



Rapport d'alternance

En vue de l'obtention du titre : Architecte logiciel, développeur
d'applications

Alternance réalisée par [Valentin ROUXEL](#)
du 06 Septembre 2021 au 31 Août 2023

[Maxime MERRIEN](#)
[Karen FÉVRIER](#)
[Marvin BELLOUARD](#)

(Maître d'alternance - Responsable technique)
(Maître d'alternance - Cheffe de projet)
(Responsable d'apprentissage EPITECH)

Remerciements

Je tiens à émettre mes remerciements à l'équipe pédagogique d'Epitech pour son enrichissement technique ainsi que l'accompagnement humain qu'ils ont pu réaliser durant le temps que j'ai passé avec eux lors de ces deux années au sein de la réalisation de ce master.

Je remercie également tous les membres de la Digital Factory pour le temps passé avec eux à monter en compétences, les échanges techniques, ainsi que toute la bienveillance présente au sein de l'équipe.

Je tiens particulièrement à remercier Maxime MERRIEN pour son temps investi à me former sur le projet PREVARISC. Je tiens par ailleurs à le remercier d'avoir réalisé son encadrement avec une grande qualité pédagogique.

Je souhaite exprimer ma reconnaissance à Erwan DE SA NOGUEIRA pour la formation et l'accompagnement qu'il m'a apporté au sein de l'apprentissage de Drupal durant la fin de mon alternance.

Mes remerciements vont également à ma famille et mes amis pour tout le soutien apporté durant la réalisation de ce cursus.

Enfin, je tiens à exprimer mes remerciements à Atos pour m'avoir donné l'opportunité de réaliser cette alternance avec Epitech ainsi que de la confiance que m'a apportée la société.

Sommaire

Remerciements	2
Sommaire	3
Table des illustrations	5
Résumé	6
Mots clés	6
Partie 01 L'entreprise	7
1.1) Présentation de l'entreprise	7
1.1.1) Histoire de l'entreprise	7
1.1.2) Secteur d'activités d'Atos	7
1.1.3) Culture d'entreprise	7
1.1.4) Quelques projets auxquels Atos a participé	8
Partie 02 L'alternance	9
2.1) Présentation de la Digital Factory	9
2.2) Contexte de la mission	10
2.2.1) Le projet PREVARISC	10
2.2.2) Le vocabulaire dans PREVARISC	11
2.2.3) L'architecture de PREVARISC	13
2.2.3.1 Les technologies utilisées	13
2.2.3.1 Les difficultés rencontrées sur l'architecture	14
2.3) Les méthodes de travail mise en place	15
2.3.1) Une semaine type	15
2.3.2) Intégration des méthodes de travail opérationnelles	15
2.3.2.1 Apprentissage du processus de traitement des tickets	15
2.3.2.2 Apprentissage de l'utilisation de l'outil redmine	18
2.3.2.3 Apprentissage des procédés de communication entre les tiers	19
2.3.3) Intégration des méthodes de travail sur github	20
2.3.3.1 Qu'est ce que git et github ?	20
2.3.3.2 Nomenclature de branch appliqué	20
2.3.3.3 Mise en place d'une intégration continue	20
2.4) Les missions qu'on m'a attribuées	21
2.5) Préambule des évolutions, explication des champs de fusions	21
2.5.1) Qu'est ce qu'un champ de fusion ?	21
2.5.2) Qu'est ce qu'un squelette de fusion ?	22
2.6) L'évolution "champ parent"	22
2.6.1) Contexte de l'évolution "champ parent"	22

2.6.2) Conception de l'évolution "champ parent"	24
2.6.3) Intégration de l'évolution "champ parent"	25
2.6.3.1) Implémentation de la partie backend	25
2.6.3.2) Réalisation de la partie administrateur	26
2.6.3.3) Réalisation de la partie utilisateur	28
2.6.3.4) Génération des documents	28
2.6.4) Rétrospective de l'évolution "champ parent"	28
2.6.4.1) Le retour client	28
2.6.4.2) Points forts de l'équipe sur cette évolution	29
2.6.4.3) Difficultés rencontrées de cette évolution	29
2.7) L'évolution "champ tableau"	29
2.7.1) Contexte de l'évolution "champ tableau"	29
2.7.2) Conception de l'évolution "champ tableau"	30
2.7.3) Intégration de l'évolution "champ tableau"	30
2.7.3.1) Implémentation de l'entité champ tableau	30
2.7.3.2) Réalisation de la partie administrateur	31
2.7.3.3) Réalisation de la partie utilisateur	31
2.7.3.4) Génération des documents	32
2.7.4) Rétrospective de l'évolution "champ tableau"	33
2.7.4.1) Le retour client	33
2.7.4.2) Points forts de l'équipe sur cette évolution	33
2.7.4.3) Difficultés rencontrées de cette évolution	34
2.8) Après PREVARISC	35
Partie 03 Rétrospective d'alternance	36
3.1) Rétrospectives des compétences techniques acquises	36
3.2) Rétrospectives du savoir être acquis	36
Conclusion	37
Glossaire	38

Table des illustrations

Figure-01, Organigramme de la composition de la Digital Factory - Établi par l'auteur.....	9
Figure-02, Carte de répartition de l'utilisation de prévarisc dans la France en 2017 et 2020 - Extrait de document interne.....	11
Figure-03, Schéma de fonctionnement interne de l'application prévarisc - Extrait du manuel d'utilisation de l'application prévarisc.....	12
Figure-04, Schéma d'architecture partiel de l'application PREVARISC - Extrait de mémoire de fin d'étude de Maxime MERRIEN année 2019-2020.....	13
Figure-05, Tableau catégorisation des anomalies - Extrait de contrat.....	16
Figure-06, Tableau sévérité et délais de remise en service - Extrait de contrat.....	17
Figure-07, Schéma processus de prise en charge d'un ticket - Extrait de formation à l'application prevarisc.....	17
Figure-08, Capture du dashboard d'accueil de redmine- Etabli par l'auteur.....	18
Figure-09, Capture d'une liste de ticket redmine- Établi par l'auteur.....	19
Figure-10, Capture partielle d'un formulaire avec un champ - Etabli par l'auteur.....	22
Figure-11, Capture des différents types de champ - Établi par l'auteur.....	23
Figure-12, Capture d'un champ parent- Établi par l'auteur.....	23
Figure-13, Diagramme de class - Établi par l'auteur.....	24
Figure-14, Capture d'un script de migration- Établi par l'auteur.....	25
Figure-15, Capture d'un script de migration- Établi par l'auteur.....	26
Figure-16, Capture de la page administrateur de création de champ- Établi par l'auteur.....	27
Figure-17, Capture page administrateur, création de champ fils - Établi par l'auteur.....	27
Figure-18, Capture du rendu d'un champ parent - Établi par l'auteur.....	28
Figure-19, Capture page administrateur, création d'un champ tableau- Établi par l'auteur.....	31
Figure-20, Capture page utilisateur, exemple de champ tableau- Établi par l'auteur.....	32
Figure-21, Capture page administrateur, liste de champ de fusion d'un tableau- Établi par l'auteur.	33

Résumé

Durant mon cursus de master à Epitech j'ai pu réaliser une alternance. Celle-ci m'a permis de rejoindre la société Atos en tant que développeur full-stack dans l'équipe Digital Factory. J'ai travaillé sur le projet Prevarisc, développé principalement en Zend 1.12, et qui est également une application Open-Source recevant des évolutions fonctionnelles de la part de la société Atos. Mon travail consistait principalement à effectuer la maintenance, et la réalisation d'évolutions de l'application en ajoutant de nouvelles fonctionnalités. Le tout en respectant les exigences de qualité et de délais. Durant cette alternance, j'ai acquis des compétences solides en programmation orientée objet et en développement sur Zend, ainsi que des compétences en TMA.

Mots clés

- **Zend**
- **SDIS**
- **Evolution**
- **Anomalie**

Partie 01 L'entreprise

1.1) Présentation de l'entreprise

1.1.1) Histoire de l'entreprise

Atos est une entreprise d'échelle internationale d'origine française. Cette dernière est une entreprise de service numérique (*ESN*). Elle réalise différentes prestations techniques telles que le conseil dans le domaine du digital, gestion d'infrastructure informatique, cybersécurité, solutions de cloud computing et d'intelligence artificielle.

La société est fondée en 1997 suite de la fusion des deux sociétés Axime et Sligos. Atos voit son siège social être localisé à Bezons. L'entreprise emploie actuellement environ 105 000 personnes dans plus de 70 pays, ce qui fait d'elle l'un des leaders mondiaux.

Atos est subdivisé en plusieurs branches appelées unités opérationnelles. Durant mon alternance, j'y ai intégré l'une d'entre elles qui est la **Business Services (BS)**. Cette même branche se voit être composée de différentes équipes, telle que la **Digital Factory (DF)** que j'ai intégré durant mon alternance.

1.1.2) Secteur d'activités d'Atos

Atos du fait de sa notoriété se voit avoir des clients d'horizons variés. Atos peut se voir collaborer avec des sociétés privées plus ou moins grandes telles qu'Orange, Airbus, PMU, etc. Cependant Atos ne se restreint pas qu'aux sociétés privées, la firme collabore aussi avec le secteur public tel que la CNAF, le ministère de la Culture, secteur défense, ministère de l'intérieur avec lequel j'ai pu collaborer au sein de cette alternance, Ministère Du Budget, Des Comptes Publics Et De La Fonction Publique et de l'action public etc...

1.1.3) Culture d'entreprise

Atos est une société qui se veut de valoriser la diversité, l'inclusion et la collaboration entre ses employés. Cette dernière cherche à encourager l'innovation, la créativité et la prise d'initiatives. La culture d'entreprise se base sur des valeurs telles que l'intégrité, la transparence, la responsabilité et le respect de l'environnement. L'entreprise accorde également une certaine importance à la formation continue de ses employés notamment en

leur rendant possible diverses formations. Atos a ce souhait de formation afin que les développeurs puissent développer leurs compétences en élargissant leur éventail technique et par ce biais leur permettre de naviguer vers d'autres projets et technologies dans leur carrière. Enfin, Atos se veut être une entreprise socialement responsable, en contribuant à des initiatives pour un monde plus durable.

1.1.4) Quelques projets auxquels Atos a participé

Atos étant une société d'une grande envergure, cette dernière a pu réaliser différents projets de tailles colossales, que ce soit des projets industriels comme dans le secteur de la recherche. Atos s'illustre aussi dans le domaine de la recherche quantique via divers projets européens orientés sur l'informatique quantique, notamment via sa participation au Quantum Flagship.

Partie 02 L'alternance

2.1) Présentation de la Digital Factory

Lors de mon arrivée au sein de l'entreprise, j'ai intégré l'équipe **Digital Factory**. Cette dernière est une composante de la branche **Business Service** d'Atos. La **Digital Factory** a comme zone d'action la réalisation et la maintenance de portail ou application web pour divers clients. L'équipe est composée à l'heure où j'écris ce rapport d'un peu plus d'une dizaine de membres. La Digital Factory suit l'organigramme suivant, à noter que chaque membre de la DF, a un fond d'étiquette bleue si ce dernier est un employé propre à Atos et mauve si ce dernier est un sous traitant.

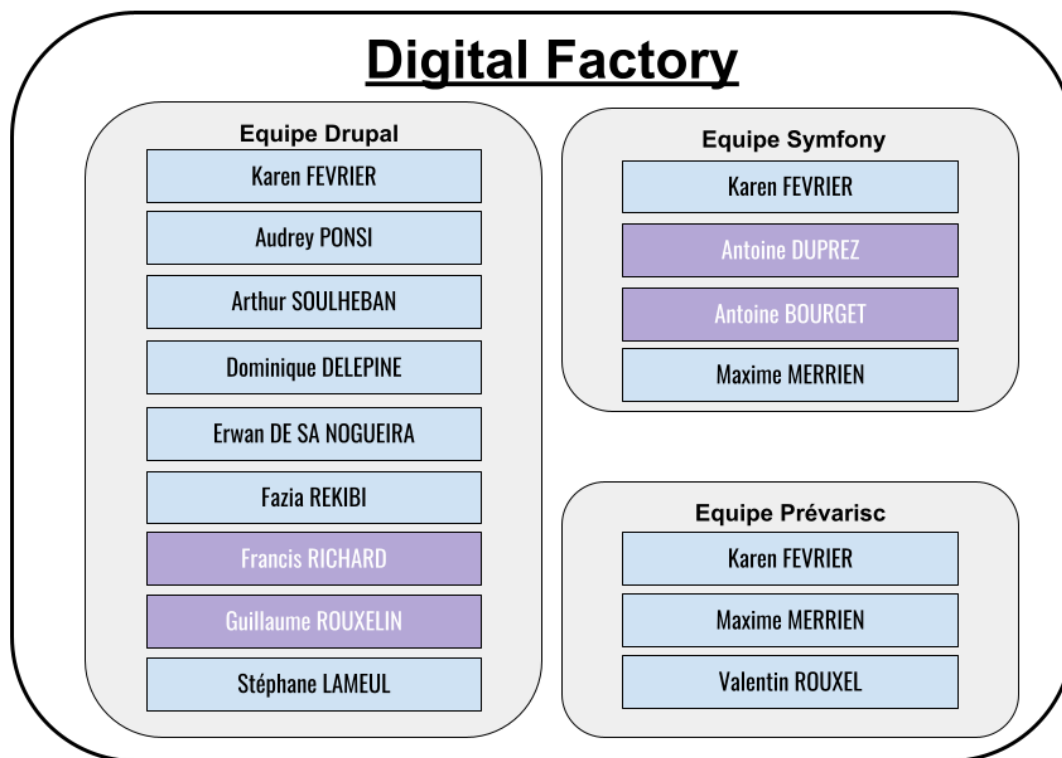


Figure-01, Organigramme de la composition de la Digital Factory - Établi par l'auteur

Nous pouvons exprimer avec le diagramme précédent que la DF(*Digital Factory*) est composée de 3 sous-équipes, une orientée vers le développement Drupal, une autre orientée vers le développement Symfony, puis la dernière orientée vers le projet prévarisc en Php. Nous pouvons remarquer que Karen FEVRIER est présente dans toutes les équipes, ceci est dû au fait qu'elle est responsable de la DF.

2.2) Contexte de la mission

2.2.1) Le projet PREVARISC

Lors du début de mon alternance, j'ai dû intégrer le projet PREVARISC. Le projet PREVARISC est un projet qui a vu sa première version sortir courant 2012, ce dernier est une refonte complète d'une précédente application nommée ERPv2. Initialement, PREVARISC fut conçu et développé par les pompiers du SDIS-62 (*Service Départemental d'Incendie et de Secours du Pas-de-calais*), dans ce même élan l'application fut créée de manière open-source, c'est-à-dire que n'importe qui peut avoir accès au code source ainsi qu'aux mises à jour de ce dernier.

PREVARISC est un outil de gestion des risques utilisé par les Services Départementaux d'Incendie et de Secours (*SDIS*) et de Secours pour planifier et suivre les activités de prévention des risques dans les établissements recevant du public. Les établissements concernés incluent les salles de spectacles, les chapiteaux, les centres commerciaux et les autres bâtiments accueillant du public.

Le suivi de prévention des établissements comprend des visites d'inspections et des études de dossier. Les dossiers sont ensuite examinés en commission, où un avis est donné sur chaque établissement (*favorable ou défavorable*). Cet avis, ainsi que les prescriptions liées, sont ensuite transmis à l'établissement et à l'administration (*commune, EPCI, département, ...*) sous forme de procès-verbal.

PREVARISC facilite la gestion de ces processus en permettant la référence des établissements dans le périmètre géographique, le suivi de l'état d'avancement des différents dossiers, la gestion du calendrier des commissions et de tous les documents liés tels que l'ordre du jour et les convocations.

L'un des grands avantages de PREVARISC est sa capacité à générer automatiquement les différents documents des dossiers tels que les procès-verbaux et les avis de commission. Cette fonctionnalité est possible grâce aux informations saisies pendant l'étude, ce qui permet d'éviter les erreurs de saisie et de gagner du temps lors de la rédaction des documents.

Ce projet, libre de droits sous licence CECIL-B, ce qui correspond à l'équivalent en droit français de la licence MIT, a été poussé par le ministère en 2012. Actuellement, 23 SDIS utilisent ce projet. Nous pouvons voir ci-dessous une carte de la France exposant les départements faisant usage de ce projet entre 2017 et 2020.

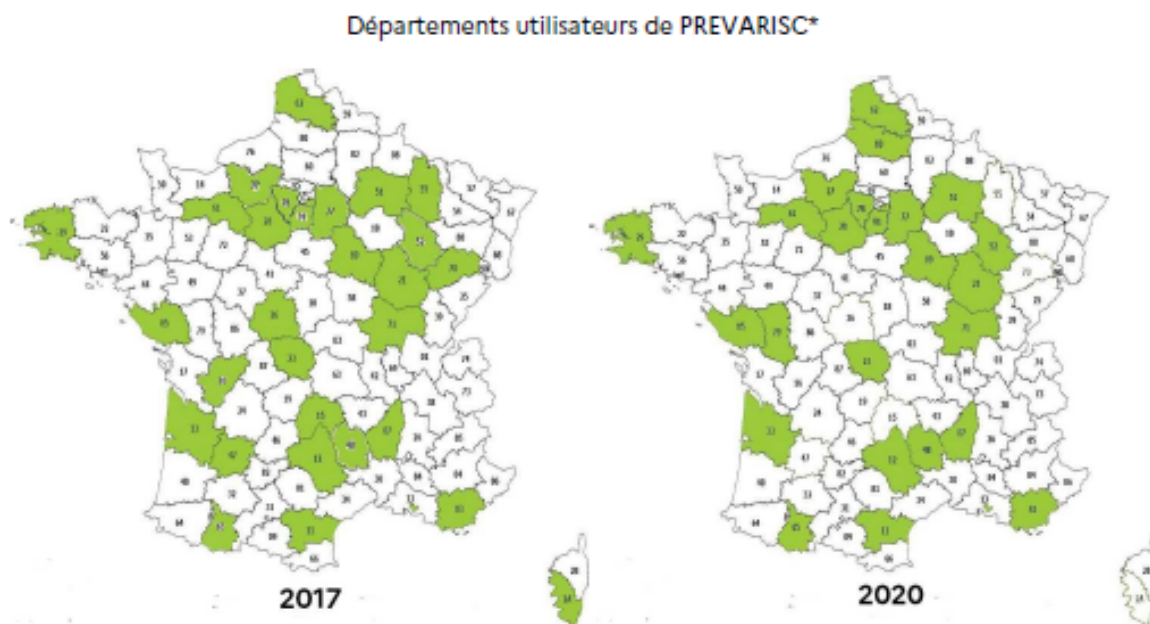


Figure-02, Carte de répartition de l'utilisation de prévarisc dans la France en 2017 et 2020 - Extrait de document interne

Le besoin d'une TMA (*Maintenance d'Application Tiers*) pour ce projet n'a pas été déclenché par un manque de mises à jour majeures, mais plutôt par une disparité dans les équipes informatiques des différents Services Départementaux d'Incendie et de Secours (*SDIS*). En effet, le travail de certains SDIS nécessitent des développements spécifiques pour leur système informatique, tandis que d'autres avaient besoin d'un support fonctionnel disponible. Le SDIS29 a été le premier à solliciter Atos pour effectuer une évolution sur son application PREVARISC, et en raison de l'utilisation de ce projet sur d'autres SDIS, Atos a créé une offre de TMA pour répondre aux besoins de tous les SDIS utilisant PREVARISC.

2.2.2) Le vocabulaire dans PREVARISC

Avant de parler plus du projet, il est important de définir différents terme afin d'éviter toute mauvaise compréhension, parmi les termes cibles, nous pouvons retrouver les termes suivants :

- **Etablissement:**

Un établissement est caractérisé par abus de langage un ou plusieurs bâtiments ; ou aire ou une zone d'étude.

- **Site:**

Un site est un regroupement d'établissements

- **Cellule:**

Les cellules sont des établissements disponibles dans un établissement de type "centre commercial".

- **Genre:**

Un genre d'établissement est la catégorisation d'un établissement, cette catégorisation ne fait pas forcément référence à une réglementation. Les libellés correspondants aux genres sont Site, Établissement, Cellule, Habitation, IGH, EIC, Camping, Manifestation Temporaire, IOP et Zone.

- **Dossier**

Un dossier est un événement qui sera tracé tout le long de la vie d'un établissement. Nous pouvons retrouver comme type de dossier les visites périodiques, permis d'aménagement etc... . Par ailleurs, un dossier peut être composé de plusieurs pièces jointes.

- **Commission**

Une commission est une instance qui a pour mission d'émettre un avis technique sur un projet de construction ou d'aménagement. Elle est composée de différents acteurs tels que des experts, des représentants de l'administration, des associations ou encore des élus.

Ainsi avec les termes définis précédemment, nous pouvons établir le schéma suivant :

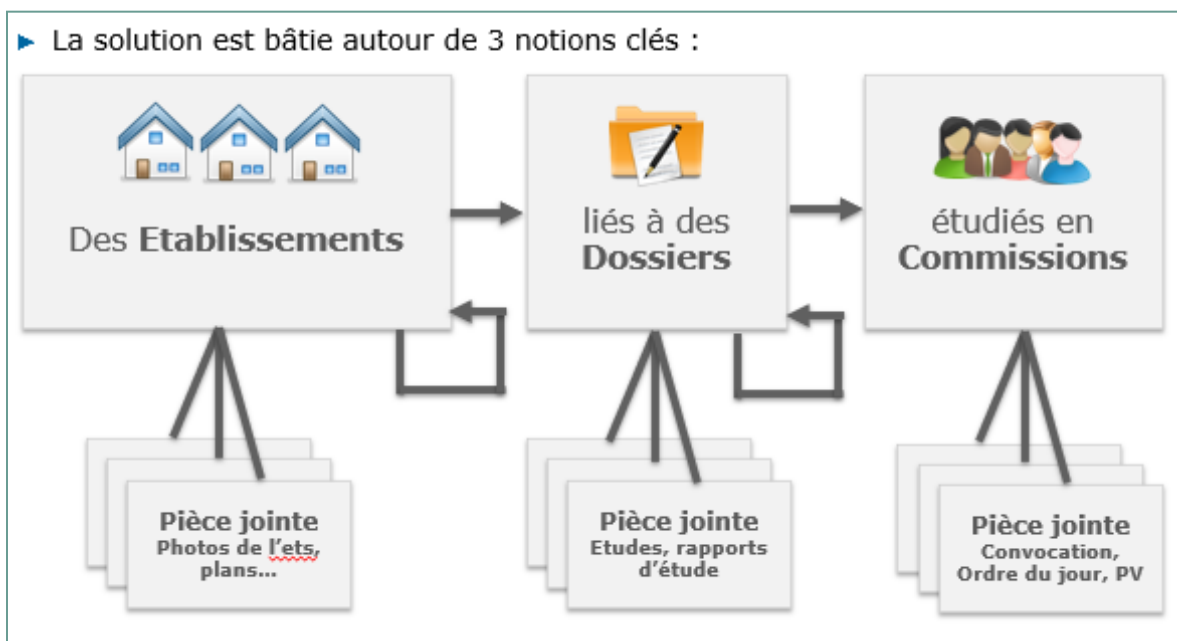


Figure-03. Schéma de fonctionnement interne de l'application prévarisc - Extrait du manuel d'utilisation de l'application prévarisc

2.2.3) L'architecture de PREVARISC

2.2.3.1 Les technologies utilisées

PREVARISC est un projet qui fut créé en 2012 avec des technologies datant de cette même époque. Nous pouvons donc retrouver comme technologie Zend 1.12 pour la gestion des données et des pages du site web. Une partie de test est disponible sur le projet, cette dernière est réalisée en Java.

Ce projet est multi-plateforme, ce qui signifie qu'il est disponible sur plusieurs systèmes d'exploitations, en l'occurrence l'OS Centos et Windows 7 & 10. En ce qui concerne la migration de données, nous avons utilisé l'outil Talend afin de pouvoir effectuer ce type d'opération plus facilement. Pour finir, nous utilisons des scripts SQL pour pouvoir effectuer des éventuelles migrations de base de données si une évolution client l'exige.

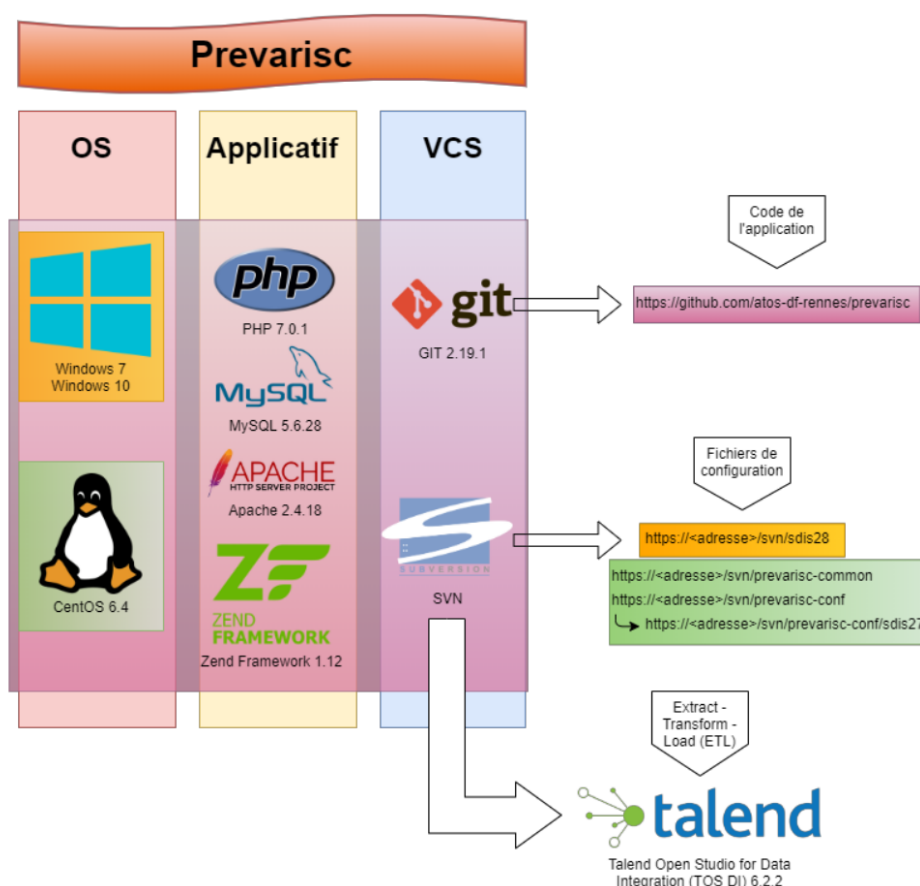


Figure-04, Schéma d'architecture partiel de l'application PREVARISC - Extrait de mémoire de fin d'étude de Maxime MERRIEN année 2019-2020

2.2.3.1 Les difficultés rencontrées sur l'architecture

Lors de l'assimilation du projet, j'ai pu rencontrer trois grosses difficultés.

La première étant l'apprentissage d'un nouveau framework sous l'une de ses premières versions. La difficulté n'était pas tant le fait d'apprendre une nouvelle technologie, mais que la ressource d'information envers cette dernière était modeste, du fait que la version du framework utilisé n'était pas mise à jour et ne pouvait pas être actualisée. Ajoutez à cela une communauté qui actuellement n'est quasiment plus existante du fait que tous les utilisateurs de cette technologie utilisent la dernière version de celle-ci, ceci rendait par conséquent les recherches pour l'apprentissage et la résolution de problèmes beaucoup plus complexes.

La seconde difficulté que j'ai rencontré est le multi versionning du projet. Dans les cas généraux, le produit a une version lors de la mise en production et ainsi la version du produit reçue chez les clients est la même. Cependant lors de l'historique du projet, les versions de produits chez le client ont pu être différentes du fait que des évolutions n'étaient pas livrées de manière globale ou que certains SDIS voulaient une spécificité qui a pu engendrer le fait d'avoir un client sous une version 2.5.40 et un autre client sous une version 2.5.42.

Ce décalage est dû à de potentielles évolutions antérieures. Ces décalages ont pu alors mener à des confusions lors de la résolution de différentes anomalies du projet par la difficulté à trouver la bonne version à corriger.

La troisième problématique identifiée est liée à un manque de rigueur dans l'architecture du code, résultant d'évolutions antérieures. Durant ma participation à ce projet, j'ai été confronté à des difficultés sur certains segments de code qui ne suivaient pas les conventions de développement ni les pratiques actuelles. Il aurait été judicieux de les retravailler, mais cela aurait exigé une quantité considérable de temps, ce qui n'était pas faisable pour respecter les délais prévus. Parmi ces manques de rigueur de codes, nous pouvons retrouver des redondances de code, des mauvaises utilisations du framework le court-circuitant, mauvaise utilisation ou partielle de l'architecture MVCS (*Model View Controller Service*) entraînant donc une réduction des capacités de production ainsi que d'assimilation des différentes parties métiers de l'application.

2.3) Les méthodes de travail mise en place

2.3.1) Une semaine type

Lors de mon alternance, j'ai dû suivre le rythme de 3 jours en entreprise (*lundi, mardi et mercredi*) puis 2 jours à l'école (*jeudi et vendredi*).

De manière générale, les semaines en entreprise suivaient ce chemin : le lundi matin j'essayais d'arriver un peu plus tôt que prévu pour me replonger sur les dernières évolutions en cours, dernières anomalies à résoudre, et lire tous les mails que j'ai pu recevoir durant le lapse de temps où je n'étais pas présent en entreprise. Hormis le lundi qui avait cette particularité, les jours suivaient la routine journalière la suivante: le matin 09h15 nous faisons un daily avec l'équipe : chaque membre de l'équipe rend compte de ce qu'il a fait, et de ce qu'il doit faire maintenant, ainsi que des difficultés rencontrées ou qu'il a peur de rencontrer sur de futur évolutions ou anomalies. Le reste de la journée nous développons, puis arrivés en fin de journée je transmettais à Maxime MERRIEN les différentes difficultés que j'avais pu rencontrer ainsi que les différentes fonctionnalités que j'ai pu terminer.

2.3.2) Intégration des méthodes de travail opérationnelles

Lors de mon arrivée sur le projet, il a fallu que je prenne le pli des différents automatismes déjà mis en place dans l'équipe afin d'être suffisamment efficace. Ceci est donc passé par les apprentissages suivants: la connaissance du processus de traitement des tickets, l'utilisation de l'outil redmine et pour finir les différents procédés de communication entre les différents tiers (*clients, membres d'équipe, collaborateur interne*).

2.3.2.1) Apprentissage du processus de traitement des tickets

Lors de la collaboration avec les différents tiers de ce projet, j'ai dû apprendre à lire et émettre des "tickets". Un ticket en informatique est un élément permettant de réaliser un suivi des demandes entre utilisateurs et développeurs. Celles-ci peuvent être de différentes natures telles que la demande de résolution d'un bug de l'application, l'interrogation d'un utilisateur suite à une mauvaise application du produit ou encore la demande d'une évolution du produit actuel. Par ailleurs, le ticket peut contenir différents vecteurs de communication comme l'urgence du ticket, l'avancement, le type du ticket etc...

Le type d'un ticket définit quelle est la nature de la demande. Dans le projet PREVARISC nous avons 3 grands types de tickets qui sont : les évolutions, les assistances et pour finir les anomalies.

Les tickets d'évolutions consistent à tenir le suivi des évolutions demandées par le client.

Les tickets d'assistance qui ont pour rôle de suivre une demande ou questionnement de l'application entre l'utilisateur et un représentant de l'équipe de développement puis pour finir les tickets de type anomalie qui permettent de suivre la résolution d'un bug de l'application ou même d'un blocage suite à ce dernier.

L'urgence d'un ticket permet de définir la criticité de ce dernier. Par exemple, un ticket de type anomalie créé pour le fait qu'un libellé d'un champ de formulaire ait une faute d'orthographe va être d'une urgence dite basse, car cela peut provoquer une gêne à l'utilisateur, cependant cela ne va pas en rendre bloquant l'application.

Dans le cas contraire, nous pouvons avoir un ticket de type anomalie qui remonte le fait que les utilisateurs n'arrivent pas à générer des documents. Ce ticket va être par conséquent d'une urgence maximale dénommée par le terme "BLOQUANT" du fait que l'anomalie bloque totalement l'une des utilisations métier de l'application, qui là est la génération de documents. Dans ce cas-là, le second ticket sera pris en charge avant le premier car leur urgence est nettement différente.

Dans la gestion de ticket de ce projet, les anomalies sont catégorisées en 3 groupes comme le montre l'extrait de contrat suivant :

► Les anomalies sont qualifiées par Niveaux de sévérité :

Niveaux de sévérité des anomalies	
Bloquante	Une anomalie est dite bloquante, si elle empêche l'utilisation de la solution pour la production sur un cas d'usage critique du service et qu'il n'existe aucune correction ou solution de contournement.
Majeure	Une anomalie est dite non bloquante lorsqu'il existe au moins une solution de contournement ou lorsque les conséquences du défaut ne bloquent pas l'utilisation de la solution. L'utilisation de la solution est toutefois dégradée.
Mineure	Une anomalie est dite mineure lorsque les conséquences du défaut n'ont pas d'incidence sur l'utilisation du produit.

Figure-05, Tableau catégorisation des anomalies - Extrait de contrat

Par ailleurs, nous pouvons lire sur l'extrait de contrat ci-suivant, les différents délais de remise en service de l'application selon la gravité des anomalies. Nous pouvons y remarquer que lorsque l'anomalie est réceptionnée est déterminée comme **bloquante**, l'équipe de développement a 2 jours pour parvenir à résoudre le ticket. Toutefois, si l'anomalie est catégorisée comme étant **majeure**, alors le délai de résolution de l'anomalie est de 5 jours. Nous remarquons ici que plus la gravité est élevée, plus le délai de réalisation de solution est court.

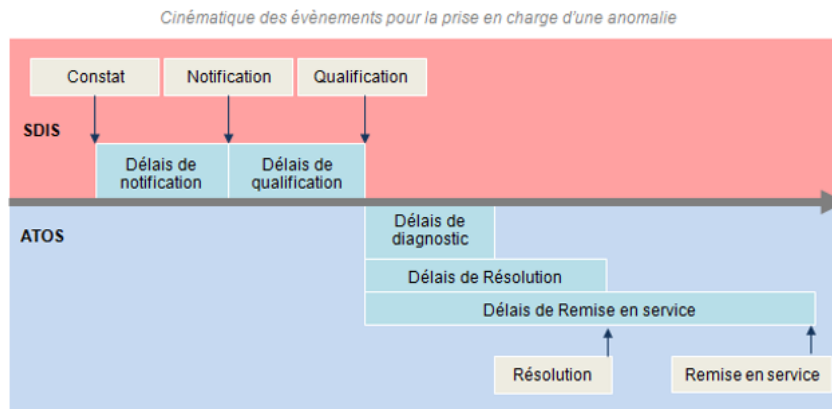
► Atos s'engage sur niveaux de services suivants, en heures et jours ouvrés :

ID	Sévérité / Délais	Délais de Remise en service
1	Délai de remise en service d'une anomalie bloquante	2j
2	Délai de remise en service d'une anomalie majeure	5j
3	Délai de prise en charge d'une demande de support	1j
4	Délai de livraison du compte-rendu de Comité de Pilotage	3j

Figure-06, Tableau sévérité et délais de remise en service - Extrait de contrat

Après avoir défini ce qu'était un ticket, nous pouvons expliquer comment ce dernier s'intègre dans le processus de développement. Nous allons voir ci-dessous un processus de résolution d'une anomalie suite à la création d'un ticket client, à noter que le schéma suivant est un extrait de contrat

► Cinématique de la prise en charge d'une anomalie :



- La **qualification** correspond à la vérification par le SDIS de l'anomalie sur l'environnement où l'anomalie est constatée, puis l'enregistrement d'une anomalie dans l'outil des demandes Atos ;
 - La **résolution** correspond à la fourniture d'un correctif à chaud ou un moyen de contournement ;
 - La **remise en service** correspond à la livraison d'un applicatif packagé et corrigé sur Redmine.
- Une demande n'est considérée comme anomalie qu'après sa qualification.

Figure-07, Schéma processus de prise en charge d'un ticket - Extrait de formation à l'application prevarisc

Nous voyons dans le processus que suite à la remontée d'un constat d'anomalie et de sa qualification, le client crée son ticket. Dans un second temps, les développeurs du côté d'Atos reçoivent ce ticket puis changent le statut du ticket à "en cours". Dans un troisième temps, les développeurs réalisent un diagnostic où ils essaient de reproduire l'anomalie décrite par le client. Ensuite, s'ensuit une phase de résolution où les développeurs vont corriger le bug en modifiant le code de l'application. Enfin, se réalise la phase de remise en service où les développeurs envoient une mise à jour aux clients permettant de corriger le bug remonté précédemment.

2.3.2.2) Apprentissage de l'utilisation de l'outil redmine

Après avoir défini comment les tickets étaient utilisés, nous allons maintenant aborder l'outil de ticketing redmine. J'ai été amené à utiliser Redmine afin de recevoir des tickets de la part des clients ainsi que des autres collaborateurs. Redmine est un logiciel de gestion de projets open-source qui permet aux équipes de gérer leurs projets en utilisant une variété de fonctionnalités. Il est souvent utilisé pour la gestion de projets informatiques, mais il peut également être utilisé pour la gestion de projets dans d'autres domaines.

Nous pouvons voir ci-après le dashboard d'accueil que nous pouvons avoir avec Redmine sur le projet prevarisc.

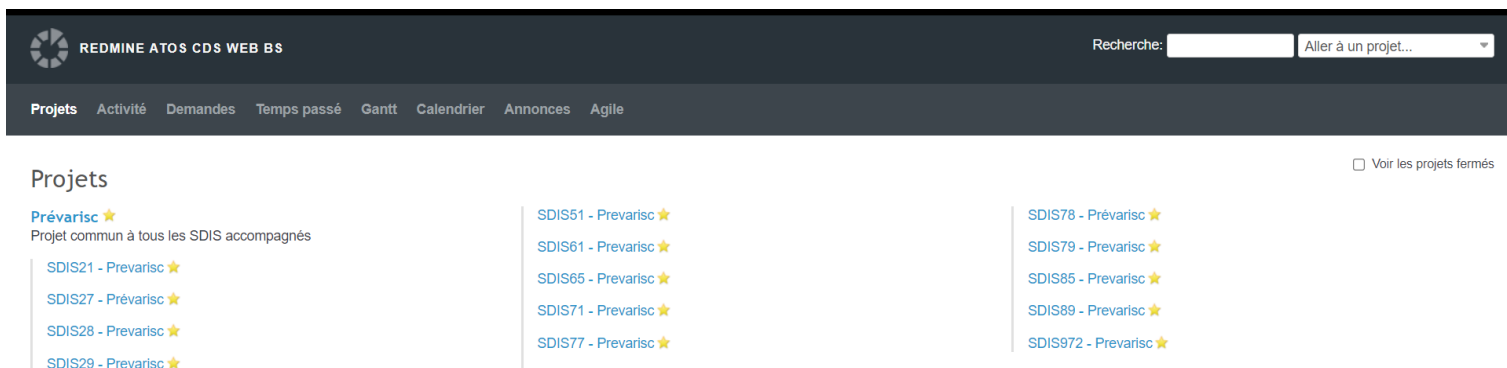


Figure-08, Capture du dashboard d'accueil de redmine- Etabli par l'auteur

Nous remarquons sur cette page, la liste des versions du projet prevarisc. Nous pouvons y visualiser un nombre conséquent de versions avec tous les SDIS pris en charge. Lorsqu'on clique sur l'un des liens d'un SDIS, nous tombons sur la liste des tickets relatifs à ce dernier, comme ci-contre :

<input type="checkbox"/>	#	Projet	Tracker	Statut	Priorité	Sujet	Auteur	Assigné à	Mis-à-jour	Version cible
<input type="checkbox"/>	4821	SDIS29 - Prevarisc	Anomalie	Résolu	Majeure	Télémaintenance		Maxime MERRIEN	07/02/2023 10:25	2.5.30
<input type="checkbox"/>	4820	SDIS29 - Prevarisc	Anomalie	Résolu	Majeure	Sauvegarde		Maxime MERRIEN	07/02/2023 10:25	2.5.30
<input type="checkbox"/>	4819	SDIS29 - Prevarisc	Anomalie	Résolu	Majeure	Dump base de PROD Prevarisc		Maxime MERRIEN	07/02/2023 10:25	
<input type="checkbox"/>	4818	SDIS29 - Prevarisc	Anomalie	Résolu	Majeure	Génération de documents		Maxime MERRIEN	07/02/2023 08:58	2.5.30
<input type="checkbox"/>	4817	SDIS29 - Prevarisc	Anomalie	Résolu	Majeure	Génération - Ordres du jour		Maxime MERRIEN	07/02/2023 10:29	2.5.29

Figure-09, Capture d'une liste de ticket redmine- Établi par l'auteur

Sur cet écran, nous visualisons que chaque ligne correspond à un ticket avec un demandeur, une personne à qui est assignée le ticket, ainsi que d'autres informations comme citées lors du point précédent.

Par ailleurs, si nous cliquons sur la ligne d'un ticket, nous avons la page détaillant ce dernier. Sur cette page de détails nous y retrouvons toutes les informations complémentaires au ticket, comme une description brève de l'anomalie ou évolution, ainsi que des captures d'écran. De plus des points cités précédemment, nous pouvons y trouver un espace d'échange sous le format d'un tchat où des échanges textuels et de fichiers peuvent se réaliser entre le client et les développeurs.

2.3.2.3) Apprentissage des procédés de communication entre les tiers

Lors de la collaboration de ce projet, j'ai dû apprendre à adapter ma communication selon le contexte et le destinataire. J'ai dû apprendre les processus qui étaient mis en place dans la société afin de répondre aux exigences clients et internes. Les différents processus de communication sont les suivants, si l'on souhaite communiquer avec un collaborateur d'une même équipe alors nous communiquons soit par tchat Microsoft Teams ou bien directement en physique.

Si le destinataire est un client, alors le vecteur de communication va dépendre du contexte.

Si nous devons communiquer avec le client, sur la planification d'un rendez-vous, alors nous passerons par la rédaction d'un email à ce dernier. Sinon, si le contexte concerne le suivi d'un ticket de maintenance ou d'évolution, alors nous passerons par une communication directe établie dans redmine envers ce même ticket.

Pour terminer, dans le cas où nous devons communiquer avec le client, ou que celui-ci souhaite communiquer pour une urgence, alors la communication se réalise directement via un appel téléphonique ou une télé-maintenance si le besoin en est.

Pour terminer avec les différents processus de communication entre les tiers, lorsque nous devons poser des questions, ou des demandes au support technique, que ce soit au niveau sécurité ou maintenance des équipements, nous envoyons dans un premiers temps un ticket au support technique puis dans certains cas des emails.

2.3.3) Intégration des méthodes de travail sur github

2.3.3.1) Qu'est ce que git et github ?

La réalisation d'évolution ou de maintenance de l'application PREVARISC engendre la création de différentes versions de ce dernier. Une gestion manuelle des différentes versions de l'application peut s'avérer longue, fastidieuse et source d'erreur. C'est pourquoi, nous utilisons les outils suivants : Git et GitHub.

Git est un outil permettant de gérer différentes versions d'un projet de manière décentralisée. L'outil va permettre de travailler sur plusieurs versions du projet en parallèle en créant des "branches". Chaque branche peut être créée afin de regrouper une à plusieurs fonctionnalités ciblées ou des résolutions de bugs. A noter que chaque branche peut avoir différents états d'avancements, chacun d'eux étant ce qu'on appelle des commits. Chaque commit contient les changements d'un ou plusieurs fichiers, ainsi qu'un nom pour pouvoir étiqueter le changement produit.

Après avoir abordé la notion de l'outil Git, nous pouvons définir ce qu'est Github. Il s'agit d'une plateforme de développement collaboratif utilisant Git pour la gestion de versions de projets. Sur Github, nous pouvons créer des projets et faire collaborer différentes personnes.

2.3.3.2) Nomenclature de branch appliqué

Pour chaque évolution effectuée ou résolution de bug du projet, nous créons une branche. Chaque branche devait être nommée de manière à suivre une nomenclature spécifique afin que l'on puisse s'y retrouver rapidement et savoir assez aisément à quoi chaque branche faisait référence. Les branches devaient suivre la nomenclature suivante:

si c'est une évolution alors elle devrait se nommer comme :

`feature/numeroTicket-nom-explicite-evolution`

si c'est une résolution de bug alors elle devrait se nommer comme :

`bugfix/numeroTicket-nom-explicite-du-bug`

2.3.3.3) Mise en place d'une intégration continue

Suite à mon arrivée au sein de PREVARISC, nous avons décidé de mettre en place une CI (*intégration continue*). Une intégration continue consiste au fait de tester de manière automatique du code envoyé par le ou les développeurs. Ces tests peuvent être des tests de qualité de code ainsi que fonctionnels. Nous avons décidé de mettre cette CI en place pour pouvoir garder une certaine qualité de code ainsi qu'une certaine rigueur.

Par ailleurs, du fait d'avoir implémenté une CI, nous avons ajouté des règles sur le projet github, nous avons ajouté la règle suivante que chaque évolution ou résolution de bug doit passer par une MR (*Merge Request*). Une MR consiste à créer une relecture avant une fusion de code. Cette relecture est nécessaire pour plusieurs raisons, la première étant de voir si la

philosophie du code respecte celle de l'architecture du code déjà en place, voir s'il est compréhensible des autres développeurs.

2.4) Les missions qu'on m'a attribuées

Lors de mon arrivée chez PREVARISC, j'ai été initialement assigné à des missions de difficultés relativement faibles. Cette approche a été mise en place pour m'aider à assimiler les premiers paradigmes de la technologie et à me familiariser avec les pratiques et les procédures de l'entreprise. Grâce à cette approche graduelle, j'ai pu acquérir de solides bases et une bonne compréhension des enjeux du projet. Au fil du temps, on m'a confié des tâches plus complexes et de plus grande envergure, qui m'ont permis de relever de nouveaux défis et d'acquérir des compétences supplémentaires. J'ai acquis plus de responsabilités, ce qui m'a permis de contribuer davantage au projet et de faire progresser mes compétences professionnelles. Dans ce rapport et les lignes qui vont suivre, je vais vous parler de deux évolutions que j'ai réalisées, qui à mon sens sont deux évolutions majeures de l'application qui m'ont permis d'accentuer mes compétences en développement.

2.5) Préambule des évolutions, explication des champs de fusions

Avant de parler des deux évolutions suivantes, il est important de comprendre la notion qui tourne autour des champs de fusion de l'application. On rappelle que l'une des utilités phares de l'outil est le fait de générer des documents de manière dynamique par un simple clic. Pour comprendre convenablement quels sont les axes de génération de ces documents, nous allons dans un premier temps définir qu'est ce qu'un champ de fusion puis dans un second temps ce qu'est un squelette de fusion.

2.5.1) Qu'est ce qu'un champ de fusion ?

Un champ de fusion est un champ qui va pouvoir à la fois, avoir une ou plusieurs valeurs de saisie du côté utilisateur par un formulaire. Ce même champ va pouvoir transmettre ces valeurs dans un document auto-généré par l'application.

Un cas concret d'utilisation des champs de fusion est le nom d'un établissement.

Les utilisateurs peuvent renseigner le champ "libellé établissement" dans un squelette de fusion qui une fois injecté dans l'application va pouvoir mapper les valeurs de la base de données correspondant au bon établissement aux champs renseignés, dans notre cas-là, ce sera le nom de l'établissement.

2.5.2) Qu'est ce qu'un squelette de fusion ?

Un squelette de fusion est un fichier permettant d'auto-générer des documents "odt" en se basant sur des valeurs de champs de fusion précédemment renseignées. Reprenons le cas précédent, quand l'utilisateur saisit le nom de l'établissement, l'information est enregistrée en base de données.

Dans un second temps, ce dernier va renseigner le squelette de notre document auto-généré en inscrivant dans le fichier "{libellé établissement}" faisant référence au champ nom établissement. A noter qu'un squelette de fusion une fois renseigné, sert de patron à tout un type de procès verbal et qu'importe les établissements sujets à la génération des documents.

Après la création du squelette et la saisie des informations, l'utilisateur peut générer le document. L'application remplace alors les champs de fusion par les informations correspondantes c'est à dire que l'application va remplacer "{libellé établissement}" par la valeur de l'établissement en question, donc si notre établissement se nommait "Boulangerie de la croix" le champ "libellé établissement" sera remplacé par "Boulangerie de la croix".

A noter que dans notre exemple, nous n'avons mis qu'un seul champ de fusion en évidence, mais qu'il est tout à fait envisageable d'en mettre plusieurs. De plus, il est tout à fait possible de boucler sur des listes de valeurs dans notre champ de fusion.

2.6) L'évolution "champ parent"

2.6.1) Contexte de l'évolution "champ parent"

Initialement les champs des formulaires étaient sous la forme suivante :

Descriptifs

Descriptif technique

[TEST-RAPPORT](#)

TEST-RAPPORT

Champ1

Figure-10. Capture partielle d'un formulaire avec un champ - Etabli par l'auteur

Sur l'image précédente, un libellé ainsi qu'un champ permettent de saisir la valeur voulue. Les champs peuvent être de différents type comme nombre, texte, paragraphe, un élément de liste ou checkbox, dans ces cas là le champ de saisie (*input*) ressemblera à l'image comme suivant :

Champ Texte	<input type="text"/>
Champ Liste	<input type="text" value="Sélectionnez un élément"/>
Champ Numérique	<input type="text"/>
Champ Checkbox	<input type="checkbox"/>

Figure-11, Capture des différents types de champ - Établi par l'auteur

Toutefois, le fait d'avoir qu'un seul et unique champ par ligne peut s'avouer être contraignant. Notamment, lorsqu'on cherche à vouloir afficher plusieurs valeurs de champs liés, cela prend plus de place, les noms des champs se ressemblent et l'expérience utilisateur n'est pas optimale. C'est pourquoi, le client (*SDIS29*) nous a demandé de pouvoir améliorer les cas de figures cités précédemment, avec la possibilité de créer n'importe quel champ comme il le souhaitait.

Ainsi, nous avons pensé à créer un champ "parent" qui pourrait être composé de plusieurs champs. De ce fait sur une même ligne, qu'ils puissent avoir plusieurs champs, puis d'une autre part, la possibilité d'avoir sur une ligne des champs de différents types comme ci-suivant :

Surface	nombre d'unité	unité
	<input type="text" value="15"/>	<input type="text" value="hectare"/>
		<div>Sélectionnez un élément</div> <div>hectare</div> <div>mètre carré</div>

Figure-12, Capture d'un champ parent- Établi par l'auteur

Nous pouvons remarquer sur notre exemple ci-précédent, un champ parent “Surface” étant composé de deux champs fils “nombre d’unité” et “unité”. Le premier champ fils est un champ de type numérique et le second est de type liste.

De plus, les clients souhaitaient pouvoir, depuis un panel d’administration, éditer la composition des champs “parents”. Ces champs devaient pouvoir transmettre leurs valeurs dans les champs de fusion afin de pouvoir intégrer les valeurs des champs dans les document auto-générés.

2.6.2) Conception de l’évolution “champ parent”

Lors de la réception de la demande de cette évolution, nous nous sommes premièrement penchés sur la partie conception de celle-ci. Nous avons donc extrait les besoins fonctionnels du client. Parmi celles-ci, nous avons extrait le fait que l’utilisateur avait comme premier besoin celui de pouvoir créer des champs à sa guise qu’importe la structure. Ce besoin nous a confortés dans le choix de faire étendre du code des entités champs déjà existants pour garder les comportements de base utilisés par le client et ajouter les fonctionnalités attendues par le client.

Une fois cette première approche faite, nous avons découpé la conception en deux parties. La première sur le développement du champ parent, puis la seconde sur l’extension de génération des champs de fusion. En ce qui concerne la partie conception COO (*Conception Orientée Objet*) nous avons décidé d’y consacrer plusieurs jours afin de concevoir au mieux avec le code existant l’évolution demandée par le client. Nous avons donc conçu le class diagram suivant pour répondre aux besoins demandés.

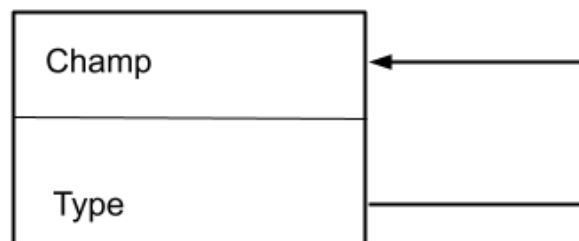


Figure-13, Diagramme de class - Établi par l’auteur

Ci-dessus, nous pouvons identifier une class Champ, nous y remarquons une flèche qui témoigne de l’auto-liaison que la class a d’elle envers elle même, cela veut dire que cette class Champ peut être composée de “zéro à N” autres champs. Par ailleurs, nous pouvons remarquer que la class est composée d’un Type, ce type indique si le champ est un champ qui a renseigné un nombre, un texte, un paragraphe, une liste, une checkbox etc ...

2.6.3) Intégration de l'évolution "champ parent"

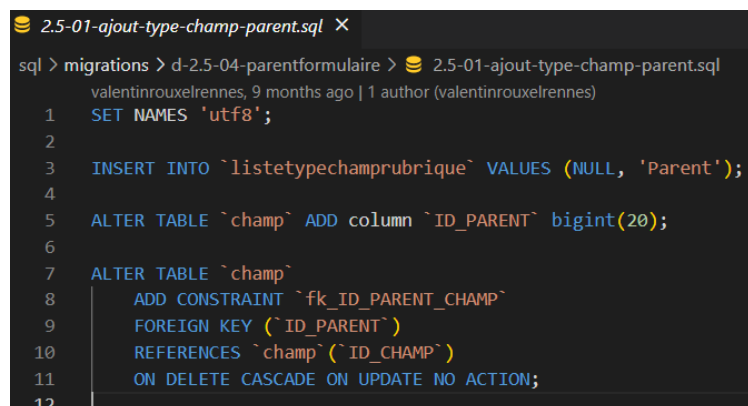
Après la réalisation de la conception de cette évolution, nous avons donc dû passer par la phase d'intégration de cette dernière. Cette intégration est découpée en plusieurs parties que nous verrons dans l'ordre suivant : l'implémentation de la partie backend, la réalisation de la partie administrateur, la réalisation de la partie utilisateur et la génération des documents.

2.6.3.1) Implémentation de la partie backend

Pour la réalisation de cette évolution, il a fallu dans un premier temps réaliser la partie serveur du code. Cette partie, plus communément appelée "backend" consiste à réaliser la communication et les traitements de données entre l'utilisateur et la base de données. Pour cette partie, nous avons transposé le class diagram exposé précédemment dans le point 2.6.2) en code. Par ailleurs, lors du point précédent, nous nous sommes basés sur l'architecture MVCS (*Model Vue Controller Service*) pour réaliser cette évolution. Cette architecture qui permet de séparer clairement les responsabilités entre les différentes couches de l'application, facilite la maintenance et l'évolution de celle-ci, et offre une grande flexibilité en permettant de remplacer facilement les différentes couches sans impacter les autres. En outre, la couche de services permet d'encapsuler la logique métier complexe, ce qui facilite la compréhension du code et la possibilité de pouvoir réutiliser des morceaux de code partout dans l'application.

Cette transposition s'est faite en plusieurs étapes.

Tout d'abord, par la création de l'entité champ "Parent" en y ajoutant les attributs et méthodes métier lui convenant. Une fois le modèle de l'entité créé dans la partie backend, nous avons dû produire un script de migration SQL permettant de créer les relations champ père et champ fils en base de données. Voici ci-contre un extrait du script de migration cité précédemment :



```

2.5-01-ajout-type-champ-parent.sql X
sql > migrations > d-2.5-04-parentformulaire > 2.5-01-ajout-type-champ-parent.sql
valentinrouxelrennes, 9 months ago | 1 author (valentinrouxelrennes)
1 SET NAMES 'utf8';
2
3 INSERT INTO `listetypechamprubrique` VALUES (NULL, 'Parent');
4
5 ALTER TABLE `champ` ADD column `ID_PARENT` bigint(20);
6
7 ALTER TABLE `champ`
8   ADD CONSTRAINT `fk_ID_PARENT_CHAMP`
9   FOREIGN KEY (`ID_PARENT`)
10  REFERENCES `parent` (`ID_PARENT`)
11  ON DELETE CASCADE ON UPDATE NO ACTION;
12

```

Figure-14, Capture d'un script de migration- Établi par l'auteur

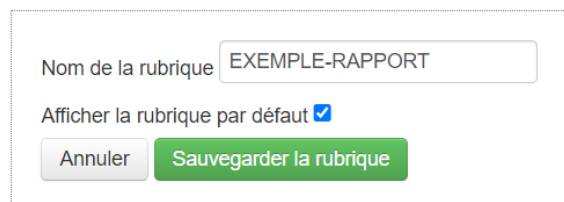
Une fois la réalisation du modèle et du script de migration, nous avons dû réaliser les deux parties de code suivantes, le controller et les services relatifs à l'entité. Le controller est une classe qui a pour rôle de gérer les différentes interactions entre l'utilisateur et l'application, dans notre cas, le controller va nous permettre de rendre notre page de formulaire ou même certains composants de la page. En ce qui concerne les services de l'entité, la réalisation de ces derniers permet d'externaliser la logique métier du controller afin de séparer les responsabilités de chaque partie de code. Ces services permettent ainsi d'interagir avec la base de données que ce soit pour y affecter des traitements ou y effectuer des lectures.

2.6.3.2) Réalisation de la partie administrateur

Suite à la conception des parties MCS, nous avons pu commencer à réaliser la partie V (*Vue*) en y réalisant les vues de cette évolution. Nous avons donc commencé par réaliser la partie administrateur de l'évolution.

Cette partie de l'application va permettre à un utilisateur de rôle administrateur de créer les champs parents accessibles dans les différents formulaires de l'application. Pour ce faire, nous avons intégré un formulaire de saisie dans lequel l'utilisateur peut taper le type de son champ, tout en ayant la possibilité de renseigner un type simple (*entier, texte, case à cocher etc...*). Dans le cas où l'utilisateur saisie un champ simple, alors il n'aura plus qu'à y renseigner son type, son nom, comme sur l'image suivante :

Informations sur la rubrique



Informations sur le champ

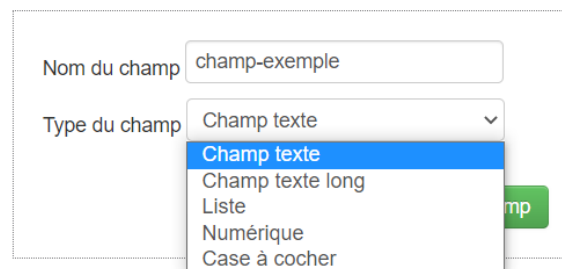


Figure-15, Capture d'un script de migration- Établi par l'auteur

Pour cette évolution, nous avons ajouté le type “parent” dans la liste des types. L’administrateur devra donc sélectionner ce type afin de créer un champ de type parent. Une fois ce type sélectionné, l’administrateur doit suivre le même procédé que lors de la création d’un champ classique.

Une fois le processus réalisé, l’administrateur devra dans un second temps alimenter le contenu du champ parent créé, ceci en remplissant un formulaire similaire au précédent. Pour ce faire, il devra cliquer sur l’icône crayon afin d’éditer le contenu du champ (*voir image ci-suivante*).

Informations sur la rubrique

Nom de la rubrique

Afficher la rubrique par défaut ☒

Informations sur le champ

Nom du champ

Type du champ

Liste des champs

	Nom du champ	Type du champ	
+	CHAMP-PARENT	Parent	

Figure-16. Capture de la page administrateur de création de champ- Établi par l’auteur

Puis, après avoir cliqué sur l’icône d’édition, l’utilisateur pourra saisir les champs fils du champ parent en remplissant au fur et à mesure le formulaire suivant comme lors de la saisie de champ classique. A l’exception près qu’un champ parent ne peut pas contenir de champ parent.

Informations sur la rubrique

TEST-RAPPORT (Affichée par défaut)

Informations sur le champ

Nom du champ

Type du champ

Informations sur le champ enfant

Nom du champ

Type du champ

Figure-17, Capture page administrateur, création de champ fils - Établi par l'auteur

2.6.3.3) Réalisation de la partie utilisateur

Après avoir réalisé la partie administrateur, nous avons réalisé la partie utilisateur. Cette partie était sujette à la réalisation de l'affichage des champs de type "Parent". Comme mentionné lors du point [2.6.1\)](#), les clients ont souhaité avoir la possibilité de remplir plusieurs champs sur une même ligne alors qu'initialement, il n'était pas possible de mettre un champ par ligne. Pour la réalisation de l'affichage des champs "Parents", nous avons bouclé sur la composition du champ parent de manière à afficher ceux qui le composent sur une et même ligne horizontale. De cette manière nous avons le rendu suivant :

Parent

Champ1	Champ2	Champ3	Champ4
<input type="text"/>	<input type="text" value="Sélectionnez un élément"/>	<input type="text"/>	<input type="checkbox"/>

Figure-18, Capture du rendu d'un champ parent - Établi par l'auteur

2.6.3.4) Génération des documents

Une fois la mise en place de la structure MVCS (*Model Vue Controller Service*) terminée, nous avons commencé à travailler sur la génération de documents contenant les champs de fusion liés aux champs parents.

Pour y parvenir, nous avons d'abord effectué une étape de reverse engineering (*analyse inverse de la logique*) du code métier destiné à la génération des documents. Une fois cette tâche réalisée, nous avons été en capacité d'effectuer les modifications nécessaires.

Pour ce faire, nous avons complété une partie de la logique métier déjà mise en place pour la génération des documents. Nous avons ainsi conservé l'ancienne logique déjà mise en place. Cela nous garantit donc de ne pas avoir de régression, c'est-à-dire éviter un comportement fonctionnel moins effectif que dans la précédente version.

Nous avons complété la logique de code antérieur en utilisant une architecture récursive, c'est-à-dire qu'un même bloc de code se répète lui-même selon une condition.

On y comprend donc que lorsque notre méthode passe sur un champ parent, la méthode va parcourir les champs fils et ainsi y affecter les valeurs adéquates comme si ces derniers étaient juste des champs simples faisant abstraction qu'ils soient champ fils.

2.6.4) Rétrospective de l'évolution "champ parent"

2.6.4.1) Le retour client

Une fois l'évolution livrée aux clients, ces derniers s'en sont directement servis. L'évolution leur a enfin permis de pouvoir mettre plusieurs champs sur une même ligne leur

permettant ainsi de réduire drastiquement la taille de certains de leurs formulaires. Du fait que les formulaires sont maintenant plus courts, cela leur permet de réduire indirectement le temps minimum nécessaire pour les remplir. Ainsi, de ce biais, un gain de temps est observé. Aujourd'hui, l'évolution est utilisée au quotidien par les SDIS utilisant l'application PREVARISC.

2.6.4.2) Points forts de l'équipe sur cette évolution

Lors de la réalisation de cette évolution, nous avons pu remarquer que l'équipe a réussi à fournir un bon travail et ce à travers trois points. Le premier est le niveau de la conception avancée fournie lors de la réalisation de la fonctionnalité, ceci par l'intégration d'un design pattern et d'adaptation de code avec le code déjà existant. Le second point sur lequel l'équipe a performé est la réalisation d'une interface homme machine (IHM) suffisamment ergonomique afin d'obtenir une bonne UX (*expérience utilisateur*). Un point sur lequel l'équipe a bien travaillé est la réalisation de démonstration client. Ces dernières servent à montrer au client où en est l'application et la manière d'interagir avec les évolutions, ces échanges servent par ailleurs à remonter des remarques clients qui peuvent être le souhait d'un changement de couleur jusqu'à la spécification de l'évolution sur l'utilisation de cette dernière. Le fait d'avoir bien réalisé ces démonstrations et échanges nous ont permis d'éviter des anomalies, d'améliorer l'UX et surtout d'éviter des aller-retours avec le client pour des mauvaises compréhensions du besoin.

Pour terminer, un point non négligeable, est que l'évolution n'a pour le moment vécue aucune anomalie. Cela illustre donc de la robustesse et de la qualité apportée à cette dernière.

2.6.4.3) Difficultés rencontrées de cette évolution

Pendant la réalisation de cette évolution, l'équipe n'a pas éprouvée de difficultés majeures ou inhabituelles. Effectivement, nous avons eu quelques contraintes techniques qui nous ont amené à effectuer une réflexion technique approfondie mais nous ne sommes pas restés bloqués face à un problème plusieurs jours consécutifs sans savoir quoi faire ni comment faire.

2.7) L'évolution "champ tableau"

2.7.1) Contexte de l'évolution "champ tableau"

Après avoir effectué l'évolution précédente pour le SDIS29, ceux-ci nous ont demandé d'améliorer cette dernière de la manière suivante. Ils souhaitent créer un champ tableau où les colonnes seraient des champs à renseigner depuis le tableau de bord de l'administration.

Puis du côté utilisateur, recevoir un formulaire de la forme d'un tableau ré-ordonnable, dans lequel l'utilisateur peut ajouter et retirer des lignes. Le tout afin de pouvoir récupérer les valeurs du tableau pour pouvoir les injecter dans les champs de fusion de manière à pouvoir renseigner les valeurs saisies par l'utilisateur dans les documents auto-générés. Pour ce qui

concerne cette évolution, nous n'avons pas reçu de wireframe ou maquette des clients: nous avons donc dû créer en partant de rien la partie tableau.

2.7.2) Conception de l'évolution "champ tableau"

De la même façon que durant la précédente évolution, nous sommes passés par une phase de conception. Nous avons dans un premier temps pris du recul sur la demande d'évolution pour voir de manière plus critique qu'elles étaient les différences avec l'évolution précédente. Après réflexion, nous avons constaté que l'unique différence qui s'exerçait entre les deux évolutions était que l'évolution des champs parent n'ont qu'une valeur de renseigner à la fois pour chacun des champs fils, tandis que l'évolution des tableaux, eux, peuvent en avoir plusieurs. Ainsi, une des solutions qu'on a pensé et qu'on va appliquer est l'ajout d'une colonne d'index afin de pouvoir lister des valeurs des champs fils d'un champ tableau.

2.7.3) Intégration de l'évolution "champ tableau"

Après avoir réalisé une conception de l'évolution, nous sommes passés par la phase d'intégration de cette dernière. L'intégration était découpée en plusieurs parties que nous verrons dans l'ordre suivant : l'implémentation de l'entité champ tableau, la partie administrateur, la partie utilisateur et la génération des documents.

2.7.3.1) Implémentation de l'entité champ tableau

En ce qui concerne l'implémentation de l'entité tableau, nous avons décidé de prendre le même style d'architecture que pour le champ parent, c'est-à-dire, en utilisant le modèle MVCS. Par ailleurs, lors de la réalisation de l'implémentation de cette entité, nous avons décidé de faire étendre la logique des champs parents, ceci afin de composer les champs tableaux. Nous avons eu cette réflexion, car en soit, un champ tableau est un type de champ parent, car celui-ci va lui-même être composé de plusieurs champs fils qui eux composeront les colonnes, comme le champ parent qui lui est composé de plusieurs champs fils.

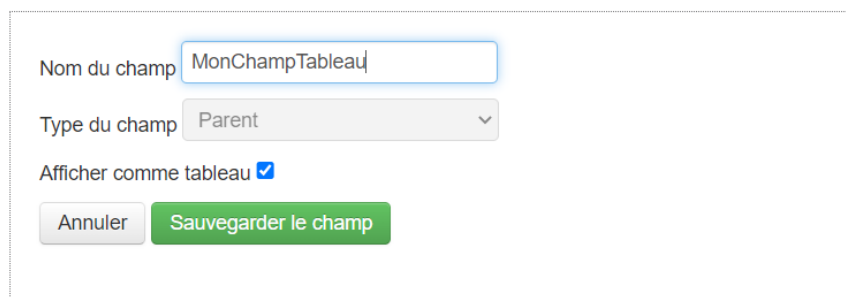
Néanmoins, la réalisation de cette implémentation nous a contraints à ajouter deux attributs en base de données. Le premier attribut ajouté était l'attribut "IS_TABLEAU" afin de distinguer les champs parents classiques des champs parent tableau. Ceci implique que si l'attribut "IS_TABLEAU" en base de données est égale à 1, alors l'élément cible est un champ parent tableau. Cependant, si la valeur de l'attribut est égale à "0" alors l'élément est un champ dit classique. Le second attribut qu'on a dû ajouter est l'attribut index, nous avons dû ajouter un attribut "INDEX" à la table "VALEUR" car, dans les cas précédents, chaque champ n'avaient qu'une et une seule valeur de rattachée. Or, dans le cas d'un champ tableau, les champs fils servant de colonnes ont possiblement plusieurs valeurs possibles de renseigner. Ainsi, en ajoutant un attribut index, nous pouvons organiser via une valeur d'index les différentes valeurs saisies dans le tableau. Par exemple, si lors de la récupération des valeurs d'un tableau, nous avons une valeur "A" avec un index de 1 et une valeur "B" avec un index de 8, alors on sait que la ligne correspondant à la valeur "B" sera en dessous de la ligne correspondant à la valeur "A".

Par ailleurs, de cette implémentation, nous pouvons remarquer que nous avons utilisé le principe d'héritage, c'est-à-dire faire étendre du code de différentes entités de manière à rendre du code commun et ne pas faire de duplication de code. Une fois la réalisation de l'entité faite, le côté back s'en est suivi comme lors de l'évolution précédente une phase de script de migration afin de faire migrer les bases de données existantes dans l'optique de pouvoir rendre disponible l'évolution sur les différentes machines clientes.

2.7.3.2) Réalisation de la partie administrateur

Une fois l'entité du champ tableau implémentée, il a fallu qu'on réalise la partie administrateur de cette évolution. Pour ce faire, nous sommes partis de la même base que l'évolution précédente des champs parents. Nous avons repris la même structure visuelle, à l'exception près, que l'administrateur doit, s'il le souhaite, cocher la case "tableau", afin d'avoir un champ de type tableau (*voir image ci-contre*).

Informations sur le champ



Nom du champ MonChampTableau

Type du champ Parent

Afficher comme tableau ☒

Annuler Sauvegarder le champ

Figure-19, Capture page administrateur, création d'un champ tableau- Établi par l'auteur

Puis une fois ce champ renseigné, l'utilisateur doit alors renseigner les champs qui serviront de colonnes à ce tableau comme l'utilisateur aurait fait pour les champs fils d'un champ parent.

2.7.3.3) Réalisation de la partie utilisateur







Après la réalisation de la partie administrateur, nous avons réalisé la partie utilisateur. La réalisation de cette partie a pour but d'afficher les champs de type tableau sous un format de tableau en ayant pour colonne les différents champs renseignés en amont par l'administrateur.

La réalisation de cette partie s'est déroulée en plusieurs phases. Tout d'abord, nous avons implémenté le fait que le rendu visuel du champ change dynamiquement selon le type renseigné en base de donnée. Après cette étape réalisée, nous nous sommes penchés sur l'affichage des colonnes. L'affichage des colonnes se réalise en plusieurs étapes. Dans une première étape, nous récupérons la liste des champs qui vont composer les colonnes en

questions. En seconde étape, une fois les différents champs récupérés, nous bouclons sur ces derniers en appliquant le même procédé étudié lors du point 2.6.3.3).

Une fois arrivés à cette étape, nous obtenons la structure de notre tableau. Ce qui nous mène à l'implémentation de l'ajout de ligne d'un champ tableau. Lorsque l'utilisateur arrive sur un formulaire ayant un ou plusieurs champ de type tableau, ces derniers n'ont qu'une ligne de pré-saisie, or le but d'un tableau est de pouvoir saisir plusieurs valeurs sur différentes lignes et colonnes. Nous avons donc ajouté le bouton "Ajouter ligne" qui, une fois cliqué, nous ajoute une ligne vierge. Ainsi, de cette manière l'utilisateur peut ajouter autant de lignes qu'il le souhaite. Une fois l'ajout de ligne réalisé, nous avons réalisé la partie suppression de ligne, pour ce faire nous avons ajouté une icône poubelle qui lorsqu'elle est cliquée supprime la ligne ciblée.

Après avoir réalisé différents tests de mise en application, nous nous sommes rendus compte que si, pour une quelconque raison, l'utilisateur doit réorganiser le tableau, alors celui-ci devait réécrire toutes les informations dans les bonnes lignes. Nous avons donc trouvé qu'il aurait put être intéressant aux utilisateurs de pouvoir faire "glisser" les différentes lignes les unes à travers les autres comme un effet dit de "drag and drop", afin de pouvoir réorganiser le tableau de manière plus simple et rapide afin d'améliorer l'expérience utilisateur. Pour ce faire, nous avons dans un premier temps ajouté une icône de croix quadri-directionnelle afin de faire comprendre à l'utilisateur que la ligne est mobile. Une fois l'icône ajoutée, nous avons ajouté du code JavaScript afin de donner le comportement de "drag & drop". De cette manière, après l'implémentation de toutes ces fonctionnalités, nous obtenons le visuel ci-contre pour un champ de type tableau.

TEST-RAPPORT					
Parent	Champ1	Champ2	Champ3	Champ4	Supprimer
+	valeur 1	Sélectionnez un élément	4150	<input type="checkbox"/>	
+		opt0	4851	<input type="checkbox"/>	
+		Sélectionnez un élément	86464	<input type="checkbox"/>	
+		opt1	1	<input type="checkbox"/>	
+	valeur aleatoire	Sélectionnez un élément	0660	<input type="checkbox"/>	
+		opt1	5557	<input type="checkbox"/>	

[Ajouter une ligne](#)

Figure-20, Capture page utilisateur, exemple de champ tableau- Établi par l'auteur

2.7.3.4) Génération des documents

En ce qui concerne la génération des documents, nous avons appliqué la même mécanique que lors de l'évolution précédente citée dans le point 2.6.3.4). A l'exception près,

qu'en plus de boucler sur chaque champ fils, nous avons bouclé sur la liste des valeurs liées à chaque. De plus, nous avons ajouté dans la partie administrateur, le squelette des champs fils afin de simplifier le traitement des utilisateurs pour les squelettes de fusion.

Liste des champs enfants

Champs de fusion :			
{descriptifEtablissem nt-test-rapport-parent- 0}	{descriptifEtablissem nt-test-rapport-parent- 1}	{descriptifEtablissem nt-test-rapport-parent- 2}	{descriptifEtablissem nt-test-rapport-parent- 3}
[!-- BEGIN DescriptifetablissementTest-rapportParentValeurs --]			
{valeur0}	{valeur1}	{valeur2}	{valeur3}
[!-- END DescriptifetablissementTest-rapportParentValeurs --]			









	Nom du champ	Type du champ	
+	Champ1	Champ texte	 
+	Champ2	Liste <ul style="list-style-type: none"> • opt0 • opt1 	 
+	Champ3	Numérique	 
+	Champ4	Case à cocher	 

Figure-21, Capture page administrateur, liste de champ de fusion d'un tableau- Établi par l'auteur

2.7.4) Rétrospective de l'évolution "champ tableau"

2.7.4.1) Le retour client

Une fois l'évolution réalisée, cette dernière a directement été utilisée par les SDIS. Ces derniers l'utilisent tous les jours dans une grande partie de leurs documents. Cette évolution leur permet maintenant de pouvoir "calquer" les tableaux de données renseignées sur PREVARIC automatiquement dans leur squelette de fusion. Cet automatisme, permet ainsi de gagner du temps, et de la qualité en supprimant le risque d'erreur de saisie manuelle lors d'un recopiage de données entre PREVARISC et le document final ciblé. Par ailleurs, un autre gros point de cette évolution est qu'elle n'a toujours pas reçu d'anomalie permettant ainsi d'éviter tout processus de correction d'anomalie faisant perdre du temps aux utilisateurs, et témoigne par ce biais de la robustesse du code produit.

2.7.4.2) Points forts de l'équipe sur cette évolution

Durant la réalisation de cette évolution l'équipe a su être forte sur différents points. Tout d'abord sur la partie conception, l'évolution des champs tableaux nous a demandé un gros travail de conception que nous avons su produire, par l'utilisation de design pattern et temps passé à concevoir. Un second point sur lequel l'équipe s'est avérée performante est la

production de code JavaScript de cette évolution. Cette production s'est avouée riche techniquement car les technologies d'aujourd'hui automatisent les mécaniques que nous avons dû programmer, ainsi le fait d'avoir programmé des composants réactifs complexes en partant de rien s'est amené à la fois très instructive d'un point de vue technique au niveau production du code JS (*JavaScript*) et conception.

2.7.4.3) Difficultés rencontrées de cette évolution

Lors de l'élaboration de cette évolution, nous avons rencontré différentes difficultés. Notamment, lorsqu'il a fallu commencer à implémenter l'ordonnancement des éléments dans le tableau. L'ordonnancement nous a posé problème à différents niveaux, que ce soit sur l'affichage dynamique lorsqu'un utilisateur réorganise un tableau, la génération des champs de fusion, le temps de développement estimé.

Les difficultés ressenties pour les deux premiers points se sont essentiellement concentrées lors de la réorganisation des tableaux du côté utilisateur était l'intégration des drag & drop entre les lignes en JS ainsi que l'actualisation automatique des composants qui ont aussi dû être fait en JS.

En ce qui concerne la difficulté lors de l'estimation du temps de production de cette tâche, nous pouvons dire qu'elle était dû au fait qu'on a dans un premier temps sous-estimé la charge de travail qu'elle allait représenter. Puis, nous avons cumulé les problèmes qui nous ont donc plus ralenti et par conséquent mis en retard.

2.8) Après PREVARISC

Après avoir effectué différentes évolutions et résolution d'anomalies, j'ai été affecté à l'équipe Drupal. J'y ai été affecté dans le but de voir un autre aspect du développement qu'on appelle le build. Le build consiste à développer une solution ou un produit, ceci en partant de rien contrairement à la TMA où l'on reprend un projet pour y apporter des évolutions ou résolutions d'anomalies. Lors de mon arrivée dans l'équipe orientée développement Drupal, j'ai dû dans un premier temps assimiler le fonctionnement de cette nouvelle technologie. Cette assimilation s'est faite en plusieurs étapes.

La première a été la participation à une revue de code entre les développeurs afin d'avoir une première vue macro du fonctionnement des applications en cours ainsi que du fonctionnement de Drupal.

La seconde, par le suivi d'une formation papier en guise de tutoriel technique, afin de pouvoir mettre en pratique les différents paradigmes s'exerçants autour de Drupal.

Partie 03 Rétrospective d'alternance

3.1) Rétrospectives des compétences techniques acquises

Cette alternance chez Atos au sein du projet Prevarisc m'a permis de monter en compétences sur la technologie Zend. Également d'améliorer ma qualité de code notamment par la production d'un code plus propre et solide, ainsi qu'en réalisant des architectures complexes.

J'ai également pu monter en compétences en reverse engineering, ceci notamment lors de la réalisation de différentes contributions ayant pour but d'améliorer le code déjà existant ainsi qu'en ré-écrivant certains morceaux de code (*refactoring*) afin d'avoir un code plus stable et solide.

3.2) Rétrospectives du savoir être acquis

Fort de cette expérience, j'ai évolué sur mes compétences techniques mais aussi sur le plan humain au niveau de mon savoir être. J'ai évolué notamment sur la gestion du stress, je peux illustrer cette évolution d'une part par le fait d'avoir dû résoudre avec Maxime MERRIEN des anomalies urgentes bloquantes. J'ai notamment progressé au niveau de ma communication et j'ai appris comment mieux communiquer avec mes pairs ainsi qu'avec le client. De la même façon, j'ai appris à adapter mon discours selon le contexte et les situations. Pour terminer sur l'évolution de mes soft skills, je dirais que cette expérience de 2 ans m'a appris à savoir couper mon temps entre les moments professionnels et personnels, chose à laquelle je n'avais pas su faire ou du moins avait du mal à faire lors de ma précédente alternance.

Conclusion

Ces deux ans d'alternance m'ont permis de vivre plein de choses. Tout d'abord, au sein de cette expérience enrichissante, j'ai appris ce qu'était de travailler dans un projet en TMA, que ce soit en passant par la résolution d'anomalies autrement dénommées bug, comme la réalisation d'évolutions. Je retiens de cette expérience de TMA au sein du projet PREVARISC essentiellement trois choses.

La première, la montée en compétence que j'ai pu réaliser sur Zend 1.12 avec Maxime MERRIEN, je tiens à souligner le travail pédagogique de qualité qu'a réalisé Maxime pour me faire monter en compétence sur cette ancienne technologie.

La seconde, je retiens de cette mission, la réalisation des deux évolutions citées dans ce rapport. L'évolution des champ parent et champ tableau. Ces deux évolutions ont ajouté d'une part une dimension non négligeable à la qualité des formulaires de PREVARISC, puis d'une autre part cette évolution m'a permis de mettre en place des architecture complexes de code en appliquant des design pattern.

Le troisième point que je retiens également de PREVARISC, est l'apprentissage de la gestion du stress. Cela a été mis à l'épreuve lors de la résolution d'une anomalie critique, qui a été une expérience formatrice pour moi. Cette situation m'a appris comment agir face à des situations difficiles et je suis reconnaissant pour cette opportunité d'apprentissage. Bien que je ne souhaite à personne de vivre une telle situation, je suis convaincu que l'expérience acquise me sera bénéfique tout au long de ma carrière professionnelle.

Pour terminer, je dirais que je sors grandi de cette aventure. Grandis du fait d'une part, par l'enrichissement technique et humain. Durant cette alternance, j'ai pu rencontrer des personnes de qualités, qui toutes m'ont apportées un petit quelque chose dans mon éventail de compétences et mon savoir être. Je sors fier de cette expérience et d'avoir pu réaliser la fin de mes études supérieures au sein de la DF.

En plus de ces points clés que j'ai retenus de mon expérience au sein du projet PREVARISC, je suis heureux de partager que j'ai récemment accepté un poste de développeur FullStack Angular Symfony chez TBS-Cobalt. Je suis enthousiaste à l'idée de continuer à monter en compétences et de mettre en pratique ce que j'ai appris durant ces deux années d'alternance au sein de la société Atos. Je suis convaincu que cette nouvelle expérience professionnelle me permettra de continuer à évoluer dans ma carrière. Je tiens à remercier toute l'équipe de la DF pour leur soutien et leur encadrement durant ces deux années passées à leurs côtés.

Glossaire

A

Application : Programme informatique répondant à divers besoins par la composition de différentes fonctionnalités.

B

C

D

E

F

Framework : *“est un ensemble cohérent de composants logiciels structurels qui sert à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel, c'est-à-dire une architecture”*. Source: <https://fr.wikipedia.org/wiki/Framework>

G

H

I

IHM (*Interface Homme Machine*) : Ensemble d'éléments graphiques composant une interface permettant de créer une interaction entre un utilisateur et un machine en ayant comme but de faciliter l'utilisation de diverses applications.

J

JavaScript : Langage côté client permettant d'interagir avec les différents composants d'une page web.

K

L

Linux : Système d'exploitation open source, souvent utilisé par les développeurs pour le développement.

M

N

O

OS : Un système d'exploitation nommé (*OS*) est une couche logiciel permettant de gérer différentes ressources d'un ordinateur, pouvant par ailleurs établir une interface entre un utilisateur et les programmes le comprenant.

P

PHP : Langage de programmation orienté dans le développement web backend.

Q

R

S

SDIS: Service départemental d'incendie et de secours

Soft skills : correspond au savoir être en langue anglaise

T

U

V

W

Wireframe : Maquette permettant d'établir les architectures des futurs écrans des applications.

X

Y

Z

Zend : Framework PHP