

Rapport d'activité

Morilleau Freddy

SN2

Stage effectué chez l'entreprise DCH-IT

Dans le cadre d'un BTS SIO option SLAM

Avec Dawid Chalasiewicz,

Gérant de l'entreprise

Du 11 janvier 2024 au 23 février 2024

Sommaire

Remerciements.....	3
Introduction.....	4
L'entreprise.....	5
Les missions.....	6
Conclusion.....	24
Annexes.....	25

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué de près ou de loin au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, j'adresse mes remerciements à mon maître de stage, Dawid Chalasiewicz, gérant de l'entreprise DCH-IT qui nous a accueilli, moi ainsi que deux autres stagiaires, Sasha Wilk et Mathéo Delaunay et qui nous a accompagné tout au long de notre période de stage.

Mes remerciements vont aussi au service relations entreprises de l'EPSI qui m'a mis en contact avec Dawid Chalasiewicz et m'a offert l'opportunité de faire un stage chez DCH-IT. J'ai ainsi eu l'occasion d'effectuer un stage en adéquation avec ce que je recherchais dans le milieu du développement.

Mes remerciements vont également à Sasha Wilk et Mathéo Delaunay, des camarades de 2ème année qui ont effectué leur stage avec moi au sein de l'entreprise DCH-IT. Leur présence a été appréciée, offrant des avis variés et enrichissants sur les différents aspects du projet.

Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage : ma famille, mes camarades stagiaires Sasha Wilk et Mathéo Delaunay.

Introduction

Dans le cadre de ma seconde année d'étude en développement informatique à l'EPSI à Nantes, j'ai eu l'occasion de réaliser un stage au sein de l'entreprise DCH-IT d'une durée de 7 semaines du 11 janvier 2024 au 23 février 2024 en tant que développeur informatique. Les langages de programmation utilisés ont essentiellement été PHP, Javascript, HTML et CSS ainsi que le framework Symfony.

Cette mission a été confiée par l'entreprise C&D Compétences et Développement qui est un réseau d'écoles et de centres de formation. Cette entreprise a confié à l'entreprise DCH-IT pour laquelle j'effectue ce stage la mission de réaliser un plan interactif du bâtiment du Campus de Nantes hébergeant différentes écoles parmi lesquelles se trouve l'EPSI, l'école où j'effectue mes études, mais aussi l'IDRAC, l'IGEFI, l'ICL, l'IEFT, SUP DE COM, WIS, l'IET et l'IFAG.

Le but de ce stage a donc exclusivement été le développement du site web contenant le plan interactif du bâtiment et donc le développement du plan interactif en lui-même.

L'entreprise

L'entreprise pour laquelle j'ai effectué ce stage est l'entreprise DCH-IT.

L'entreprise est située au 2 rue Charles Deslescluze à Saint-Nazaire.

Il s'agit d'une entreprise de services informatiques pour particuliers et professionnels.

Les missions

Au début du stage, nous nous retrouvions tous les matins avec les autres stagiaires et Dawid notre maître de stage pour mettre en place le projet et définir les différentes missions de chaque stagiaire durant le stage.

Par la suite, nous nous retrouvions plusieurs fois par semaine pour discuter de l'avancée de chacun dans les différentes missions qui lui ont été confiées. On discutait aussi des potentiels problèmes que l'on pouvait rencontrer lors du développement, l'expérience de Dawid en développement et le fait d'être 3 stagiaires à aussi beaucoup aidé à la résolution de problèmes, car cela apportait plus de points de vue sur la question et ainsi une meilleure visibilité sur les problèmes en général ce qui a permis de trouver des solutions à ces problèmes plus facilement. Une fois des missions terminées, on discutait des missions à effectuer par la suite

Durant ce stage, les missions qui nous ont été confiées ont essentiellement été le développement des différentes fonctionnalités du projet de site de plan interactif. Il y avait donc tout ce qui concerne la création de compte par les utilisateurs, la connexion, le plan interactif qui est la fonctionnalité majeure du projet et bien sûr le design de toutes ces différentes pages.

Et pour ce qui est des missions qui m'ont été confiées, je me suis principalement occupé de la partie développement et des fonctionnalités d'inscription, c'est-à-dire de création de compte, et du plan interactif et des différentes fonctionnalités en rapport avec le plan.

- Organisation du projet

Notre première importante mission durant ce début de stage a donc été de nous occuper, moi et les autres stagiaires, de l'organisation du projet.

Pour organiser le projet et définir les missions de chacun, nous avons donc commencé à utiliser Trello, un site de gestion de projet en ligne qui fonctionne à l'aide de cartes représentant chacune une mission à faire ou une fonctionnalité à développer. (Voir annexe n°1)

Trello est un outil très utile et possède de nombreuses fonctionnalités, la principale est la création de cartes avec un titre et une description pour définir en détail la mission afin de savoir précisément ce qu'il y a à faire et à quoi cela va servir.

Une autre fonctionnalité du site très utile est la possibilité de faire des check list pour chaque carte, autrement dit des listes à cocher permettant de structurer encore plus les différentes missions. Cela permet de définir les différentes choses à implémenter pour le bon fonctionnement de la fonctionnalité et permet ainsi de simplifier le suivi de l'avancé d'une mission. Il y a aussi possibilité d'ajouter des documents en rapport avec les missions, par exemple pour le développement du plan interactif, il y avait les documents des plans du bâtiment.

Une autre fonctionnalité majeure de Trello est la possibilité d'assigner des tâches à des personnes, ce qui nous a permis de définir clairement les missions de chacun durant le stage afin que l'organisation du projet soit bien structurée et que l'on n'empiète pas sur le travail des uns et des autres.

On peut aussi mettre des dates d'échéances pour définir quand certaines fonctionnalités doivent être terminées. Cela permet d'établir un ordre de priorité entre les missions et cela est très utile car certaines fonctionnalités dépendent d'autres et ne peuvent pas être développées avant les autres.

Pour le début de ce stage, nous avons donc commencé par réfléchir aux différentes fonctionnalités à développer et déterminer à qui attribuer les différentes missions.

Nous avons aussi déterminé le langage de programmation avec lequel on allait développer le projet. Notre choix s'est porté sur Symfony, un framework PHP, c'est-à-dire une infrastructure de développement en PHP

dont on a appris l'utilisation il y a peu durant des cours à l'EPIS. Nous avons donc choisi Symfony car nos notions étaient fraîches et aussi pour s'améliorer car ce framework PHP est beaucoup utilisé dans le développement de sites ou d'applications web. Il est très populaire car il permet la création de sites ou d'applications Web complexes, robustes, fiables, évolutifs, maintenables et performants.

- Développement du menu et du footer

La première mission de développement que j'ai effectuée a été la création du menu et du footer du site (voir annexe n°2). Le menu consiste en une liste structurée de liens cliquables permettant aux utilisateurs de naviguer facilement entre les différentes parties et fonctionnalités du site. Ce menu est situé à gauche de l'écran sur la page principale.

Cette fonctionnalité est codée essentiellement en HTML avec un peu de PHP pour personnaliser le menu en fonction de si l'utilisateur est connecté ou non et si le compte connecté est un administrateur ou non.

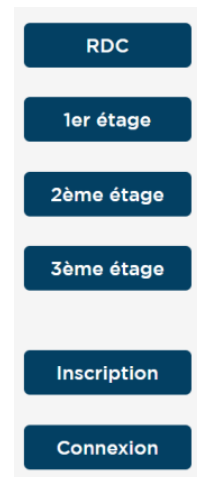
Au niveau du code, pour faire la liste on utilise des balises `` et `` qui permettent de créer des listes d'éléments. On utilise d'abord une balise `` avec plusieurs balises `` qui représente des éléments de la liste et qui contiennent les liens des différents étages du plan interactif avec le rez-de-chaussée et les 3 étages.

On crée ensuite une autre balise `` pour afficher tout ce qui concerne la connexion, l'inscription et l'administration. Pour gérer la personnalisation, on implémente du code PHP à l'aide des balises `{% %}` afin de vérifier le statut de connexion de l'utilisateur. Pour vérifier cela, on fait le test `'if app.user'` qui teste si un utilisateur est authentifié dans l'application.

Donc si l'utilisateur n'est connecté à aucun compte, on affiche uniquement les boutons 'Inscription' et 'Connexion' qui renvoient aux fonctionnalités correspondantes qui lui permettent donc soit de créer un compte soit de se connecter à un compte déjà existant dans la base de données.

Dans le cas où l'utilisateur serait connecté, on lui affiche plus de choses, on lui affiche déjà le bouton 'Déconnexion' qui offre la possibilité de se déconnecter ce qui le renverra sur la page de connexion. De plus, on affiche le bouton 'Mon compte' qui renvoie à une page permettant à l'utilisateur d'avoir accès à toutes les informations le concernant, son prénom, son nom, son mail, si son compte est vérifié et le bureau dans lequel il s'est enregistré.

Un autre bouton s'affiche aux utilisateurs si leur compte possède le rôle administrateur, on vérifie cela avec le test `"app.user.roles[0] ==`



“ROLE_ADMIN”” et cela donne l'accès au bouton 'Admin' qui donne accès à une page qui offre plusieurs actions telles que la possibilité de gérer les utilisateurs ainsi qu'avoir accès aux informations de tous les utilisateurs, la possibilité de gérer les salles du plan interactif, c'est-à-dire d'en ajouter et la possibilité de gérer les étages et d'en ajouter.

```
<div>
  <ul class="menu-ul">
    <li class="menu-li"><a href="{{ path('app_show_plan_by_id', {'id': 0}) }}">RDC</a></li>
    <li class="menu-li"><a href="{{ path('app_show_plan_by_id', {'id': 1}) }}">1er étage</a></li>
    <li class="menu-li"><a href="{{ path('app_show_plan_by_id', {'id': 2}) }}">2ème étage</a></li>
    <li class="menu-li"><a href="{{ path('app_show_plan_by_id', {'id': 3}) }}">3ème étage</a></li>
  </ul>

  <ul class="menu-ul">
    <li class="menu-li"><a href="{{ path('app_register') }}">Inscription</a></li>
    {% if app.user %}
      <li class="menu-li"><a href="{{ path('app_logout') }}">Déconnexion</a></li>
      <!-- Utilisation de app.user.id pour récupérer l'ID de l'utilisateur connecté -->
      <li class="menu-li"><a href="{{ path('app_admin_user_show', {'id': app.user.id}) }}">Mon Compte</a></li>
      {% if app.user.roles[0] == "ROLE_ADMIN" %}
        <li class="menu-li"><a href="{{ path('app_admin_dashboard') }}">Admin</a></li>
      {% endif %}
    {% else %}
      <li class="menu-li"><a href="{{ path('app_login') }}">Connexion</a></li>
    {% endif %}
  </ul>
</div>
```

Pour ce qui est du footer, autrement dit le pied de page, il est codé exclusivement en HTML et affiche différentes informations comme la mention de droit d'auteur C&D ainsi que les différents liens vers les pages LinkedIn des stagiaires et développeurs du site que sont Sasha, Mathéo et moi, Freddy. Quand on affiche les droits d'auteur, on utilise la syntaxe “{{ "now"|date("Y") }}" qui récupère la date d'aujourd'hui et affiche l'année.

```
<footer>
  <p>© {{ "now"|date("Y") }} C&D - Tous droits réservés</p>
  <div class="dev-link">
    <p>Devs : </p>
    <a href="https://www.linkedin.com/in/sasha-wilk/" target="_blank" alt="Sasha Wilk">Sasha</a>
    <p> - </p>
    <a href="https://www.linkedin.com/in/matheo-de-launay/" target="_blank" alt="Mathéo DeLaunay">Mathéo</a>
    <p> - </p>
    <a href="https://www.linkedin.com/in/freddy-morilleau/" target="_blank" alt="Freddy Morilleau">Freddy</a>
  </div>
</footer>
```

- Création de l'inscription

Ma première grosse mission a été de mettre en place la fonctionnalité d'inscription sur le site (voir annexe n°3).

L'inscription est une fonctionnalité très importante car comme son nom l'indique, elle permet aux utilisateurs de créer leur propre compte, leur donnant ainsi accès à l'ensemble des fonctionnalités du site. Lorsque

l'utilisateur arrive sur le site, on lui propose d'abord de se connecter (voir annexe n°4) et dans le cas où il n'a pas encore de compte, on lui propose d'en créer un en cliquant sur "Pas encore de compte ?". Une fois ce lien activé, l'utilisateur est automatiquement redirigé vers la page d'inscription, où il peut procéder à la création de son compte en fournissant les informations nécessaires.

COMPÉTENCES
& DÉVELOPPEMENT
Réseau d'écoles et de centres de formation

Bonjour ! [Pas encore de compte ?](#)

Email

Password

☐ Se souvenir de moi [Mot de passe oublié ?](#)

La page d'inscription consiste simplement en un formulaire où l'on demande à l'utilisateur différentes informations que sont son prénom, son nom, son bureau si la personne en possède un dans le bâtiment, si elle n'en a pas elle ne met rien, ensuite, on demande l'email et le mot de passe de l'utilisateur et on lui demande d'accepter les conditions. Par la suite l'utilisateur à juste à cliquer sur le bouton "S'inscrire" et il est redirigé vers la page de connexion. A son inscription, un mail est envoyé à un administrateur pour lui permettre d'accepter ou de refuser l'inscription de l'utilisateur. Après la connexion, si l'inscription a été acceptée par l'administrateur, il peut alors accéder à la page

COMPÉTENCES
& DÉVELOPPEMENT
Réseau d'écoles et de centres de formation

Bonjour ! [Déjà un compte ?](#)

Prénom

Nom

Bureau

Email

Mot de passe

Agree terms ☐

principale du site contenant le plan interactif. En revanche, si son inscription n'a pas encore été acceptée, alors il est redirigé vers une page lui signalant que sa demande d'inscription est toujours en cours et doit attendre qu'un administrateur étudie sa demande. Il reçoit par la suite un mail lui signalant si son compte a été accepté ou refusé. Si son compte est accepté, il peut alors se connecter au site et avoir accès au plan interactif.

```
#[Route('/register', name: 'app_register')]
public function register(Request $request, UserPasswordHasherInterface $userPasswordHasher, EntityManagerInterface $entityManager): Response
{
    $user = new User();
    $form = $this->createForm( type: RegistrationFormType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        // encode le mot de passe en clair
        $user->setPassword(
            $userPasswordHasher->hashPassword(
                $user,
                $form->get('plainPassword')->getData()
            )
        );

        $entityManager->persist($user);
        $entityManager->flush();
    }
}
```

Au niveau du code, on crée d'abord une nouvelle instance de l'entité User qui va nous permettre de créer un nouvel utilisateur et de stocker ses données. Ensuite, on crée le formulaire d'inscription.

On fait ensuite un test pour savoir si le formulaire a été envoyé par l'utilisateur et qu'il est valide, autrement dit que l'utilisateur a cliqué sur "S'inscrire" et à entrer des données valides. Si ce test est un succès alors on commence d'abord par modifier le mot de passe que l'utilisateur a choisi afin de le hacher, le hacher signifie le transformer en une chaîne de caractères, cela permet de protéger sa confidentialité. Lorsqu'un mot de passe est haché, le résultat est une valeur de hachage qui ne peut généralement pas être inversée pour retrouver le mot de passe original.

Une fois le mot de passe haché, on prépare notre nouvel utilisateur à être ajouté à la base de données puis on l'ajoute.

- Création des mails de validation et de refus

Cette mission suit la mission précédente mais la fonctionnalité développée est destinée aux administrateurs afin de gérer les inscriptions des utilisateurs.

Après avoir créé le nouvel utilisateur dans la base de données, on envoie un mail à un administrateur contenant les informations de la nouvelle inscription tels que le prénom, le nom, le mail et la salle dans laquelle l'utilisateur s'est enregistré. Ce mail contient aussi deux boutons, un bouton "Accepter" qui permet d'accepter l'inscription de l'utilisateur en question, et un bouton "Refuser" qui permet de refuser son inscription.

```
// Envoie un mail de confirmation d'inscription à Dawid
$email = (new TemplatedEmail())
->from(new Address(address: 'noreply@dch-it.fr', name: 'DCH-IT'))
->to($_ENV['EMAIL_ADDRESS'])
->subject('Nouvelle inscription')
->htmlTemplate('registration/confirmation_register.html.twig')
->context([
    'user' => $user,
    'acceptRegistrationUrl' => $this->generateUrl( route: 'accept_registration', ['id' => $user->getId()], referenceType: UrlGeneratorInterface::ABSOLUTE_URL),
    'refuseRegistrationUrl' => $this->generateUrl( route: 'refuse_registration', ['id' => $user->getId()], referenceType: UrlGeneratorInterface::ABSOLUTE_URL),
]);
$this->mailer->send($email);
```

Au niveau du code de l'envoi d'e-mail, on utilise `TemplatedEmail` afin de composer et d'envoyer l'email. Pour composer l'email, on commence d'abord par déterminer l'adresse expéditrice qui est "noreply@dch-it.fr" et on définit le nom qui est "DCH-IT", ensuite, on détermine le destinataire de l'email qui sera un administrateur et on définit l'objet de l'email qui est "Nouvelle inscription". Pour ce qui est du contenu, on utilise la méthode "`htmlTemplate`" dans laquelle on spécifie le fichier "`confirmation_register.html.twig`" qui contient le code HTML de l'email. Enfin, avec la méthode "`context`" on fournit les données nécessaires au code du contenu du mail. Les données envoyées sont les données de l'utilisateur ainsi que les URLs auxquels mènent les boutons "Accepter" et "Refuser" du mail.

```
<h1>Nouvelle inscription</h1>

<p>Un nouvel utilisateur s'est inscrit</p>

<p>Prénom : {{ user.getFirstName() }}</p>
<p>Nom : {{ user.getLastName() }}</p>
<p>Email : {{ user.getEmail() }}</p>
<p>Salle : {{ user.getOffice() }}</p>

<table>
  <tr>
    <td><a href="{{ acceptRegistrationUrl }}">Accepter</a></td>
    <td><a href="{{ refuseRegistrationUrl }}">Refuser</a></td>
  </tr>
</table>
```

```
#[Route('/accept-registration/{id}', name: 'accept_registration')]
public function acceptRegistration(User $user, EntityManagerInterface $entityManager, SessionInterface $session): Response
{
    $user->setIsVerified( isVerified: true);
    $entityManager->persist($user);
    $entityManager->flush();

    // Envoie un mail pour prévenir de la validation de son compte à l'utilisateur
    $email = (new TemplatedEmail())
        // À changer par la bonne adresse expéditrice
        ->from(new Address( address: 'noreply@dch-it.fr', name: 'DCH-IT'))
        ->to($user->getEmail())
        ->subject( subject: 'Inscription acceptée')
        ->htmlTemplate( template: 'registration/accept_registration.html.twig')
        ->context([
            'user' => $user,
            'loginUrl' => $this->generateUrl( route: 'app_login', [], referenceType: UrlGeneratorInterface::ABSOLUTE_URL),
        ]);
    $this->mailer->send($email);

    $session->set('registration_accepted', true);

    return $this->redirectToRoute( route: 'app_home');
}
```

Lorsque l'administrateur clique sur le bouton "Accepter", on passe l'utilisateur en "vérifié" dans la base de données puis on lui envoie un mail de la même manière qu'à l'administrateur.

Dans ce mail, on envoie comme contenu un message lui signalant que son inscription a été acceptée et lui proposant de se connecter (voir annexe n°5).

```
<h1>Inscription acceptée</h1>
<p>Votre inscription a été validée</p>
<p>Connecter-vous dès maintenant à votre compte</p>
<a href="{{ loginUrl }}">Se connecter</a>
```

```
#[Route('/refuse-registration/{id}', name: 'refuse_registration')]
public function refuseRegistration(User $user, EntityManagerInterface $entityManager, SessionInterface $session): Response
{
    $entityManager->remove($user);
    $entityManager->flush();

    // Envoie un mail pour prévenir du refus d'inscription à l'utilisateur
    $email = (new TemplatedEmail())
        // À changer par la bonne adresse expéditrice
        ->from(new Address( address: 'noreply@dch-it.fr', name: 'DCH-IT'))
        ->to($user->getEmail())
        ->subject( subject: 'Inscription refusée')
        ->htmlTemplate( template: 'registration/refuse_registration.html.twig')
        ->context([
            'user' => $user,
        ]);
    $this->mailer->send($email);

    $session->set('registration_refused', true);

    return $this->redirectToRoute( route: 'app_home');
}
```

En revanche, lorsque l'administrateur clique sur le bouton "Refuser", on supprime l'utilisateur de la base de données et on lui envoie un mail (voir annexe n°6).

Dans ce mail, on envoie comme contenu un message lui signalant que son inscription a été refusée.

```
<h1>Inscription refusé</h1>

<p>Votre inscription a été refusée</p>
```

- Création de pop-up de validation et de refus

Cette mission est elle aussi liée à l'inscription.

Dawid, notre maître de stage, nous a signalé que lorsque l'administrateur clique sur les boutons "Accepter" et "Refuser", il n'a pas de confirmation sur le fait que cela a marché ou non. Pour remédier à cela, on a créé une pop-up, c'est-à-dire une fenêtre qui s'ouvre devant la fenêtre principale du site pour informer l'administrateur du succès de l'acceptation ou du refus de l'inscription.

Premièrement, on commence par créer les deux pop-up en HTML. À savoir que la balise `` possède du code HTML permettant la fermeture de la pop-up affiché.

```
<!-- Popup d'acceptation -->
<div id="popup" class="customModal">
  <div class="customModal-content">
    <p>L'inscription a été acceptée avec succès.</p>
    <span class="customClose">✕</span>
  </div>
</div>

<!-- Popup de refus -->
<div id="refusedPopup" class="customModal">
  <div class="customModal-content">
    <p>L'inscription a été refusée.</p>
    <span class="customClose">✕</span>
  </div>
</div>
```

D'abord, pour savoir si l'inscription de l'utilisateur a été acceptée ou refusée, on a créé deux variables de sessions pour l'acceptation et le refus qui passent à "vrai" en fonction de l'action qu'a effectuée l'administrateur.

Ensuite, en Javascript, on récupère nos variables de session avec “var registrationAccepted” et “var registrationRefused” et on effectue des tests. Si l’inscription a été acceptée alors on récupère notre pop-up d’acceptation et on l’affiche sur la page lorsque l’administrateur retourne sur le site. En revanche, si l’inscription a été refusée alors on récupère notre pop-up de refus et on l’affiche de la même manière.

```
document.addEventListener('DOMContentLoaded', function() {  
    // Vérifier si l'inscription a été acceptée ou refusée  
    var registrationAccepted = "{{ app.session.get('registration_accepted') }}";  
    var registrationRefused = "{{ app.session.get('registration_refused') }}";  
  
    // Afficher la popup d'acceptation si nécessaire  
    if (registrationAccepted) {  
        var popup = document.getElementById('popup');  
        if (popup) {  
            popup.style.display = 'block';  
        }  
    }  
  
    // Afficher la popup de refus si nécessaire  
    if (registrationRefused) {  
        var refusedPopup = document.getElementById('refusedPopup');  
        if (refusedPopup) {  
            refusedPopup.style.display = 'block';  
        }  
    }  
}
```


- Plan interactif - Création du SVG

Ma plus grosse mission durant ce stage a été la création du plan interactif.

Le but du plan interactif est de pouvoir voir où se situent les salles dans le bâtiment et aussi de pouvoir cliquer sur chaque salle afin de connaître les personnes qui se situent dedans, et notamment pour savoir où se situe le bureau de telle ou telle personne.

Pour faire ce plan interactif, la première chose que nous avons eue à faire a été la création des SVG, chaque SVG représentant un étage du bâtiment. SVG signifie Scalable Vector Graphics, ce qui se traduit littéralement par "Graphiques vectoriels évolutifs" en français. C'est un format de fichier graphique, qui est utilisé pour décrire des images vectorielles.

L'intérêt de créer un SVG est d'abord d'avoir une image utilisant des vecteurs ce qui permet un redimensionnement sans perte de qualité car les vecteurs représentent des formes géométriques mathématiques plutôt que des pixels. Ainsi, quelle que soit la taille à laquelle l'image SVG est agrandie ou réduite, sa netteté et sa clarté sont préservées.

De plus, et il s'agit là tout l'intérêt d'un SVG, c'est que l'on peut rendre ces images interactives avec du Javascript et les personnaliser avec du CSS. C'est donc avec cela que l'on va créer notre plan interactif.

Pour faire nos différents SVG correspondant aux différents étages, nous avons utilisé Photoshop qui est un logiciel de retouche d'images et de création graphique capable de créer des SVG.

L'intérêt de Photoshop est de pouvoir créer des calques qui permettent de superposer des éléments graphiques et de les éditer individuellement.

Malheureusement, notre premier essai n'a pas été concluant car on pensait qu'il suffisait d'un seul calque contenant l'entièreté du plan, or, on se trompait.

Pour bien réussir à créer un SVG, il a fallu créer un calque par salle en plus du calque contenant tous les murs de l'étage (voir annexe n°7). Cela fait en sorte que lorsque l'on exporte le fichier sous format SVG, cela crée une zone distincte pour chaque salle que l'on peut exploiter distinctement. Pour un souci de visibilité lors du développement, on a mis une couleur différente à chaque calque.

- Plan interactif - Exploitation du SVG

Une fois nos fichiers SVG terminés, j'ai pu passer à l'exploitation de ces fichiers.

Lorsque l'on exporte le fichier SVG, on obtient un fichier contenant des balises XML. Comme HTML, XML utilise des balises pour structurer les données ce qui fait que l'on peut inclure du XML dans nos pages HTML afin de les exploiter.

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  width="12258" height="6088" viewBox="0 0 12258 6088">
  <g>
    <!-- Zone de l'ascenseur -->
    <path id="Ascenseur" data-name="Ascenseur" class="cls-1"
      d="M2288,3259L327-1-3,356-321,8Z"/>
    <!-- Texte de l'ascenseur -->
    <text x="2350" y="3470">Asc.</text>
  </g>
  <g>
    <!-- Zone de la cafétéria -->
    <path id="Cafétéria" data-name="Cafétéria" class="cls-2"
      d="M9968,4877L-62-3047,1589,22s230-4.34,270,415c0,0
      137,934.66-85,1523s-553.4,828.73-904,919S10237.9,4830.48,9968,4877Z"/>
    <!-- Texte de la cafétéria -->
    <text x="10500" y="3300">Cafétéria</text>
  </g>
```

Au niveau du code, on retrouve d'abord une balise <svg> qui permet d'intégrer notre SVG dans notre page. Dans cette balise, on retrouve les attributs "width" et "height" qui définissent respectivement la largeur et la hauteur de notre élément SVG et on retrouve l'attribut "viewBox" qui définit la boîte de vue de notre élément, c'est-à-dire la partie de l'espace de dessin SVG qui sera visible à l'écran. Les quatre nombres dans cet attribut représentent les coordonnées x et y du coin supérieur gauche de la boîte de vue, ainsi que sa largeur et sa hauteur.

Ensuite, et ce, pour chaque salle du bâtiment, on va retrouver une balise <g> qu'on utilise pour regrouper plusieurs éléments SVG ensemble. On va aussi retrouver une balise <path> utilisée pour dessiner des formes complexes en utilisant des courbes et des segments de ligne. Cela va nous permettre de créer nos différentes zones. L'attribut "d" lui va nous permettre de spécifier les instructions de dessin qui décrivent la forme

voulue. Pour ce faire, on va utiliser des coordonnées représentées par des nombres et des actions représentés par des lettres minuscules ou majuscules.


Par exemple, pour créer la zone de notre ascenseur, on précise le point de départ de notre zone avec "M2255, 3259", M définit le point de départ de notre chemin aux coordonnées 2255 en abscisse et 3259 en ordonnée. Ensuite avec "l327-1" on trace une ligne vers un point relatif situé à une distance de 327 unités horizontales vers la droite (axe des abscisses) et à une distance de 1 unité verticale vers le haut (axe des ordonnées) par rapport à la position actuelle du point. Par la suite, on effectue d'autres actions pour tracer d'autres lignes et pour finir, on utilise la lettre "Z" qui ferme le chemin en traçant une ligne de la position actuelle à la position de départ du chemin, formant ainsi la zone de notre salle.

Maintenant que nos zones sont créées, on ajoute des balises <text> qui vont nous permettre comme leur nom l'indique d'ajouter du texte à des positions spécifiques. Pour ce faire, il suffit simplement de spécifier respectivement dans les attributs "x" et "y" la position en abscisses et en ordonnées du texte.

Dans notre cas, on va d'abord utiliser ces balises pour afficher le nom des salles et on va ensuite les utiliser pour afficher à quelle école appartiennent certaines salles.

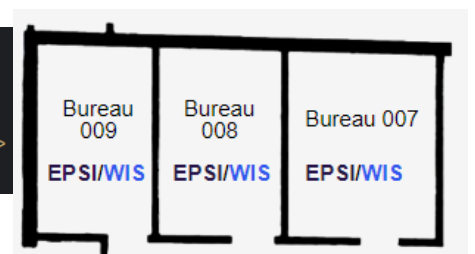
Par exemple, pour notre ascenseur, on va afficher le texte "Asc." aux coordonnées d'abscisse 2350 et d'ordonnées 3470

```
<!-- Texte de l'ascenseur -->  
<text x="2350" y="3470">Asc.</text>
```



Et pour l'affichage des écoles, le principe est le même en rajoutant de la couleur au texte avec du CSS

```
<text x="2650" y="1100">Bureau</text>  
<text x="2730" y="1200">008</text>  
<text x="2600" y="1400" style="...">  
  <tspan fill="#271549">EPSI</tspan><tspan fill="#3058ed">WIS</tspan>  
</text>
```



À noter que pour savoir à qui appartenait chaque bureau, je me suis baladé dans tout le bâtiment.



Ensuite, je me suis occupé d'insérer dans le plan interactif des icônes de vidéoprojecteurs et d'imprimantes aux endroits où il y en a. Les vidéoprojecteurs se situent dans certaines salles et on retrouve une imprimante à chaque étage.

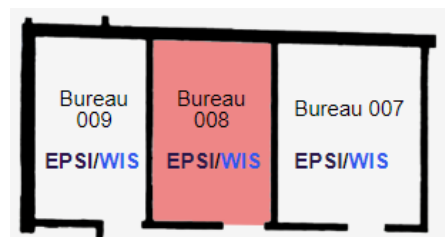
```
<image x="4150" y="2300" xlink:href="{ asset('img/projector.png') }"
width="200" height="200" />
```

Pour ce faire, on utilise des balises `<image>` qui permettent donc d'insérer des images dans le SVG, on retrouve dans ces balises les attributs "x" et "y" pour gérer la position, ainsi que les attributs "width" et "height" pour gérer la taille de l'image, enfin, on a un attribut "href" où l'on met le lien de notre icône, ici il s'agit d'un icône de projecteur.

```
<!-- Icône d'imprimante -->
<image x="5640" y="1600" xlink:href="{ asset('img/printer.jpg') }" width="250" height="250" />
```

On fait donc la même chose pour les icônes d'imprimantes.

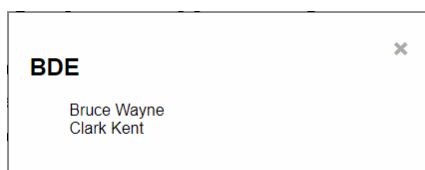
Enfin, pour l'aspect visuel du plan, on ajoute notre image représentant tous les différents murs des étages et on ajoute du CSS afin d'ajouter de la couleur aux salles lorsque l'on passe le curseur de la souris par dessus.



- Plan interactif - Création des pop-up utilisateurs

La dernière mission que j'ai effectuée au cours de ce stage a été la création de pop-up qui apparaissent lorsque l'on clique sur une salle afin de savoir qui se situe dedans.

Tout d'abord, il a fallu créer la pop-up, pour se faire, on la crée en HTML et en utilisant le moteur de template Twig qui permet de créer des modèles HTML dynamiques et nous permet de faire des tests p et ainsi savoir quoi afficher. On utilise donc twig pour savoir si la page reçoit des utilisateurs envoyés par notre contrôleur avec le test "{% if users %}". Si on en reçoit, on



affiche donc pour chaque utilisateur son prénom et son nom. En revanche, si on ne reçoit aucun utilisateur, on affiche "Aucun utilisateur ici".



```
<!-- Modal -->
<div id="myModal" class="modal">
  <div class="modal-content">
    <span class="close"></span>
    <p id="modalContent"></p>
    {% if users %}
    <ul>
      {% for user in users %}
        <li id="{{ user.office }}">{{ user.firstName }} {{ user.lastName }}</li>
      {% endfor %}
      <li id="noUser">Aucun utilisateur ici</li>
    </ul>
    {% else %}
      <p>Aucun utilisateur dans cette salle</p>
    {% endif %}
  </div>
</div>
```

Ensuite, pour envoyer des utilisateurs à notre pop-up, on a un contrôleur PHP, qui en fonction de l'étage contenu dans la variable "\$id", va chercher dans la base de données et met dans une liste tous les utilisateurs ayant pour salle une salle de l'étage en question. On envoie enfin cette liste à notre pop-up.

```
#[Route('/plan/{id}', name: 'app_show_plan_by_id', methods: ['GET'])]
public function showPlanById($id, UserRepository $userRepository): Response
{
    // Récupérer le numéro de plan à partir de la requête HTTP
    $planNumber = $id;

    // Si aucun numéro de plan n'est spécifié, afficher le premier plan par défaut
    if ($planNumber === null) {
        $planNumber = 0; // Numéro de plan par défaut
    }

    // Créer une requête personnalisée pour sélectionner les utilisateurs avec un bureau commençant par $planNumber
    $query = $this->entityManager->createQuery(
        'SELECT u FROM App\Entity\User u WHERE u.office LIKE :office'
    )->setParameter('office', $planNumber . '%');

    // Exécuter la requête
    $users = $query->getResult();

    // Sélectionner le bon template Twig en fonction du plan
    $templateName = 'home/etage' . $planNumber . '.html.twig';

    // Rendre le template Twig approprié avec les utilisateurs filtrés
    return $this->render($templateName, [
        'users' => $users,
    ]);
}
```

Enfin, pour afficher la pop-up lorsque l'on clique sur une salle, on utilise du Javascript.

Tout d'abord, on récupère toutes les balises <path> de SVG de l'étage affiché qui représentent nos salles. Ensuite, pour chaque <path> on ajoute un événement qui s'active lorsque l'utilisateur clique sur la salle.

Cet événement fait plusieurs choses, d'abord pour chaque utilisateur envoyé par le contrôleur, on va vérifier si l'utilisateur est dans cette salle, si oui, on l'affiche et on passe une variable qui vérifie la présence d'au moins une personne dans la salle à "vrai", sinon on ne l'affiche pas.

Ensuite, on regarde si des utilisateurs ont été trouvés grâce à notre variable et si aucun n'a été trouvé, on affiche le message "Aucun utilisateur ici" présent dans notre pop-up.

Et pour finir, l'événement affiche la pop-up avec le contenu modifié en fonction que l'on ait trouvé des utilisateurs ou non.

```

// Ajout d'un event listener pour chaque path
var paths = document.querySelectorAll('svg path');
paths.forEach(function(path) {
    path.addEventListener('click', function(event) {
        var pathName = event.target.dataset.name;
        var pathId = event.target.id;

        // Filtrer les utilisateurs en fonction de l'id du path
        var usersInRoom = document.querySelectorAll('#myModal .modal-content ul li');
        var userFound = false; // Variable pour indiquer si au moins un utilisateur est affiché

        usersInRoom.forEach(function(user) {
            if (user.id === pathId) {
                user.style.display = 'block'; // Afficher l'utilisateur si l'id correspond
                userFound = true; // Marquer qu'un utilisateur a été trouvé
            } else {
                user.style.display = 'none'; // Cacher l'utilisateur si l'id ne correspond pas
            }
        });

        // Si aucun utilisateur n'est trouvé, afficher "Aucun utilisateur ici"
        if (!userFound) {
            var noUserMessage = document.getElementById('noUser');
            noUserMessage.style.display = 'block';
        }

        // Get the modal element
        var modal = document.getElementById('myModal');
        // Get the content element inside the modal
        var modalContent = document.getElementById('modalContent');

        modalContent.innerHTML = '';

        var roomName = document.createElement('h2');
        roomName.innerHTML = pathName;
        modalContent.appendChild(roomName);

        modal.style.display = 'block';
    });
});

```

Conclusion

Bien que le projet fut très intéressant, celui-ci n'a pas été conduit à terme, le plan interactif n'est donc pas consultable aujourd'hui.

Malgré cela, j'ai tout de même appris des choses intéressantes au cours de ce stage qui me seront utiles à l'avenir, j'ai d'abord beaucoup appris en ce qui concerne la gestion de projet à plusieurs grâce à l'utilisation de Trello, je me vois d'ailleurs clairement utiliser un outil du genre pour de futurs projets.

J'ai aussi appris la coopération entre stagiaires ainsi que le partage des tâches tout en pouvant compter sur l'avis des autres lorsque l'on fait face à des problèmes ou lorsque l'on a des doutes sur certaines choses.

Sur un niveau plus technique, j'ai appris à faire des plans interactifs et cela a été pour moi la partie la plus stimulante et intéressante. J'ai appris à créer des SVG et à les exploiter et je suis reconnaissant d'avoir eu l'occasion de travailler sur ce sujet. J'ai de plus appris à mieux développer sur le framework Symfony qui est un outil très utile.

Et j'ai aussi acquis une meilleure connaissance du bâtiment ce qui me sera sans nul doute plutôt utile.

Je remercie de nouveau David Chalasiewicz, notre maître de stage pour nous avoir accueilli au sein de l'entreprise DCH-IT et de nous avoir accompagné durant le stage.

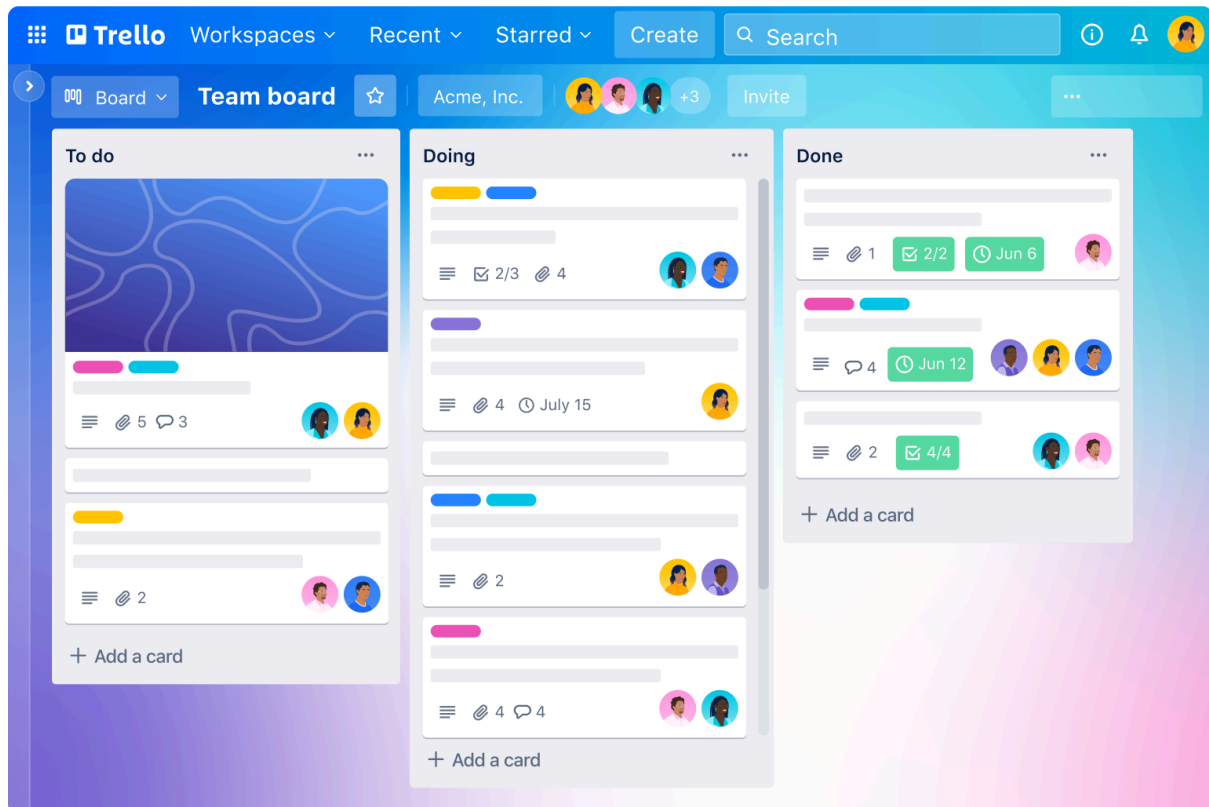
Merci aussi à Sasha Wilk et Mathéo Delaunay pour m'avoir accompagné tout au long de ce stage.

Pour finir, ce stage a été intéressant et m'a permis de découvrir de nouvelles choses en développement qui me seront certainement utiles à l'avenir.

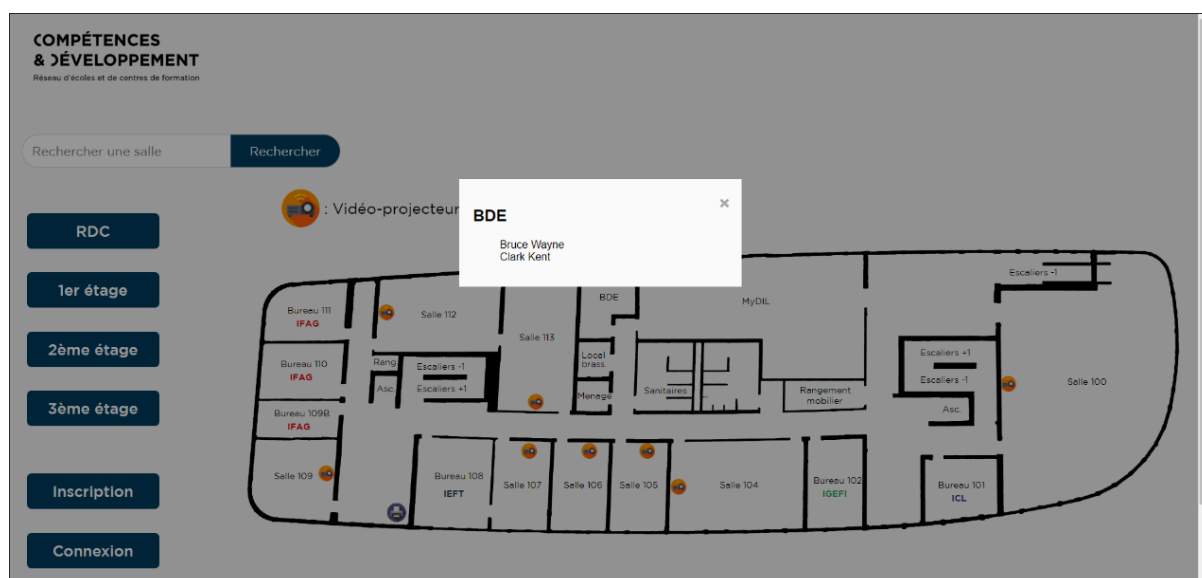
Bien que le projet n'ait pas pu aboutir, cela reste une expérience intéressante et je remercie Dawid pour cela.

Annexes

Annexe n°1 - Exemple de Trello



Annexe n°2 - page principale avec les différents étages



Rechercher une salle

Rechercher



: Vidéo-projecteur



: Imprimante

RDC

1er étage

2ème étage

3ème étage

Inscription

Connexion



: Vidéo-projecteur



: Imprimante

RDC

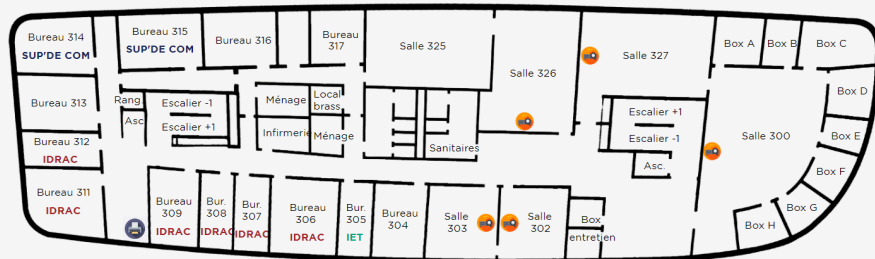
1er étage

2ème étage

3ème étage

Inscription

Connexion



Annexe n°3 - page d'inscription du site

COMPÉTENCES
& DÉVELOPPEMENT
Niveau : 2 (Niveau 1 et 2 de compétences formation)

Bonjour ! [Déjà un compte ?](#)

Prénom

Nom

Bureau

Email

Mot de passe

Agree terms ☐

Sign up

Annexe n°4 - page de connexion du site

COMPÉTENCES
& DÉVELOPPEMENT
Niveau : 2 (Niveau 1 et 2 de compétences formation)

Bonjour ! [Pas encore de compte ?](#)

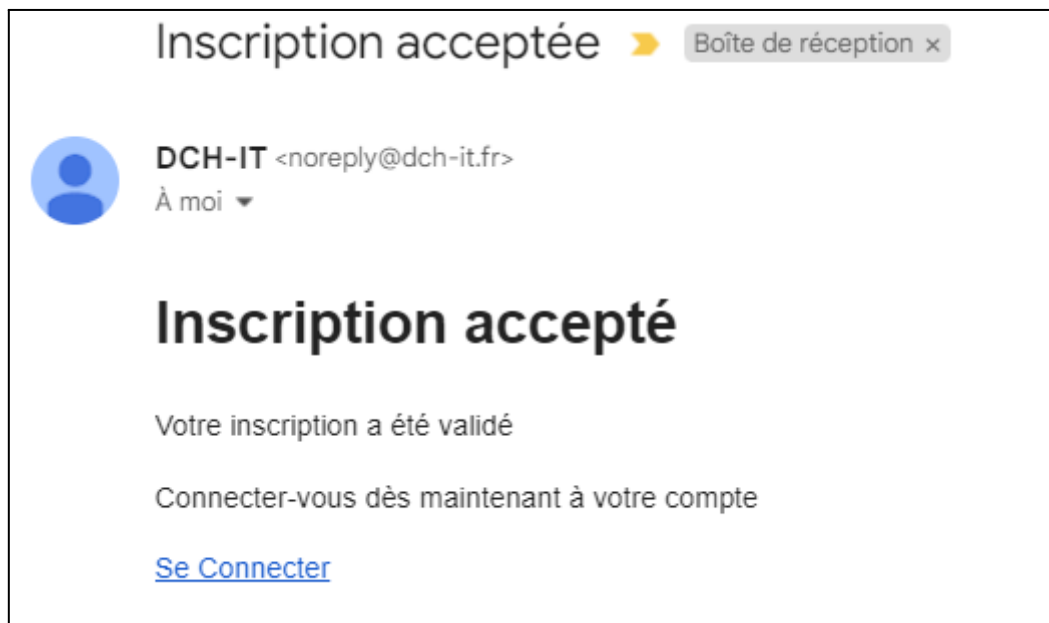
Email

Password

☐ Se souvenir de moi [Mot de passe oublié ?](#)

Sign in

Annexe n°5 - Mail d'inscription validé



Annexe n°6 - Mail d'inscription refusé



Annexe n°7 - Calques du SVG

