



# Rapport de Stage

## Développeur Web et Web Mobile

### AFPA

20 Rue du Luxembourg  
59100 ROUBAIX

<https://afpa.fr>

*Période en centre de formation :*  
**24/08/2020 au 12/02/2021**

### INSY2S

51 Rue Marcel Hénaux  
59000 LILLE

<https://insy2s.com>

*Période de stage:*  
**15/02/2021 au 12/05/2021**

**Yoan DECONINCK**

## Sommaire

Remerciements.....	3
Introduction.....	4
Abstract.....	5

## INSY2S

Présentation de l'entreprise.....	6
Présentation de l'équipe.....	7
Présentation Projet ELM.....	8

## Environnement de travail

Outils Utilisés.....	12
- Outils de communication.....	12
- Outils de développement.....	13
Technologies utilisées.....	14
Organisation Scrum.....	16

## Projet ELM

Architecture.....	18
Module « Projets ».....	22
Module « Tableau Chiffrage ».....	32
Cahiers de tests.....	35

## Bilan

Bilan.....	37
------------	----

## Annexes

Annexes.....	38
--------------	----

## Remerciements

Premièrement, je tiens à remercier mon formateur Monsieur BENSARI Chakib pour sa pédagogie, son soutien et sa disponibilité lors de la formation en ces temps difficiles de crise sanitaire.

Je remercie également Monsieur TORCHE Nezar pour m'avoir permis d'effectuer mon stage dans son entreprise.

Je tiens à également remercier Monsieur MOLLET Peter pour sa formation accélérée sur les technologies utilisées dans l'entreprise ainsi que Madame ZIMMER Marine et Monsieur LEMOINE Jovany pour leur aide et disponibilité pendant tout le stage en tant que Scrum Master.

Et pour finir je tiens à remercier mes camarades de la Team Akatsuki avec qui j'ai effectué le stage ainsi que tous mes camarades de la session de formation de DWWM.

## Introduction

Ma formation de développeur web et web mobile a débuté en août. J'ai été initié pendant ma formation à plusieurs langages de programmation qui sont JavaScript ainsi que sa librairie JQuery, PHP et le framework Symfony ainsi que MySQL pour les requêtes en base de données.

Pour compléter cette formation, nous avons dû effectuer un stage de 3 mois environ. A notre arrivé dans l'entreprise INSY2S, nous avons eu une semaine pour apprendre React puis lors de la seconde semaine un petit projet qui reprend dans les grandes lignes le site allo-ciné (c'est-à-dire consulter des résumés de films avec possibilité de les ajouter en favoris) afin de nous familiariser davantage avec celui-ci.

A la suite de ces 2 premières semaines nous avons été assignés à un projet déjà existant qui est le projet ELM (Enterprise Learning Management). Le projet ELM est un site interne à l'entreprise qui sert pour les différents projets en cours, gérer le travail de ses employés ainsi que pour avoir une liste des stagiaires et formateurs. Nous étions 6 stagiaires affectés à ce projet.

Lors du stage notre travail était de nous concentrer sur la partie « Projets » du site ainsi qu'à la création d'un nouveau module « Chiffrage » .

Pendant ce stage j'ai pu utiliser des technologies apprises lors de ma formation (JavaScript, HTML, CSS) ainsi qu'une nouvelle technologie qui est React.js, une librairie JavaScript frontend.

## Abstract

My training course to become a web developer began in august. I was initiated during my training course to many programming languages that are JavaScript and its library JQuery, PHP and the framework Symfony and MySQL as well for the databases requests.

To complete this training course, we had to complete an internship of approximately three months. To our arrival to the company called INSY2S, we had a week to learn React. Then at the second week of our internship we had a small project to do, the website needed to follow the main lines of the website “Allo-ciné” (it means that we could be able to see the summary of a movie with the possibility to add them to our favorites list) to familiarize more with React.

Following those two weeks, we were assigned to a project already existing that is the ELM project. The ELM project is a website that is internal to the company used for the current projects as well to have a list of the interns and the instructors.

Throughout the internship, our role was to concentrate ourselves to the “project” part of the website as well as the creation of a new module, the “Costing” module.

During the internship, I was able to use some of the technologies that I learned during my training course (JavaScript, HTML, CSS) but also a new technology which is React.js, a frontend JavaScript library.

# INSY2S

## Présentation de l'entreprise



INSY2S est fondée en 2014 par Monsieur TORCHE Nezar. C'est une société de services du numérique qui a pour but d'aider des sociétés clientes dans la réalisation de projets. Elle se spécialise également dans le secteur du conseil en systèmes et logiciels informatiques.

Les différents domaines d'intervention sont :

- Data (IA / Machine Learning)
- Développement Web et Web Mobile
- Protection et gestion des données (RGPD)

Une liste de certains partenaires :



## Présentation de l'équipe



**Nezar TORCHE**

*PDG de  
l'entreprise*



**Marinne ZIMMER**

*Scrum Master*



**Peter MOLLET**

*Référent Technique  
/ Scrum Master*



**Jovany LEMOINE**

*Scrum Master*

### Team Akatsuki



**Florent  
MARTENS**



**Mathias  
DERACHE**



**Yoan  
DECONINCK**



**Romain  
WYON**



**René-Jean  
RUGABA**



**Steeven  
GHEERAERT**

## Présentation du projet ELM

Lors de mon stage, j'ai travaillé sur le projet ELM (Enterprise Learning Management). Ce projet sert à l'entreprise à gérer ses activités ainsi que celles de ses employés, il est utilisé également par des organismes de formation comme l'AFPA pour effectuer des demandes de formation auprès de INSY2S mais aussi de suivre les sessions en cours les concernant. Les stagiaires des sessions ont aussi accès pour valider leur présence chaque jour et pour communiquer avec leur formateur pour les travaux à réaliser.

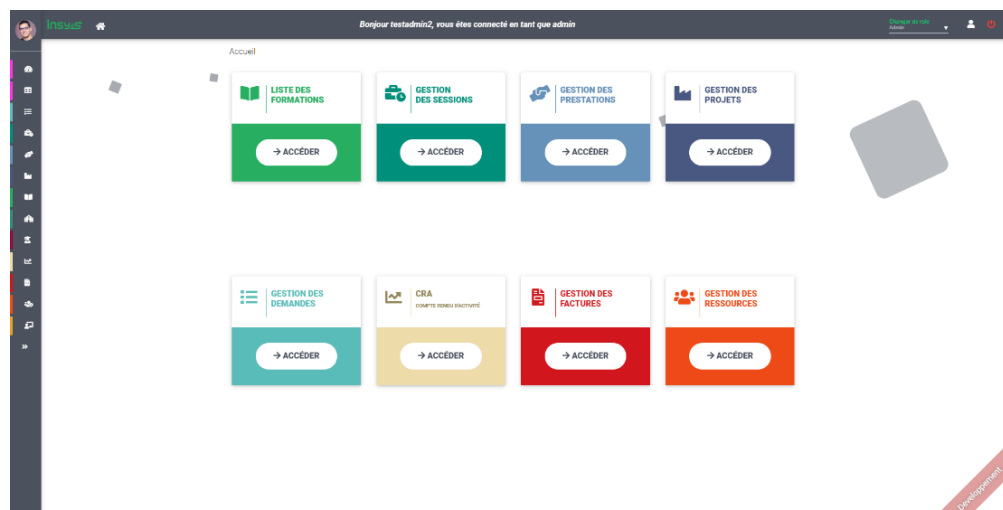
L'application possède différents rôles pour les utilisateurs. Un utilisateur peut avoir plusieurs rôles, selon le(s) rôle(s) de l'utilisateur il a accès ou non à certains modules.

Les rôles sont :

- Admin : Les personnes de l'entreprise ayant les droits administrateurs sur l'application.
- Manager : Les managers de l'entreprise.
- Employé : Les employés de l'entreprise.
- Centre : Les organismes de formation comme l'AFPA.
- Stagiaire : Les stagiaires des sessions.

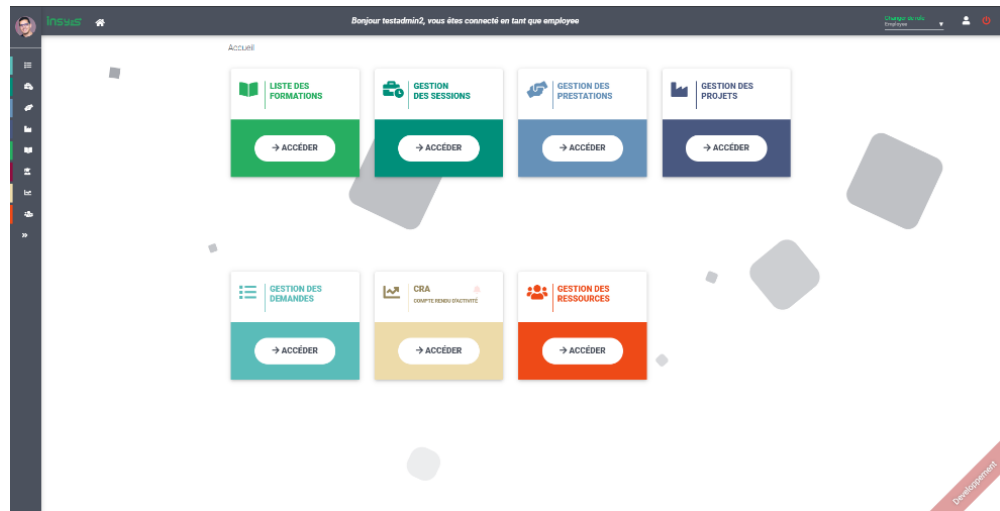
Voici une vue de l'écran d'accueil pour :

Un Admin/Manager :





Un Employé :



L'application est séparée en différents modules :

- **Formations**

Ce module regroupe l'ensemble des formations proposées par INSY2S. L'admin et le manager peuvent créer de nouvelles formations et modifier les formations existantes. L'admin, le manager, l'employé et le centre peuvent consulter les formations et voir les sessions en cours pour chaque formation (l'employé et le centre voient uniquement les formations qui les concernent). Le stagiaire n'a pas accès à ce module.

- **Sessions**

Ce module permet de visualiser les sessions et leurs différentes informations comme la liste des stagiaires inscrits à la session, les employés intervenants auprès des stagiaires. L'admin et le manager peuvent visualiser et modifier l'ensemble des sessions, ajouter des intervenants et gérer les sessions (ajouter des stagiaires, modifier le pointage des stagiaires, ajouter des tâches à faire aux stagiaires...). L'employé peut visualiser et gérer les sessions où il intervient. Le centre peut visualiser uniquement les sessions dont il est à l'origine de la demande. Le stagiaire n'a pas accès à ce module.

- **Prestations**

Ce module permet de gérer les prestations de l'entreprise. Une prestation correspond à une mission réalisée chez un client de type prestation comme Décathlon. L'admin et le manager peuvent visualiser l'ensemble des prestations, modifier une prestation existante ou en créer une nouvelle, il peut aussi postuler à une prestation. L'employé peut visualiser les prestations où il intervient et postuler à une prestation qui n'a pas encore d'intervenant. Le stagiaire et le centre n'ont pas accès à ce module.

- **Projets**

Ce module permet de gérer les projets internes de l'entreprise. L'admin et le manager peuvent visualiser les projets, les modifier et créer un nouveau projet. L'employé peut visualiser les projets où il intervient. Le stagiaire et le centre n'ont pas accès à ce module. C'est sur ce module que moi et mon équipe devons travailler.

- **Demandes**

Ce module permet de gérer les différentes demandes (congés, formation). L'admin et le manager peuvent visualiser les demandes et les modifier (accepter ou refuser la demande). L'admin peut créer une nouvelle demande de congé ou de formation. L'employé peut visualiser les demandes qui le concerne (postulation à une prestation, demande de congés), créer une demande de congés. Le centre peut visualiser les demandes qui le concerne (demande de formation), créer une demande de formation. Le stagiaire n'a pas accès à ce module.

- **CRA (Compte Rendu d'Activité)**

Ce module permet de gérer les Comptes Rendu d'Activité (CRA) des employés. L'admin et le manager peuvent visualiser les CRA de l'ensemble des employés ainsi que ceux des missions, ils peuvent valider ou rejeter les CRA des employés une fois qu'ils ont été soumis. L'admin peut en plus modifier les CRA des employés. L'employé peut visualiser et éditer son CRA et le soumettre pour validation. Le stagiaire et le centre n'ont pas accès à ce module.

- **Factures**

Ce module permet une gestion simplifiée des factures, elle a pour objectif seulement d'avoir une vision sur les factures des missions (session, prestation ou projet). L'admin et le manager peuvent consulter et éditer les factures et en créer de nouvelles. Chaque facture doit être rattachée à une mission. L'employé, le centre et le stagiaire n'ont pas accès à ce module.

- **Utilisateur**

Ce module permet de gérer les utilisateurs de l'application. L'admin et le manager peuvent visualiser l'ensemble des utilisateurs et créer un nouvel utilisateur, ils ont aussi la possibilité de modifier leurs propres informations personnelles et ajouter/retirer des compétences aux employés. L'employé, le centre et le stagiaire peuvent modifier leurs informations personnelles. L'employé peut en plus modifier ses compétences.

- **Employé**

Ce module ressemble au module utilisateurs. Il permet de consulter l'ensemble des employés de l'entreprise et ajouter/retirer des compétences

aux employés. L'employé, le centre et le stagiaire n'ont pas accès à ce module.

- **Dashboard**

Ce module permet d'avoir un résumé de certaines données de l'application sous forme de représentation graphique ainsi que les actions à réaliser. L'employé, le centre et le stagiaire n'ont pas accès à ce module.

- **Clients**

Ce module permet de gérer les clients de l'entreprise, il y a 2 types de clients : prestation (lorsqu'un employé effectue une mission chez un client) et formation (les organismes de formation qui font des demandes de formation auprès Insy2s comme l'AFPA). L'admin et le manager peuvent visualiser et modifier les clients, rattacher un compte centre à un client de type formation et créer de nouveaux clients. Ils ont aussi la possibilité de créer des sites clients (c'est une filiale du client, par exemple le client formation AFPA peut avoir comme site client AFPA Roubaix, AFPA Lille...). L'employé, le centre et le stagiaire n'ont pas accès à ce module.

- **Compétences**

Ce module permet de gérer les compétences. L'admin et le manager peuvent ajouter ou supprimer des compétences. L'employé peut visualiser les compétences. Le centre et le stagiaire n'ont pas accès à ce module.

- **Tableaux Chiffrage**

Ce module créé durant notre stage permet de gérer les tables de référence utilisées pour le chiffrage. Seul l'admin peut accéder et modifier les tables de référence.

Les autres rôles n'ont pas accès à ce module.

# Environnement de travail

## Outils utilisés

### Outils de communication

Pour communiquer entre nous lors de ce stage se déroulant à distance en raison de la crise sanitaire nous avons utilisés 2 outils différents :



Nous avons tout d'abord utilisé Microsoft Teams pour les daily meeting ainsi que les sprint reviews.



Ensuite nous avons utilisé Discord pour simuler un workspace tout en étant à distance. Chaque équipe avait son propre canal de communication ainsi que des canaux communs à l'entreprise pour de l'entraide entre tous les membres de l'entreprise, des annonces ou des suggestions.

## Outils de développement



Visual Studio Code

Pour pouvoir coder nos différentes fonctionnalités, nous avons utilisé Visual Studio Code.



Gitlab nous sert pour le versioning des différents projets. Un dépôt front et back pour chaque projet. J'utilise Git Bash pour effectuer les différentes commandes git.



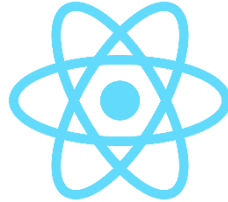
Pour gérer les projets, nous avons utilisés Jira. Un backlog est créé avec les différentes tâches à effectuer pendant le projet. Chaque feature est un ticket à effectuer. Lors de chaque sprint le scrum master choisi les tickets que l'on doit faire durant le sprint. Chaque membre choisissait sa tâche et déplaçait son ticket selon l'état d'avancement (à faire, en cours, fait). (Voir l'annexe A pour un affichage de l'écran de Jira)



Comme nous travaillons en méthode SCRUM, nous utilisons PlanItPoker pour pokeriser (attribuer de point d'effort aux tickets du sprint). (Voir l'annexe B pour un affichage de l'écran de PlanItPoker)

## Technologies utilisées

Notre stage étant uniquement axé sur la partie front-end voici la liste des technologies utilisées lors de celui-ci :



**React** : React est une bibliothèque JavaScript qui permet de créer des interfaces utilisateur en Single Page Application. Le navigateur charge l'application lors du 1<sup>er</sup> accès et ne fait qu'afficher les différents composants. Cela permet d'avoir une expérience utilisateur (UX) plus fluide.

React utilise un langage hybride appelé JSX (JavaScript XML) qui permet de générer des objets JavaScript avec une notation similaire à HTML.



**HTML5/CSS3** : HTML (HyperText Markup Language) est utilisé pour créer et représenter la structure et le contenu d'une page web. Et CSS (Cascading Style Sheets) permet de créer des pages web avec une apparence soignée.



**JavaScript** : Javascript permet d'implémenter des mécaniques complexes à une page web.



MDB (Material Design for Bootstrap) : MDB aide au design et à la responsivité de l'application, nous avons utilisé la version pro de MDB et pour être plus précis sa version React.

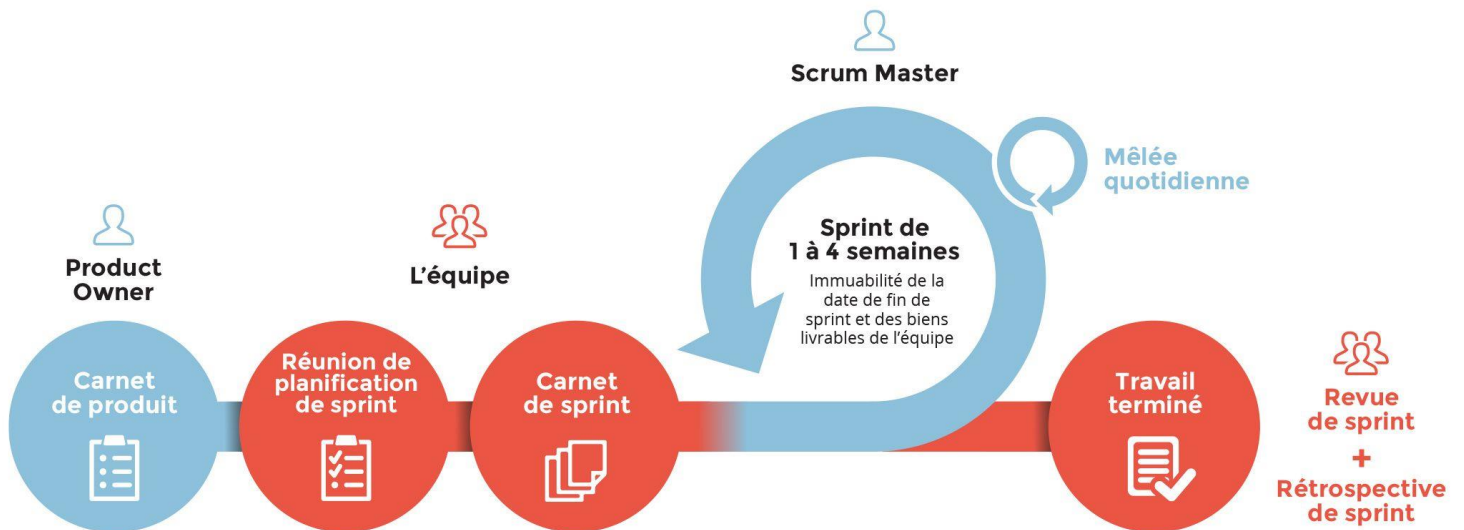


Axios : Axios est une bibliothèque JavaScript qui fonctionne comme un client HTTP. Elle permet de communiquer avec des API en utilisant des requêtes http. Comme cette bibliothèque est basée sur des Promises (promesses), nous avons la possibilité d'effectuer des appels asynchrones. Nous avons également la possibilité d'intercepter et d'annuler une demande.



Formik/YUP : Formik est une librairie facilitant la création de formulaires sur React. YUP quant à elle permet de faire des schémas de validation afin de contrôler les différentes données du formulaire et d'afficher des messages d'erreur quand celles-ci ne correspondent pas aux exigences.

# Organisation Scrum



Pendant le stage, nous avons utilisé la méthodologie scrum voici son déroulement lors du stage :

Notre Product Owner organise des tickets comportant des tâches à effectuer pour l'application. Une fois le backlog crée, l'équipe dirigée par le scrum master organise un « poker planning ». Le but du poker planning a pour but de présenter les tickets à l'équipe et de les noter en point d'effort allant de 0.5 à 13 points d'effort.

0.5 est une tâche très simple et très rapide à effectuer comme modifier un placeholder dans un formulaire et une tâche à 13 points d'effort est une tâche très complexe et/ou très longue à effectuer comme la création d'un formulaire complexe.

Une fois le poker planning effectué, le sprint commence. Nous avons tous les matins à 9h un daily meeting, c'est une réunion d'environ 15min où chaque membre de l'équipe doit répondre tour après tour à 3 questions :

- « Qu'est-ce que j'ai fait hier ? »
- « Qu'est-ce que je vais faire aujourd'hui ? »
- « Quelles sont les difficultés que j'ai rencontrées ? »



A la fin du sprint, nous effectuons un « sprint review », cela permet de montrer tour à tour la partie fonctionnelle des tickets effectués lors du sprint. Et enfin nous terminons avec le « sprint rétrospective », qui permet de faire le point avec l'équipe sur ce qui a marché ou non dans l'organisation, cohésion de l'équipe mais aussi les choses qu'il faut ou ne faut plus faire. Les questions sont:

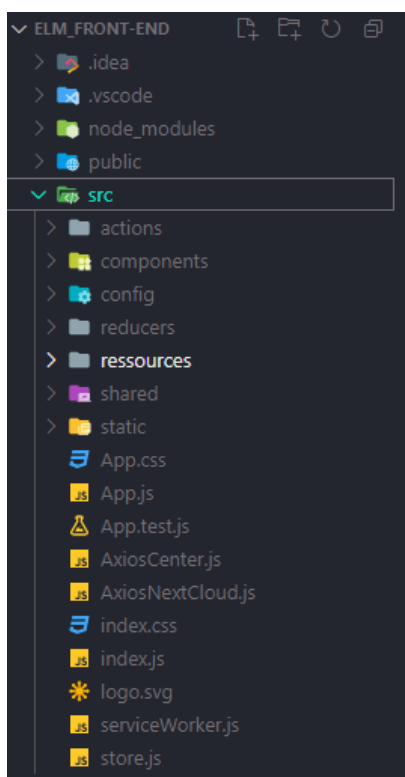
- « More of » (plus de)
- « Less of » (moins de)
- « Start doing » (commencer à faire)
- « Keep doing » (continuer à faire)
- « Stop doing » (arrêter de faire)

# Projet ELM

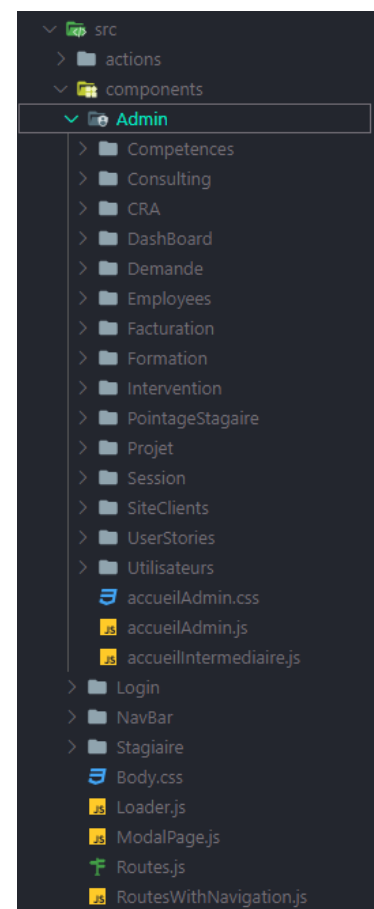
## Architecture

La partie front-end de l'application ELM a été créée sous la bibliothèque React avec sa technologie Single Page Application. Pour une bonne maintenabilité et compréhension du code, l'application ELM est séparée en différents packages.

### Architecture Générale :



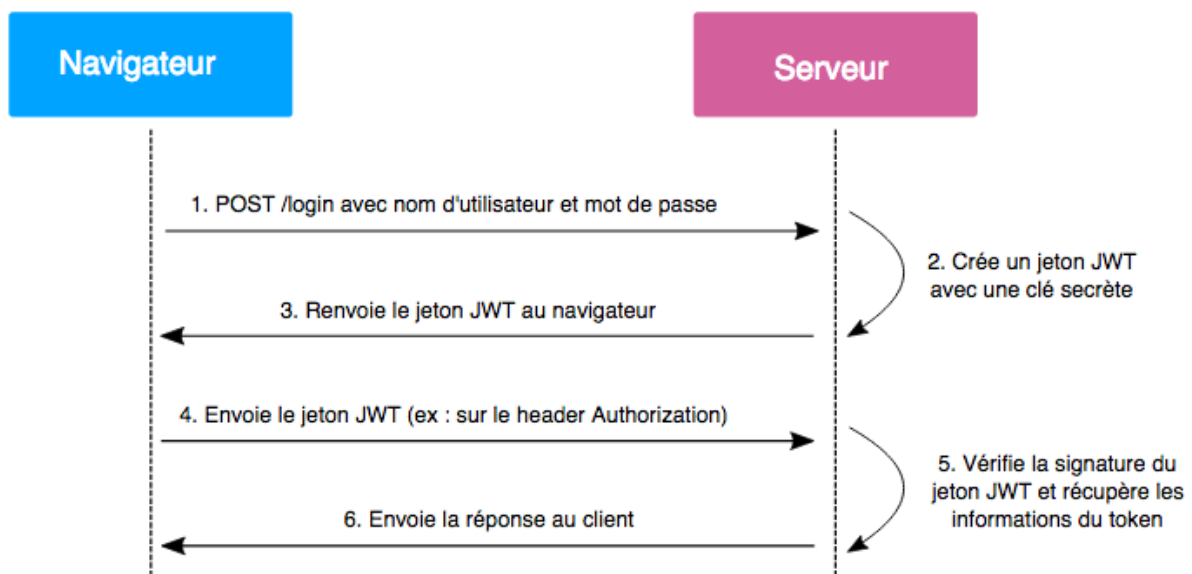
### Architecture Components :



- **Action** : regroupe les actions de react-redux
- **Components** : regroupe tous les composants affichés sur l'image « Architecture Components » ci-dessus. Le dossier components comporte plusieurs sous dossiers qui représentent chaque module de l'application.
- **Config** : regroupe tous les fichiers de configuration

- **Reducers** : regroupe les reducers de react-redux
- **Ressources** : regroupe les ressources de l'application notamment les images
- **Shared** : regroupe les services communs à l'application
- **Static** : regroupe les ressources statiques de l'application comme les images de fond
- **AxiosCenter.js** : c'est un fichier regroupant toutes les requêtes faites au Back en utilisant la bibliothèque Axios.

Pour sécuriser les échanges du front et du back de l'application utilise des JWT (JSON Web Token) qui est un standard permettant de sécuriser les échanges avec un système de token. Les web services sont uniquement disponibles si l'utilisateur envoie son token pour s'authentifier. Les seuls web services disponibles sans authentification sont la connexion et l'activation de compte.



Une autre sécurité cette fois effectuée uniquement du côté front de l'application est par rapport aux routes de l'application car certaines pages doivent être accessibles par des rôles bien spécifiques (exemple : certaines pages sont seulement accessibles pour les Admins et/ou les Managers). Dans ce cas si un utilisateur n'a pas l'autorisation requise pour accéder à la page il est redirigé soit à la page de connexion s'il n'était pas connecté, soit à la page d'accueil ou sinon sur une page d'erreur. Pour cela un composant « PrivateRoute » qui vérifie si l'utilisateur a les droits d'accès à la page qu'il veut est créé.

```
// vérifie que l'utilisateur est connecté
const PrivateRoute = ({ component: Component, token, roles, role, ...rest }) => (
  <Route
    {...rest}
    render={props =>
      (token && token !== "" && roles.includes(role)) ? (
        <Component {...props} />
      ) : (
        <Redirect to={{ pathname: "/", }} />
      )
    }
  />
);
```

D'abord on récupère les props que le composant doit recevoir avec la ligne : ({component : Component, token, roles, role, ...rest })

- Component : le composant qui doit être affiché. Component permet de spécifier à React qu'il s'agit d'un composant et non pas d'une donnée
- token : le token envoyé par le back lors de l'authentification, il permet au client de prouver son authentification lorsqu'il effectue une requête au Back.
- roles : la liste des rôles autorisés à accéder à la route demandée
- role : le rôle de la personne connectée
- ...rest : le reste des props qui seront attribués au composant.

Dans un second temps on crée la route avec les props ...rest. Ensuite nous faisons un rendu selon si l'utilisateur est autorisé ou non à accéder à la route :

- S'il est autorisé, c'est-à-dire si le token est présent et que son rôle fait partie de la liste des rôles autorisés à accéder à la route, on le redirige vers le composant voulu.
- Sinon on le redirige vers la page de login.

### Exemple du composant PrivateRoute :

```
<PrivateRoute
  token={token}
  path={"/"+pathRole+"/chifffrage"}
  component={DetailsChiffragesParent}
  roles={["ROLE_ADMIN", "ROLE_MANAGER"]}
  role={role}
  exact />
```

Dans cet exemple seuls les utilisateurs authentifiés et possédant les rôles Admin et Manager peuvent accéder au composant « DetailsChiffragesParent ».

## Module « Projets »

Durant la majorité de notre stage, nous avons travaillé sur le module « Projets » de l'application. Ce module était déjà existant mais nous devons faire une refonte totale du module à la suite de changements importants dans le back survenus peu avant notre arrivée.

### Page d'accueil d'un projet du module « Projets » :

The screenshot shows the 'Projet Entreprise Learning Management' page. At the top, a dark blue header contains the project title and the mission date: 'Date de Mission: 02/09/2019 au 31/12/2020'. Below this, a navigation bar has four tabs: 'Users stories', 'Projets', 'Params Users stories', and 'Statistiques projet'. The 'Projets' tab is active. A 'Filtre' section contains three dropdown menus for 'Etat User Story', 'Priorité User Story', and 'Urgence User Story', each with a 'Select...' option. Below the filters, a blue bar contains three tabs: 'Liste users stories', 'diagramme Users stories', and 'Trello users stories'. The 'Liste users stories' tab is active, showing a list of user stories. The list has two entries: 'Lot - lot1 - début : 14/01/2021 - fin prévisionnelle : 14/05/2021 - à faire - poids : 5 - priorité : forte - urgence : moyen' and 'Lot - lot2 - début : 13/01/2021 - fin prévisionnelle : 13/06/2021 - à faire - poids : 4 - priorité : moyen - urgence : moyen'. A green plus icon and the text 'Ajouter une user story' are visible next to the list. The page is part of the 'insus' application, as indicated by the logo in the top left corner. The user is logged in as 'testadmin2' with the role of 'Admin'. The page is labeled 'Development' in the bottom right corner.

Pour ma part dans ce module j'ai surtout dû m'occuper de l'affichage du détail et de la modification d'une user story.

## Affichage du détail d'une User Story :

Détails du user story : lot1Test

Date de début  
23 avril, 2021

Type du User Story  
Lot

Date de fin estimée  
27 avril, 2021

Date de fin réelle  
user story non terminée

Etat  
ToDo

Poids  
2

Nom User Story  
lot1Test

Description

Tags  
Front

User Story Parent :  
Pas de user story parent

Sous type User Story  
Feature

Priorité User Story  
moyen

Urgence User Story  
moyen

Sous user stories :

Ajouter une sous user story

Pour le design du détail et de la modification d'une User Story, je suis resté sur quelque chose de semblable au modèle donné lors du tout premier sprint. (Voir l'annexe C)

Pour récupérer les bonnes informations dans le parent, je récupère l'id d'une User Story fourni par un autre composant créé par un de mes collègues puis dans le composant parent des User Stories j'utilise l'id avec un appel Axios pour avoir les informations et les passer dans le composant détails/modification User Story.

### Fonction pour afficher le bon détail :

```
getUserStoryByIdAndWrapper = (id) => {
  axiosCenter
    .getUserStoryByIdAndWrapper(this.props.token, id)
    .then((response) => {
      if (response.status === 200) {
        this.setState({
          UserStorieUseForDetail: null,
        });

        if (!this.state.userStoryUseForDetail) {
          this.setState({
            UserStorieUseForDetail: response.data,
          });
        }
      }
    });
};
```

A la suite de cela je devais m'occuper de pouvoir donner la possibilité de modifier une User Story. Un Admin ou Manager peut modifier toutes les informations d'une User Story tandis qu'un employé simple ne peut modifier que l'état et la date de fin réelle d'une User Story. Pour la modification d'une User Story j'utilise la bibliothèque Formik qui est une bibliothèque de React qui permet de simplifier les formulaires.



## Modification du côté d'un Admin/Manager :

Détails du user story : lot1 Test

Date de début

23 avril, 2021

Type du User Story

Lot

Date de fin estimée

27 avril, 2021

Date de fin réelle

user story non terminée

Etat

ToDo

Poids

2

Nom User Story

lot1 Test

Description

Tags

Front

User Story Parent :

Pas de user story parent

Sous Type User Story

Feature

Priorité User Story

moyen

Urgence User Story

moyen

ANNULER

VALIDER

Sous user stories :

Ajouter une sous user story

```
<Formik
  onSubmit={(values, e) => this.submit(values, e)}
  initialValues={{
    id: this.state.userStory.id,
    name: this.state.userStory.name,
    description: this.state.userStory.description,
    createdAt: this.state.userStory.createdAt,
    beginDate: moment(this.state.userStory.beginDate).format("YYYY-MM-DD"),
    estimatedEndDate: moment(this.state.userStory.estimatedEndDate).format("YYYY-MM-DD"),
    realEndDate: this.state.userStory.idEtat === 3 ? moment(this.state.userStory.realEndDate).format("YYYY-MM-DD") : null,
    userStoryParentId: this.state.userStory.idUserStoryParent,
    etatId: this.state.userStory.idEtat,
    typeId: this.state.userStory.idType,
    subTypeId: this.state.userStory.idSubType,
    tags: this.state.tags.filter(tag => this.state.userStory.listIdTags.includes(tag.id)).map(tag => { return { value: tag.id, label: tag.libelle } }),
    listIntervenants: this.state.userStory.listIntervenants // [],
    listIdTags: this.state.userStory.listIdTags // [],
    produitId: this.state.userStory.idProduit,
    archiver: this.state.userStory.archiver,
    poids: this.state.userStory.poids,
    prioriteId: this.state.userStory.idPriorite,
    urgenceId: this.state.userStory.idUrgence
  }}
  validationSchema={this.validationSchema}>
```

Ici nous avons les valeurs initiales pour le formulaire de modification de la User Story. Pour que certains des inputs ne changent pas si l'utilisateur est un employé quand on modifie une User Story nous faisons une vérification du rôle de l'utilisateur.

```

<MDBCol className="mt-2">
  <small>Date de début</small>
  <Field
    className="form-control"
    name="beginDate"
    component={INSYDatePicker}
    value={values.beginDate}
    setFieldValue={setFieldValue}
    disabled={!this.state.enable} // editByAdminOrManager
    themeDatePicker={this.themeDatePicker}/>
</MDBCol>

<MDBCol>
  {editByAdminOrManager && this.state.enable ?
    <Field type='select' name='typeId' options={typeUserStory} component={INSYSelect} Label='Type du User Story' />
    :
    <MDBInput Label="Type du User Story" value={this.state.userStory.type} disabled />
  }
</MDBCol>
</MDBRow>

<MDBRow>
  <MDBCol className="mt-2">
    <small>Date de fin estimée</small>
    <Field
      className="mt-n1"
      name="estimatedEndDate"
      component={INSYDatePicker}
      value={values.estimatedEndDate}
      setFieldValue={setFieldValue}
      disabled={!this.state.enable} // editByAdminOrManager
      themeDatePicker={this.themeDatePicker}/>
    </MDBCol>
  </MDBRow>

```

Grâce à Yup, nous pouvons faire des schémas de validation qui vérifient à ce que les données rentrées par l'utilisateur correspondent à ce que l'on veut (nombre minimum ou maximum de caractères, un certain type de donnée, etc...) dans le cas contraire un message d'erreur s'affiche pour indiquer à celui-ci ce qui ne va pas.

```

validationSchema = Yup.object().shape({
  name: Yup.string()
    .trim()
    .typeError("Erreur de type")
    .required("Champ Requis")
    .min(3, "3 caractères minimum")
    .max(255, "255 caractères maximum"),
  description: Yup.string()
    .trim()
    .typeError("Erreur de type")
    .max(255, "255 caractères maximum"),
  beginDate: Yup.date()
    .required("Champ Requis")
    .max(Yup.ref("estimatedEndDate"), "La date de début doit être avant la fin prévisionnelle."),
  estimatedEndDate: Yup.date()
    .required("Champ Requis")
    .min(Yup.ref('beginDate'), "La date de fin prévisionnelle doit être après la date de début."),
  realEndDate: Yup.date()
    .nullable()
    .min(Yup.ref('beginDate'), "La date de fin réelle doit être après la date de début."),
  etatId: Yup.string().required("Champ Requis"),
  typeId: Yup.string().required("Champ Requis"),
  subTypeId: Yup.string().required("Champ Requis"),
  tags: Yup.array(),
  poids: Yup.number().integer("Le poids doit être un entier.").min(0, "Le poids doit être un nombre positif.").strict().required("Champ Requis"),
  prioritId: Yup.string().required("Champ Requis"),
  urgenceId: Yup.string().required("Champ Requis")
})

```

Détails du user story : lot1Test

Date de début  
23 avril, 2021

Type du User Story  
Lot

Date de fin estimée  
27 avril, 2021

Date de fin réelle  
27 avril, 2021

Etat  
Done

Poids  
-5

Nom User Story  
l

Description  
Ceci est un test de la description.

Tags  
Front Back

User Story Parent :  
Pas de user story parent

Priorité User Story  
faible

Sous Type User Story  
Feature

Urgence User Story  
forte

Sous user stories :  
Ajouter une sous user story

ANNULER

VALIDER

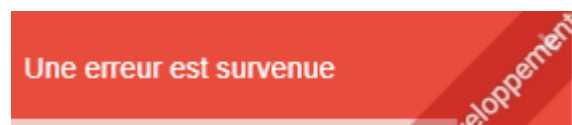
Le poids doit être un nombre positif.

Quand nous avons remplis toutes les informations requises et que nous appuyons sur le bouton valider un « toast » s’affiche pour nous indiquer si l’opération a réussie ou échouée.

### Modal de réussite :



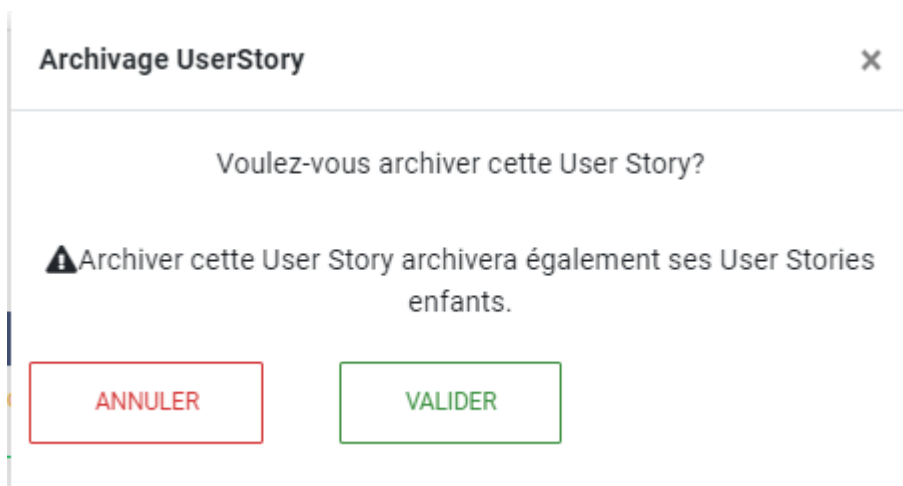
### Modal d'échec :



Ensuite j'ai dû m'occuper de l'archivage d'une User Story qu'il fallait archiver à l'aide d'un modal. Pour l'archivage ce qu'il fallait du côté back-end était fait (une colonne ARCHIVER dans la base de données qui comporte un booléen TRUE (si la User Story est archivée) ou FALSE (valeur de base qui correspond à une User Story non archivée)). Le code du composant DétailsUS commençant à être long j'ai décidé de créer un nouveau composant qui est appelé dans le composant DétailsUS.

```
<MDBModal
  size="md"
  isOpen={this.state.showModalArchivage}
  toggle={this.toggleModalArchivage}
  backdrop={false}
  backdropClassName="teal"
  centered>
  <MDBModalHeader toggle={this.toggleModalArchivage} className="text-center">Archive UserStory</MDBModalHeader>
  <MDBCardBody className="text-center">
    <ArchiveUserStory
      toggle={this.toggleModalArchivage}
      userStory={this.state.userStory}
      getListeUserStories={this.props.getListeUserStories}
      setNullToUserStorieUseForDetail={this.props.setNullToUserStorieUseForDetail}
      tags={this.state.tags}/>
  </MDBCardBody>
</MDBModal>
```

## Modal d'archivage d'une User Story:



Quand on clique sur le bouton valider le formulaire dans le modal renvoie toutes les informations de base et ne fait que modifier le booléen « archiver » de FALSE à TRUE.

```
class ArchivageUserStory extends React.Component{
  constructor(props){
    super(props)
    this.state = {
      userStory: this.props.userStory,
      loaded: false,
      tags: this.props.tags
    }
  }
}
```

```
<Formik
  onSubmit={values => this.submit(values)}
  initialValues={{
    id: this.state.userStory.id,
    name: this.state.userStory.name,
    description: this.state.userStory.description,
    createdAt: this.state.userStory.createdAt,
    beginDate: moment(this.state.userStory.beginDate).format("YYYY-MM-DD"),
    estimatedEndDate: moment(this.state.userStory.estimatedEndDate).format("YYYY-MM-DD"),
    realEndDate: this.state.userStory.idEtat === 3 ? moment(this.state.userStory.realEndDate).format("YYYY-MM-DD") : null,
    userStoryParentId: this.state.userStory.idUserStoryParent,
    etatId: this.state.userStory.idEtat,
    typeId: this.state.userStory.idType,
    subTypeId: this.state.userStory.idSubType,
    tags: this.state.tags.filter(tag => this.state.userStory.listIdTags.includes(tag.id)).map(tag => { return { value: tag.id, label: tag.libelle } }),
    listeIntervenants: this.state.userStory.listeIntervenants // [],
    listIdTags: this.state.userStory.listIdTags // [],
    produitId: this.state.userStory.idProduit,
    archiver: true,
    poids: this.state.userStory.poids,
    prioriteId: this.state.userStory.idPriorite,
    urgenceId: this.state.userStory.idUrgence
  }}>
```

Quand la User Story est archivée un « toast » de réussite d’affiche, le détail se ferme et la liste des User Stories se met à jour. (Voir l’annexe D)

Au départ chaque composant du module « Projets » avait ses propres appels Axios (GET, POST, PUT ou DELETE) et certains appels dans différents composants se répétaient et alourdissaient le code ainsi que le temps de réponse du back car ils se faisaient dès le lancement de la page. Pour contrer cela une autre de mes tâches fut de regrouper les appels effectués plusieurs fois dans de fonctions (surtout les GET) dans le parent pour les effectuer une seule et unique fois au lancement de la page. Ensuite nous avons la possibilité de passer la ou les fonctions nécessaires pour mettre à jour un composant lors d’un ajout (POST) ou modification (PUT).

## Fonctions GET dans le composant parent des User Stories :

```
getEtatsUserStory = () => {
  const { token } = this.props;

  axiosCenter.getUserStoryEtats(token).then((response) => {
    this.setState({
      etatsUserStory: response.data,
    });
  });
};

getTypesUserStory = () => {
  const { token } = this.props;

  axiosCenter.getTypeUserStories(token).then((response) => {
    this.setState({
      typesUserStory: response.data,
    });
  });
};

getSousTypesUserStory = () => {
  const { token } = this.props;

  axiosCenter.getSubTypeId(token).then((response) => {
    this.setState({
      sousTypesUserStory: response.data,
    });
  });
};

getTagsUserStory = () => {
  const { token } = this.props;

  axiosCenter.getTagsUserStory(token).then((response) => {
    this.setState({ tags: response.data });
  });
};
```

```
{this.state.UserStorieUseForDetail ? (
  <MDBCol md="12" Lg="4">
    <DetailsUSProjet
      handleClose={this.deselectTache}
      history={this.props.history}
      interventionsProduit={this.state.interventionsProduit}
      missionDTO={this.state.wrapper.missionDTO}
      setNullToUserStorieUseForDetail={
        this.setNullToUserStorieUseForDetail
      }
      getUserStoryByIdAndWrapper={
        this.getUserStoryByIdAndWrapper
      }
      getListeUserStories={this.getListeUserStories}
      toggle={this.toggle}
      userStoryUseForDetail={
        this.state.UserStorieUseForDetail
      }
      etatsUserStory={this.state.etatsUserStory}
      typeUserStory={this.state.typesUserStory}
      sousTypeUserStory={this.state.sousTypesUserStory}
      tags={this.state.tags}
      updatePage={this.updatePage}
      allUserStories={this.state.alluserStories}
      prioritesUserStory={this.state.prioritesUserStory}
      urgencesUserStory={this.state.urgencesUserStory}
    />
  </MDBCol>
) : null}
```

## State du composant parent des User Stories :

```
this.state = {
  loaded: false,
  wrapper: {},
  activeItem: "1",
  idTacheSelectionne: null,
  vueTaches: true,
  modal: false,
  etatsUserStory: [],
  typesUserStory: [],
  sousTypesUserStory: [],
  tags: [],
  idProduit: "",
  UserStoryUseForDetail: null,
  userStoryIdParent: null,
  produitId: this.props.match.params.id,
  userStories: [],
  allUsersStories: [],
  dataPie: {},
  typeUserStories: [],
  alluserStories: [],
  etatUserStories: [],
  activeItemJustified: "1",
  allPrioriteProduit: [],
  allUrgenceProduit: [],
  prioritesUserStory: [],
  urgencesUserStory: [],
  userStoriesFilter: null,
  valuesFilter: {},
  employes:[]
};
```

## Module « Tableaux Chiffrages »

Lors de l'avant dernier sprint, nous avons dû créer un nouveau module qui est le module « Tableaux Chiffrage ». Ce module comporte plusieurs tableaux utilisés pour le « Chiffrage ». J'ai personnellement dû m'occuper du tableau des « Complexités ». Pour créer ce module un collègue a créé une maquette de ce que donnerait le module après une consultation et débat avec les membres de notre groupe (voir l'annexe E).

### Tableau Complexités du module « Tableaux Chiffrage » :

Catégories

Complexités

Contextes

Paramètres













Phases Projet

Technologies

Types Intervenant

Valeurs Contexte

Abaques

CODE REF	LIBELLE	ACTIONS
TS	très simple	 
S	simple	 
M	moyen	 
C	complexe	 
TC	très complexe	 
TE	Test	 
<div>+ AJOUTER</div>		

Pour le design nous sommes restés sur quelque chose de simple comme seul l'admin ou le manager ont accès au module « Chiffrage ».

### Affichage des informations du tableau :

```
render(){
  if (!this.state.loaded) {
    return(<></>)
  }
  return(
    <MDBContainer className="container text-center mt-3">
      <MDBCard className="bg-primary p-3 text-white">
        <div className="row">
          <div className="col-4">CODE REF</div>
          <div className="col-4">LIBELLE</div>
          <div className="col-4">ACTIONS</div>
        </div>
      </MDBCard>
      {this.props.complexitesRef.map((complexiteRef) => {
        return(
          <MDBContainer key={complexiteRef.id}>
            <MDBRow className="border-light">
              <MDBCol md="4" className="my-2">{complexiteRef.codeRef}</MDBCol>
              <MDBCol md="4" className="my-2">{complexiteRef.libelle}</MDBCol>
              <MDBCol md="4" className="my-2">
                <MDBIcon icon="pen" className="green-text zoomIcon pointeur" title="modifier" onClick={()=>this.setUpdate(complexiteRef)} />
                <MDBIcon className="ml-3 text-danger zoomIcon pointeur" icon="trash-alt" title="supprimer" />
              </MDBCol>
            </MDBRow>
          </MDBContainer>
        )
      })}
    )
  )
}
```



En appuyant sur le bouton d'ajout ou de modification un input s'affiche et nous pouvons ajouter/modifier les différentes informations.

### Ajout/Modification de complexités :

[illegible]

### Formulaire de l'ajout/modification d'une complexité :

```

<MDBRow className="d-flex justify-content-center mt-3">
  {this.state.creation?(
    <Formik
      initialValues={this.state.initialValues}
      onSubmit={ this.submit }
      validationSchema={ this.validationSchema}
    >
      {({ handleChange, handleSubmit, handleBlur }) => (
        <MDBFormInline onSubmit={handleSubmit}>
          <div>
            submit = (values) => {
              let codeRefNoSpace=values.codeRef.replace(/\s/g,'');
              values.codeRef=codeRefNoSpace
              if (this.state.update) {
                axiosCenter.putComplexiteChiffrage(this.props.token, values).then((response) => {
                  toast.success("Modification réussie !");
                  this.props.getComplexitesRef();
                }, err => {
                  toast.error("Erreur lors de la modification !");
                })
              } else {
                axiosCenter.postComplexiteChiffrage(this.props.token, values).then((response) => {
                  toast.success("Ajout de la complexité réussi !");
                  this.props.getComplexitesRef();
                },err => {
                  toast.error("Erreur lors de l'ajout !");
                })
              }
            }
          </div>
        </MDBFormInline>
      )}
    </Formik>
  )}
</MDBRow>

```

## Le schéma de validation du Tableau Complexité :

```
validationSchema = Yup.object().shape({  
  codeRef: Yup.string().trim().typeError("Erreur de type").required("Champ Requis !").matches(/^[^\\d\\w_]*$/, 'Caractère non autorisé').min(1, "1 caractère minimum"),  
  libelle: Yup.string().trim().typeError("Erreur de type").required("Champ Requis !").min(2, "2 caractères minimum")  
})
```

Dans ce cas précis grâce à Yup dans ce schéma de validation le codeRef n'accepte que les lettres et n'accepte pas d'espace entre elles.

Juste après la création des différents tableaux nos scrum masters ont remarqués que certains d'entre eux (3 pour être plus précis) comportaient les mêmes types d'informations (codeRef, Libelle) et que par conséquent nous pouvions améliorer le code en utilisant un composant générique au lieu de 3 composants distincts.

C'est un collègue qui s'est occupé de cette tâche néanmoins vous trouverez en annexe (Annexe F) des captures d'écran de ce code car le principe reste le même que dans le code précédent nous faisons juste l'appel pour recevoir les informations dans le parent puis nous passons les informations dans le composant générique.

# Cahiers de Tests

L'une de nos dernières tâches fut de rédiger et exécuter un cahier de tests. J'ai personnellement choisi de rédiger les cahiers de tests du détail/modification et de l'archivage d'une US (User Story) et d'exécuter le cahier de tests de l'ajout/modification du sous type d'une US. Le principe est simple pour la rédaction il faut indiquer avec des étapes les actions à effectuer et le résultat attendu des suites de cette action. Et pour l'exécution il faut effectuer les actions, regarder le résultat attendu et indiquer si le résultat obtenu correspond à celui attendu et indiquer le ticket crée et la correction effectuée si ce n'est pas le cas.

## Rédaction cahier de tests du détail/modification d'une US :

		Déroulement
N°	Action	Résultat Attendu
<b>Etape 1 : Modification d'une User Story</b>		
0	Se connecter avec un profil Admin ou Manager	Affichage de la page d'accueil
1	Vérifier si la carte " GESTION DES PROJETS " s'affiche	la carte " GESTION DES PROJETS " s'affiche
2	Cliquer sur le bouton " Accéder "	la carte " GESTION DES PROJETS " tourne et le bouton " Consulter les produits " s'affiche
3	Vérifier si dans le menu latéral " Projets " s'affiche	" Projets " s'affiche dans le menu latéral
4	Cliquer sur le bouton " Consulter les produits "	Affichage de l'écran comportant la liste des produits
5	Cliquer sur un produit dans la liste des produits	Affichage de l'écran comportant la liste des User Story
6	Cliquer sur une User Story dans la liste des User Story	Affichage du détail d'une User Story à droite de la liste des User Story
7	Cliquer sur le bouton " modifier " dans le détail d'une User Story	Activation de tout les champs pour la modification
8	Saisir des données invalides (Champs vides, nombre négatif pour le poids, etc...)	Message d'erreur sur les différents champs invalides
9	Saisir des données valides et cliquer sur le bouton " Valider "	Toast modification réussie + données modifiées dans la BDD + modifications visibles dans la liste et le détail User Story
10	Modifier les champs disponibles et cliquer sur le bouton " Annuler "	le formulaire sort du mode édition + affiche les données d'origine
<b>Etape 2 : Affectation d'intervenants sur une User Story</b>		
0	Se connecter avec un profil Admin ou Manager	Affichage de la page d'accueil

## Rédaction cahier de tests de l'archivage d'une US :

		Déroulement
N°	Action	Résultat Attendu
<b>Etape 1 : Archivage d'une User Story (Admin/Manager)</b>		
0	Se connecter avec un profil Admin / Manager	Affichage de la page d'accueil
1	Vérifier si dans le menu latéral " Projets " s'affiche	" Projets " s'affiche dans le menu latéral
2	Vérifier si la carte " GESTION DES PROJETS " s'affiche	la carte " GESTION DES PROJETS " s'affiche
3	Cliquer sur le bouton " Accéder "	la carte " GESTION DES PROJETS " tourne et les boutons " Créer projet ", " Consulter projet " et " Consulter les produits " s'affichent
4	Cliquer sur le bouton " Consulter les produits "	Affichage de l'écran comportant la liste des produits
5	Cliquer sur un produit dans la liste des produits	Affichage de l'écran comportant la liste des User Story
6	Cliquer sur une User Story dans la liste des User Story	Affichage du détail d'une User Story à droite de la liste des User Story
7	Cliquer sur le bouton " modifier " dans le détail d'une User Story	Activation de tout les champs et des bouton " intervenants " et " archivage " pour la modification
8	Cliquer sur le bouton " archiver "	Affichage de la modal d'archivage d'une User Story
9	Cliquer sur le bouton " Valider "	Toast archivage réussi + donnée " ARCHIVER " passé à " TRUE " dans la BDD + le détail de la User Story disparaît + la User Story disparaît de la liste des User Story
10	Cliquer sur le bouton " Annuler "	la modal d'archivage disparaît + retour à la modification d'une User Story + la donnée " ARCHIVER " reste à " FALSE " dans la BDD

## Exécution cahier de tests de l'ajout/modification d'une US :

	Action	Résultat Attendu	Cas de tests	Résultat obtenu	Ticket	Correction effectuée
<b>Etape 1 : Accéder a l'affichage des sous type en tant qu'Admin/Manager</b>						
0	Se connecter avec un profil administrateur/manager	Affichage de la page d'accueil	Connexion en tant qu'admin	Affichage de la page d'accueil		
1	Vérifier si la carte "GESTION DE PROJETS" s'affiche	La carte "GESTION DE PROJET " s'affiche	Vérification de l'affichage de la carte	La carte "GESTION DE PROJET" s'affiche		
2	Cliquer sur le bouton "Accéder"	La carte "GESTION DE PROJET" tourne et les boutons "Créer un projet", "Consulter un projet" et "Consulter les produits" s'affichent	Click bouton "Accéder"	La carte "GESTION DE PROJET" tourne et les boutons "Créer un projet", "Consulter un projet" et "Consulter les produits" s'affichent		
3	Cliquer sur le bouton "Consulter les produits"	Affichage de la liste de tous les produits	Click bouton "Consulter les produits"	Affichage de la liste de tous les produits		
4	Cliquer sur un produit	Affichage des Users Stories du produit	Click sur un produit	Affichage des Users Stories du produit		
5	Cliquer sur l'onglet "Params Users stories"	Affichage des Sous Types	Click onglet "Params User Story"	Affichage des Sous Types		
<b>Etape 2 : Ajouter un sous type à une user story en tant qu'Admin/Manager</b>						
0	Se connecter avec un profil administrateur/manager	Affichage de la page d'accueil	Connexion en tant qu'admin	Affichage de la page d'accueil		
1	Vérifier si la carte "GESTION DE PROJETS" s'affiche	La carte "GESTION DE PROJET " s'affiche	Vérification de l'affichage de la carte	La carte "GESTION DE PROJET" s'affiche		
2	Cliquer sur le bouton "Accéder"	La carte "GESTION DE PROJET" tourne et les boutons "Créer un projet", "Consulter un projet" et "Consulter les produits" s'affichent	Click bouton "Accéder"	La carte "GESTION DE PROJET" tourne et les boutons "Créer un projet", "Consulter un projet" et "Consulter les produits" s'affichent		
3	Cliquer sur le bouton "Consulter les produits"	Affichage de la liste de tous les produits	Click bouton "Consulter les produits"	Affichage de la liste de tous les produits		
4	Cliquer sur un produit	Affichage des Users Stories du produit	Click sur un produit	Affichage des Users Stories du produit		
5	Cliquer sur l'onglet "Params Users stories"	Affichage des Sous Types	Click onglet "Params User Story"	Affichage des Sous Types		
6	Cliquer sur le bouton "Ajouter un sous type"	un modal "Créer un Sous Type" s'ouvre	Click bouton "Ajouter un sous type"	Affichage du modal de création de sous type		
7	Entrer des données valide pour le sous type dans l'input et appuyer sur "AJOUTER"	Toast d'ajout + ajout du sous type	Données entrées pour la création d'un sous type	Toast d'ajout + ajout du sous type		
<b>Etape 3 : Modification d'un sous type d'une user story en tant qu'Admin/Manager</b>						
0	Se connecter avec un profil administrateur/manager	Affichage de la page d'accueil	Connexion en tant qu'admin	Affichage de la page d'accueil		
1	Vérifier si la carte "GESTION DE PROJETS" s'affiche	La carte "GESTION DE PROJET " s'affiche	Vérification de l'affichage de la carte	La carte "GESTION DE PROJET" s'affiche		
2	Cliquer sur le bouton "Accéder"	La carte "GESTION DE PROJET" tourne et les boutons "Créer un projet", "Consulter un projet" et "Consulter les produits" s'affichent	Click bouton "Accéder"	La carte "GESTION DE PROJET" tourne et les boutons "Créer un projet", "Consulter un projet" et "Consulter les produits" s'affichent		

## Bilan

Ce stage m'a été fortement bénéfique. J'ai beaucoup appris.

Le projet ELM m'a permis d'observer que sur un projet existant les besoins du client peuvent changer et qu'il faut pouvoir créer, recréer, remodeler et s'adapter à ce qui est déjà existant pour correspondre aux différents besoins.

L'accueil d'INSY2S a été incroyable, ils étaient toujours là quand nous avions besoins d'aide ou pour nous partager des connaissances. Pour cela je les remercie.

J'ai aussi appris à utiliser React que je trouve fortement intéressant. J'ai pu consolider mes connaissances du front-end. J'ai aussi eu de la chance d'être dans une équipe dans laquelle l'entraide était la priorité. Même si cette équipe était composée de certaines personnes qui sont issues de ma formation c'étaient majoritairement des personnes avec qui j'avais assez peu discuté lors de la formation, cela m'a permis de les connaître beaucoup mieux.

# Annexes

## Annexe A (Écrans Jira) :

### Écran d'un sprint sur Jira :

The screenshot shows a Jira sprint board for 'Team Akatsuki- Sprint 4'. The board is divided into three columns: TO DO, IN PROGRESS, and DONE. Each column contains several task cards with details like issue ID, title, and assignee.

TO DO	IN PROGRESS	DONE
ELM-1713 FRONT - Table de Référence chiffage - Table ContexteRef Feature Gestion de projet sous ELM Assigné à: 10	ELM-1723 Florent - FRONT - Filtre User Story Feature Gestion de projet sous ELM Assigné à: 13	ELM-1720 Mathias - FRONT - Update Produit - Ajout nouveaux Champs Feature Gestion de projet sous ELM Assigné à: 5
ELM-1719 FRONT - Table de Référence chiffage - Table ParametreChiffageRef Feature Gestion de projet sous ELM Assigné à: 10	ELM-1711 Mathias - FRONT - Table Référence chiffage - Table CategoryRef Feature Gestion de projet sous ELM Assigné à: 8	ELM-1721 Romain - FRONT - Create User Story - Ajout nouveaux champs Feature Gestion de projet sous ELM Assigné à: 6
ELM-1714 FRONT - Table de Référence chiffage - Table PhaseProjetRef Feature Gestion de projet sous ELM Assigné à: 10	ELM-1717 Steeven - FRONT - Table de Référence chiffage - Table TechnologieRef Feature Gestion de projet sous ELM Assigné à: 8	ELM-1722 Yoan - FRONT - Update User Story - Ajout nouveaux champs Feature Gestion de projet sous ELM Assigné à: 6
ELM-1718 FRONT - Table de Référence chiffage - Table AbaqueRef Feature Gestion de projet sous ELM Assigné à: 13	ELM-1712 Yoan - FRONT - Table de Référence chiffage - Table ComplexiteRef Feature Gestion de projet sous ELM Assigné à: 8	ELM-1724 Steeven - FRONT - affichage US type Trello nouveaux champs + lien Feature Gestion de projet sous ELM Assigné à: 3
	ELM-1715 Romain - FRONT - Table de Référence chiffage - Table TypeIntervenantRef Feature Gestion de projet sous ELM Assigné à: 10	ELM-1726 Florent - FRONT - affichage US type liste Feature Gestion de projet sous ELM Assigné à: 3
	ELM-1716 RJ-FRONT - Table de Référence chiffage - Table valeurContexteRef Feature Gestion de projet sous ELM	ELM-1727 Romain - FRONT - affichage produit tableau Feature Gestion de projet sous ELM

### Capture d'écran d'un ticket Jira :

The screenshot shows a Jira ticket detail view for issue ELM-1722. The ticket is titled 'Yoan - FRONT - Update User Story - Ajout nouveaux champs' and has an estimate of 6. The description section contains details about the user story and the new fields to be added.

**ELM / ELM-1722**

Yoan - FRONT - Update User Story - Ajout nouveaux champs

Estimate: 6

**Description**

En tant que **Admin Manager** ou **Employe** je souhaite **visualiser** les **nouveaux champs** d'une user story.

En tant que **Admin** ou **Manager** je souhaite **modifier** les **nouveaux champs** d'une user story

Les nouveaux champs sont :

- Poids (entier positif)
- Priorité (table de référence)
- Urgent (table d référence)

**Règle de gestions**

- Seul l'admin et le manager peuvent modifier les champs
- Le poids est un entier positif

**Composant à modifier :**

**Comments**

There are no comments yet on this issue.

## Annexe B (Écrans PlanItPoker) :

### Écran de choix de points d'efforts pour un ticket sur PlanItPoker :

The screenshot shows the PlanItPoker interface for a ticket titled "ELM - Akatsuki - Sprint 3". The main area displays a grid of story points: 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100, and a wildcard card with a question mark and a coffee cup icon. On the right, a "Waiting for moderator" panel lists players: Marine, Yoan, Mathias, Romain, Steeven, and rugabarj@gmail..., each with a timer. Below the grid, there are tabs for "Active Stories" (11), "Completed Stories" (0), and "All Stories" (11). A list of stories is visible, including "ELM-1695 FRONT > Fix > Creation US > Ne pas faire le filtre pour le choix du type en dur" and "ELM-1696 FRONT > Fix > Modification US > modification type US". A sidebar on the left contains a "Feedback & Support" button. A bottom right panel offers help with the project, stating "Adding developers to your team is easy with CodeFirst". A notification bar at the bottom right indicates "Marine joined the board." and "Marine left the board."




### Écran de résultat du choix des points d'efforts d'un ticket :

The screenshot shows the PlanItPoker interface for a ticket titled "ELM - Akatsuki - Sprint 3". The main area displays a donut chart showing the results of the story voting. The chart is divided into three segments: 5 points (28.6%, 2 players), 8 points (57.1%, 4 players), and 1 point (14.3%, 1 player). The text "7 Players voted Avg: 7" is displayed in the center of the chart. On the right, a "Story voting completed" panel lists players and their votes: Marine (5), Yoan (8), Mathias (8), Romain (8), Steeven (8), rugabarj@gmail... (8), and florent (5). Below the chart, there are tabs for "Active Stories" (8), "Completed Stories" (3), and "All Stories" (11). A list of stories is visible, including "ELM-1698 FRONT > Refact > Creation US", "ELM-1699 FRONT > Refact > détail/modification US", and "ELM-1700 FRONT > Refact > liste type". A sidebar on the left contains a "Feedback & Support" button. A bottom right panel offers help with the project, stating "Need help with the project?". A green notification bar at the bottom right indicates "Story estimate was saved."

### Annexe C (Modèle Détails/Modification d'une User Story) :

## Details du module : module 1 lot 1

---

**Date de Début**

09 juin, 2020

---

**Date de Fin prévisionnelle**

09 juin, 2020

---

**type de l' user story :**

Module ▼

---

**Date de Fin Réel**

la tâche n'est pas finie

---

**Etat :**

To Do

---

**Libelle**

dev front module projet

---

**Commentaire**

---

**Human deposit**

---

**Tags :**

---

**User story Parent :**

lot 1

**Sous type de l' user story :**

Feature ▼

---

**Sous user stories :** + Ajouter une sous user story

---

- > user story 1 - début : 10/01/21 - fin : 11/01/21 - en cours
  - > tâche 1 user story 1 - début : 10/01/21 - fin : 11/01/21 - en cours
    - > sous tâche 1 - début : 10/01/21 - fin : 11/01/21 - terminé
    - > sous tâche 2 - début : 10/01/21 - fin : 11/01/21 - en cours
  - > tâche 2 user story 1 - début : 10/01/21 - fin : 11/01/21 - à faire
- > user story 2 - début : 10/01/21 - fin : 11/01/21 - à faire



## Annexe D (Liste des User Stories avant et après archivage) :

Liste des User Stories


Ajouter une user story 

> Lot - lot1 - début : 14/01/2021 - fin prévisionnelle : 14/05/2021 - à faire - poids : 5 - priorité : forte - urgence : moyen

> Lot - lot2 - début : 13/01/2021 - fin prévisionnelle : 13/06/2021 - à faire - poids : 4 - priorité : moyen - urgence : moyen

Lot - lot1Test - début : 23/04/2021 - fin : 27/04/2021 - terminé - poids : 5 - priorité : faible - urgence : forte

Liste des User Stories

Ajouter une user story 

> Lot - lot1 - début : 14/01/2021 - fin prévisionnelle : 14/05/2021 - à faire - poids : 5 - priorité : forte - urgence : moyen

Lot - lot1Test - début : 23/04/2021 - fin : 27/04/2021 - terminé - poids : 5 - priorité : faible - urgence : forte

## Annexe E (Maquette Tableaux chiffrages) :

[illegible]

## "AJOUTER/MODIFIER"

TABLES REFERENCES CHIFFRAGES						
CategoryRef	ComplexiteRef	<u>ParametreChiffrageRef</u>	PhaseProjetRef	TechnologieRef	TypeIntervenantRef	AbaqueRef
Productivité (heures)		Taux journalier Moyen (euros)		Archivé		
7		300,00		FALSE		
<input type="text" value="Productivité"/>		<input type="text" value="TJM"/>		<input type="text" value="False"/> <input checked="" type="checkbox"/>		

## Annexe F (Composant Générique) :

```
<ComposantGeneriqueTableRef
  tableauRef={this.state.complexitesRef}
  methodePut={this.PutComplexitesRef}
  methodePost={this.PostComplexitesRef}
  />
```

```
render() {
  if (!this.state.loaded) {
    return(<></>)
  }
  return (
    <MDBContainer className='container text-center mt-3'>
      <MDBCard className='bg-primary p-3 text-white'>
        <div className='row'>
          <div className='col-4'>CODE REF</div>
          <div className='col-4'>LIBELLE</div>
          <div className='col-4'>ACTIONS</div>
        </div>
      </MDBCard>
      {this.props.tableauRef.map((ref) => {
        return(
          <MDBContainer key={ref.id}>
            <MDBRow className='border-light'>
              <MDBCol md='4' className='my-2'>{ref.codeRef}</MDBCol>
              <MDBCol md='4' className='my-2'>{ref.libelle}</MDBCol>
              <MDBCol md='4' className='my-2'>
                <MDBIcon icon='pen' className='green-text zoomIcon pointeur' title='modifier'
                  onClick={this.props.modifLibelle ? ()=>this.setUpdate(ref.id) : ()=>this.setUpdate(ref)} />
                <MDBIcon className='ml-3 text-danger zoomIcon pointeur' icon='trash-alt' title='supprimer' />
              </MDBCol>
            </MDBRow>
          </MDBContainer>
        )
      })}
    )
  )
}
```

```
<MDBRow className='d-flex justify-content-center mt-3'>
  {this.state.creation?
    <Formik
      initialValues={this.state.initialValues}
      onSubmit={this.submit}
      validationSchema={this.validationSchema}
    >
      {({ handleChange, handleSubmit, handleBlur }) => (
        <MDBFormInline onSubmit={handleSubmit}>
          <div>
            <Field className={this.state.update? 'mr-3 form-control grey-text' : 'mr-3'} name='codeRef' type='text' placeholder='Code Ref' errorRight component={INSYInput} disabled={this.state.update} />
          </div>
          <div>
            <Field className='mr-3' name='libelle' type='text' placeholder='Libelle' errorRight component={INSYInput} />
          </div>
          <MDBBtn
            size='sm'
            floating
            type='submit'
            color='white'
          >
            <MDBIcon
              style={{cursor:'pointer'}}
              size='1x' title='valider'
              className='green-text'
              icon='check'
            />
          </MDBBtn>
          <MDBBtn
            onClick={this.stopSubmit}
            size='sm'
            floating
            type='submit'
            color='white'
          >
            <MDBIcon
              style={{cursor:'pointer'}}
              icon='times'
              title='Annuler'
              className='red-text'
            />
          </MDBBtn>
        </MDBFormInline>
      )}
    </Formik>
  )
  :
  <MDBBtn size='sm' rounded type='submit' color='white' onClick={() => this.setState({creation:true})}>
    <MDBIcon icon='plus' title='Ajouter Module' className='green-text mr-2' /> Ajouter
  </MDBBtn>
</MDBRow>
</MDBContainer>
```