



Amélioration de l'expérience client sur le formulaire de déclaration FNoL Home-in-One

Table des matières

1	Analyse préliminaire	5
1.1	Introduction	5
1.2	Objectifs.....	5
1.3	Planification initiale	5
2	Analyse / Conception.....	8
2.1	Organisation	8
2.2	Concept	8
2.2.1	Structure de l'application.....	8
2.2.2	Le formulaire.....	9
2.2.1	Persistance de données	11
2.2.2	Ajout d'étapes	11
2.2.3	Champ date	11
2.2.4	Info-bulle.....	11
2.3	Stratégie de test.....	12
2.4	Risques techniques	13
2.5	Dossier de conception	13
2.5.1	Matériel.....	13
2.5.2	Logiciel et outils informatique.....	13
3	Réalisation.....	15
3.1	Dossier de réalisation	15
3.1.1	Le formulaire.....	15
3.1.2	Persistance de donnée	19
3.1.3	Librairie va-ngx-shared	20
3.1.4	Librairie va-ngx-primeng	21
3.1.5	Etape dans le formulaire	22
3.1.6	Champ date	24
3.1.7	Infobulle	25
3.1.8	Déploiement sur l'environnement DCI	25
3.2	Description des tests effectués.....	25

3.3	Erreurs restantes	26
3.3.1	Persistance de données	26
3.3.2	Les étapes dans le formulaire.....	26
3.3.3	Infobulles	27
3.4	Liste des documents fournis	27
4	Conclusions	28
4.1	Objectifs.....	28
4.2	Points positifs / points négatifs.....	28
4.2.1	Le positif	28
4.2.2	Le négatif.....	28
4.3	Difficultés rencontrées	28
4.4	Evolutions et suites possibles du projet.....	29
4.4.1	La correction des bugs	29
4.4.2	Implémentation des modifications sur la partie Responsabilité Civile.....	29
4.4.3	Ajout d'un composant de navigation.....	29
4.5	Bilan personnel.....	29
5	Annexes.....	30
5.1	Résumé du rapport du TPI / version succincte de la documentation	30
5.2	Sources – Bibliographie.....	31
5.2.1	Webographie	31
5.2.2	Illustration	31
6	Journal de travail	33
6.1	Jour 1 – 11 mars 2024.....	33
6.1.1	Bilan de la journée	33
6.2	Jour 2 – 12 mars 2024.....	35
6.2.1	Bilan de la journée	36
6.3	Jour 3 – 13 mars 2024.....	37
6.3.1	Bilan de la journée	38
6.4	Jour 4 – 14 mars 2024.....	39
6.4.1	Bilan de la journée	39

6.5	Jour 5 – 18 mars 2024.....	40
6.5.1	Bilan de la journée.....	40
6.6	Jour 6 – 19 mars 2024.....	41
6.6.1	Bilan de la journée.....	41
6.7	Jour 7 – 20 mars 2024.....	42
6.7.1	Bilan de la journée.....	42
6.8	Jour 8 – 21 mars 2024.....	44
6.8.1	Bilan de la journée.....	44
6.9	Jour 9 – 25 mars 2024.....	45
6.9.1	Bilan de la journée.....	45
6.10	Jour 10 – 26 mars 2024.....	46
6.10.1	Bilan de la journée.....	46
7	Glossaire	47

1 Analyse préliminaire

1.1 Introduction

Le site de la Vaudoise Assurances est géré avec un CMS du marché développé en .NET. Grâce à cet outil, il est possible de délivrer l'entier du contenu du site à plus de 35'000 utilisateurs par mois. Site d'information, de contact et d'e-commerce, il sert de plateforme centrale dans la stratégie numérique de la Vaudoise Assurances. Actuellement, le site offre plusieurs formulaires en ligne permettant aux visiteurs de rapidement déclarer un sinistre en rapport avec leur police d'assurance.

Dans le cadre de l'amélioration du processus de déclaration de sinistre ou également surnommé FNoL (First Notice of Loss), l'équipe en charge du site a créé un formulaire en ligne. Ledit formulaire se trouve être relativement simple et l'équipe Online Expérience souhaite apporter un lot d'améliorations afin d'augmenter la qualité et l'expérience client.

1.2 Objectifs

Objectifs relatifs au projet fournis par le Chef de Projet et décrits dans le cahier des charges :

1. Persistance des données → Lorsqu'un client consulte le formulaire en ligne et qu'il ferme son navigateur, actuellement le contenu est perdu et ne persiste pas. Dans la nouvelle version, lorsque le client revient sur le formulaire, il devrait lui être proposer de reprendre sa déclaration là où il en était.
2. Étapes dans le formulaire → Point crucial d'expérience utilisateur, le souhaite serait de récupérer le composant *step* déjà développé par la Vaudoise Assurance et l'intégrer dans le formulaire. Par anticipation, les champs avaient déjà été ordonnancés dans une bonne logique :
 - a. **Les informations du déclarant.**
 - b. **Les informations du sinistre.**
 - c. **Les informations complémentaires.**
3. Champs date → Par manque de temps, les champs de type date avaient été gardés dans leur état simple lors de l'implémentation initiale. Dans cette nouvelle version, il s'agit de récupérer la librairie *datepicker* et de les ajouter sur les champs « date de naissance » et « date du sinistre ».
4. Ajout d'info-bulle → Certains champs dans le formulaire se doivent d'être accompagnés d'une info-bulle afin de leurs donner plus de sens. Cela sera fait avec un composant pioché dans la librairie de composants UI *PrimeNG*.
5. Lorsqu'un client drag & drop un fichier sur la page, il faudrait améliorer graphiquement le rendu du composant avec l'ajout d'un masque gris

1.3 Planification initiale

Chaque bloc dessiné dans le planning initial représente un temps de ~2h.

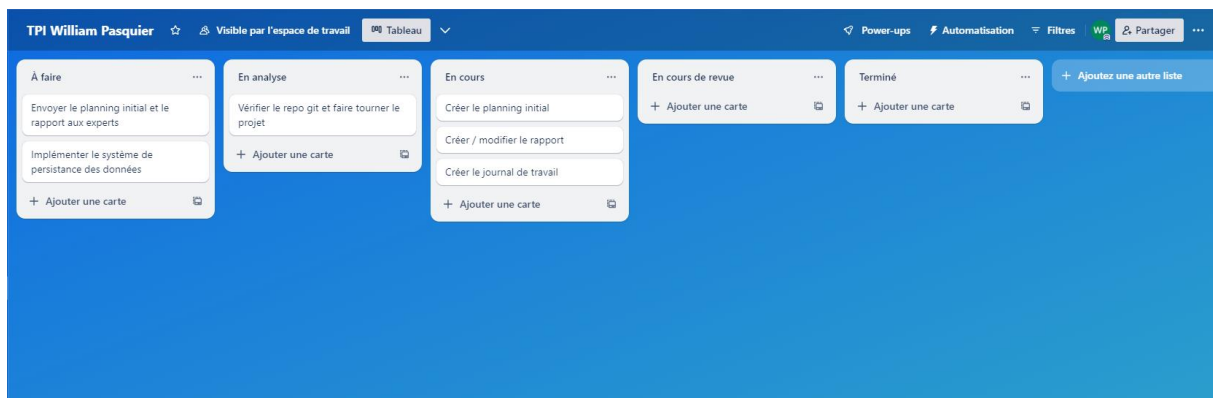
Tâches :	Type de tâches	Dates				
		11.03.2024	12.03.2024	13.03.2024	14.03.2024	18.03.2024
Découvertes / discussion du cahier des charges	Documentation					
Création du planning initial	Documentation					
Création du rapport de TPI	Documentation					
Création du journal de travail	Documentation					
Configuration de l'environnement de développement	Implémentation					
Analyse du fonctionnement de la persistance de données	Analyse					
Question liée à l'analyse si besoin	Autre					
Implémentation de la persistance de données	Implémentation					
Implémentation des test unitaires	Implémentation					
Test de l'application	Test					
Demander les traductions des champs dans les différentes langues	Autre					
Changement des champs de date	Implémentation					
Ajout des infos bulles sur les champs nécessaires	Implémentation					
Analyse du masque gris sur le FNOL VM	Analyse					
Ajout du masque gris lors du Drag & Drop	Implémentation					
Implémentation des tests si nécessaire	Implémentation					
Test de l'application	Test					
Analyse du fonctionnement des étapes du formulaire déjà développer	Analyse					
Question liée à l'analyse si besoin	Autre					
Mise à jour du rapport, du journal et du planning	Documentation					

Tâches :	Type de tâches	Dates				
		19.03.2024	20.03.2024	21.03.2024	25.03.2024	26.03.2024
Implémentation des étapes du formulaire	Implémentation					
Implémentation des test sur les étapes du formulaire	Implémentation					
Test de l'application	Test					
Vérification de l'utilisation des bonnes pratiques dans l'implémentation du code	Implémentation					
Ajouter les traductions des champs	Implémentation					
Déployer sur l'instance DCI le projet	Implémentation					
Tester l'application et les modifications sur l'instance DCI	Test					
Préparation des livrables attendus	Documentation					
Mise à jour du rapport, du journal et du planning	Documentation					

2 Analyse / Conception

2.1 Organisation

Dès lors que le planning de travail initial fût créé la prochaine étape est d'organiser les différentes tâches pour pouvoir avoir un suivi de ces dernières. Il existe différent choix pouvant répondre au besoin comme les *GitHub Projects* ou *JIRA*. Le choix final sera *Trello*, une application web permettant de créer des tâches sous formes de cartes et de les classer selon la catégorie d'avancement. Rapide et facile à mettre en place, elle convient parfaitement pour ce cas d'utilisation.



Un journal de travail est également créé et tenu à jour au minimum 2 fois par jour (matin et après-midi) pour avoir un suivi détaillé des actions et des discussions / décisions. Tous les différents livrables et documents sont sauvegardés dans un cloud one drive et un repo git pour garantir une redondance de donnée en cas de suppression accidentelle.

2.2 Concept

L'application est basée sur le framework front-end **Angular**. **Angular** permet de créer des SPA (single page applications) web côté clients. Grâce à ce framework, il est possible de créer des applications dynamique grâce au système de composants et de service qu'il propose. En combinant ces différents systèmes, il est possible de créer des applications puissantes.

2.2.1 Structure de l'application

L'application possède une structure spéciale. Au sein d'une seule application en existe 2 autres. Lorsqu'une nouvelle application Angular est créée grâce au CLI, une architecture de base est implémentée avec un composant de base. L'*app.component*. A l'intérieur de ce répertoire s'y trouve un fichier *app.html*, il est la structure du composant. Il y également un contrôleur *app.component.ts* accompagné de son fichier de test *app.components.spec.ts*. Ces fichiers servent à ajouter de la logique dans le composant (action lors d'un clic sur un bouton, ...) et finalement un fichier de style *app.component.scss*. De plus le composant parent possède 2 fichiers en plus, le *app.module.ts* qui contient toute les déclarations de module interne ou externe, des

composants de l'applications ou encore différents service à utiliser dans l'applications. Généralement, c'est l'intérieur du composant *app.component* que l'on écrit les bases de l'applications.

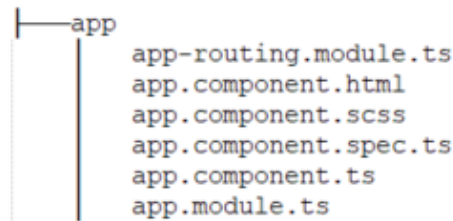


Figure 1 : Arborescence app

Notre cas est plus spécifique. Comme dit précédemment, il existe de 2 applications permettant de faire 2 types de déclarations différentes, le *damage-claim* et le *third-party-liability*. Ces 2 fonctionnalités (*features*) se trouvent dans le fichier portant le même nom et sont référés à 2 routes spécifiques de l'application. Chaque « sous-applications possèdent » la même structure que l'*app.component*. Le composant contenant la déclaration de sinistre dans un ménage concerne le composant *damage-claim*

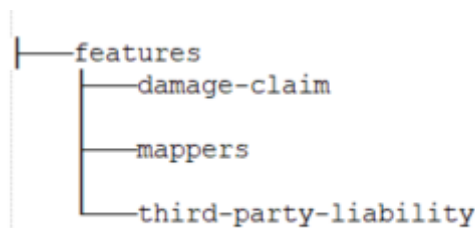


Figure 2 : Arborescence features

2.2.2 Le formulaire

Angular fournit un système formulaire puissant avec lequel il est possible d'en tirer parti pour le rendre entièrement dynamique (afficher ou cacher des champs lors de la modification d'un autre ou alors lorsqu'une certaine valeur est rentrée). Actuellement le formulaire s'initialise en créant un *FormGroup* global dans le composant parent. Le formulaire est divisé en 3 étapes :

- Informations concernant le déclarant et l'assuré

Informations concernant le déclarant et l'assuré

Vous êtes

Preneur/Assuré	Conseiller	Courtier
----------------	------------	----------

Coordonnées de l'assuré

Nom

Prénom

Figure 3 : Etape "Informations concernant le déclarant et l'assuré" du formulaire

- Information concernant le sinistre

Information concernant le sinistre

Cause du sinistre

Description du sinistre

Date du sinistre

Figure 4 : Etape "Information concernant le sinistre" du formulaire

- Informations complémentaires

Informations complémentaires

Une autre compagnie d'assurance est-elle concernée par ce sinistre ?

Oui	Non
-----	-----

Le dommage a-t-il été provoqué par un tiers ?

Oui	Non	Peut-être
-----	-----	-----------

Figure 5 : Etape "Informations complémentaires" du formulaire

Chaque étape du formulaire est dans un composant dans lequel il y génère et gère les *FormControl* qui le composent dans un *FormGroup*. Une fois généré, le *FormGroup* est référencé au formulaire parent. En effet, c'est depuis le formulaire parent que la requête d'envoi des données de réclamations se fait. Cette forme actuelle ne convient pas pour effectuer une persistance de données facile avec une possibilité d'amélioration plus simple. Un remaniement du système de création de formulaire est à prévoir.

2.2.1 Persistance de données

La persistance de donnée permet, dans ce cas, de sauvegarder les données entrées par l'utilisateur lors de la complétion de sa déclaration de sinistre. Si ce dernier venait à quitter la page web ou à la rafraîchir, les données se retrouvent à nouveau dans les champs afin de pouvoir continuer d'effectuer sa déclaration. Dans le cahier des charges, il est stipulé que les données se doivent d'être enregistrées dans le local storage du navigateur web du client.

Pour pouvoir obtenir ce résultat, un service devrait être créer pour gérer l'état globale de du formulaire tout en souscrivant aux modifications des champs du formulaire. Après chaque changement d'état, un autre service s'occupant d'enregistrer, récupérer et supprimer les données dans le local storage. Ce service de gestion du local storage existe déjà et est déjà présent dans l'application VM FNoL, un projet fortement similaire au notre l'HiO FNoL.

Afin de laisser le choix à l'utilisateur, un popup de confirmation devrait être ajouté avec la proposition de récupérer ou non les données enregistrer. Ce popup peut être récupéré grâce à la librairie *PrimeNG* qui est déjà référencé dans le projet.

2.2.2 Ajout d'étapes

Le formulaire dans sa nouvelle forme doit ajouter un système d'étapes. Les 3 étapes spécifiées dans [Le formulaire](#) sont actuellement affichées sur une seule page. Le but est d'ajouter un système de navigation et d'afficher une étape après l'autre. Ce qu'il faut mettre en place c'est un système qui permet de cacher les autres étapes non-nécessaires. Cet ajout requiert également des nouveaux boutons de navigation « Suivant » et « Précédent » permettant d'effectuer les actions.

2.2.3 Champ date

Les champs date actuelle dans le formulaire sont des champ text avec une validation de format. Le souhait serait de migrer ces champs à ceux contenu dans le formulaire VM FNoL permettant d'afficher une fenêtre calendrier pour la sélection de la date.

2.2.4 Info-bulle

Les info-bulles sont des composants *PrimeNG* qu'il faut intégrer dans le projet. Etant donné que la librairie est déjà installée il suffit de faire un appel de ce dernier.

2.3 Stratégie de test

Lorsque le CLI Angular est installé sur un poste, il installe automatiquement tout ce qui est nécessaire pour performer des tests que ce soit ceux unitaires ou encore ceux d'intégrations.

Dès lors que l'on crée un composant ou un service en s'aidant du CLI, ils sont créés avec des fichiers de test dans lequel il est possible d'y écrire les futurs tests unitaires permettant de garantir le fonctionnement attendu du module ajouté en cas de modification. Cette philosophie de test permet aux développeurs de s'assurer que les futures implémentations ne doivent perturber que les tests présents dans le scope de ces modifications et de les adapter si la logique change.

Les tests qui sont rédigés dans l'application sont des tests unitaires. Les tests unitaires se concentrent sur une seule partie d'une application complète. Généralement, elle teste une fonction ou une classe de l'application et la philosophie est de faire 1 test pour 1 fonctionnalité ou 1 cas d'utilisation. Idéalement, le composant testé est dépourvu d'effets secondaires, de sorte qu'il peut être facilement isolé et testé.

Ces tests sont intégrés dans la *pipeline CI/CD (Continuous Integration / Continuous Deployment)* afin d'éviter que des modifications ne casse l'application. Les tests sont exécutés à chaque *commit* sur le *repository* GIT de l'application.

Le but est qu'à chaque création de composants ou services d'implémenter des tests capables de tester la logique de l'application en prenant des cas concrets. L'objectif serait de couvrir une bonne partie des cas possible et important de l'application afin de garantir qu'un changement, par exemple, ne vienne pas casser le fonctionnement de la détection de changement d'état vital à la persistance de donnée. En implémentant des tests de cette manière, il est possible de faire tendre le taux de risque d'ajout de bug à un taux acceptable pour le déploiement de l'application.

Les tests ne devant réellement appeler des services ou model de l'application, ils seront « *mockés* » afin de pouvoir en appeler des versions contrôlées.



```
TestBed.configureTestingModule({
  providers: [
    MockProvider(DamageClaimFormStateService, formStateServiceMock),
    MockProvider(LocalStorageService, localStorageServiceMock),
  ],
});
service = TestBed.inject(DamageClaimStateStorageService);
```

Figure 6 : Exemple de mock dans les tests unitaires

Aucunes données de test ne sont à prévoir.

2.4 Risques techniques

Malheureusement, mes connaissances en tests unitaires actuelle ne me permettent pas de pouvoir en rédiger de manière fluide contrairement à l'implémentation des différents objectifs du cahier des charges. Cependant, en m'aidant des différents exemples déjà présents au sein de l'application et en regardant différentes formations écrites ou vidéos, je serai en mesure de réussir d'ajouter des tests capables de couvrir un pourcentage satisfaisant de l'application. De plus grâce aux pulls requests et à l'expertise de mes collègues, les erreurs pourront être évitées.

Un autre point pouvant poser problèmes peut être l'implémentation de composants ou de « système » requérant plus de *refactoring* que nécessaire ou de devoir implémenter des librairies apportant plus de travaux. La solution serait d'analyser les différentes possibilités afin de convier efficacité et possibilité d'implémentation future au sein du monorepo avec les librairies faites pas la Vaudoise Assurances. De plus en analysant l'implémentation dans les autres projets, il sera plus facile de les ajouter au sein du projet.

2.5 Dossier de conception

Afin de pouvoir conceptualisé l'application et effectuer les travaux, différentes ressources et matériels ont dû être utilisé pour parvenir à ces résultats.

2.5.1 Matériel

- Tablette Dell fourni par la Vaudoise Assurances.
- Poste de développement avec accès administrateur afin de pouvoir installer les ressources nécessaires au développement.

2.5.2 Logiciel et outils informatique

2.5.2.1 Environnement de développement

Afin de pouvoir effectuer le développement, l'IDE Visual Studio Code est installé avec différentes extensions comme Prettier me permet d'écrire du code et de pouvoir automatiquement le formater. Ce qui permet d'avoir les mêmes standards entre tous les développeurs. Il est mon premier choix car il offre différentes extensions capables de simplifier la vie d'un développeur. De plus, il est un IDE capable de travailler avec de nombreux langages comme le TypeScript ou l'HTML grâce à sa grande polyvalence.

2.5.2.2 Logiciel de conceptualisation

Dans ce rapport certaine partie seront représentée graphiquement pour pouvoir expliquer graphiquement des concepts de l'application ou des futures approches possibles. J'ai utilisé le site web draw.io qui permet, gratuitement de faire toute sorte de diagramme comme des UML et bien plus.

2.5.2.3 Git

L'application est disponible sur un répertoire git. Git permet de faire de la gestion de version sur son application. Entre autres, Git permet également de créer des branches,

des sortes de copies de l'application permettant de faire des modifications sans réellement affecter l'application de base. Ce répertoire est stocké sur un hébergeur en ligne, dans notre cas BitBucket qui permet donc d'avoir une sauvegarde en ligne du code source de l'application.

2.5.2.4 Librairie ou outils de développements

Comme expliqué précédemment, l'application web est faites en Angular. Pour pouvoir bénéficier de l'utilisation des commandes Angular, l'installation du CLI est nécessaire. Il faut également installer une version de NodeJS et de NPM installée sur le poste de développement.

3 Réalisation

3.1 Dossier de réalisation

3.1.1 Le formulaire

En ce qui concerne le formulaire. Un remaniement du système de création doit être fait. Cela permettra d'implémenter plus facilement le système de persistance et celui des étapes dans le formulaire tout en regroupant en 1 point la gestion sa création. Comme expliqué, actuellement chaque étape du formulaire est dans un composant dans lequel il y génère et gère les *FormControl* qui le composent dans un *FormGroup*. Une fois généré, le *FormGroup* est référencé au formulaire parent. En effet, c'est depuis le formulaire parent que la requête d'envoi des données de réclamations se fait.

Ce système est fonctionnel mais dès lors qu'on y ajoute de la complexité, comme de la persistance de donnée pour permettre de retrouver l'état du formulaire combiné à des étapes afin de garder une utilisation fluide du service de déclaration de dégâts alors leurs ajouts devient plus complexe.

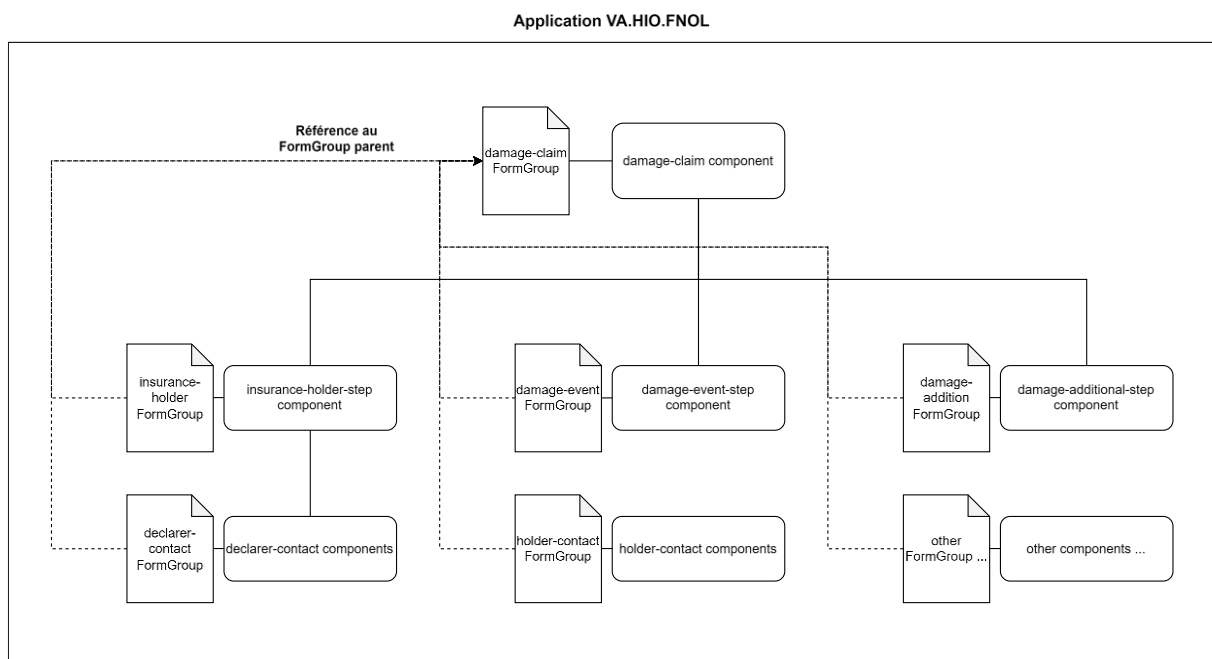


Figure 7 : Schéma d'explication de création du formulaire

En effet, afin de permettre de sauvegarder dans le local storage les données fournies par l'utilisateur, il faudrait se souscrire à chaque changement des champs (*FormControl*) pour ensuite le stocker. Dans l'état actuelle de l'application et de la génération du formulaire, il aurait fallu manuellement ajouter chaque souscription. Cette méthode n'est pas viable en cas de changement dans le formulaire qui obligent à parcourir les composants qui composent l'applications.

La manière optimale, qui est utilisée dans le FNoL VM, est de centralisé la création du formulaire dans un service injecté dans l'application. Ce service permet de gérer la création du formulaire et d'en sauvegarder l'état. Ce service offre également la

possibilité de souscrire aux modifications de l'état. Dès lors que le composant parent est initialisé, on y injecte le service qui va automatiquement créer notre *FormGroup* pour toute l'application.

Maintenant que le service servant à la création du formulaire est implémenté, il existe 2 solutions pour instancier les composants enfant avec leurs *FormGroup* respectif.

La première consiste à générer le *FormGroup* avec chaque *FormGroup* et chaque *FormControl* qui se trouve à l'intérieur pour ensuite l'instancier dans une propriété du composant parent. Pour les transmettre aux enfants, on crée des propriétés Input pour ensuite l'y ajouter la référence. Cette méthode est fonctionnelle mais ajoute de nouvelles contraintes. Tout d'abord le formulaire est créé en entier mais l'utilisateur n'a peut-être entré qu'une valeur dans le champ. Dès lors que l'on sauvegarde l'état, tous les champs, que ce soit ceux avec une valeur mais également ceux qui ne possèdent aucune valeur, finissent dans le local storage ce qui ne sert à rien.

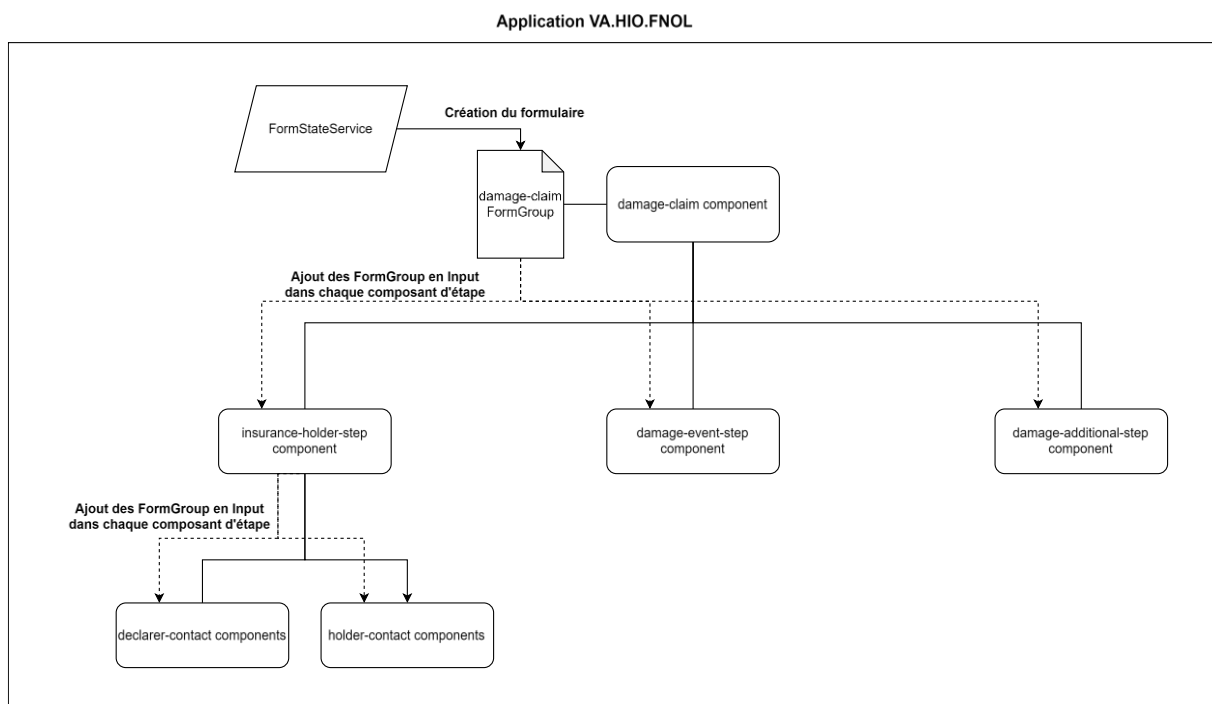


Figure 8 : Schéma d'explication de la nouvelle implémentation théorique du formulaire

La seconde serait, comme dans le précédent, de créer le *FormGroup* mais uniquement le général lors de l'instanciation du service dans le composant parent. Pour ce qui est des enfants, étant donné que l'application est découpée en étape, à chaque appel d'étape, il suffit d'appeler le service afin de créer ou récupérer le *FormGroup* correspondant. Si le composant de l'étape possède des composants enfants, étant donné que ces derniers sont déjà créés dans le *FormGroup* de celui de l'étape, il suffit de le transmettre avec un *Input*. Cette solution facilite les appels et évite d'enregistrer des données inutiles.

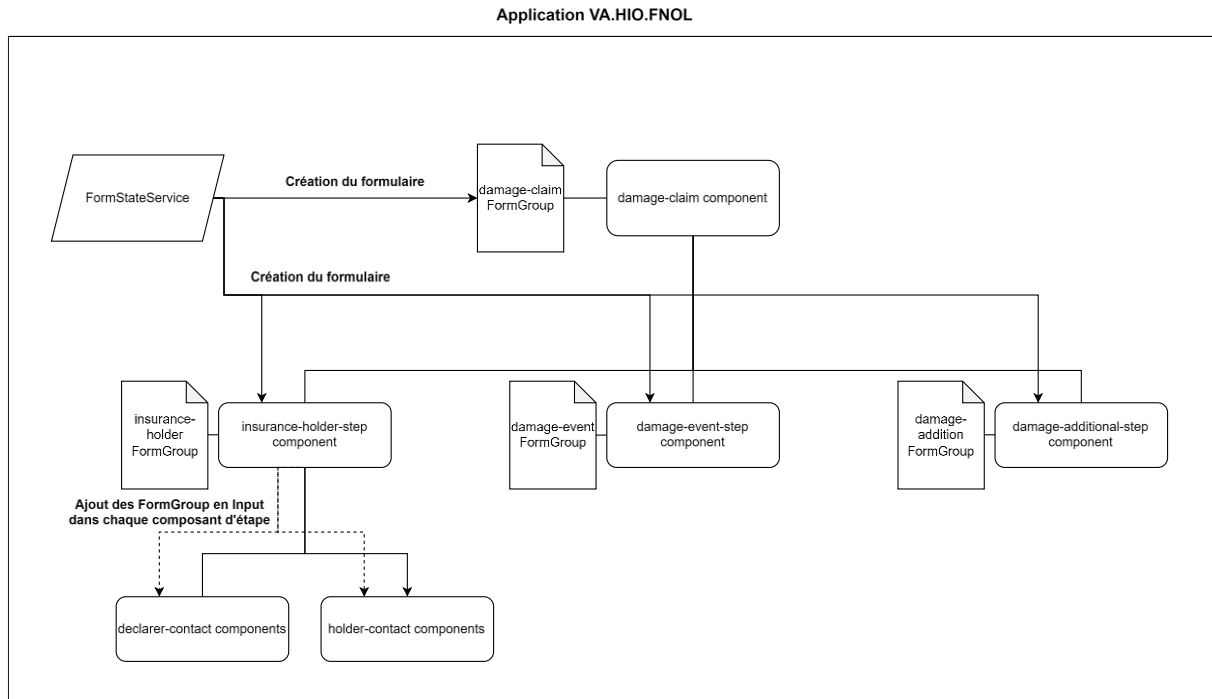


Figure 9 : Schéma d'explication de la nouvelle implémentation choisie du formulaire

Aux vues de ces avantages, cette seconde solution est la plus optimale et sera finalement choisie dans la nouvelle implémentation.

```

/**
 * Gets the requested form group.
 * @param propertyName - the name of the form group to retrieve.
 * @returns the form group or undefined if it is not yet defined.
 */
getOrCreateFormGroup<T extends keyof DamageClaimFormGroup>(
  propertyName: T,
): Required<DamageClaimFormGroup>[T] {
  let formGroup = this.state.controls[propertyName];
  if (!formGroup) {
    formGroup = this.formBuilderService.buildFormGroup(propertyName);

    this.state.setControl(propertyName, formGroup, {
      emitEvent: false,
    });
  }

  return formGroup as Required<DamageClaimFormGroup>[T];
}

```

Figure 10 : Code pour la création ou la récupération du FormGroup

```

/**
 * Build rge form group for the requested step.
 * @param stepName - The name of the step for which the form group should be built.
 * @returns The requested form group.
 * @throws An error if the requested name is not supported.
 */
buildFormGroup<T extends keyof DamageClaimFormGroup>(
  stepName: T,
): DamageClaimFormGroup[T] {
  switch (stepName) {
    case 'insuranceHolder':
      return this.insuranceHolderStepFormGroup() as DamageClaimFormGroup[T];
    case 'damageEvent':
      return this.damageEventStepFormGroup() as DamageClaimFormGroup[T];
    case 'additionalInformation':
      return this.additionalInformationStepFormGroup() as DamageClaimFormGroup[T];
  }

  throw new Error(
    `Invalid property name ${stepName} provided to buildFormGroup`,
  );
}

```

Figure 12 : Création du FormGroup selon l'étape choisie

```

ngOnInit(): void {
  // Get step form group.
  this.formGroup =
    this.formStateService.getOrCreateFormGroup('insuranceHolder');

  this.declarerType = this.formGroup.controls.declarerType;

  this.declarerContact = this.formGroup.controls.declarerContact;

  this.holderContact = this.formGroup.controls.holderContact;
}

```

Figure 11 : Création ou récupération du formulaire depuis l'état dans le composant parent ou dans celui d'une étape

3.1.2 Persistance de donnée

Pour pouvoir effectuer la persistance de données, avec les changements ajouter pour la création du formulaire, une souscription des modifications dans l'état du formulaire doit être faites pour permettre l'enregistrement.

```
/**
 * Indicates if state can be saved when form group value changes.
 * It is used to prevent saving state when form groups are being
 * created and when state is being loaded from local storage.
 */
canSaveState = true;

constructor() {
    this.formStateService.valueChanged.subscribe(
        this.onValueChanges.bind(this),
    );
}
```

Figure 14 : Souscription des modifications de l'état du FormGroup dans le constructeur du service

```
/**
 * Callback method that is invoked when from group value changes.
 * Saves the state to local storage.
 */
onValueChanges(): void {
    if (this.canSaveState) {
        const stateValue: FormGroupModel<DamageClaimFormGroup> =
            this.formStateService.getState().getRawValue();

        stateValue.recaptcha = null;

        this.localStorageService.set(localStorageKey, stateValue);
    }
}
```

Figure 13 : Sauvegarde de l'état du Formulaire

A chaque chargement du composant parent, il y recherche le contenu du local storage avant d'ajouter à chaque contrôle sa valeur. Afin de permettre à l'utilisateur d'avoir le choix entre reprendre sa déclaration ou de la redémarrer à zéro. Un pop-up s'affiche lorsqu'un état est possible d'être récupéré.

```

ngAfterViewInit(): void {
  if (this.stateStorageService.hasSavedState()) {
    setTimeout(
      () =>
        this.confirmationService.confirm({
          message: this.loadStateConfirmationText,
          icon: DialogIcon.InfoCircle,
          accept: this.loadSavedState.bind(this),
          reject: this.resetSavedState.bind(this),
        }),
      200,
    );
  }
}

```

Figure 15 : Contrôle si un état existe et affiche le pop-up

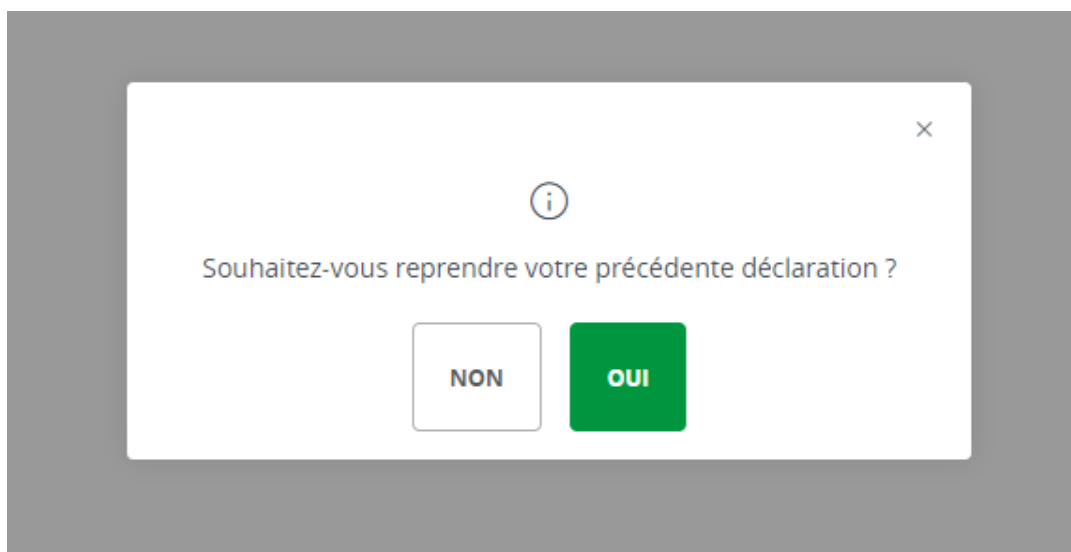
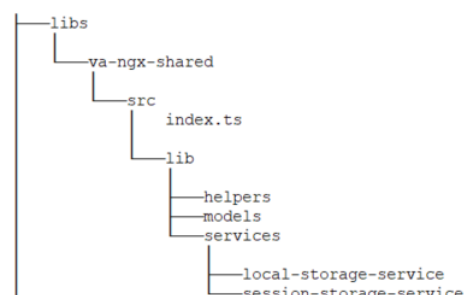


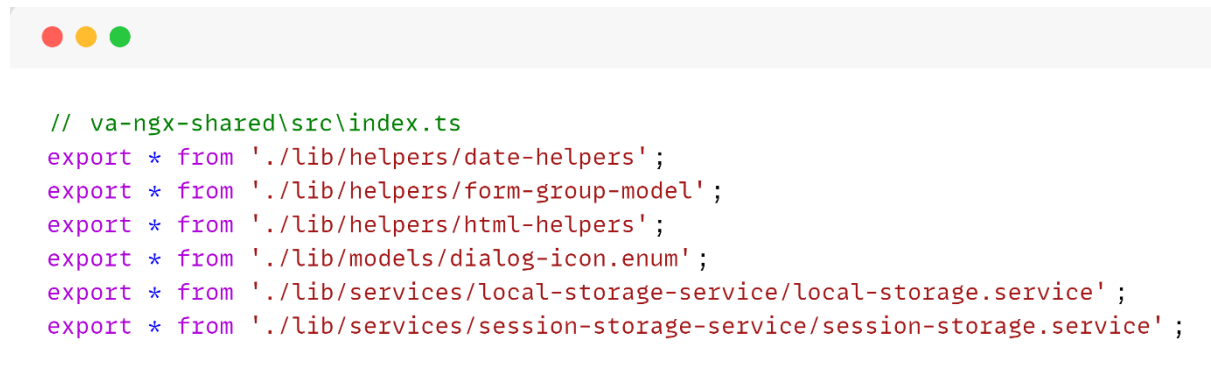
Figure 16 : Pop-up affiché à l'utilisateur

3.1.3 Librairie va-ngx-shared

Le service s'occupant de la gestion du local storage se trouve dans une librairie développée par la Vaudoise Assurances. Pour pouvoir l'utiliser, il faut en faire la référence au sein du projet. De base, la librairie est dans un mono repository NX dans lequel il s'y trouve toutes les librairies nécessaires aux différents projets. Pour



l'instant le projet est dans un repository à part. Pour permettre d'appeler les services nécessaires, l'import de cette librairie est fait à la main. Mais pour simplifier une future migration sur le mono repository, elle est référencée dans la configuration pour l'entier du projet dans le fichier `tsconfig.json`. Lors d'une future migration, il suffira de changer le chemin d'accès à celui du monorepo.



```
// va-ngx-shared\src\index.ts
export * from './lib/helpers/date-helpers';
export * from './lib/helpers/form-group-model';
export * from './lib/helpers/html-helpers';
export * from './lib/models/dialog-icon.enum';
export * from './lib/services/local-storage-service/local-storage.service';
export * from './lib/services/session-storage-service/session-storage.service';
```

Figure 17 : Contenu de librairie que l'on souhaite intégrer dans l'application



```
// Intégration de la librairie dans le fichier tsconfig.json
"paths": {
  "@va-ngx-shared": ["src/libs/va-ngx-shared/src/index.ts"]
},

// Import simplifié des models / services depuis la librairie au sein de l'application
import { FormGroupModel, LocalStorageService } from '@va-ngx-shared';
```

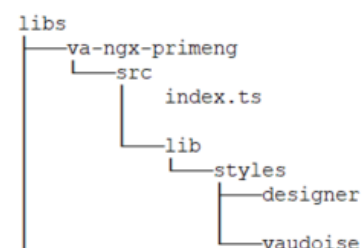
Figure 18 : Implémentation de la librairie contenant des composants, services, models, ... utile à l'application

Le contenu de la librairie ne se trouve pas en entier mais on y importe uniquement ce qui est utilisé au sein de l'application sans y apporter de modification afin de garder une cohérence entre celle présente dans le monorepo et celle dans le projet HiO. Ceci est dû à des dépendances dans l'application amenant des erreurs inutiles ainsi qu'un alourdissement non nécessaire.

3.1.4 Librairie va-ngx-primeng

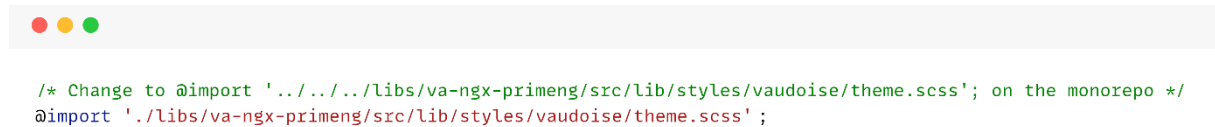
L'application utilise PrimeNG, une bibliothèque de composant UI flexible et accessible pour Angular. Les composants que l'application peuvent être la dropdown, la zone de dépôt des fichiers (file dropzone) ou encore le popup de confirmation de reprise du formulaire. Cette librairie de composant s'installe directement dans l'application grâce au package NPM (voir [PrimeNG – Installation](#)).

Cette librairie ajoute également sa propre couche de style qui ne correspond pas à la charte Vaudoise Assurances. La librairie va-ngx-primeng permet de modifier les composants et d'y ajouter une sur-couche de style. Dans le projet, le



contenu ajouter par la librairie ne sont que les fichiers de style permettant de modifier les contenus PrimeNG ajouté dans le projet HiO FNoL.

Les fichiers de style présent dans la librairie se trouve dans un fichier *themes.scss*. Il suffit de l'importer dans le fichier de style *styles.scss* du projet pour pouvoir l'utiliser dans l'application.



```
/* Change to @import '../libs/va-ngx-primeng/src/lib/styles/vaudoise/theme.scss'; on the monorepo */  
@import './libs/va-ngx-primeng/src/lib/styles/vaudoise/theme.scss';
```

Figure 19 : Import de la librairie de style dans l'application

3.1.5 Etape dans le formulaire

L'ajout du système d'étape fait partie du second changement le plus massif au sein du projet. Grâce à l'ajout du service pour la gestion de création du formulaire et de gestion d'état, l'implémentation des étapes ne requiert pas de grand *refactoring* de l'application.

Cependant, un des choix à faire est de savoir comme afficher le composant de chaque étape dès lors que l'on souhaite l'atteindre tout en cachant ceux que l'on ne souhaite pas afficher. Une des possibilités serait de garder la forme actuelle avec le composant principal qui appelle tous les composants de chaque étape nécessaire.



```
<hf-insurance-holder-step></hf-insurance-holder-step>  
<hf-damage-event-step></hf-damage-event-step>  
<hf-damage-additional-info-step></hf-damage-additional-info-step>
```

Figure 20 : Appel des composants d'étapes dans le composant principale

Avec cette solution, dès lors que l'on détecte quelle étape à afficher, on cache l'élément du DOM ne correspondant pas à cette étape pour ne plus les afficher. A chaque changement d'étape, on récupère la prochaine étape pour cacher l'ancienne et afficher la nouvelle. Cette solution est viable mais n'offre pas de résultats satisfaisants. Par exemple, le *FormGroup* qui contient les *FormGroup* de chaque étape sera instancié avec les tous les contenus et finira inutilement dans le local storage.

Angular offre des solutions capables d'obtenir un résultat convenable. Par exemple, le système de *routing* permet d'associer un composant à une route de l'URL. Grâce aux routes, il est possible d'associer chaque étape à une URL, ce qui évite de devoir instancier tous les composants mais de le faire uniquement à chaque fois qu'une route appelée spécifique est appelée et d'ajouter le *FormGroup* à celui principal.

Angular offre également un système de protection de route nommé les *guards*. Grâce aux *guards*, il est possible, dans notre cas d'utilisation, de bloquer l'accès aux routes des prochaines étapes en vérifiant que les formulaires relia à la route soit bien valide.

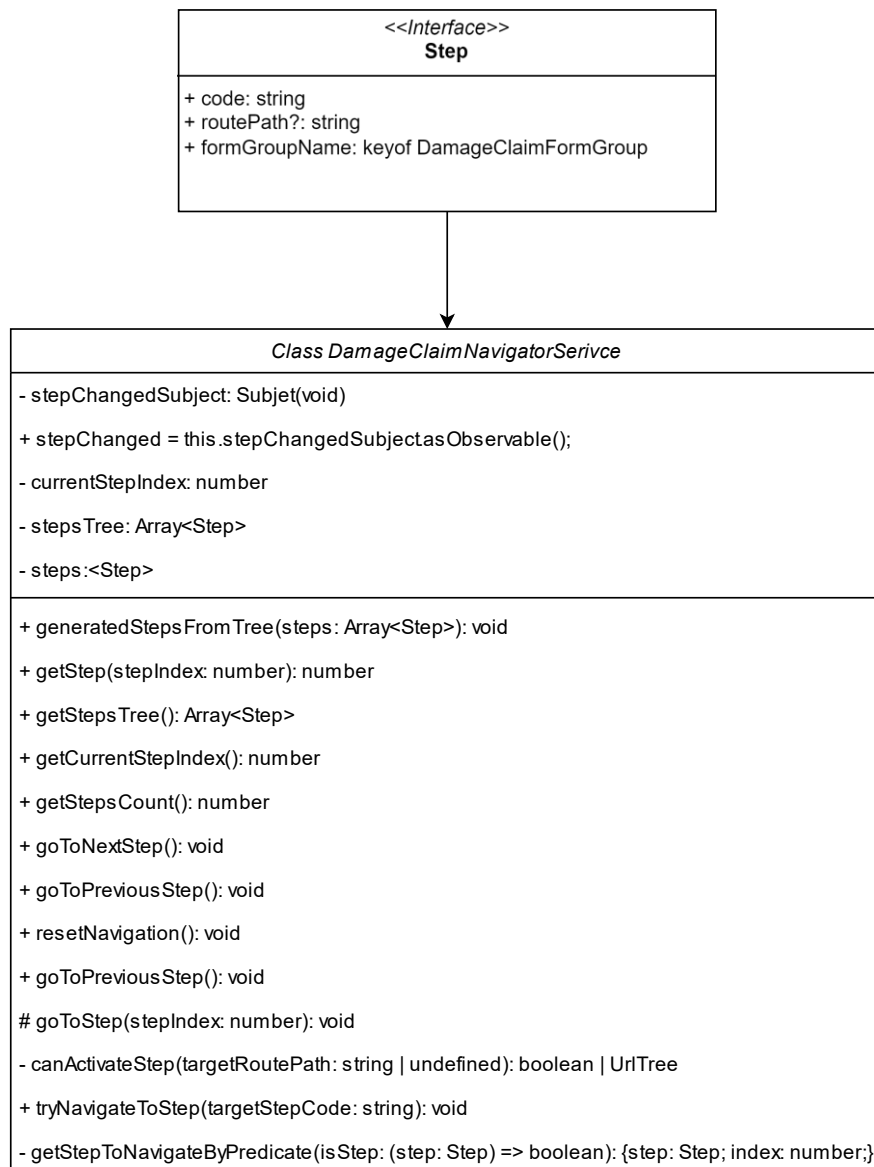
Si ce n'est pas le cas, la route est alors bloquée pour permettre de corriger une étape non-correct.



```
// Liste des routes enfants de la route parent avec les guards.
{
  path: '',
  component: DamageClaimComponent,
  children: [
    {
      path: '',
      redirectTo: DamageClaimRoute.insuranceHolderStepRoute,
      pathMatch: 'full',
    },
    {
      path: DamageClaimRoute.insuranceHolderStepRoute,
      component: InsuranceHolderStepComponent,
    },
    {
      path: DamageClaimRoute.damageEventStepRoute,
      component: DamageEventStepComponent,
      // Guards
      canActivate: [canActivateStep],
      runGuardsAndResolvers: 'always',
    },
    {
      path: DamageClaimRoute.additionalInfoStepRoute,
      component: DamageAdditionalInfoStepComponent,
      // Guards
      canActivate: [canActivateStep],
      runGuardsAndResolvers: 'always',
    },
  ],
},
```

Figure 21 : Définition des routes et des Guards des étapes du formulaire

Afin de pouvoir gérer la liste d'étape à parcourir et effectuer les changements d'étape, il faut, comme pour la gestion de l'état du formulaire, un service qui créera une liste des futures étapes avec la route à atteindre à chaque changement d'étapes et le nom du *FormGroup*. A l'intérieur de ce service, il existe différentes méthodes permettant de passer à la prochaine étape, l'étape précédente ou encore réinitialiser la navigation. Chaque action est rattachée à un bouton permettant d'avancer ou reculer dans les étapes dans le formulaire.

Figure 22 : UML service *DamageClaimNavigatorService*

3.1.6 Champ date

Le champ date utilisé dans le VM FNoL est un composant personnalisé créé par la Vaudoise Assurances ajouté dans le package npm *va-ext-front*. Dès lors que la librairie est installée, il ne suffit que d'appeler le composant `<va-datepicker>` dans la page HTML en y ajoutant les valeurs minimums et maximales possible.

Une des adaptations nécessaires à la suite de l'ajout fut le type du *FormControl* relié aux champs de date passant de type *string* à un type *date*. Il fut également nécessaire d'ajouter des nouveaux *Validators* sur les *FormControl* afin de bloquer, par exemple, la date d'anniversaire

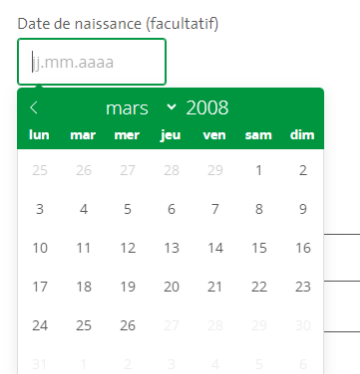


Figure 23 : Démonstration du date picker

minimal à celle d'une personne ayant 18 ans. Ces adaptations sont nécessaires au bon fonctionnement du composant appelé.

3.1.7 Infobulle

Les infobulles proviennent de la librairie PrimeNG sous le nom de tooltip. Ces tooltips doivent être importés dans les modules du projet afin de pouvoir être utilisés et appelés dans les composants.

Le contenu des infobulles m'a été fourni par mon *Product Owner* ainsi que les traductions de ces dernières. Le tout a été ajouté aux fichiers `.json` contenant les textes présents sur l'application.

3.1.8 Déploiement sur l'environnement DCI

Malgré une demande d'accès à Jenkins, la plateforme de build, test et déploiement de l'application, les accès ne me sont toujours pas donnés et il m'est donc impossible de le faire moi-même.

Afin de pouvoir le faire, j'ai pu demander à mon chef de projet de le faire à ma place après que j'ai terminé d'intégrer les changements dans le répertoire et que la construction de l'application soit validée après que les tests ont eu été soumis.

3.2 Description des tests effectués

Comme expliqué dans la rubrique [Stratégie de test](#), les tests présents dans l'application permettent de s'assurer que des futures modifications ne puissent interférer dans le bon fonctionnement de l'application en modifiant une zone non souhaitée. Une partie des tests ayant été mis en place ont été intégrés dans les nouveaux services et guards. Ces derniers sont devenus des acteurs majeurs au bon fonctionnement de l'application, et garantir qu'il ne soit modifié dans leurs logiques ce qui peut mener à des erreurs dans l'application permet de couvrir des futures erreurs.

Des tests ont également été implémentés au sein des composants de l'application permettant de garantir le bon fonctionnement de ces derniers.

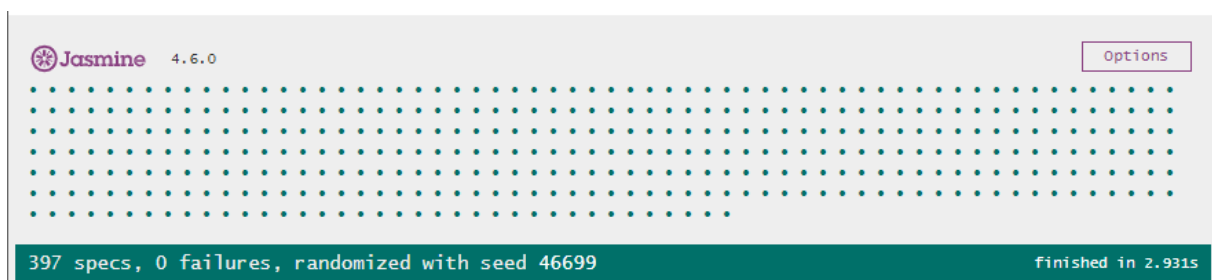


Figure 24 : Tests soumis et valide

3.3 Erreurs restantes

3.3.1 Persistance de données

3.3.1.1 Description détaillée

L'ajout de la persistance de données n'est pas entièrement fonctionnel. Dans la majorité du travail fourni, les données sont bels et biens sauvegardées et récupérées. Ce qui pose problèmes ce sont certains champs qui actuellement ne sont pas en mesure de soit formater correctement le contenu récupérer. C'est le cas du champ téléphone qui récupère le numéro enregistré depuis le local storage dans un format qu'il n'accepte pas ce qui le fait avoir une erreur. Ou alors soit le champ date comme celui de la date de naissance de l'assuré qui n'est pas capable de récupérer afin de la réafficher dans le formulaire. Les bugs que je viens de citer sont des bugs connus sur les composants et ne font donc pas partie du scope de mon TPI.

Screen des erreurs

3.3.1.2 Conséquences

Les conséquences sont que l'expérience utilisateur est amoindrie. Dès lors que l'on souhaite récupérer le formulaire, les utilisateurs sont forcés de devoir retaper ou d'interagir avec des champs qu'ils ont déjà remplis. Ceci nous fait perdre en fluidité.

3.3.1.3 Actions envisagées

Même si ces erreurs sont connues et ne font pas partie du scope de mon TPI puisque ces erreurs ne sont pas de moi directement. Malgré tout, il perturbe le fonctionnement du formulaire. Ces mêmes composants sont aussi utilisés sur d'autres applications, donc pouvoir les améliorer n'est pas négligeable. Premièrement, plusieurs tickets JIRA ont été créer afin de garder une trace des bugs et pouvoir discuter des bugs et des actions possibles à performer.

3.3.2 Les étapes dans le formulaire

3.3.2.1 Description détaillée

Le problème relié à l'étapes du formulaire provient du champ téléphone. Comme expliqué précédemment, le champ formulaire ne reçoit pas le numéro avec un format qu'il accepte, ce qui refait monter une erreur. Lorsque l'on souhaite retourner sur la route d'une étape, le formulaire, s'il détecte une erreur dans une des routes, force le retour sur cette page pour la corriger. Donc si à la 3^{ème} route l'on souhaite y retourner, étant donné que le numéro de téléphone se trouve à la 1^{ère} route, l'utilisateur est forcé de retourner à la première route pour corriger le champ du formulaire.

3.3.2.2 Conséquences

Il s'agit encore un problème perturbant l'expérience utilisateur devant forcer l'utilisateur à repasser sur les champs qu'il a déjà parcouru pendant sa déclaration.

3.3.2.3 Actions envisagées

Même ticket ouvert que celui du téléphone.

3.3.3 Infobulles

3.3.3.1 Description détaillée

Le composant PrimeNG servant à l'intégration de l'infobulle ne permet pas de modifier sa taille. Etant donné que les textes à afficher possèdent pas mal de caractères, leurs affichages n'est pas idéal. Une requête à été faites par mon *Product Owner* afin de le modifier mais malheureusement la solution trouvée provoque plus de bug.

Également, les infobulles ne sont pas activables avec les touches du clavier comme *tab* ce qui n'offre pas non plus une accessibilité et une navigation propre.

3.3.3.2 Conséquences

L'affichage n'est pas idéal pour une lecture fluide.

3.3.3.3 Actions envisagées

Vérifier si une customisation du composant est envisageable et si ce n'est pas le cas alors voir pour en créer un nouveau depuis une base comme le popup de confirmation. Un ticket a été ouvert pour en discuter après le TPI.

3.4 Liste des documents fournis

Les documents fournis sont :

- Le rapport du TPI (PDF)
- Le journal de travail (PDF et intégré au rapport : [Journal de travail](#))
- Le résumé (intégré au rapport : [Résumé du rapport du TPI / version succincte de la documentation](#))
- Le planning Initial (PDF et intégré au rapport : [Planification initiale](#))
- Le planning Exécutif (PDF)
- Code source de l'application (.zip)

Les fichiers seront disponibles depuis un répertoire OneDrive téléchargeable. La Vaudoise Assurance ne permet pas d'envoyer des fichiers .zip afin d'éviter les fuites de données.

4 Conclusions

Au cours de ce TPI, j'ai pu mettre en pratique les 4 années de formation que j'ai eu au cours de mon apprentissage au sein de la Vaudoise Assurances. Ce projet m'a permis de progresser et d'en apprendre plus.

4.1 Objectifs

Parmi les différents objectifs du TPI, j'ai pu en mettre en place plus de 90%. Malgré la grosse phase de refactoring pour la création du formulaire et la sauvegarde de l'état, j'ai pu maintenir un bon flux de travail. Même si des bugs sont encore persistant, tous les objectifs ont été atteints, que ce soient les étapes, la persistance, les champs de date ou encore les infobulles. Le 10% restant concerne principalement la résolution de bug au sein de l'application et tester sur le l'environnement DCI plus en détails.

4.2 Points positifs / points négatifs

4.2.1 Le positif

Premièrement le projet était un vrai petit challenge qui nécessitait réflexion. Ce genre de projet sont ceux que j'aime le plus faire, à devoir se creuser la tête et expérimenter en pesant le pour et le contre de chaque solution possible. Deuxièmement, les connaissances apprises sont multiples, au cours de ce TPI, j'ai pu intégrer des librairies interne à l'entreprise tout en devant en intégrer des externes et les faire cohabiter ensemble dans le projet. Et finalement, le point où je me suis nettement amélioré c'est l'écriture de tests unitaires. Au début en écrire me prenait du temps et il m'arrivait d'avoir des erreurs mais sur les 4 derniers, il m'était plus fluide de les concevoir, de les écrire et de les arranger en cas de bug.

J'ai également appris à devoir documenter mes réflexions tout en les justifiant, le but étant de permettre une transmissibilité fluide du projet à d'autres futurs acteurs du projet.

4.2.2 Le négatif

Le négatif du projet ne concerne pas le directement le projet mais malheureusement la phase de refactoring m'ayant pris plus de temps que prévu. Ce qui fait totalement parti d'un projet. Grâce à cette « épreuve », j'en ressors avec de nouveaux outils et bonnes pratiques afin de réussir à surmonter plus vite ce style de problèmes à l'avenir.

4.3 Difficultés rencontrées

Sur la documentation et la manière dont la mettre en place afin de pouvoir bien expliquer les différents points et les différentes pistes de réflexions m'a amené plus de conflit que prévu mais après discussion avec des collègues et consulter les anciennes documentations rédigées lors des cours professionnels EPSIC, j'ai pu trouver une architecture de document me convenant.

L'intégration des composants externes m'a aussi posé des problèmes dû à un petit manque de connaissances. Mais après quelques communications avec des collègues et mon chef de projet, j'ai pu facilement m'en sortir seul.

4.4 Evolutions et suites possibles du projet

Le projet peut évoluer de plusieurs manières. Voici quelques changements possibles qui pourraient être fait dans l'avenir.

4.4.1 La correction des bugs

Premièrement et le plus important, il s'agirait de corriger les bugs répertoriés présent dans l'application et qui freinent l'expériences des utilisateurs. Tout est documenté dans ce chapitre [Erreurs restantes](#)

4.4.2 Implémentation des modifications sur la partie Responsabilité Civile

Comme expliqué, l'application contient 2 formulaires à l'intérieur. Celui des dommages liés à l'assurance ménage mais également celui des dommages liés à la responsabilité civile. Le but serait d'ajouter un d'implémenter les mêmes modifications dans le celui de la responsabilité afin d'aligner les 2 formulaires. Le projet de ce TPI ayant déjà ouvert les pistes de réflexions, il ne suffit que de reproduire les mêmes modifications que ce celles présentes sur celui de la déclaration de sinistre ménage.

4.4.3 Ajout d'un composant de navigation.

Désormais le formulaire possède différentes étapes et les affiche une par une. Pour l'utilisateur cela décharge l'interface en traitant étape par étape sa déclaration. Mais avec l'interface actuelle, il n'est pas capable de savoir dans quelle étape il se trouve ou combien d'étape il lui reste. De plus, il n'y a pour l'instant que 3 étapes, donc revenir à la première cela est relativement rapide. Mais imaginons que dans le futur le formulaire récupère 5 nouvelles étapes, la navigation n'est plus optimale. La solution serait d'ajouter un composant de navigation avec la possibilité de voir l'étape en cours et celle à venir. Ce qui permettra à l'utilisateur de naviguer plus librement sans devoir cliquer de nombreuses fois sur les boutons suivant ou précédant.

4.5 Bilan personnel

Durant cette période de TPI de 10 jours, j'ai eu du plaisir à pouvoir travailler comme un employé et non plus comme un apprenti et d'avoir pu montrer mes preuves à travers ce projet. Malgré un départ un peu plus compliqué, je suis fier de moi d'avoir pu remplir, à mon avis, plus de 90% des objectifs du cahier des charges tout en arrivant à améliorer l'application actuelle pour de futures améliorations possibles. Ce qui me fait encore plus plaisir, c'est que c'est un projet avec un but qui sera destiné à de la production et non pas comme un simple exercice. Cela permet d'encore plus valorisé le travail que j'ai fourni.

5 Annexes

5.1 Résumé du rapport du TPI / version succincte de la documentation

Situation de départ

La mission principale de ce TPI était d'améliorer l'expérience utilisateur sur les formulaires de déclarations de sinistre ménage et de responsabilité civile présent sur le site de la Vaudoise Assurances. Ce formulaire est un point central dans la stratégie digitale de l'entreprise puisqu'il sert de point d'entrée pour permettre la relation entre un sinistré et l'assureur. Les modifications visées serviront à fluidifier les interactions de l'utilisateur afin de faciliter son utilisation. Pour parvenir à ce but, j'ai dû en premier analyser le fonctionnement et l'architecture du formulaire afin de pouvoir prévoir et anticiper les futures modification, implémentation ainsi que les potentiels points bloquants.

Mise en œuvre

A la suite de la phase de l'analyse, j'ai dû mettre en place un nouveau système capable de gérer la création du formulaire de manière centralisée afin d'être synchronisé avec les standards des autres applications Vaudoise tout en performant les modifications nécessaires afin de parvenir aux objectifs du cahier des charges. A la suite de ces changements j'ai dû implémenter le système de gestion de l'état du formulaire tout en le sauvegardant. Afin de simplifier le déroulement de la déclaration, l'ajout d'étapes dans le formulaire a dû être fait accompagnés des boutons de navigations. Et finalement des modifications de champs afin de les rendre plus simple et agréable à utiliser ont dû être implémenter comme la modification des champs dates ou l'ajout d'infobulles.

Résultat

Les nouveaux ajouts dans le formulaire permettent aux futurs utilisateurs de pouvoir, en cas de fermeture intentionnelle ou non de la page, de récupérer l'état du formulaire avant qu'ils ne le quittent. Il ajoute également un système d'étape ce qui rend plus léger l'interfaces et permet d'effectuer sa demande étape par étape. Différents champs ont également été modifié afin d'être plus clairs comme le champ de la date qui affiche directement un calendrier interactif permettant de choisir une date avec les contraintes de validation nécessaires. D'autres champs bénéficient d'une infobulle permettant d'ajouter du détail à ce dernier. Pour finir la zone de dépôt de fichiers possède désormais un masque gris lorsqu'un fichier est en train d'être drag & drop. Tous ces changements permettent de fluidifier l'expérience utilisateur.

En ce qui concerne la partie développement les changements ont été intégrés avec la volonté de synchronisé le fonctionnement de l'application avec celui des autres formulaires facilitant l'intégration de nouvelles fonctionnalités.

5.2 **Sources – Bibliographie**

5.2.1 **Webographie**

Introduction to forms in Angular – Angular : <https://angular.io/guide/forms-overview>

FormGroup – Angular : <https://angular.io/api/forms/FormGroup>

FormControl – Angular : <https://angular.io/api/forms/FormControl>

Introduction to services and dependency injection – Angular :
<https://angular.io/guide/architecture-services>

Angular components overview – Angular : <https://angular.io/guide/component-overview>

Angular Routing – Angular : <https://angular.io/guide/routing-overview>

CanActivate – Angular : <https://angular.io/api/router/CanActivate>

ConfirmDialog – PrimeNG : <https://primeng.org/confirmdialog>

Angular PrimeNG ConfirmDialog ConfirmationService – Geek for Geeks :
<https://www.geeksforgeeks.org/angular-primeng-confirmdialog-confirmation-service/>

Dropdown – PrimeNG : <https://primeng.org/dropdown>

Tooltip – PrimeNG : <https://primeng.org/tooltip>

Intro to Nx – Nx : <https://nx.dev/getting-started/intro>

5.2.2 **Illustration**

Figure 1 : Arborescence app	9
Figure 2 : Arborescence features	9
Figure 3 : Etape "Informations concernant le déclarant et l'assuré" du formulaire	10
Figure 4 : Etape "Information concernant le sinistre" du formulaire	10
Figure 5 : Etape "Informations complémentaires" du formulaire	10
Figure 6 : Exemple de mock dans les tests unitaires.....	12
Figure 7 : Schéma d'explication de création du formulaire	15
Figure 8 : Schéma d'explication de la nouvelle implémentation théorique du formulaire	16
Figure 9 : Schéma d'explication de la nouvelle implémentation choisie du formulaire	17
Figure 10 : Code pour la création ou la récupération du FormGroup	17
Figure 11 : Création ou récupération du formulaire depuis l'état dans le composant parent ou dans celui d'une étape.....	18
Figure 12 : Création du FormGroup selon l'étape choisie.....	18
Figure 13 : Sauvegarde de l'état du Formulaire.....	19
Figure 14 : Souscription des modifications de l'état du FormGroup dans le constructeur du service	19
Figure 15 : Contrôle si un état existe et affiche le pop-up.....	20
Figure 16 : Pop-up affiché à l'utilisateur.....	20

Figure 17 : Contenu de librairie que l'on souhaite intégrer dans l'application	21
Figure 18 : Implémentation de la librairie contenant des composants, services, models, ... utile à l'application.....	21
Figure 19 : Import de la librairie de style dans l'application	22
Figure 20 : Appel des composants d'étapes dans le composant principale.....	22
Figure 21 : Définition des routes et des Guards des étapes du formulaire	23
Figure 22 : UML service DamageClaimNavigatorService.....	24
Figure 23 : Démonstration du date picker.....	24
Figure 24 : Tests soumis et valide	25

6 Journal de travail

6.1 Jour 1 – 11 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	1h	Découverte et prise de connaissance du cahier des charges du TPI
09:00 – 10:00	1h	Entretien avec l'expert et début de rédaction du planning initial
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	1h30	Rédaction du planning initial
11:00 – 12:00	30min	Création du Journal de travail et de sa mise en page
12:00 – 12:30	30min	REPAS
12:30 – 13:00	30min	Création du Trello pour avoir un suivi des tâches à faire de celles en cours
13:00 – 14:00	2h	Création et rédaction du rapport de TPI
14:00 – 15:00		
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	30min	Création et rédaction du rapport de TPI
16:00 – 17:24	1h24	Installation de dépendances nécessaire au fonctionnement du projet. Erreur d'affichage avec style manquant. Tentative de résolution en mettant à jour les packages, mais erreur persistante. Demande d'aide au CdP et résolution en remarquant une erreur dans l'URL du proxy ayant changé de dev à dci. Commit et Pull Request crée pour corriger un bug risquant d'impacter tout le monde.
TOTAL	8h24	

6.1.1 Bilan de la journée

Création des différents documents pour les rendus hebdomadaires et celui final. Leurs création et modification fut plus rapides que prévue ce qui m'a laissé du temps pour les commit sur un repository GitHub et un dossier sur le Cloud OneDrive (expliqué dans le rapport chapitre organisation).

J'ai pu prendre le temps d'installer les dépendances et de faire fonctionner la solution et d'explorer son fonctionnement ainsi que son implémentation sur le site de la Vaudoise Assurances. Nous avons remarqué, moi et mon Chef de Projet, une erreur quant à la demande des ressources de style ce qui empêchait l'app de s'afficher correctement. Ceci était dû à une migration des API incluant un changement des URLs. Une Pull Request a été faite et est en cours de revue.

6.2 Jour 2 – 12 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	1h	Analyse du système de persistance de donnée dans le FNoL VM
09:00 – 10:00	45min 15min	Analyse du système de persistance de donnée dans le FNoL VM Question et discussion sur la manière d'implémenter les librairies nécessaires afin de garantir une A COMPLETER
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	10min 30min	Modification du Journal de travail en fonction des retours reçus par mail et complétions des actions faites durant la matinée. Contrôle des
11:00 – 12:00	1h	Analyse du fonctionnement de création du formulaire pour voir s'il est possible de répliquer le même fonctionnement. Ce système pourrait simplifier l'ajout du système de persistance de données et d'étapes de formulaire.
12:00 – 12:30	30min	REPAS
12:30 – 13:00	30min	Analyse du fonctionnement de création du formulaire pour voir s'il est possible de répliquer le même fonctionnement. Ce système pourrait simplifier l'ajout du système de persistance de données et d'étapes de formulaire.
13:00 – 14:00	2h	Tentative d'implémentation du système de persistance des données avec modifications du système de création de formulaires. Malgré l'analyse et le comparatif, l'architecture implémentée m'a induit en erreur ce qui m'a fait prendre du retard. Pour corriger ce problème, je vais refaire une phase d'analyse et proposer ma solution au chef de projet afin de voir si le chemin est correct.
14:00 – 15:00		
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	45min	Suite de tentative d'implémentation
16:00 – 17:35	1h	Analyse du fonctionnement du FNoL VM

	35min	Modification du journal.
TOTAL	8h51	

6.2.1 Bilan de la journée

Grâce au temps gagné hier, j'ai pu prendre plus de temps dans la phase d'analyse du fonctionnement des autres projets similaires duquel je peux m'inspirer. Malheureusement, mon objectif de finir la partie d'implémentation de la persistance de donnée n'a pas pu être abouti en entier, mais j'ai tout de même une base de laquelle partir.

Je prévois demain de finaliser l'analyse et de proposer une manière d'aborder les différentes problématiques telles que la création du formulaire avec Angular et de voir s'il est possible de recréer la philosophie intégrée dans le FNoL VM. Une adaptation du planning doit être faite.

6.3 Jour 3 – 13 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	1h45 15min	Analyse du fonctionnement de la persistance sur le VM FNoL. Refactoring de la logique de création du formulaire. Actuellement, chaque composant créait son propre formulaire en l'envoyant au formulaire parent. Cette solution empêche de gérer l'état de chaque champ et complique l'ajout de la persistance et des étapes du formulaire, ce qui nécessite un remaniement de la logique de création.
09:00 – 10:00		Modification du planning pour correspondre avec ces phases de refactoring plus conséquentes que prévu
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	20min	Discussion et démonstration de la solution choisie avec le chef de projet afin de voir ce qui est à faire. On constate que le refactoring est plus grand que prévu.
11:00 – 12:00	1h	Poursuite de la phase de refactoring en ajoutant un système de création de formulaires généralisé à l'application. Facilite la gestion de l'état du formulaire pour le créer, détecter les changements attribués par l'utilisateur et en retrouver l'état. L'étape visée est la création du formulaire.
12:00 – 12:30	30min	REPAS
12:30 – 13:00	15min 1h45	Discussion avec le Product Owner quant au placement des infobulles. Les textes en français sont fournis et les traductions ont été demandées et en attentes d'être transmises. Une demande supplémentaire m'a été demandée, celle de corriger une petite faute d'orthographe.
13:00 – 14:00		Implémentation du système de création du formulaire généralisé. Il est fonctionnel et le crée tout en l'injectant dans le composant parent de l'application. Suppression de la méthode de création précédente en y ajoutant la nouvelle. Pour ce faire, les composants d'étapes enfants reçoivent en paramètre le groupe de formulaire leur correspondant. Malheureusement, des erreurs se répliquent en cascade ralentissant ma progression.

14:00 – 15:00	1h	Après modification, je constate un problème risquant de complexifier la phase d'implémentation de la détection de changement d'état et de sauvegarde ainsi que celle des étapes. Cela est dû au système de création de formulaires. Il doit être modifié afin de faciliter l'implémentation future de ces 2 autres fonctionnalités.
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	2h09	Modification du rapport de TPI
16:00 – 17:24		Modification du Journal de travail
TOTAL	8h29	

6.3.1 Bilan de la journée

À la suite des analyses faites durant le 2^e jour, j'ai compris que le système actuel pour la création du formulaire n'est pas viable pour une implémentation de persistance de donnée et d'ajout d'étape sans ajouter plus de complexité. Après avoir réfléchi de mon côté et implémenté un morceau de logique afin d'en faire une démonstration à mon chef de projet pour lui indiquer ma nouvelle direction et l'améliorer avec son expertise.

Nous remarquons assez rapidement que cette implémentation sera plus chronophage que prévu, mais que la valeur ajoutée que ce soit sur l'apprentissage obtenu, mais le grain de temps sur l'implémentation des fonctionnalités en vaut le jeu. Au cours de cette journée j'ai pu mettre en place le système de création de formulaires centralisé en 1 point et effectuer les modifications sur les composants enfants.

À la suite de ces changements (détails dans le tableau du jour), je constate que l'implémentation est correcte, mais qu'elle génère le formulaire complet. Le meilleur serait de créer le groupe général avec au sein un valeur de l'étape courante et l'ajout des groupes de chaque étape par le composant lui-même une fois qu'il est initialisé. Ce qui évite de mettre dans le local storage les contrôles d'une étape pas encore initialisée.

Les modifications et l'avancement seront faits demain matin avant d'en rediscuter avec le chef de projet.

6.4 Jour 4 – 14 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	2h	Modification du système de génération pour faciliter l'implémentation de la persistance de donnée et des étapes dans le formulaire. Ajout de la librairie partagée sur le projet.
09:00 – 10:00		
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	20min	Discussion et mise au point avec le chef de projet. A la suite de la découverte des modifications plus grande, il me propose de s'occuper des tests unitaires et de les aligner avec la nouvelle version le vendredi 15 mars lors de ma journée de cours.
11:00 – 12:00	1h	Modification générale du système de génération dans les composants
12:00 – 13:00	1h	REPAS
12:30 – 13:00	2h30	Ajout du système de persistance. Système fonctionnel sur les premiers champs du formulaire. Modification sur le reste du formulaire.
13:00 – 14:00		
14:00 – 15:00		
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	2h09	Modification du rapport de TPI
16:00 – 17:54		Modification du journal
TOTAL	8h29	

6.4.1 Bilan de la journée

Réussite de la création du système de persistance de donnée à la suite du refactoring de la génération du formulaire. Ce qui offre une plus grande flexibilité quant à la modification de l'application. Gain de temps dans les futures implémentations. À la suite des discussions avec le chef de projet, il est convenu qu'il m'aidera dans la mise en place des tests unitaire en reprenant la suite le vendredi 14 mars lors de mes cours étant donné que les modifications se retrouvent agrandies étant donné l'implémentation de l'application avant mes modifications.

6.5 Jour 5 – 18 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	1h	Ajout du pop-up de récupération de données
09:00 – 10:00	30min	Ajout du pop-up de récupération de données
	30min	Visite du second expert
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	1h40	Ajout du pop-up de récupération de données. Le composant s'affiche avec un défaut au niveau du style, mais n'est pas critique. Sera modifié dans un second temps.
11:00 – 12:00		
12:00 – 12:30	30min	REPAS
12:30 – 13:00	1h	Implémentation des tests unitaires concernant la récupération de l'état et du pop-up de confirmation
13:00 – 14:00	1h	Résolution des erreurs dans le test
14:00 – 15:00	1h	Analyse du fonctionnement du système d'étape sur le VM FNoL
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	30min	Début d'implémentation des composants d'étapes
	15min	Modification du journal
16:00 – 17:24	1h24	Modification du rapport de TPI
TOTAL	8h49	

6.5.1 Bilan de la journée

Au cours de cette journée, le système de sauvegarde d'état et de récupération est, cette fois-ci, fonctionnel avec un pop-up demandant à l'utilisateur s'il souhaiterait reprendre ou non sa déclaration afin de pouvoir récupérer les données sauvegardées dans le local storage. Si ce dernier ne le souhaite pas, sa déclaration est vidée.

J'ai pu ensuite prendre du temps pour analyser le fonctionnement du système d'étape et voir comme l'implémenter au sein de l'application. Avec le temps restant, j'ai pu déjà démarrer l'implémentation.

6.6 Jour 6 – 19 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	1h	Nettoyage final du code et pull request de la persistance de donnée
09:00 – 10:00	1h	Implémentation des étapes dans le formulaire
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	10min 1h30	Discussion avec le chef de projet sur la méthode choisie pour l'ajout des étapes
11:00 – 12:00		Implémentation des étapes dans le formulaire. Étape au sein du formulaire fonctionnel, mais ne revient pas sur la page en cours lors de la récupération des données
12:00 – 12:30	30min	REPAS
12:30 – 13:00	30min	Analyse du fonctionnement de récupération de l'étape courante dans le VM FNoL
13:00 – 14:00	1h	Modification pour parvenir à revenir sur la page courante. Fonctionnel, mais il y a un problème lorsque l'on ne souhaite pas reprendre la déclaration. La page affichée n'est pas la première, mais toujours celle courante.
14:00 – 15:00	30min	Analyse du fonctionnement
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	45min	Modification du rapport de TPI
16:00 – 17:24	1h04	Correction de la pull request
	20min	Modification du journal
TOTAL	8h49	

6.6.1 Bilan de la journée

Avancée satisfaisante, l'ajout du système d'étape s'est fait avec plus de facilité grâce aux modifications précédentes. Presque, il me reste qu'à corriger quelques bugs, comme la page qui ne revient pas sur la première étape, et ajouter des tests unitaires avant de pouvoir faire une pull request de mon code.

6.7 Jour 7 – 20 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	2h	Implémentation des suggestions de la pull request pour l'ajout de persistance. Discussion avec les différentes personnes qui ont ajouté des commentaires.
09:00 – 10:00		
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	40 min	Avancée sur la finition d'ajout des étapes
11:00 – 12:00	1h	Découverte d'un bug sur la reprise des données sur les dropdown du formulaire. En attendant, début de la tâche d'intégration des nouveaux champs date. Intégration réussie, mais quelques problèmes à régler plus tard.
12:00 – 12:30	30min	REPAS
12:30 – 13:00	30min	Discussion avec le chef de projet sur le bug trouvé pour les dropdown. Après analyse proposition d'ajout du composant similaire à celui du VM FNoL qui permet d'être défini plus facilement
13:00 – 14:00	1h	Ajout de la dropdown ainsi que de la librairie de style va-ngx-primeng dans le projet permettant de donner style aux composants PrimeNG utilisé
14:00 – 15:00	1h	Problème de style apparu à la suite de l'ajout de la librairie de style sur la zone de dépôt des fichiers. Investigation afin de trouver le problème.
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	45min	Nettoyage du code et unifications des modifications apportées
16:00 – 17:24	1h00 30min	Modification du rapport de TPI Discussion avec un collègue sur des changements à apporter sur la pull request et sur les champs date erronés.
TOTAL	8h49	

6.7.1 Bilan de la journée

Résolution de plusieurs problèmes en 1 fois après avoir ajouté la librairie de style comme le style du popup de confirmation ou encore l'ajout du masque gris sur la zone de dépôt de fichier en drag & drop.

6.8 Jour 8 – 21 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	2h	Ajout des guards dans les routes de l'application pour éviter à une personne qui n'a pas fait le formulaire depuis l'étape 1 d'accéder aux autres étapes / routes.
09:00 – 10:00		
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	40min	Modification du style pour corriger la zone de dépôt de fichier.
11:00 – 12:00	1h	Après la phase de test, détection d'un bug causé par le champ du numéro de téléphone dans la page de l'étape 1 qui ne récupère pas la valeur avec un format qui lui plait. Ce champ détecte alors une erreur, ce qui dès lors ramène l'utilisateur à l'étape 1 pour la corriger.
12:00 – 12:30	30min	REPAS
12:30 – 13:00	30min	Discussion avec un collègue sur le problème et m'indique que le problème vient du composant et non de mes modifications. Un ticket a été créé pour pouvoir travailler dessus plus tard.
13:00 – 14:00	2h	Ajout de nouveaux tests unitaires dans l'application
14:00 – 15:00		
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	45min	Investigation sur les dates pour essayer de trouver la cause du problème. Donnée non retournée peut être causée à cause d'un problème de « parse » de la donnée.
16:00 – 17:24	1h24	Modification du journal Modification du rapport de TPI
TOTAL	8h49	

6.8.1 Bilan de la journée

Nouveau problème ajouté qui risque d'ajouter un bug jusqu'à ce que des modifications puissent être faites. Des tickets ont été créés afin de notifier le travail à faire et que les changements puissent être ajoutés.

6.9 Jour 9 – 25 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	1h	Ajustement des étapes du formulaire et des guards
09:00 – 10:00	1h	Discussion d'une solution pour le problème de la date qui ne se remplit pas dans le champ. L'erreur est connue et ne fait pas partie du scope du TPI. Un Ticket a été créé pour pouvoir en discuter afin d'améliorer le composant.
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	40min	Investigation sur l'erreur du champ de téléphone. Après discussion, problème connue comme le champ de la date. Création d'un ticket pour en discuter après le TPI.
11:00 – 12:00	1h	Création des derniers tests unitaires.
12:00 – 12:30	30min	REPAS
12:30 – 13:00	30min	Correction de la dernière
13:00 – 14:00	1h	Attente d'une réponse. Rédaction du rapport de TPI
14:00 – 15:00	1h	Analyse du fonctionnement du système d'infobulle. Implémentation dans les différentes langues Création des pull request pour la correction de la zone de dépôt de fichiers, de la modification des champs dates et de l'ajout des infobulles.
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	2h09	Rédaction du rapport de TPI
16:00 – 17:24		Modification du journal
TOTAL	8h59	

6.9.1 Bilan de la journée

Fin de l'ajout des derniers points du cahier des charges. Dernier objectif est la correction du code et de le déployer sur l'environnement DCI.

6.10 Jour 10 – 26 mars 2024

HEURE	DUREE	TRAVAIL EFFECTUE
08:00 – 09:00	2h	Correction des pulls request de la veille. Ajout des tests unitaires.
09:00 – 10:00		
10:00 – 10:20	20min	PAUSE
10:20 – 11:00	40min	Investigation et discussions concernant l'accessibilité des infobulles. N'ayant pas le temps un ticket a été créé afin de vérifier le problème et comment améliorer ces dernières.
11:00 – 12:00	30min	Ajout des dernières traductions Modification du rapport de TPI
12:00 – 12:30	30min	REPAS
12:30 – 13:00	30min	Correction de la dernière pull request et déploiement sur l'environnement DCI
13:00 – 14:00	2h	Modification du rapport de TPI
14:00 – 15:00		
15:00 – 15:15	15min	PAUSE
15:15 – 16:00	45min	Rédaction du cahier des charges
16:00 – 17:24	1h24	Corrections et préparation des livrables.
TOTAL	8h59	

6.10.1 Bilan de la journée

Correction des dernières pull requests. Déploiement sur l'environnement DCI.
Préparation des livrables.

7 Glossaire

A

Angular

est un framework pour clients, open source, basé sur TypeScript · 8

C

CLI

est une interface homme-machine dans laquelle la communication entre l'utilisateur et l'ordinateur s'effectue en mode texte · 8

F

FormControl

Suivi de la valeur et de l'état de validation d'un contrôle de formulaire individuel. · 11

FormGroup

Suivi de la valeur et de l'état de validité d'un groupe d'instances de FormControl. · 9

framework

n ensemble cohérent de composants logiciels structurels · 8

front-end

correspond aux productions HTML, CSS et JavaScript d'une page internet ou d'une application qu'un utilisateur peut voir · 8

G

GIT

est un logiciel de gestion de versions décentralisé. · 12

I

IDE

st une application logicielle qui aide les programmeurs à développer efficacement le code logiciel. · 13

L

local storage

est une technique d'enregistrement de données dans un navigateur web. · 11

P

pipeline CI/CD

est la combinaison des pratiques d'intégration continue et de livraison continue ou de déploiement continu. · 12

pulls requests

sont un mécanisme qui permet à un développeur de prévenir les membres de son équipe qu'il a terminé une fonctionnalité. · 13

R

refactoring

est l'opération consistant à retravailler le code source d'un programme informatique · 13

S

SPA (single page applications)

est une application web accessible via une page web unique. Le but est d'éviter le chargement d'une nouvelle page à chaque action demandée · 8

U

UML

est un langage de modélisation graphique à base de pictogrammes conçu comme une méthode normalisée de visualisation dans les domaines du développement logiciel et en conception orientée objet. · 14