes déroulantes
- Créer des commande php (symfony) pour importer les données dans la base
- Concevoir un design (respecter un code couleur, m'inspirer du visuel de l'interface du logiciel de frais évoqué précédemment afin de d'obtenir un visuel cohérent avec les solutions précédemment développé pour tous les écrans)
- Pour les utilisateurs, j'ai déterminé 7 écrans :
    - 3 écrans pour les demandeurs
    - 3 écrans pour les achats
    - 3 écrans pour la comptabilité

    } Une liste de demandes et un formulaire d'ajout/édition et un autre pour visualiser les demandes seulement
    - 1 écran pour la connexion
- Pour les Administrateurs, j'ai déterminé 15 écrans :
    - 14 écrans pour simplement afficher le contenu des tables (surtout utile pour les tables importées quotidiennement de l'ERP). La table user sera un peu plus travaillé (des actions concernant la gestion des utilisateurs, réinitialiser un mot de passe, voir ses demande ect…)
    - 1 écran avec une liste de toute les demandes en cours et passées
- Une documentation suffisante assurant la maintenabilité du code
- Faire des commits de façon récurrente pour de nombreuses raisons évidentes
- Apprendre Symfony, pour cela, mon Tuteur de stage m'avait conseillé plusieurs vidéos afin de me familiariser avec ce Framework. Ces différentes vidéos ont été un atout majeur, et m'ont permis de presque finir ce projet de bout en bout. Cependant j'ai tout autant appris en codant le projet grâce à la documentation officielle de Symfony (très complète), à divers sites/forums (comme Stack overflow) pour des problématiques similaires mais aussi avec des codes développés par mon Maître de stage pour des projets antérieurs.

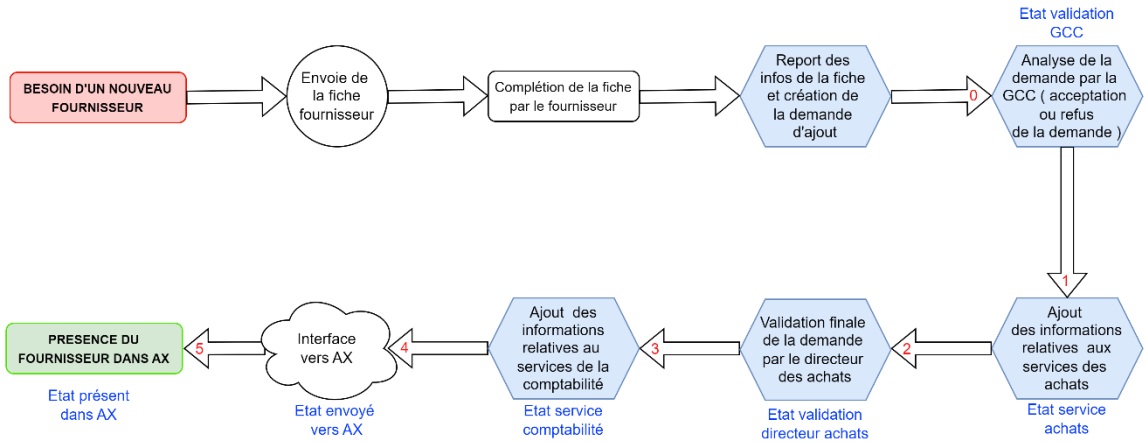
2.3 Outils de développement

Pour les interfaces web, Nicoll, Aliaxis France utilise le framework Symfony. Mon tuteur m'a donc laissé le choix : accomplir cette tâche en PHP « classique » ou bien me former, et utiliser ce framework. J'ai opté pour Symfony plutôt que PHP pur pour plusieurs raisons. Tout d'abord, Symfony offre une structure solide et suit les meilleures pratiques du développement web, ce qui enrichira mes compétences. Ensuite, ses fonctionnalités avancées simplifient le développement et améliorent la maintenabilité du code. De plus, grâce à sa documentation et à sa communauté, l'équipe pourra facilement récupérer mon travail après mon stage. Pour faire simple, Symfony garantit non seulement la qualité du projet, mais aussi une collaboration et une maintenance efficaces à long terme.

Les solutions apportées

3.1 Le workflow

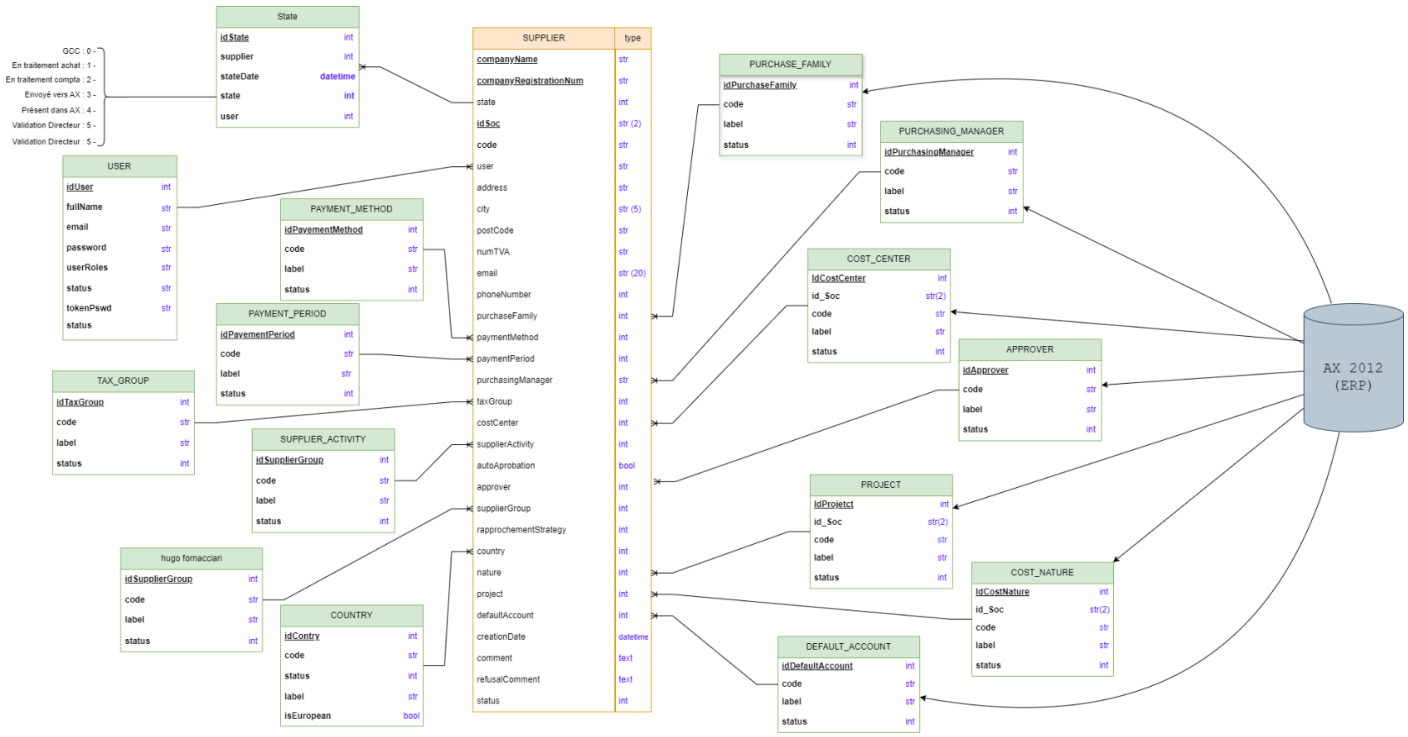
Pour partir sur de bonnes bases, bien comprendre le projet, j'ai réalisé la modélisation du workflow qui a permit ensuite de définir les écrans nécessaires.



Le workflow de la demande d'ajout de fournisseur :

Ensuite, après avoir plus ou moins compris le process utilisé et avoir listé la plupart des champs, est venu le moment de commencer le corps de l'application :

3.2 La conception de la base de données



J'ai donc réalisé cette base de données de A à Z, sur draw.io a l'aide des informations morcelés que j'ai récupérées. En effet à chaque échange avec les clients des changements était nécessaire jusqu'à la dernière présentation (où il a fallu échanger deux états). Une fois la modélisation faite, je suis passé à la console Symfony qui me permet rapidement et simplement de créer une entité, ensuite je fais une « migration » (sorte de conversion entre les changements sur les entités qui sont traduit en ordres SQL. Pour finir je migre cette migration ce qui vient exécuter les ordres de toutes les migrations appliquant les changements (création, modifications)

Documentation Symfony Bdd

3.2.1 La gestion des utilisateurs

Pour cette application, il est nécessaire de prévoir une bonne gestion des utilisateurs. En effet, le service achat doit voir les demandes qui sont à l'état qui leur correspond, de même pour la comptabilité et les utilisateurs eux doivent pouvoir suivre l'état de leur demande. Pour cela s'est posé la question : comment gérer ces rôles/droits sachant qu'ils peuvent en avoir plusieurs ? (ex : ROLE_USER et ROLE_ACHAT).

La bonne réponse ou du moins la plus « jolie » était donc de les gérer via un tableau JSON dans la base.

Le grand cylindre nommé AX représente leur ERP, à partir duquel seront extraites quotidiennement les différentes tables qui lui sont associées. Cette base a été créée en local sur laragon, mais les tables ont été entrées à l'aide du Framework Symfony (plus précisément à l'aide de l'ORM Doctrine par défaut avec Symfony), qui simplifie grandement les liens et interactions entre une base de données et un site web. La propriété '' status '' récurrente et présente dans toute les tables a un intérêt bien précis.

NB : Ces deux modélisations ont été faite sur le logiciel de graphs draw.io

3.3 Les commandes d'import

Le framework Symfony fournit de nombreuses commandes via le script bin/console (par exemple la célèbre commande bin/console cache:clear). Ces commandes sont créées avec le composant Console. On peut également l'utiliser pour créer nos propres commandes. Documentation commandes Symfony

Afin d'importer les informations provenant de l'ERP (qu'elles soient importées quotidiennement ou une seule fois), J'ai dû coder une commande par table. Grâce à un modèle d'une commande réalisé pour un projet antérieur et la documentation Symfony, j'ai pu sans trop de difficultés développer ces différentes commandes. Pour résumer le procédé, je '' désactive '' tous les enregistrement (status à 0) c'est une simple lecture de fichier CSV, j'entre ensuite les informations dans la base met le status à 1 lorsque c'est

un nouvel élément ou s'il est déjà présent je le réactive. Ensuite, pour faciliter les différents imports, j'avais besoin d'une commande pour tout importer d'un coup. Et c'est que présente un problème alors que j'imaginais cette tache comme quelque chose de simple : un parcourt de tableau qui exécute chacune des commandes. J'ai donc simplement créé un tableau avec le nom de toutes les commandes, et avec une simple boucle et la fonction '' run '' pour les exécuter une par une.

3.4 Les controllers

Un contrôleur est une fonction PHP que vous créez qui lit les informations de l'objet Request et crée et renvoie un objet Response. La réponse peut être une page HTML, JSON, XML, un téléchargement de fichier, une redirection, une erreur 404 ou toute autre chose. Le contrôleur exécute la logique arbitraire dont votre application a besoin pour restituer le contenu d'une page. Ces fichiers sont dans un dossier spécifique Controller et sont composé de fonctions. Il faut donc créer une fonction par page, ces fonctions peuvent être organisée via plusieurs fichiers Controller.

J'ai créé quatre fichiers contrôleurs pour mon application. Chaque fichier est dédié à une partie spécifique de l'application :

1. AdminController : Gère toutes les fonctionnalités et opérations liées à la partie administrateur (ex :affichage de toute les tables).
2. UserController : Prend en charge toutes les fonctionnalités destinées aux utilisateurs (ex : affichage des demandes en cours des utilisateurs) .
3. LoginController : Responsable de la gestion des processus d'authentification.
4. RegistrationController : Gère les processus d'inscription.

Les fichiers de contrôle pour la connexion et l'inscription m'ont été fournis, et je les ai adaptés à mon code existant pour répondre à mes besoins spécifiques.

Ces contrôleurs sont organisés de manière à assurer une séparation claire des responsabilités, facilitant ainsi la maintenance et l'évolution de l'application.

3.5 Les différents écrans

L'affichage des demandes en fonction du rôle de l'utilisateur et donc de l'état de

```
<tbody>
    {% for cs in datas %}
    <tr>
        <th>{{cs.id}}</th>
        <th>{{cs.code}}</th>
        {% if cs.label is empty %}
            <th>{{ '{ Non renseigné }' }}</th>
        {% else %}
            <th>{{ cs.label }}</th>
        {% endif %}
        {% if cs.status == 1 %}
            <th>{{ 'Actif' }}</th>
        {% else %}
            <th>{{ 'Inactif' }}</th>
        {% endif %}
    </tr>
    {% endfor %}
</tbody>
```

Pour le design, j'ai donc repris comme modèle la demande d'achat et essayer de faire un affichage le plus fidèle possible à cette dernière (j'ai aussi récupéré son footer, header, navbar) avec bootstrap.

la demande n’a pas été un casse-tête, avec une méthode je viens récupérer l’utilisateur authentifié, son identifiant, son rôle et plus qu’à afficher les demandes avec ces conditions. Les vues Administrateur, qui permettent d’afficher les tables de la base, sont vraiment peu complexe et comportent un simple tableau avec un "foreach" Twig.

3.6 Le formulaire d’ajout

Pour créer un formulaire avec Symfony, l'intégration des entités et des relations est simplifiée grâce à ses fonctionnalités intégrées (documentation form Symfony):

- Génération Automatique :

  Symfony permet de générer facilement des formulaires à partir des entités existantes en utilisant la commande ‘php bin/console make:form’. Cela crée un formulaire prérempli basé sur la structure de votre classe d'entité.

- Personnalisation des Labels :

  Par défaut, Symfony utilise les noms des propriétés de l'entité comme labels pour les champs du formulaire. Pour des labels plus explicites, il est nécessaire de les renommer manuellement dans la définition du formulaire :

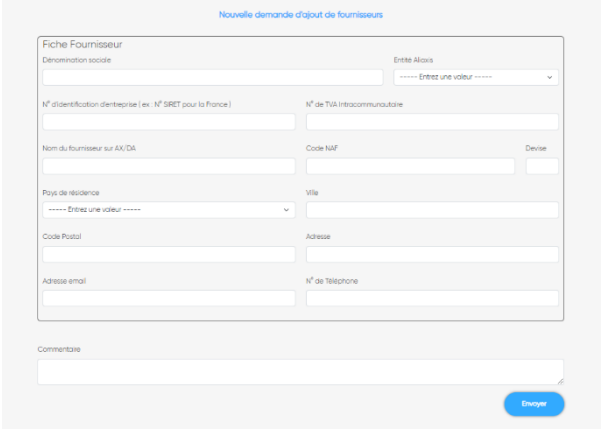
```php
class SupplierType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('companyName',options:[
                'required' => true,
                'label' => 'Dénomination sociale',
            ])
```

- Gestion des Relations :

  Pour les champs qui représentent des relations, comme ManyToOne ou OneToMany, Symfony crée automatiquement des listes déroulantes mais il manque de nombreuses options. Vous pouvez spécifier comment ces entités doivent être affichées où utiliser une méthode qui concatène plusieurs champs comme moi :

```php
// d'origine :
->add('purchaseFamily')
//une fois les modifications apportée (sinon une erreur se produit
//symfony ne sait pas quoi affiche, l'id, le label ect) :
->add('purchaseFamily', EntityType::class,[
    'class' => PurchaseFamily::class,
    'placeholder' => '----- Entrez une valeur -----',
    'required' => true,
    'label' => 'Famille d\'achat',
    'choice_label' => 'codeAndLabel'
])
```

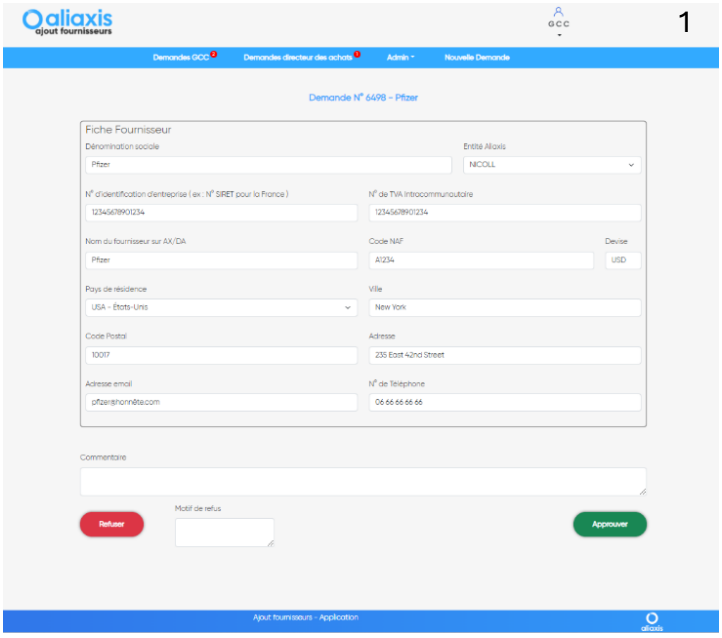
Pour faire simple, Symfony simplifie la création de formulaires en automatisant une grande partie du processus, mais nécessite parfois des ajustements manuels comme la personnalisation des labels et la gestion fine des relations (qui devrait être amélioré nativement à mon avis). A droite, une capture du formulaire d'ajout, tous ces champs sont obligatoires et sont la retranscription de la fiche fournisseur

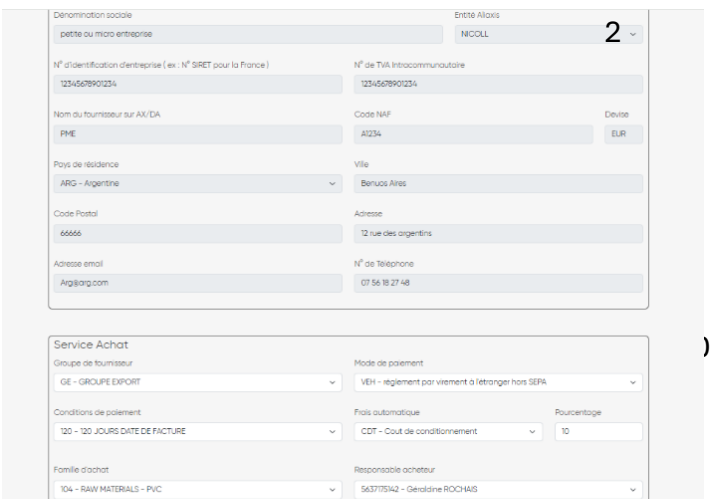


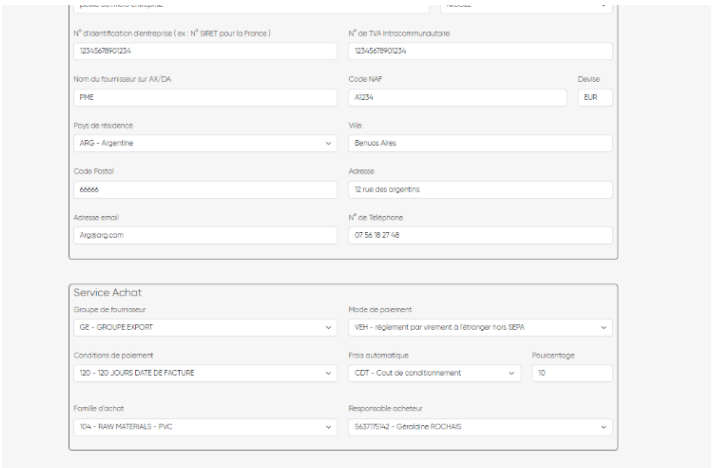
3.7 Formulaire d'édition

Ce formulaire permet d'ajouter les informations (liées à chaque étape du workflow). Il reprend la même template (même fichier twig) que l'ajout de fournisseur, toutes les étapes et captures viennent de cette template. La seule différence entre l'ajout et la modification est le Controller.

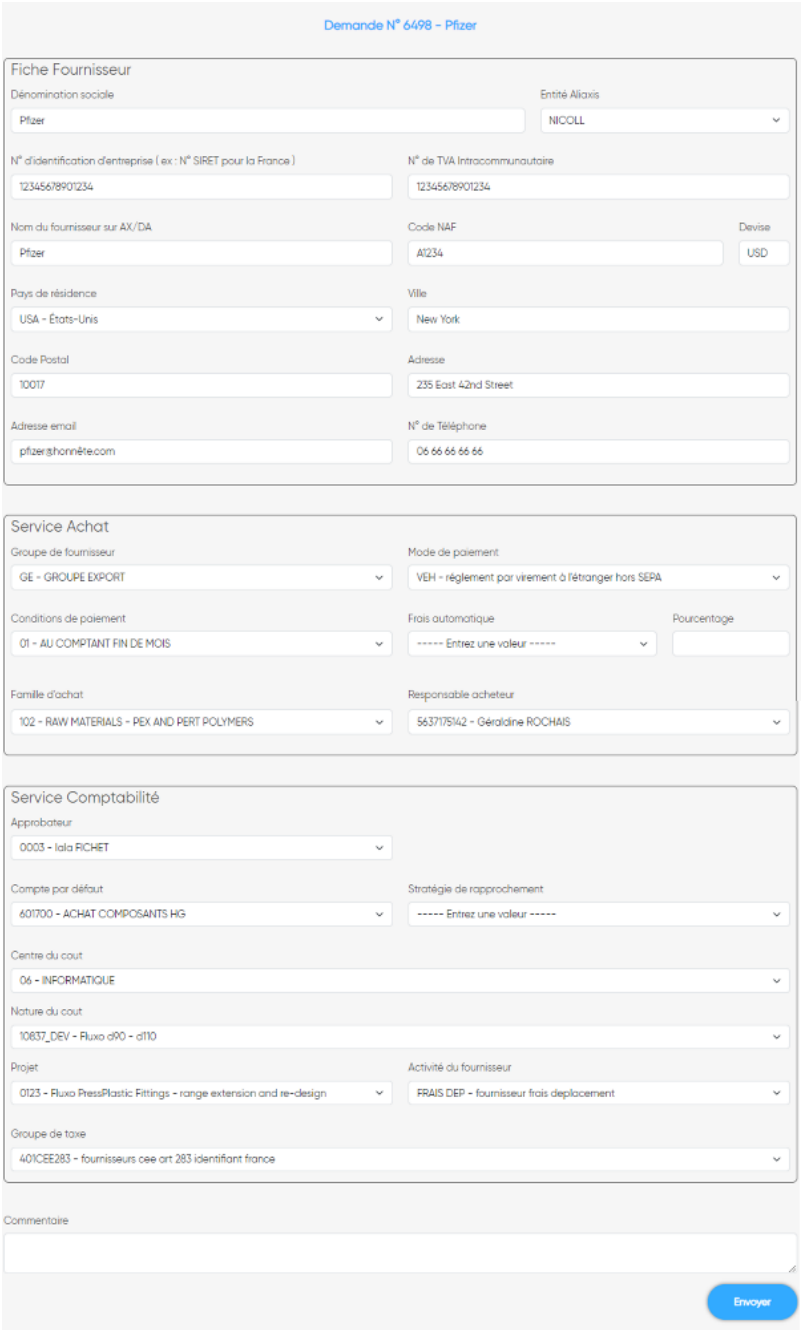
- La première capture, correspond à la validation GCC qui permet de vérifier la légitimité de la demande et de l'entreprise concerné.
- La deuxième correspond à l'étape du service des achats qui apporte des informations liées au paiement, au type d'achat, responsable correspondant...
- La troisième capture, correspond à la validation par le directeur des achats, c'est une confirmation de la création d'un fournisseur.







Pour finir le formulaire de la comptabilité, qui peut vérifier l'exactitude de tous les champs avant son importation dans AX. Ils peuvent modifier toutes les informations entrées jusqu'ici, et doivent aussi remplir les champs relatifs à la comptabilité. Une fois cette étape passé, l'état passe à « envoyé vers AX » et la demande sera retranscrite dans un fichier csv qui sera ensuite envoyé dans AX via une interface SSIS.



```twig
{% if supplier.state is defined and (supplier.state == 2 or supplier.state == 0) %}
                                {# 2 = validation directeur et 0 = validation CGG #}
    <div class="d-flex align-items-center justify-content-between flex-row">
    <div class="d-flex align-items-center">
        <div class="c">
            {{ form_row(form.refuser, {'attr': {'class': 'btn btn-danger'}}) }}
        </div>/.c
        <div class="ml-5">
            {{form_row(form.refusalComment, {'attr': {'class' : 'form-control'}}) }}
        </div>/.ml-5
    </div>/.d-flex.align-items-center
        <div class="droite">
            {{ form_row(form.valider, {'attr': {'class': 'btn btn-success'}}) }}
        </div>/.droite
    </div>/.d-flex.align-items-center.justify-content-between.flex-row

        {{ form_row(form.save, {'attr': {'class': 'hidden'}}) }}
{% else %}
    <div class="d-flex justify-content-end">
        {{ form_row(form.refuser, {'attr': {'class': 'hidden'}}) }}
        {{ form_row(form.valider, {'attr': {'class': 'hidden'}}) }}
        {{ form_row(form.save) }}
    </div>/.d-flex.justify-content-end
    <div class="hidden">
        {{ form_row(form.refusalComment) }}
    </div>/.hidden
{% endif %}
```

Afin que l'affichage soit dynamique en fonction de l'état de la demande (la visibilité des boutons, des fieldsets ainsi que des disabled sur les fieldsets ou non) j'ai utilisé les conditions dans la template Twig.

Ce code permet donc d'afficher les bons boutons selon l'état de la demande, cela est possible car j'ai passé dans mon Controller des paramètres :

```php
$page = $this->render('add_form.html.twig', [
    'supplier' => $demande,
    'form' => $form,
    'nbDmd' => $getSupplierRqstNumberService->getSupplierRqstNumberService($state),
    'nb_dmd_gcc' => $getSupplierRqstNumberService->getSupplierRqstNumberService(0),
    'nb_dmd_directeur' => $getSupplierRqstNumberService->getSupplierRqstNumberService(2)
]);
```

Grâce à l'id je retrouve la demande et le passe à la template, je passe le form afin de pouvoir l'afficher et 3 autres paramètres permettant de faire des info-bulles avec le nombre de demandes à traiter.

Je passe donc la demande en question récupéré grâce à une route dynamique :

```php
#[Route('edit/{id}', name: 'edit_demand')]
```

Les difficultés rencontrées

4.1 L'environnement financier

Les premiers obstacles n'ont pas tardé à se présenter, en effet, n'étant pas familier avec l'univers de la comptabilité, le seul fait de bien assimiler le processus d'ajout d'un fournisseur était loin d'être évident, mais après avoir dialogué avec les clients j'ai naturellement réussi saisir le concept.

4.2 La chasse aux données

La conception de la base n'a été une promenade de plaisance, non pas parce qu'elle était particulièrement complexe (des cardinalités 1,N partout) mais davantage car j'ai reçu les règles, champs (tables) en décousu. Aujourd'hui, j'ai appris que l'on

devait pouvoir retracer une demande de A à Z, alors que lorsque j'avais posé la question (plusieurs semaines) on m'avait dit que non. Malgré cela voila le modèle conceptuel que j'ai produit (validé et utilisé).

4.3 La factorisation des controllers admin

Au début, j'ai commencé par coder une fonction par table, arrivé la moitié je me suis rendu compte que développer une fonction me permettant d'afficher toutes les tables de manière dynamique dans une interface d'administration serait la bienvenue. Je voulais éviter d'écrire une méthode spécifique pour chaque table, et plutôt créer une solution générique et réutilisable, mais cela semblait complexe.

Pour répondre à ce besoin, j'ai créé une méthode dans mon contrôleur Symfony qui utilise un paramètre dynamique pour déterminer quelle table afficher. Voici comment cela fonctionne :

Route Dynamique :

```
#[Route('/admin/{file}', name: 'admin')]
```

J'ai défini une route dynamique où {file} représente un paramètre variable dans l'URL. Cela permet de générer des URL comme /admin/cost-center ou /admin/approver. Elle est ensuite associée à une méthode qui prend le paramètre dynamique {file} de l'URL et l'interface EntityManager d'ORM pour interagir avec la base de données.

Puis je me heurte à un problème, comment passer de la variable à la classe correspondante ? Mais grâce à stackoverflow j'ai trouver quelqu'un qui avait également besoin de convertir une chaîne de caractères en une classe. Cette personne a procédé en ajoutant "App\\Entity\\" devant la chaîne, ce qui correspond au dossier où se trouvent les entités dans Symfony, c'est grâce à cette astuce que j'ai pu factoriser mes controllers alors je n'étais pas sûr que cela était même possible.

Je récupère le repository correspondant à la classe d'entité déterminée grâce à la méthode, ce qui me permet de faire des requêtes sur cette table.

J'utilise le repository pour récupérer toutes les entrées de la table, triées par status de manière décroissante et par label de manière croissante.

```
$datas = $repoIdClass->findBy(
            [/* filters */],
            ['status' => 'DESC', 'label' => 'ASC']
        );
```

Enfin pour l'affiche de la page/ du HTML j'utilise Twig, Twig est un moteur de templates puissant et intuitif intégré dans Symfony. Il offre une syntaxe simple et lisible, échappe automatiquement les variables pour prévenir les attaques XSS, et compile les templates en PHP natif pour des performances optimales. Twig supporte l'héritage de templates

pour une réutilisation efficace du code et est facilement extensible grâce à des extensions. Intégré nativement dans Symfony. Naturellement, je rends une vue Twig spécifique pour chaque table en utilisant le paramètre {file} pour déterminer la template à utiliser. Les données récupérées sont passées à la vue.

4.4 Mise en pratique des rôles

Pour sécuriser une application Symfony, le fichier 'security.yaml', souvent configuré par le biais d'un bundle comme 'Symfony Security Bundle', joue un rôle crucial en définissant les règles d'accès basées sur les rôles des utilisateurs. Voici une explication du concept et de mon expérience avec ce fichier :

Ce fichier permet de configurer les règles d'accès sécurisé aux différentes parties de l'application en fonction des rôles des utilisateurs. On y définit notamment les chemins d'accès et les rôles nécessaires pour y accéder.

Voilà ce que j'avais produit à l'aide de la doc :

```
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/, roles: ROLE_USER }
```

Cette configuration indique que seuls les utilisateurs ayant le rôle 'ROLE_ADMIN' peuvent accéder aux chemins commençant par '/admin', tandis que tous les utilisateurs authentifiés ('ROLE_USER') peuvent accéder aux autres chemins.

2. Problème rencontré :

Au début, j'ai fait face à un problème où l'accès à la page de connexion ('/login') était également bloqué, car elle était incluse dans le chemin général ('^/'). Cela signifiait que même pour accéder à la page de connexion, l'utilisateur devait être authentifié, ce qui est contre-productif.

3. Solution apportée :

Après avoir identifié le problème, j'ai ajusté la configuration pour exclure explicitement le chemin vers la page de connexion ('^/login') en lui assignant un rôle spécial ('PUBLIC_ACCESS' dans votre exemple) qui permet un accès public sans authentification préalable.

Puis en vérifiant avec mon tuteur :

```
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/login, roles: PUBLIC_ACCESS }
    - { path: ^/, roles: ROLE_USER }
```

Avec cette correction, les utilisateurs peuvent maintenant accéder librement à la page de connexion sans être redirigés vers une authentification préalable, tout en maintenant la sécurité sur les autres parties de l'application.

Pour faire simple, le fichier ‘security.yamlpm’, est essentiel pour définir qui peut accéder à quoi dans votre application Symfony en fonction des rôles (pas seulement). Il est important de configurer correctement les chemins d'accès pour éviter les blocages involontaires comme celui rencontré avec la page de connexion.

4.5 L’import des fournisseurs

L'import des fournisseurs dans notre application se fait via une commande Symfony utilisant un fichier CSV comportant quatre colonnes : Société concernée (Nicoll/Girpi/AUI), Code (nom court affiché lors d’un achat), Dénomination Sociale, et numéro de TVA intracommunautaire. Cette fonctionnalité a été développée en dernier car la structure de la table n'était pas stable.

Une fois la table modélisée et l'entité créée (" dans " Symfony en premier) dans la base de données, j'ai développé la commande d'import sur le modèle des précédentes, et les fournisseurs ont commencé à apparaître dans la table. J'ai ensuite travaillé sur la vue Admin pour afficher les champs de la table. Cependant, j'ai constaté que le champ du numéro de TVA était vide. Après investigation, j'ai réalisé que bien que je lisais la valeur, je ne l’ajoutais pas à l’objet, ce qui expliquait son absence dans la base de données.

Une autre difficulté est apparue à l'importation : une erreur signalait une valeur trop grande pour le champ. Les données du fichier n'étaient pas toutes conformes, et les formats de numéros de TVA variaient selon les pays. J'ai donc augmenté la taille du champ à 14 caractères pour accueillir les numéros de TVA néerlandais, les plus longs, par rapport aux 13 caractères utilisés en France.

Malgré ces ajustements, l'erreur persistait en raison de la qualité médiocre des informations des fournisseurs dans AX. Sous les conseils de mon tuteur, j'ai limité le numéro de TVA aux 14 premiers caractères. Finalement, les fournisseurs étaient correctement importés dans la base de données.

4.5.2 Changement majeur en cours de route

Lors de la quatrième semaine du projet, j'ai participé à une réunion avec les clients, accompagné de mon maître de stage ainsi qu'un autre collègue, chargé du développement de l'interface SSIS et de l'envoi des fichiers CSV vers notre base de données. Pendant cette réunion, une question cruciale a été soulevée : « Est-il nécessaire d’historiser les changements d’états ? ». La réponse fut affirmative, soulignant l'importance de la traçabilité.

En réponse à cette demande, j'ai ajouté une nouvelle entité intitulée « state », composée des attributs suivants :

- Date
- Demande (fournisseur)
- État
- Utilisateur

Cette modification a eu un impact sur les contrôleurs (Controllers), car dorénavant, à chaque changement d'état, il est nécessaire d'ajouter une entrée dans la table "state". Cette nouvelle procédure s'applique également lors de l'importation des données.

Processus d'Importation des Fournisseurs

Le processus d'importation des fournisseurs, dans sa version finale, se déroule comme suit :

Lecture et Comparaison des Données :

Lors de la lecture du fichier CSV, les numéros de TVA sont comparés à ceux déjà présents dans la base de données.

Si un numéro de TVA n'est pas trouvé dans la base, l'état de ce fournisseur est défini à 5 dans la table "supplier". (Sous conseil de mon maître de stage j'ai conservé l'état dans cette table malgré la redondance pour faciliter son utilisation lors de conditions dans une template par exemple)

+ Un nouvel enregistrement est créé dans la table "state" avec les informations suivantes :

- Heure de l'état
- Utilisateur AX (utilisé uniquement pour les imports)
- Fournisseur correspondant
- État 5

Gestion des États Existants :

- Si une correspondance est trouvée (égalité de numéro de TVA), l'état est récupéré.

- Si l'état est déjà égal à 5, aucune action supplémentaire n'est requise.

- Si l'état n'est pas 5, il est incrémenté de 1, et un nouvel état est créé dans la - table "state" de manière similaire au premier cas de figure.

Gestion des Fournisseurs Absents dans le Fichier CSV :

- Les fournisseurs présents dans la base de données mais non trouvés dans le fichier CSV sont considérés comme ayant été supprimés ou retirés d'AX.

- Leur état est alors mis à jour à 0.

4.6 Récupération des données du formulaire

Lorsque j'ai dû m'attaquer au formulaire, j'ai commencé par écrire cela dans mon Controller "add_supplier" :

```
$demande = new Supplier();
$form = $this->createForm(SupplierType::class;
```

Puis pour l'afficher, mon Controller renvoie :

```
$page = $this->render('add_form.html.twig', ['form' => $form]);
```

Ensuite pour récupérer les informations saisies :

```
$form->handleRequest($request);
if ($form->isSubmitted() && $form->isValid()) {
      //$form->getData() holds the submitted values
      // but, the object thas also been updated
      $demand = $form->getData();
```

Et quand bien même ce code provient de la documentation officielle, lorsque fais un dump de mon objet, ce dernier est vide... Je suis resté bloqué un certain temps, cherché la cause sans jamais la trouver. Finalement après avoir revu avec mon tuteur de stage, il s'est avéré que le problème venait de la création du formulaire, il manquait un paramètre primordial, l'objet fraichement crée :

```
$form = $this->createForm(SupplierType::class, $demande);
```

4.7 La boite de pandore

Récemment, en travaillant sur un formulaire avec Symfony, j'ai ouvert une véritable boîte de Pandore. Malgré tous mes efforts et ceux de mon maître de stage, je n'ai pas réussi à résoudre un problème frustrant. Les champs requis du formulaire ne s'activaient pas lorsque je cliquais sur le bouton valider, bien que tout semblait correct dans le code.

J'ai vérifié à plusieurs reprises que l'option « `required` » était bien définie sur « `true` » dans mon SupplierType (classe du formulaire) :

```php
class SupplierType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('companyName',options:[
                'required' => true,
                'label' => 'Dénomination sociale',
            ])
```

J'ai également vérifié que les attributs required étaient bien présents dans l'HTML généré. Chaque champ avait l'attribut `required="required"` comme attendu. Nous avons aussi exploré différentes méthodes pour gérer le bouton de soumission dans Symfony, que ce soit directement dans le SupplierType ou dans le template Twig, et en récupérant l'information avec un get Symfony. Malgré cela, le problème persistait. Ce qui est particulièrement étrange, c'est que les champs requis fonctionnaient correctement lors de la validation GCC pour le refus comme pour l'approbation, alors que c'est le même contrôleur et le même template Twig. Ce mystère reste non résolu.

Après avoir exploré toutes ces pistes, la solution qui nous semble la plus probable est d'implémenter une validation en JavaScript. Cela permettrait de contourner ce bug incompréhensible et de garantir que les champs requis sont bien vérifiés avant la soumission du formulaire.

Ce problème reste un mystère pour nous, et bien que nous ayons essayé de le résoudre de toutes les manières possibles avec Symfony, il semble que la validation côté client via JavaScript soit notre meilleure option pour l'instant. Ouvrir cette boîte de Pandore nous a certainement posé plus de questions que fourni de réponses.

Conclusion – Bilan

Mon stage au sein du service informatique de Nicoll a été une expérience plus qu'intéressante et enrichissante. Durant cette période, j'ai eu l'opportunité d'apprendre de nombreuses choses et de me confronter à diverses problématiques. Bien que certaines aient été résolues avec succès, d'autres n'ont malheureusement pas pu l'être complètement. Cela dit, chaque défi rencontré a été une occasion d'apprentissage et de développement professionnel.

Je tiens à remercier chaleureusement toute l'équipe du service informatique de Nicoll pour leur soutien et leur encadrement tout au long de ce stage. Leur expertise et leur disponibilité ont grandement facilité mon intégration et m'ont permis de progresser efficacement sur le projet.

Malgré mes efforts, je n'ai pas pu finaliser complètement le projet. Il reste quelques points à perfectionner :

- Activer les champs "required".
- Filtrer les listes déroulantes pour exclure les valeurs inactives (status = 1).
- Convertir les demandes en fichier CSV pour l’export vers AX
- Intégrer des pièces jointes pour les documents KBIS, RIB et URSSAF.
- Compléter l'import des utilisateurs malgré les difficultés rencontrées.
- Mettre en place une vue de l'historique des états d'un fournisseur ainsi qu'une fonction de recherche dans les fournisseurs.
- Améliorer le tri et le filtrage dans la vue des demandes.
- Implémenter Select2 pour une recherche intuitive dans les champs avec de nombreuses valeurs comme la famille d'achat.
- Ajouter le rôle GCC pour une gestion optimale.
- Faciliter l'accès à l'ajout d'un fournisseur depuis une demande d'achat.

Malgré ces points en suspens, je suis globalement très satisfait de cette expérience de stage. Avoir pu travailler sur un projet aussi vaste de A à Z, qui aura un impact concret et positif pour l'entreprise, a été extrêmement gratifiant. Ce stage m'a également offert l'occasion précieuse de me former sur Symfony, une compétence qui constituera un atout majeur pour ma deuxième année et au-delà, enrichissant ainsi mon parcours professionnel.

Ce stage chez Nicoll m'a non seulement permis de développer mes compétences techniques, mais il m'a également permis de mieux comprendre les défis opérationnels et stratégiques auxquels une entreprise comme Nicoll fait face quotidiennement (notamment avec l'évolution de leur ERP).

Je suis persuadé que ce stage chez Nicoll a été une étape cruciale dans mon développement professionnel, me préparant efficacement aux défis futurs et renforçant mes compétences dans le domaine du développement informatique.