Alexandre PASCAL

Année 2023 - 2024

RAPPORT DE STAGE

3ème année BUT Informatique, Parcours A, Initial

Tuteur de stage : Driss Makhout

Enseignant référent : Philippe Lopisteguy

Enseignant lecteur : Marie Bruyère

Table des matières

I. Remerciements 3
II. Contexte 4
1. Présentation personnelle 4
2. Présentation de l'hôpital 5
2.1 Organisation Globale 5
2.2 Forme Juridique et organisation de la direction 5
2.3 Fonctionnement des services de soins et administratifs 5
3. Présentation du service informatique 7
4. Présentation de mon tuteur de stage 7
5. Présentation du contexte matériel 8
6. Présentation du sujet et du commanditaire 10
III. Analyse préalable 11
1. Redéfinition détaillée du sujet et des consignes 11
2. Public ciblé 12
IV. Méthode de travail 13
1. La démarche 13
1.1 Organisation et gestion de projet 13
1.2 Découverte et choix de l'environnement 16
2. Analyse 18
3. Maquettage 20
4. Développement 25
4.1 En-tête 25
4.2 Création de pages 34
4.2.1 Découverte d'éditeurs 34
4.2.2 Choix de TipTap et création du premier menu 35
4.2.3 Deuxième menu 37
4.2.4 Stockage des pages 38
4.2.5 Modification et redirection vers une page 39
4.2.6 Gestion de fichiers 40
4.2.7 Finalisation des fonctionnalités d'un menu 42
4.3 Gestion de la connexion 44
4.4 Gestion des rôles 45
4.5 Gestion de la sécurité 48
4.6 Ajout de plusieurs nouvelles fonctionnalités 49
4.7 Phase de tests 50
4.8 Déploiement 51
V. Résultats 53
1. Ce qui a été fait 53
1.1 Utilisateur déconnecté ou Lecteur 53
1.2 Le rôle d'Éditeur 57

1.3 Le rôle d’Administrateur 64
2. Ce qui devra être fait 66
3. Ce qu'il aurait été bien de faire 66
VI. Bilan 67
VII. Résumé en anglais 68
VIII. Glossaire 69
IX. Bibliographie 71
X. Annexes 71

Table des figures

Figure 1 : Plan de l'hôpital 10
Figure 2 : Grande salle informatique 10
Figure 3 : Mon premier bureau 11
Figure 4 : Mon nouveau grand bureau 11
Figure 5 : Utilisation d'un tableau blanc 11
Figure 6 : Gantt Monday 15
Figure 7 : Tableau Monday 16
Figure 8 : Kanban Monday 16
Figure 9 : Liste des éléments de l'ancien intranet 20
Figure 10 : Premiers wireframes 22
Figure 11 : Test 2 Représentation des informations dans des bulles 23
Figure 12 : Test 1 Représentation des informations dans des bulles 23
Figure 13 : Proposition de styles de menus 23
Figure 14 : Partie haute de la maquête finale 24
Figure 15 : Proposition d'arrières plans 24
Figure 16 : Création du menu 25
Figure 17 : Détails pour la maquette finale 25
Figure 18 : Représentation des animations Figma 26
Figure 19 : Liste des tables de l'ancien intranet 27
Figure 20 : Génération des fichiers NextJS 28
Figure 21 : Comparaison requêtes SQL vs Prisma 29
Figure 22 : Fichier layout.tsx 30
Figure 23 : Schéma explicatif API 32
Figure 24 : Exemple requête API 34
Figure 25 : Architecture dossier API 35
Figure 26 : Tests 52
Figure 27 : Nouveau matériel à disposition 53
Figure 28 : Page d'accueil utilisateur non connecté 55
Figure 29 : Menu utilisateur non connecté 56
Figure 30 : Exemple de page utilisateur non connecté 57
Figure 31 : Téléchargement de fichiers utilisateur non connecté 57
Figure 32 : Formulaire de connexion 58
Figure 33 : Formulaire d'inscription 58
Figure 34 : Page d'accueil Éditeur connecté 59
Figure 35 : Modification de la partie haute 60
Figure 36 : Modification des autres raccourcis 60
Figure 37 Menu Éditeur connecté 61
Figure 38 : Actions dans le menu 61
Figure 39 : Suppression dans le menu 62
Figure 40 : Création d'une page 62
Figure 41: Gestionnaire de fichiers 63
Figure 42 : Menu '/' 63

Figure 43 : Menu des tableaux 64
Figure 44 : Interface dépôt d'images 64
Figure 45 : Menu images 65
Figure 46 : Lecteur vidéos 65
Figure 47 : Menu sélection 66
Figure 48 : Affichage de la page 66
Figure 49 : Accueil Administrateur connecté 67
Figure 50 : Page d'administration 67
Figure 51: Création de rôles 68
Figure 52 : Modification de rôles 68

I. Remerciements

En premier lieu, je tenais à remercier ma famille qui a pu m'aider à chercher, et à trouver ce stage.

Je tiens également à exprimer ma plus profonde gratitude à tout le personnel de l'hôpital qui a contribué au bon déroulement de mon stage et à la réussite de mon travail.

Tout d'abord, je remercie Monsieur Pierre Nogrette, Directeur de l'hôpital de Cahors, pour m'avoir permis de réaliser mon stage au sein de son établissement.

Je remercie également Monsieur Vincent Goutines, Chef du Service Informatique, pour m'avoir accueilli dans son service, et accompagné tout au long de mon stage.

Je tiens à remercier mon maître de stage Monsieur Driss MAKHOUT, qui m'a permis de découvrir le monde de l'informatique dans un hôpital, et qui m'a donné l'opportunité de travailler sur un projet concret.

Cela a été un vrai plaisir pour moi de pouvoir travailler dans un cadre où je me sentais bien et à ma place. La bienveillance de toute l'équipe m'a vraiment permis de me sentir à l'aise et de progresser dans mon travail.

Je tenais également à remercier toute l'équipe pédagogique de l'IUT de Bayonne et du Pays Basque pour m'avoir aidé et motivé tout le long de ma recherche de stage qui n'a pas été facile.

Enfin, comme l'année précédente, je tenais à remercier à nouveau mes grands-parents qui m'ont hébergé pendant toute la durée de mon stage, et qui m'ont permis de me concentrer pleinement sur mon travail durant toutes ces semaines.

II. Contexte

1. Présentation personnelle

Pour commencer, je vais me présenter rapidement, car je pense que c'est important pour que vous puissiez mieux comprendre mon rapport de stage.

Je m'appelle Alexandre PASCAL, j'ai 21 ans et j'ai choisi le parcours A, pour cette dernière année de BUT Informatique. Une des choses que j'aime le plus dans l'informatique, c'est la diversité des métiers et des compétences que l'on peut acquérir. J'aime également le fait que l'on puisse travailler dans des domaines très différents, et que l'on puisse continuer à apprendre tout au long de sa vie.

J'ai choisi de faire des études en informatique par passion, mais également par rapport aux opportunités que ce secteur pourrait m'apporter. Comme vous le savez, l'informatique est un secteur qui est en constante évolution, et selon moi encore pour des décennies.

Concernant mes traits de caractère, je me considère comme patient et persévérant. J'aime apprendre de nouvelles choses, et je suis très curieux. J'aime également travailler en équipe, et je pense que c'est une des choses les plus importantes dans le monde du travail, mais il est vrai que durant ce stage, j'ai su apprécier le confort et la tranquillité d'un travail individuel.

Cette année encore plus que les précédentes, j'ai beaucoup réfléchi à mon avenir professionnel. Après de nombreuses recherches, discussions et comparaisons, j'ai pris la décision de ne pas poursuivre les études en école d'ingénieur ou en master. Je sais que si j'avais voulu poursuivre mes études, j'aurais très probablement réussi, mais par rapport à mes envies et mes besoins, j'ai décidé de me lancer dans la vie active.

Dans la vie, ce que j'aime par-dessus tout c'est tester et apprendre de nouvelles choses. Pour moi, le plus grand problème est qu'il n'y a que 24 heures dans une journée dont 8 qu'il faudrait passer à « ne rien faire ».

Si j'avais plusieurs vies, j'aurais probablement eu un parcours très différent à chaque fois.

Je pense que c'est essentiel de se poser des questions sur soi-même, et de se remettre en question. C'est ce que j'ai fait cette année, et c'est ce qui me permettra d'avancer au mieux dans ma vie.

2. Présentation de l’hôpital

2.1 Organisation Globale

L’hôpital de Cahors est un établissement au cœur de la ville, situé dans le département du Lot, à une heure de Toulouse où se situe le CHU. Il est l’établissement de référence pour le département. Il est le plus gros employeur de la ville avec environ 1200 employés répartis parmi les différents services que compte l’hôpital, dont 10 dans le service informatique, et un parc de 750 ordinateurs.

2.2 Forme Juridique et organisation de la direction

Le Centre Hospitalier est un établissement public de santé, avec un statut de personne morale et de droit public. Il est doté d’une autonomie administrative et financière, relève de la juridiction administrative, et doit respecter les missions et les valeurs publiques. L’établissement est dirigé par un directeur, assisté d’une équipe de direction. Il a autorité sur l’ensemble du personnel. Il s’appuie sur des instances décisionnelles et consultatives et des commissions ou comités spécialisés.

2.3 Fonctionnement des services de soins et administratifs

Les services de l’hôpital sont regroupés sous différents pôles, que ce soit les services médicaux ou les services administratifs et techniques.

Les différents pôles des soins sont :

- Pôle viscéral et (Gastro-entérologie / Oncologie, Chirurgie vasculaire…)
- Pôle des spécialités (Cardiologie / Médecine du sport, Dermatologie…)
- Pôle (Neurologie, Rhumatologie…)
- Pôle Gériatrie (Diabétologie / Endocrinologie, Maladies métaboliques…)
- Pôle Mère (Gynécologie / Obstétrique, Pédiatrie / Néonatologie…)
- Pôle Urgence (Réanimation, Urgences / SAMU / SMUR / CESU)
- Pôle Bloc opératoire – Anesthésie (Bloc opératoire, O.R.L. …)

Les équipes transversales :

- Centre de cicatrisation
- Douleurs / Soins palliatifs
- Equipe Opérationnelle d’Hygiène (EOH)
- Recherche clinique
- Unité Mobile de Gériatrie (UMG)
- Unité de Tabacologie et de Liaison en Addictologie (UTLA)

Les unités Médico-techniques :

- Pôle Logistique médicale prestation de (Imagerie médicale…)

3. Présentation du service informatique

Le service informatique est composé de 10 personnes.

Il est dirigé par un responsable : Monsieur Vincent Goutines

Le service est composé de 3 équipes :

- Équipe de développement applicatif, gestion des applications de l'hôpital : 2 personnes

- Équipe de support technique, gestion des incidents et des demandes : 2 personnes

- Équipe de gestion des infrastructures, gestion des serveurs et des réseaux : 2 personnes

- Équipe transversale DPI (Dossier Patients Informatisés) : Une infirmière référente et une secrétaire référente

- Un RSSI (Responsable Sécurité Système d'Information) / DPO (Data Protection Officer)

La moyenne d'âge du service informatique est de 43 ans.

4. Présentation de mon tuteur de stage

Comme je l'ai dit plus tôt, mon tuteur de stage s'appelle Driss Makhout, a 48 ans, et a un poste d'ingénieur système et réseau au sein de l'établissement.

Pour en arriver là, il a obtenu une licence en Informatique Optimisation des Processus Industriels, double reconnaissance : métallurgie et national. Puis, en même temps, il a validé plusieurs modules du CNAM (Conservatoire National des Arts et Métiers). Il s'est réorienté ensuite sur de l'informatique plus généraliste au sein de l'entreprise Allium, devenue SCC France (plusieurs formations certifiantes systèmes et réseaux).

Lors d'une mission avec SCC au Centre Hospitalier de Villeneuve-sur-Lot, il a obtenu une proposition d'embauche au sein de l'hôpital, et il l'a acceptée. Un des principaux avantages a été la sédentarité du poste. Il a travaillé un an dans un data center, avant de continuer son parcours jusqu'à l'hôpital de Cahors, à domicile. Aujourd'hui, il est ingénieur systèmes et réseaux, et cela depuis 12 ans.

5. Présentation du contexte matériel

L'hôpital de Cahors a été créé en 1684. Il est situé au centre de la ville de Cahors, ce qui est un très bon emplacement stratégique, mais pose aujourd'hui des problèmes de places.

En effet, l'hôpital ne cesse de se développer, et donc le nombre d'employés dans chaque service aussi. Cependant, l'hôpital ne peut pas s'agrandir, car il est entouré de bâtiments et de routes.

C'est à cause de cela que le service informatique ne peut pas être regroupé dans un seul endroit spécifique. Même s'il y a une salle où l'on peut retrouver la majeure partie du personnel informatique, trois personnes sont dans un bâtiment différent et moi également.

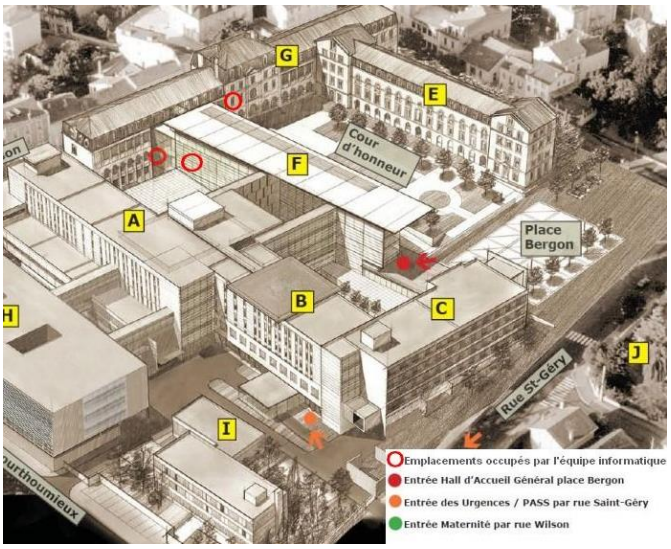


Figure 1 : Plan de l'hôpital

Nous ne sommes pas vraiment très loin, mais ce n'est pas pratique pour nous de devoir nous déplacer, changer de bâtiment, pour aller voir un collègue. C'est pour cela que le système de messagerie instantanée est très important pour nous. Cela s'applique également à l'ensemble de l'hôpital, du fait de sa taille. C'est-à-dire que s'il y a un message à faire passer, cela se fera la plupart du temps par mail ou par téléphone plutôt qu'en physique.

Me concernant, j'ai commencé le stage dans une grande salle informatique du bâtiment F qui est une salle généralement utilisée lorsque des personnes externes viennent à l'hôpital.



Figure 2 : Grande salle informatique

Par la suite, j'ai été déplacé dans un bureau du bâtiment G, qui était plus petit, plus éloigné, mais plus calme et plus pratique pour moi. Ce bureau était certes petit, mais sachant que je travaillais seul sur un projet, je préférais être seul dans un bureau et avoir à me déplacer, plutôt que dans une salle informatique avec des personnes qui viennent et qui partent à tout moment de la journée.

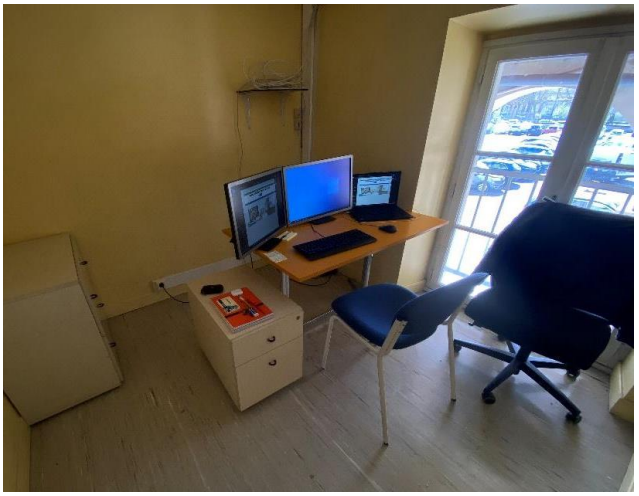


Figure 3 : Mon premier bureau

Le lendemain, Driss, mon tuteur, m'a proposé de me déplacer dans un autre bureau, qui était plus grand, plus lumineux, juste à côté de mon ancien. Celui-ci devait être occupé, mais sans nouvelles d'un potentiel occupant, il m'a proposé de m'y installer. J'ai donc accepté, et nous avons transféré les affaires dans ce nouveau bureau. C'était l'ancien bureau du chef du service informatique donc il était vraiment très grand.



Figure 4 : Mon nouveau grand bureau

J'ai donc pu être équipé d'une station d'accueil pour pouvoir brancher l'ordinateur portable qui m'a été attribué, deux écrans supplémentaires, un clavier et une souris. J'ai pu également demander à avoir un cahier, un bloc note et un tableau blanc pour pouvoir représenter et réfléchir à mes idées.

C'est donc dans ce bureau que j'ai pu travailler dans les meilleures conditions possibles.

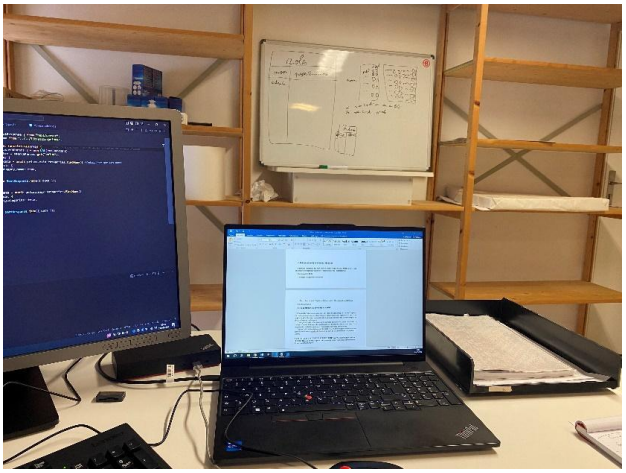


Figure 5 : Utilisation d'un tableau blanc

6. Présentation du sujet et du commanditaire

La mission qui m'a été confiée était donc la refonte de l'intranet de l'hôpital de Cahors.

Pour réaliser cette tâche, j'ai donc dû réaliser plusieurs étapes telles que : l'étude de l'existant, l'analyse du marché, le choix des outils, la création de maquettes, le développement, les tests, le déploiement et la formation des utilisateurs.

Pour commencer, il est important de comprendre ce qu'est un intranet. Il s'agit d'un réseau informatique interne à une organisation, qui permet de partager des informations, des ressources et des services entre les membres de cette organisation.

Donc effectivement, par opposition à un site internet, un intranet est un réseau privé, qui n'est accessible qu'aux membres de l'organisation. Cela permet de protéger les données sensibles et de limiter l'accès à certaines informations.

Pour remettre en contexte, l'actuel intranet de l'hôpital de Cahors a été créé en 2009. Il a été fait avec un CMS nommé Joomla, qui est un outil pour permettre la création de sites sans avoir de connaissances en développement (il est comparable à d'autres comme WordPress ou Drupal).

Depuis sa création, l'intranet de l'hôpital de Cahors n'avait pas été mis à jour en termes de design et de fonctionnalités. De nombreux liens étaient morts (voir glossaire), des informations étaient obsolètes. Il était donc nécessaire de le refondre complètement pour le rendre plus performant et plus adapté aux besoins des utilisateurs.

III. Analyse préalable

1. Redéfinition détaillée du sujet et des consignes

Comme je l'ai dit précédemment, il fallait reprendre entièrement le site afin d'avoir des informations à jour mais aussi de le rendre plus attractif et moderne.

Pour faire cela j'avais plusieurs solutions possibles. J'aurais pu utiliser un CMS comme WordPress afin d'avoir une base fiable et seulement coder tous les modules nécessaires. Mais après réflexion, j'ai considéré, que ce soit par rapport au cadre du stage, ou bien pour mon expérience personnelle, que le fait de créer ce site « à partir de rien » était la meilleure des solutions.

Effectivement, cette possibilité m'a bien évidemment permis d'améliorer mes compétences en tant que développeur, mais la vrai plus-value c'est que j'ai pu réaliser ce projet de A à Z c'est-à-dire, de l'idée jusqu'au déploiement

Je pense que je n'aurais pas pu rêver mieux comme situation car j'ai pour ambition de travailler l'année prochaine en tant que développeur full stack, donc le fait d'avoir touché à tout en terme de développement, de gestion de données et de déploiement, me permettra, pour le futur, de mieux appréhender les problématiques que je pourrais rencontrer.

J'ai pu ainsi constater que ce projet était très enrichissant pour moi, mais j'ai aussi compris que j'allais être confronté très rapidement à des problèmes.

Ce projet est, selon moi, très important, ambitieux et peut réellement apporter du positif pour l'hôpital. Mais comme vous l'aurez surement compris, mon plus gros problème va être le temps, sachant que je n'ai eu que 14 semaines de stage. Le délai était donc très court.

Vous découvrirez ma réalisation dans la partie résultat, mais je pense que ce projet aurait pu être bien plus perfectionné et complet avec plus de temps. Je n'en reste pas moins fier, et je pense avoir réalisé un bon travail pour ce sujet qui n'était vraiment pas simple.

2. Public ciblé

Pour continuer les rappels sur les principes d'un intranet de manière générale, et plus précisément de celui de l'hôpital de Cahors, l'utilité est de pouvoir permettre à tous les employés de consulter différentes informations liées à l'hôpital. On peut notamment penser à l'annuaire, qui est très souvent utilisé pour pouvoir trouver le numéro de téléphone d'une certaine personne, d'un certain service. Le site peut également permettre de diffuser des informations actuelles comme pour annoncer un évènement.

Donc dans le cadre de l'utilisation de l'intranet de l'hôpital de Cahors, il est important de pouvoir consulter les informations de l'hôpital, mais il faut également que certaines personnes puissent les modifier sans avoir à modifier du code. En termes de types d'utilisateurs, on retrouve donc les visiteurs qui peuvent consulter les informations, les éditeurs, qui peuvent modifier certaines pages et les administrateurs qui peuvent modifier ce qu'ils souhaitent, comme les rôles ou les pages.

IV. Méthode de travail

1. La démarche

1.1 Organisation et gestion de projet

Pour atteindre mon objectif qui était de développer l'intranet de l'hôpital de Cahors, j'ai dû mettre en place une organisation et une gestion de projet. Pour cela, j'ai donc utilisé l'outil Monday, qui m'a permis de pouvoir lister, trier, et suivre l'avancement de mes tâches.

Après l'avoir choisi, j'ai commencé à créer mon espace de travail qui allait contenir mon tableau de bord. Cet espace m'a permis de recevoir des notifications tous les matins, afin de me signaler où j'en étais par rapport à mes tâches en cours, combien j'en avais réalisé, combien il m'en restaient à faire, et donc tout ce qui pourrait me rappeler quotidiennement mon avancement.

Dans la foulée, j'ai choisi les styles de vue que je voulais avoir pour lister mes tâches : tableau, Gantt et Kanban. Ces trois styles de vue me permettaient de voir mes tâches sous différents angles, et donc de pouvoir les suivre de manière plus précise. Ce sont mes préférences personnelles, mais j'aimais les trois vues pour différentes choses : tableau pour avoir une vue d'ensemble sur toutes les tâches, Gantt pour visualiser au mieux les tâches réalisées par rapport aux estimations de départ et Kanban pour me rappeler les tâches en cours, restantes et terminées.



Figure 6 : Gantt Monday

Ensuite, j'ai commencé à estimer de manière large les différentes "catégories" de tâches telles que l'analyse, le maquettage, le développement, le déploiement, la phase de test par les utilisateurs et de corrections des erreurs, les problèmes, les demandes de la part des utilisateurs.

Cela m'a permis d'avoir une vision globale de mon projet, et donc de pouvoir le découper en plusieurs grandes étapes

Par la suite, j'ai commencé à découper ces grandes étapes en tâches. Donc pour la partie analyse, j'ai pu discerner et prioriser plusieurs tâches telles que l'étude de l'existant, la veille concurrentielle, l'analyse des besoins, des données, des contraintes, des forces et des faiblesses. Cette liste n'est pas exhaustive, mais elle regroupe les tâches principales que j'ai dû réaliser pour la partie analyse.

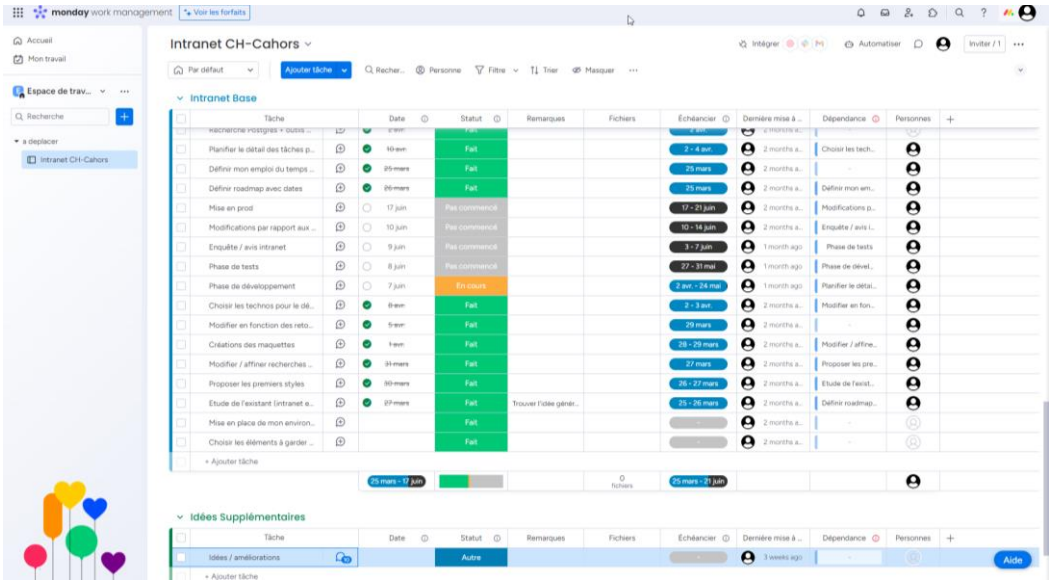


Figure 7 : Tableau Monday

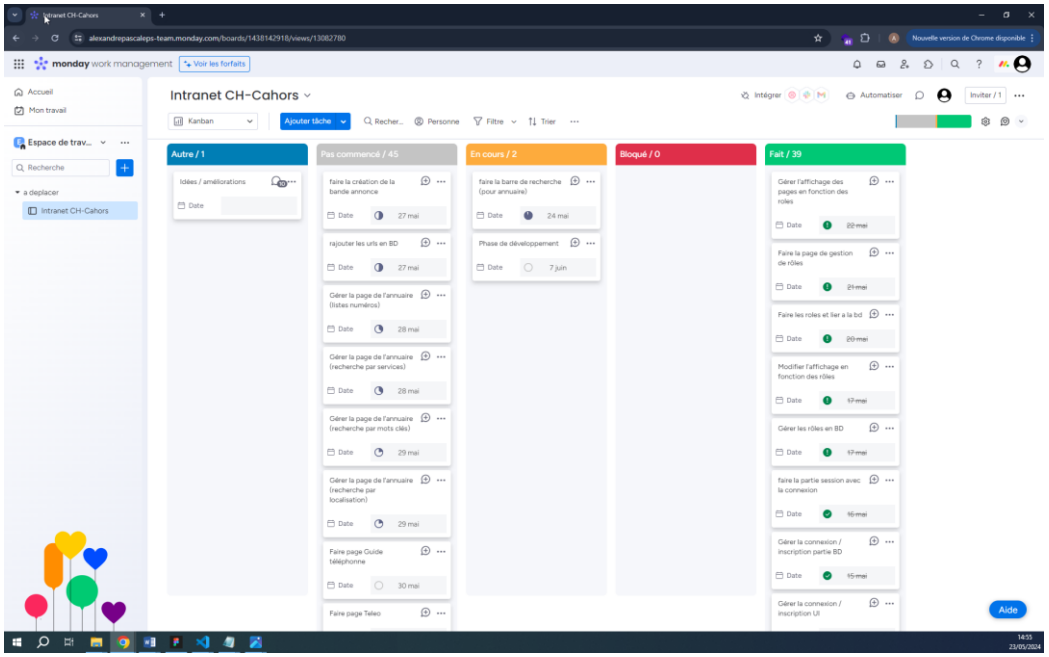


Figure 8 : Kanban Monday

Et J'ai réalisé le même processus pour la phase de maquettage en créant des tâches telles que la création des wireframes, des maquettes, des prototypes et bien d'autres.

Pour le développement, j'ai listé et priorisé toutes les fonctionnalités qu'il fallait retrouver dans l'intranet comme par exemple, le fait de pouvoir se connecter / s'inscrire, gérer des rôles, pouvoir créer des nouvelles pages, ajouter du contenu à ces pages très simplement tel que des fichiers, des tableaux, des images, des vidéos etc. Toutes les fonctionnalités que j'ai et avais prévu de réaliser seront détaillées dans la partie résultats.

En plus de développer les fonctionnalités, il me fallait également recréer toutes les pages déjà existantes dans l'intranet afin de ne perdre aucune information importante anciennement présente dans l'intranet.

Concernant le déploiement, j'étais beaucoup moins en capacité d'estimer et de découper les tâches, car je n'avais jamais déployé un site en production sur un serveur. J'ai dû me renseigner sur les différentes étapes à suivre pour déployer un site en production, et c'est pourquoi je n'ai pu lister et prioriser que les tâches principales telles que la mise en place de l'hébergement, la configuration du serveur et la mise en place de la base de données. Par conséquent, j'ai pris soin de laisser un temps suffisamment long pour cette phase pour le cas où cela serait bien plus difficile que prévu.

Pour finir, j'ai listé et priorisé les tâches pour la phase de test et de correction des erreurs, problèmes, demandes des utilisateurs. J'ai alors prévu de laisser une semaine, afin que les utilisateurs puissent tester l'intranet et me remonter les problèmes, erreurs, demandes qu'ils pourraient rencontrer. J'ai donc prévu de corriger ces éléments dans la semaine qui suivra.

Voilà dans les grandes lignes, la manière dont j'ai organisé et géré mon projet grâce à l'outil Monday.

Pour ce qui est des réunions ou de la présentation de l'avancement du projet, je montrais environ chaque semaine à mon tuteur l'avancement de mon projet, les tâches réalisées, celles en cours et celles restantes.

1.2 Découverte et choix de l’environnement

Pour ce projet, j'ai eu la chance de pouvoir être libre du choix des technologies. Comme j'ai pu le dire plus tôt, j'ai décidé que pour ce stage, j'allais reprendre à zéro l'intranet et le coder entièrement.

J'ai donc commencé à réfléchir quels outils j'allais utiliser pour ce projet et à lister les technologies que je connaissais et que je maitrisais. Après avoir regardé des retours sur différents sites pour chaque technologies, j’ai fini par faire ces choix :

Pour la création de maquettes, j’ai décidé de rester fidèle à Figma qui est un outil auquel je ne trouve aucun défaut et que j’ai l’habitude d’utiliser pour tous les projets.

Pour le développement j'ai décidé d'utiliser React, la bibliothèque Java Script, que je connaissais déjà un petit peu car nous avons utilisé le Framework React Native lors du développement de notre application mobile Valpineta. J'ai commencé à chercher s’il existait des Framework pour React, permettant de faciliter la création de ce site, en me fournissant donc tous les outils nécessaires au bon développement, en une seule installation. J'ai fini par découvrir l'existence du Framework NextJS, qui est donc un Framework utilisant la bibliothèque React, open source, et créé par Vercel (j'y reviendrai plus tard). NextJS est donc un Framework que l'on pourrait définir comme full stack, car il gère le front-end, avec de nombreuses choses comme le rendu côté client permis par React, afin de fluidifier les pages, mais il gère également le back-end, avec un serveur Node.js, configuré afin de faciliter des tâches qui sont complexes ou qui peuvent prendre du temps, comme l'utilisation d'API routes.

Pour la base de données, j'avais prévu d'utiliser MySQL, car c'est un système de base de données relationnel que je connais bien, qui est fiable et qui était déjà utilisé sur l'ancien intranet. Cependant mon tuteur m'a convaincu de me renseigner sur PostGreSQL. Après de nombreuses recherches, j'ai pu constater qu’il était également très apprécié, soutenu par la communauté open source et bien documenté. En termes de performances, il est très proche de MySQL, mais il est décrit comme plus complet en termes de fonctionnalités proposées.

Pour faciliter la consultation et l'édition de ma base de données, j'ai choisi PGAdmin4 qui est l'outil le plus populaire pour gérer les bases de données PostGreSQL. Il est très efficace et facile à utiliser.

Pour la gestion de version, j'ai décidé d'utiliser Git, qui est un outil que je connais bien, et qui est très utilisé dans le monde du développement. J'ai donc créé un répertoire sur GitHub, qui est une plateforme de développement collaboratif, qui permet de stocker des projets facilement.

Par la suite, j'ai découvert Vercel, qui est une plateforme de déploiement continue, qui permet de déployer des applications, et qui est très simple d'utilisation. Par conséquent, j'ai décidé de l'utiliser pour ce projet, car il me permettait de déployer mon site en un clic, et de le mettre à jour automatiquement à chaque mise à jour sur le répertoire GitHub.

J'ai également découvert Prisma, qui est un ORM (Object-Relational Mapping) open source pour Node.js et TypeScript. Il permet de simplifier l'accès à la base de données, en générant automatiquement des requêtes SQL à partir de requêtes JavaScript. Il permet également de faciliter la gestion des relations entre les tables. J'ai donc décidé de l'utiliser pour ce projet, car il me permettait de gagner du temps, et de me concentrer sur le développement du site.

Pour me faciliter le test de mes routes API, j'ai utilisé l'outil Postman, permettant d'exécuter n'importe quels types de requêtes http et d'avoir une interface très facilement lisible.

Intégré à NextJS, ESLint m'a permis de définir des règles, qui permettent de formater le code automatiquement, ou de m'obliger à respecter des bonnes pratiques de développement. C'est-à-dire que si je ne les respecte pas, je suis notifié grâce à des erreurs ou avertissements. D'ailleurs, si ces règles ne sont pas respectées, il est impossible de mettre à jour la version déployée sur Vercel.

Pour ce qui est de la communication, nous n'utilisons pas d'outils particulier, nous communiquons par mail, par téléphone ou directement en face à face.

Pour terminer, j'ai également utilisé plusieurs fois l'outil utilisant l'Intelligence Artificielle, ChatGPT (version gratuite). Je l'utilise depuis un certain temps, il ne cesse de progresser, mais malgré cela, je constate que le métier de développeur à encore de belles années devant lui, car il y a encore beaucoup d'erreurs, mais surtout, un manque "d'imagination". Je dis cela car pour le moment, si on lui demande de réaliser un beau site, il mettra simplement quelques couleurs à des endroits pas adaptés. C'est pour cela que pour ma part, je l'utilise en complément à StackOverflow et Github, afin de résoudre des erreurs, et de recroiser les informations. En plus de cela, je le trouve utile et efficace, dans le cas où, par exemple, on souhaite créer une fonction qui génère un nombre aléatoire, ou bien si par exemple on doit remplir une liste avec des prénoms. Tout cela pour dire que, ChatGPT est un outil très pratique, mais ce n'est pas (du moins, pas encore) possible de créer une application ou un site, qui soit complexe et "beau", en écrivant une phrase dans un outil qui utilise de l'intelligence artificielle comme ChatGPT.

Après avoir bien défini toutes les technologies et outils que j'allais pouvoir utiliser, j'ai pu enfin commencer à travailler sur le projet, en débutant par la phase d'analyse.

2. Analyse

Lors de ce stage, la chose la plus intéressante a été le fait de créer en partant seulement d'une idée, un vrai projet fonctionnel. Et donc pour réaliser cela, j'ai commencé par de l'analyse. L'analyse étant un terme très large, j'ai pu découper cette partie en plusieurs étapes comme dit plus tôt. Au préalable, j'ai commencé par l'étude de l'existant (voir glossaire).

Pour ce projet, l'étude de l'existant a été une partie relativement ennuyante mais fondamentale. La partie la plus longue a consisté à lister toutes les fonctionnalités, pages, liens présents dans cet intranet, et cela ne transparaît pas au premier coup d'œil mais il y en a vraiment beaucoup.

Le but de ce travail était de pouvoir définir les éléments que l'on veut retrouver, ou bien supprimer : ils sont toujours sur le site, mais plus fonctionnels. Il fallait également faire attention aux doublons, qui nuisent à l'organisation du site, et bien évidemment, supprimer celles que l'on ne veut simplement pas retrouver dans le nouvel intranet.

Pour réaliser ce travail j'ai donc accédé à un ordinateur connecté au réseau de l'hôpital, afin de pouvoir aller sur l'intranet. J'avais sur un écran, l'intranet, et sur l'autre, un document Word où je listais tous les éléments fonctionnels, non fonctionnels, douteux (dans le cas où je n'étais pas sûr de leurs bons fonctionnements). Je prenais le temps de bien les colorer afin de pouvoir aisément les distinguer. Ces informations étaient ainsi listées, avec des titres correspondant aux différentes sections du site.

Une fois tout listé, j'ai présenté la liste de toutes les informations à mon tuteur, qui a pu confirmer tout ce qu'il ne souhaitait plus voir sur le nouvel intranet. J'ai donc adapté mon document au fur et à mesure qu'il me donnait des informations supplémentaires à supprimer.

Ce document m'a pris plus d'une journée, du moment où j'ai commencé à le rédiger, jusqu'à la fin de la vérification de mon tuteur.

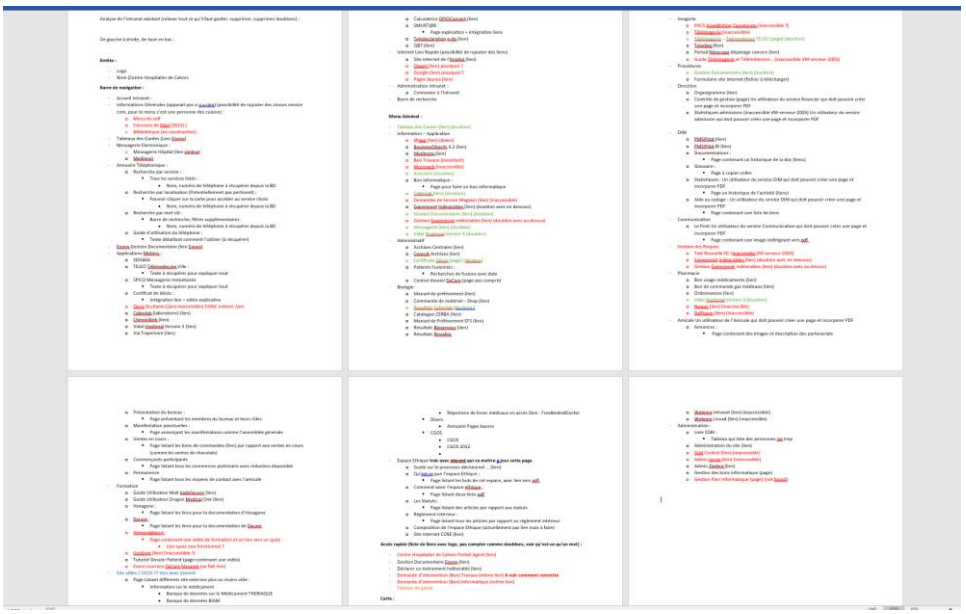


Figure 9 : Liste des éléments de l'ancien intranet

Même si vous ne voyez probablement pas les détails, vous pouvez voir comment étaient structurées les informations, et donc au final cela a occupé presque 6 pages complètes.

A côté de cette liste, lors de la discussion avec mon tuteur, j'ai rajouté une petite liste des fonctionnalités que l'on souhaite retrouver sur le site, car elles étaient des points forts dans l'ancien intranet. On peut notamment parler de la barre de recherche, ou bien le fait d'avoir dès l'accueil, des boutons permettant d'accéder aux éléments les plus consultés par le personnel.

Une fois que j'étais débarrassé de cette première tâche, j'ai pu passer à une partie que je considère bien plus intéressante, qui s'appelle la veille concurrentielle (voir glossaire).

Dans mon cas, le but était donc de voir ce qui se faisait en termes d'intranet, dans le domaine de la santé, et si possible des hôpitaux. Mais effectivement, dès le début, j'ai été confronté à un petit problème… Pour rappel, le principe d'un intranet étant d'avoir un site accessible depuis un navigateur, seulement par les personnes connectées au réseau de l'établissement, il m'était donc impossible de voir ce qui pouvait se faire par exemple dans les hôpitaux de Toulouse ou de Paris par exemple.

Par conséquent, j'ai décidé que, pour essayer de me rapprocher un maximum de ce qui pouvait se faire, je chercherai sur internet, des exemples d'intranet typiques d'entreprises. Lors de ma recherche, j'ai également pu trouver sur des sites, des listes de templates (voir glossaire), qui pourraient être pertinentes pour un intranet. J' ai donc sélectionné trois sites ou templates, qui pouvaient être intéressants, autant dans la manière dont sont disposées les informations, que dans le style global du site.

Après cela, j'ai regardé tout ce qui pouvait exister comme site internet d'hôpital, mais j'ai pu constater qu'une grande majorité de sites internet étaient peu ergonomiques, pour certains très vétustes. J'en ai donc retenu qu'un seul, celui de l'APHP (Assistance Publique - Hopitaux de Paris). Suite à cette découverte, j'ai décidé de ne pas seulement me cantonner à des sites internet comme ceux des hôpitaux, mais aussi à des sites que les gens pourraient percevoir comme modernes et faciles à utiliser. Ceux que j'ai retenus étaient donc Apple, Uber et Airbnb.

Je me suis ainsi retrouvé avec sept idées de sites que j'allais devoir présenter à mon tuteur. Je me suis rappelé qu'il m'avait donné pour consigne de lui en présenter trois. J'ai à nouveau fait le tri dans les derniers sites et j'ai fini par garder : le site de l'APHP, Airbnb et parmi les templates proposés par Microsoft SharePoint (voir glossaire), j'ai choisi celui qui a le plus retenu mon attention. J'ai fait des captures d'écran que j'ai collées sur Figma, afin de pouvoir présenter facilement mes trouvailles à mon tuteur.

3. Maquettage

Je me suis fais la réflexion que pour pouvoir se projeter peut-être encore plus facilement, cela serait une bonne idée si je réalisais un wireframe (voir glossaire) pour chaque site afin qu'il voit en premier lieu, la disposition globale des informations, puis à quoi un site réel pourrait ressembler dans ces conditions.

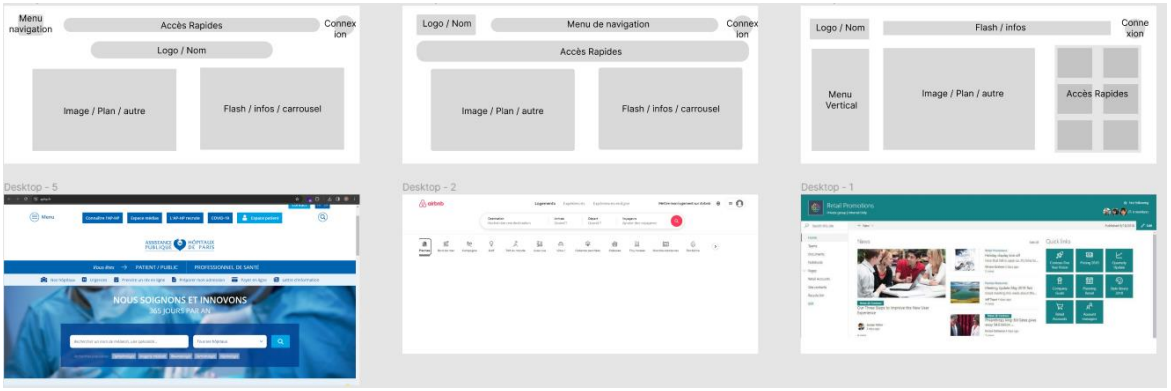


Figure 10 : Premiers wireframes

Lors de ma présentation, je lui ai bien expliqué que les critères qu'il devait étudier étaient seulement les fonctionnalités présentes à l'écran et l'allure générale, et non pas les couleurs utilisées qui ne correspondaient pas à celles de l'hôpital.

Relativement rapidement, il m'a affirmé que celui qu'il aimait le plus était celui de l'APHP (le premier à gauche), qu'il lui plaisait vraiment, presque à tous les niveaux. Il a même été surpris que l'APHP ait un site si « moderne ». J'étais donc assez satisfait parce qu'il me semblait également que c'était le plus intéressant des trois.

Après cela, il m'a fait part d'une idée qu'il avait eu pour disposer l'information dans le site. L'idée était donc d'afficher, dans le centre de la page, des bulles cliquables, qui pourraient se déployer et donc afficher plus d'informations et de détails pour une catégorie d'informations. J'aimais beaucoup l'idée car je trouvais cela original, simple d'utilisation et assez sympathique pour l'expérience utilisateur.

J'ai donc conservé la partie haute du wireframe du site internet de l'APHP, et j'ai commencé à faire pleins de tests sur la manière de diffuser l'information dans cette forme-là. J'ai finalement présenté un résultat avec beaucoup d'animations afin de pouvoir se projeter un maximum. Malgré un certain temps passé à réaliser ces wireframe, je n'étais pas tellement convaincu du résultat, car je gardais à l'idée le fait que tout devait être modulable, mais que dans cette disposition, l'information pourrait devenir rapidement illisible.

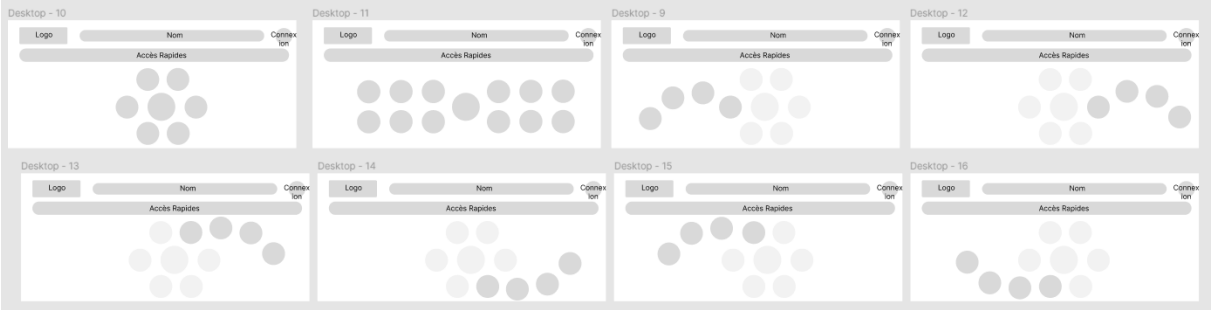


Figure 12 : Test 2 Représentation des informations dans des bulles

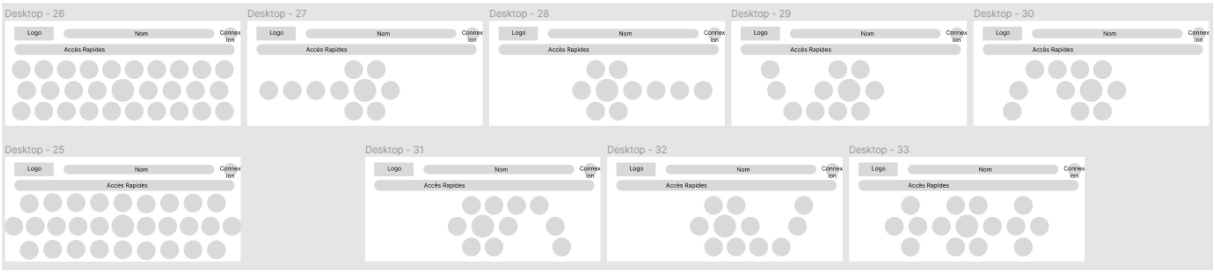


Figure 11 : Test 1 Représentation des informations dans des bulles

Suite à cette présentation, il a rapidement été d'accord avec moi : cela ne serait pas assez clair et de manière générale, trop complexe pour un rendu pas si intéressant au final.

Après cela, je lui ai proposé différents styles de menus qui apparaîtraient quand on clique sur le bouton en haut à gauche. Parmi eux, j'ai repris celui proposé par l'APHP, mais j'ai également proposé celui d' Apple et celui de Samsung qui me paraissaient intéressants, car ils sont tous les deux épurés et sont organisés différemment en termes de catégories, sous catégories etc…

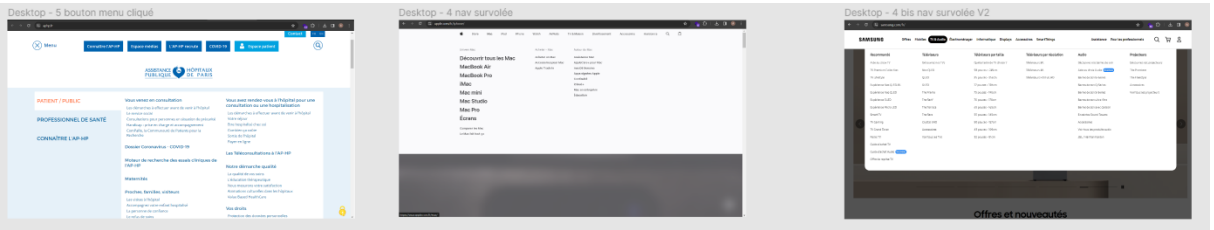


Figure 13 : Proposition de styles de menus

Il a choisi de conserver le premier style de menu, celui de l'APHP, mais cette fois-ci, je n'avais pas vraiment d'avis sur la question, la disposition des trois me paraissait intéressante.

Ensuite je n'avais plus qu'à réaliser les maquettes pour visualiser correctement à quoi pourrait ressembler le site. J'ai réalisé les maquettes les plus importantes (pas toutes car il y avait beaucoup de pages qui se ressemblaient). Bien

évidemment, j'en ai animé un maximum pour que mon tuteur puisse tester comment réagirait le site.

J'ai donc commencé par réaliser la première partie de ma maquette correspondant à la partie haute et j'étais plutôt satisfait du résultat.



Figure 14 : Partie haute de la maquête finale

J'ai continué en choisissant une image de fond libre de droits pour habiller la partie information. Je n'arrivais pas trop à me décider donc j'ai préféré les proposer toutes à mon tuteur. J'ai disposé une image, qui n'est pas très importante, car elle ne sera pas là au final, simplement pour imaginer comment rendrait du contenu sur ce fond.



Figure 15 : Proposition d'arrières plans

Je lui ai présenté la page d'accueil. Elle lui plaisait mais par rapport aux images de fond, il ne savait pas trop, mais avait une petite préférence pour la première à gauche. Ce serait donc celle-là que j'utiliserai pour les prochaines maquettes, mais j'ai bien conservé les deux autres également.

J'ai continué à améliorer le site puis je suis passé au menu. J'ai repris le principe de l'organisation des informations du menu choisi par mon tuteur, et je me suis retrouvé avec trois types d'information : La première étant ce que j'appellerai une "catégorie principale", qui est donc affichée avec du texte grand et en majuscule. Lorsque cette catégorie principale serait choisie lors du clic sur le nom, elle

afficherait donc à droite en gras ce que je nommerai des “sous-catégories” qui pourraient ou non, contenir du texte en plus petit et non en gras, que j’appellerai « titre ».

J’ai par conséquent repris toutes les informations que nous avions listées dans la partie étude de l’existant, et j’ai commencé à organiser les informations dans le menu.

Pour rappel, je réalise un site permettant de modifier / ajouter des informations facilement, mais le but n’est pas de devoir forcer les employés à remettre les informations qui étaient déjà présentes sur l’ancien intranet. Je dois donc créer de base le menu contenant les informations choisies.

Une fois avoir réalisé cette tâche, qu’il faut dire, relativement pénible et longue, j’ai présenté le résultat à mon tuteur, toujours en permettant de cliquer sur les différentes catégories principales, affichant des sous catégories et titres différents à chaque fois, avec des informations de l’ancien intranet.



Figure 16 : Création du menu

J’ai ainsi fini mon travail de maquettage avec quelques autres maquettes comme le formulaire de connexion, la barre de recherche ou une page typique que l’on pourrait retrouver sur le site.

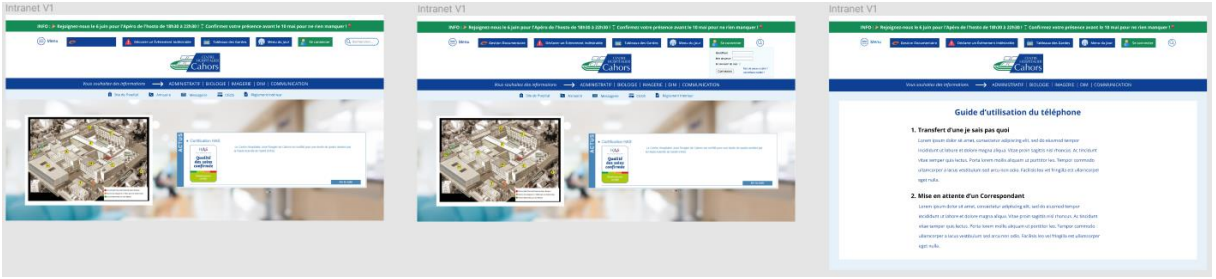


Figure 17 : Détails pour la maquette finale

Finalement, après toutes ces présentations, il est revenu sur l’ensemble du site et m’a confirmé que cela lui convenait, et que je pouvais passer à la suite.

Ce travail de maquettage a été très important afin d'être sûr et certain de ce qu'il attendait de moi. Même si par la suite, des petites choses ont été modifiées, l'allure générale du site a été conservée.

Je voulais rajouter une petite chose à propos de ces maquettes. Avant, lorsque je réalisais des maquettes, je n'animais pas vraiment les composants. Je me suis rendu compte ici que, par exemple, lorsque l'on clique sur le bouton menu, on peut vraiment visualiser comment le futur utilisateur pourrait vivre cette expérience. Cela permet de se projeter vraiment plus facilement, mais effectivement, il faut prendre le temps de le faire. Malgré l'investissement nécessaire, je pense aujourd'hui que cela en vaut la peine.

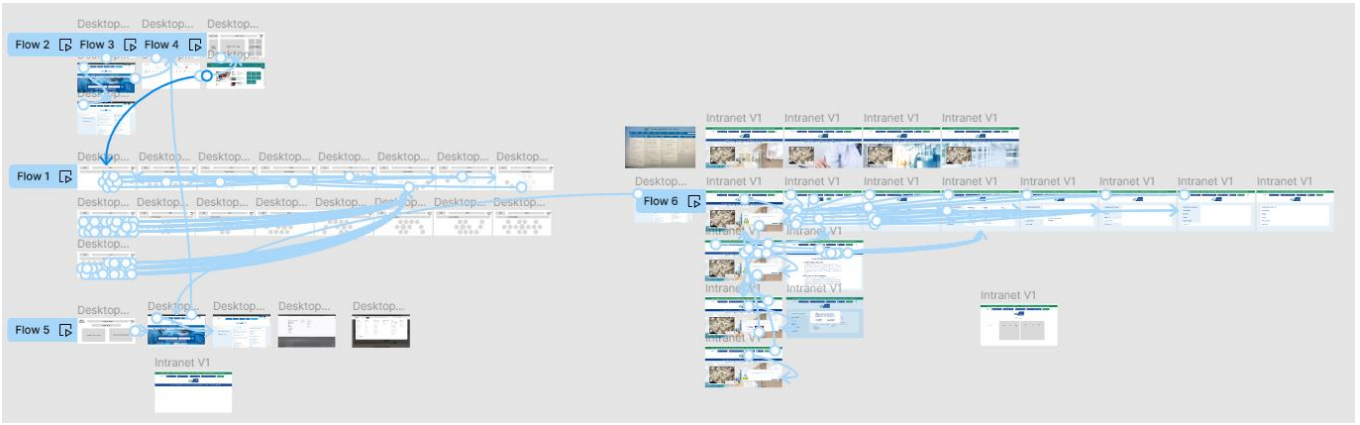


Figure 18 : Représentation des animations Figma

Chaque trait bleu ciel que vous pouvez voir, correspond à une animation permettant de passer d'une maquette à une autre en cliquant sur un bouton, élément, ou quoi que ce soit. Cela permet donc de visualiser le rendu final mais également de fluidifier la présentation ce qui n'est pas non plus négligeable.

Après l'étape maquette, j'ai commencé à développer le site avec NextJS, le Framework choisi, présenté plus tôt dans ce rapport.

4. Développement

4.1 En-tête

Juste avant de commencer à développer, je me suis posé plusieurs questions. Parmi elles : est-ce possible de récupérer les pages et les données de manière générale de l'ancien site ?

Pour répondre à cette question, j'ai donc exploré la base de données de l'ancien intranet, qui avait été réalisée avec le CMS Joomla. Les données étaient stockées dans une base de données MySQL. J'ai rapidement extrait la base de données pour éviter de causer des problèmes qui pourraient arriver. Une fois avoir fait ça, j'ai dû donc la convertir en base de données PostGreSQL. Pour ce faire, j'ai essayé plusieurs outils. Mais au moment de tester, il y avait toujours des problèmes, par rapport à des mots clés qui n'existaient pas sous PostGreSQL. J'ai finalement dû les modifier à la main, et cela a fini par fonctionner. Cela a été possible grâce à l'outil PGAdmin4 qui m'a permis de visualiser, créer modifier des bases de données facilement. J'ai fini par voir à quoi ressemblait la base de données. Comme je me l'imaginais, la base de données contenait cent seize tables.

J'ai pu ainsi comprendre où était stocké le contenu des pages mais il me paraissait difficilement exploitable. J'ai préféré repartir d'une base de données vierge mais j'ai bien conservé l'ancienne base pour récupérer les données plus tard.

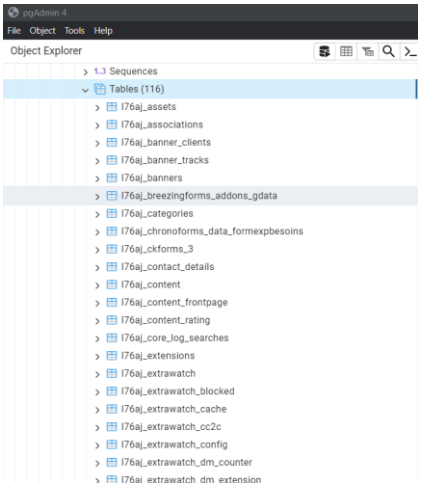


Figure 19 : Liste des tables de l'ancien intranet

Ma base de données étant prête, j'ai commencé à créer mon projet avec NextJS. L'installation est très simple, simplement une ligne de code suffit, puis une liste de choix nous est proposée, afin de générer une structure de fichier comme on le souhaite.

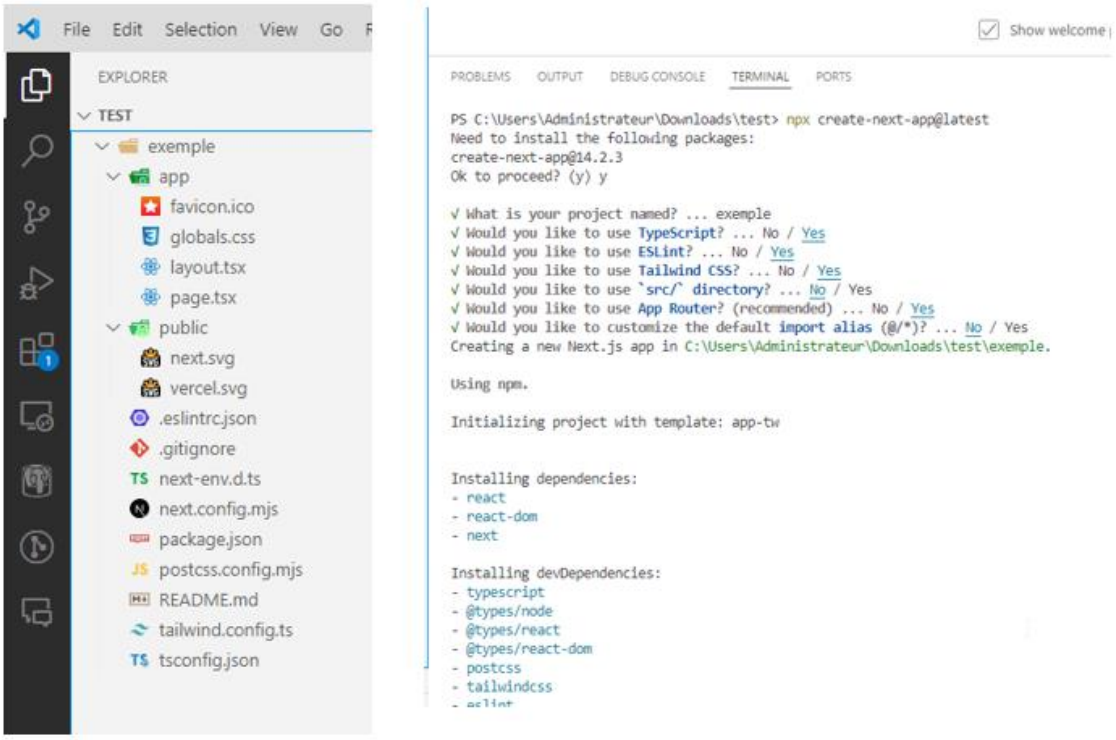


Figure 20 : Génération des fichiers NextJS

J'ai rapidement créé mon répertoire sur GitHub que j'ai laissé en privé pour éviter tous problèmes. Par la suite, j'ai paramétré tout ce qui m'intéressait, et donc parmi les choix, j'ai choisi de déployer mon projet sur Vercel. C'est-à-dire que la base de données serait donc hébergée sur Vercel, mais bien évidemment je pourrais la consulter / modifier depuis PGAdmin4 en m'y connectant avec des accès fournis. Ce que m'apporte le plus cette plateforme, c'est de pouvoir déployer mon application, et qu'elle soit donc accessible depuis internet le temps du développement. Cela me permettait de pouvoir montrer à mon tuteur l'avancée, sans avoir à démarrer le serveur NodeJS. Il me suffisait simplement d'accéder à mon répertoire GitHub, dans lequel il y avait une partie déploiement, avec le statut (échec / en cours / déployé). Grâce à cela, je pouvais consulter mon site en cliquant simplement sur un lien. Pour plus de sécurité, depuis Vercel, je pouvais configurer les adresses mails que j'autorisais à accéder à ce site.

Une fois que Vercel était paramétré, dès que je mettais à jour mon répertoire sur GitHub, l'application était construite et déployée, et j'avais un retour sur la réussite ou l'échec du déploiement.

J'ai donc pu enfin commencer à créer l'intranet, et j'ai débuté directement par la partie haute du site. J'ai assez rapidement recrée les différents boutons, bandeaux, logos affichés, et je me suis alors assez rapidement confronté au menu.

Après réflexion, l'idée qui m'a paru la plus intéressante pour pouvoir stocker des informations qui devront pouvoir être modifiables, était bien de tout stocker en base

de données. J'ai donc créé la première table de ma base de données. C'était simplement pour commencer à tester, donc je l'ai simplement appelé « test » avec deux attributs, un identifiant « id » de type nombre et un « nom » pour une chaine de caractère. J'ai peuplé rapidement à la main cette table juste pour pouvoir tester des petites opérations très simples.

Pour commencer à afficher ces informations, j'ai utilisé l'ORM Prisma que j'ai trouvé vraiment très bien fait, et simple d'utilisation. Il a pu me permettre d'éviter d'écrire des requêtes en SQL.

Pour des requêtes très simples la différence n'est pas énorme, mais dès que l'on passe à quelque chose de plus complexe, on comprend mieux l'avantage.

```
//Recupère les catégories principales de la base de données
const roles = await prisma.main_categories.findMany();

const [rows] = await connection.execute("SELECT * FROM main_categories");

//Recupère les catégories principales et chacune de ses sous-catégories
const data = await prisma.main_categories.findMany({
  include: {
    sub_categories: true,
  },
});

const [rows2] = await connection.execute(`
    SELECT main_categories.*, sub_categories.*
    FROM main_categories
    LEFT JOIN sub_categories ON main_categories.id = sub_categories.main_category_id
  `);
```

Figure 21 : Comparaison requêtes SQL vs Prisma

Grâce à cela, j'ai rapidement réussi à afficher mes quelques données.

A ce moment-là j'avais donc besoin de les afficher dans mon menu. Avant de faire cela, j'ai pensé qu'il fallait que l'entête (partie haute) soit affichée absolument sur chaque page. J'étais donc bien embêté car je n'avais pas envie de réécrire le code dans chaque fichier. Je me suis très rapidement rappelé que React permettait de créer des composants, qui sont donc des morceaux de code, qui peuvent être réutilisés très facilement juste en écrivant <NomDuComposant/>, et bien évidemment en important le fichier correspondant. Cela me permettrait donc d'éviter les duplications de code dans chacune de mes pages. Mais cela ne me plaisait toujours pas car j'aurais bien aimé utiliser ce composant à un seul endroit et qu'il soit automatiquement « ajouté » à toutes les pages. En recherchant sur la documentation de NextJS, j'ai pu comprendre l'utilité du fichier « layout.tsx » (voir glossaire) qui avait été généré automatiquement lors de la création du projet. Effectivement, ce fichier me permettait de faire exactement ce que je voulais. Pour faire simple, on peut dire que quand on arrive sur une page, ce n'est pas

directement la page qui est appelée, mais layout.tsx en premier, et lui, rajoute par la suite le contenu de la page.

J'ai donc ajouté mon composant entête dans layout.tsx et effectivement, ma page s'affichait effectivement toujours avec l'entête. Cependant au début, je n'étais pas parfaitement sûr d'avoir compris ce fichier layout, donc j'ai simplement créé une deuxième page, ajouté un bouton pour pouvoir passer de la première page à la deuxième, et j'ai pu voir que mon entête était bien présente également sur cette nouvelle page.



Figure 22 : Fichier layout.tsx

Une fois cela fait, j'ai pu revenir au problème initial, qui était donc d'afficher du contenu dans mon menu. Pour cela j'ai commencé par créer un nouveau composant menu que j'ai rajouté dans mon composant entête. J'ai rajouté quelques lignes de code pour pouvoir avoir d'affiché un fond bleu qui soit par-dessus les informations qui seront dans la partie basse du site. Après avoir codé cela, j'avais donc bien mon menu qui s'affichait par-dessus le reste de la page, mais je voulais aussi gérer l'ouverture lors du clic sur le bouton menu. Pour cela, j'ai donc utilisé un des outils les plus utiles proposés par React : les « hooks ». Parmi les hooks, il en existe un qui s'appelle UseState.

UseState permet d'ajouter une « variable d'état » à notre composant qui sera composé de deux valeurs qui seront liées : la première que l'on peut donc définir par défaut avec le type que l'on veut et à la valeur que l'on veut, et la deuxième qui permet de modifier la valeur de la première.

Par exemple, dans mon cas, j'ai donc eu à créer un useState qui contient [isActive, setIsActive], qui prendra comme valeur par défaut un état, représenté sous forme de booléen (voir glossaire), à false par défaut, pour définir le fait que le menu ne doit pas être affiché à l'ouverture de la page.

On a donc cela : const [isActive, setIsActive] = useState<boolean>(false);

D'après ce que je viens d'expliquer, cela pourrait paraître comme une simple variable et donc remplaçable par quelque chose ressemblant à cela : let isActive : boolean = false;

Qui serait donc modifié en faisant : isActive = true;

Sauf que la grosse différence repose sur le fait que comme je l'ai dit plus tôt, c'est une variable d'état, et donc cela va permettre que, lorsque l'on va mettre à jour isActive en utilisant setIsActive, cela va réafficher le composant, et donc potentiellement le modifier.

Donc au final, pour afficher mon menu, j'ai mis une condition, afin qu'il faille que isActive soit défini à True, si je veux qu'il s'affiche. J' ai dû également définir que dès que je clique sur le bouton menu cela inverse la valeur de isActive. Cela pour éviter de devoir dire : si isActive est sur True je passe à False et si isActive est sur False je passe à True.

Grâce à cela le menu s'est affiché correctement, c'est-à-dire qu'au début il n'est pas affiché, car isActive n'est pas à True, et donc après avoir cliqué sur le bouton du menu, la valeur de isActive va être passée à True, ce qui va entraîner le réaffichage de mon composant, qui va venir tester à nouveau ma condition sur l'affichage de mon menu, et qui va donc cette fois être concluant.

Je tiens à préciser que quand je dis que la page va être réaffichée, pour l'utilisateur, la page ne va pas se « recharger / rafraichir » c'est-à-dire qu'il n'y a pas de chargement pour l'utilisateur, cela se fait instantanément.

Je considère que cette fonctionnalité nécessitait une explication, car c'est une fonctionnalité très utilisée dans React de manière générale et donc avec NextJS aussi. Pour vous dire, à l'heure où je rédige ce rapport, j'ai pu l'utiliser plus de cinquante fois.

A ce moment-là, le menu s'affichait bien comme je le souhaitais, mais j'ai été confronté en suivant à un nouveau problème. Effectivement, plus tôt, j'affichais le contenu de ma table grâce à Prisma. Mais désormais, je voulais afficher mon texte dans ce composant. J'ai pris un certain temps à comprendre que, lorsque l'on va utiliser un hook comme useState par exemple, ce que l'on affiche est obligatoirement « côté client ».

Petite explication à ce niveau-là : on dit que l'on peut afficher du texte géré de deux manières différentes : côté client et côté serveur.

Pour faire simple, côté client c'est le navigateur de l'utilisateur, et côté serveur c'est donc le serveur / l'ordinateur qui héberge le site. Le serveur est uniquement capable d'afficher des pages comme on le lui indique dès le départ. Ces pages pourront être affichées pour tout utilisateur qui se connecte au site. Le navigateur (côté client), lui, permet de pouvoir faire des actions qui n'ont donc pas vraiment de lien avec le serveur c'est-à-dire que dans mon cas, si j'ouvre ou pas mon menu, si je rafraichis la page, ma page d'accueil reviendra comme à l'état initial avant ouverture du menu.

Pour essayer d'être plus clair, je vais donc expliquer ce qui se passerait si je voulais gérer l'affichage du menu, donc sans le hook UseState, car techniquement c'est possible.

Pour faire cela, il faudrait gérer l'état de l'affichage en base de données, et donc modifier cet état lorsque l'on clique sur le bouton. Pour le moment, on peut se dire,

pourquoi pas, mais le problème est que si on le fait, cela implique de recharger la page, donc d'avoir un chargement et quelque chose de pas naturel pour une ouverture de menu. Mais le plus gros problème, est que si l'on fait ça, tous les utilisateurs qui seraient connectés au site, qui n'auraient rien demandé, verraient le menu s'afficher devant leurs yeux après un changement de page ou un rechargement. On remarque donc, qu'en plus du fait d'exécuter une requête pour pas grand-chose, de ne pas être pratique, cela cause des problèmes.

Je pense que cela a pu vous aider à comprendre la différence entre exécuter une tâche côté client et une côté serveur. A présent, je vais aborder un autre fonctionnement / problème que l'on rencontre lorsque l'on utilise un composant / page côté client.

Donc, pour récapituler, je veux afficher des informations contenues en base de données, dans un menu dont l'affichage est géré côté client. Cette opération est impossible directement car le navigateur ne peut pas communiquer directement à la base de données. C'est le serveur qui communique avec la base de données. En conséquence, pour pallier ce problème, il va falloir utiliser le système d'API. Le schéma suivant permet de comprendre très facilement comment la communication se fait et de bien comprendre ce que l'on peut ou ne pas faire côté client.

Pour faire très simple sur le fonctionnement d'une API (Interface de Programmation d'Applications), comme le nom l'indique, cela sert d'interface entre le client et le serveur qui communiquent avec la base de données. C'est donc grâce à cela que n'importe quel composant affiché grâce au navigateur (côté client), va pouvoir récupérer des données d'une base de données.

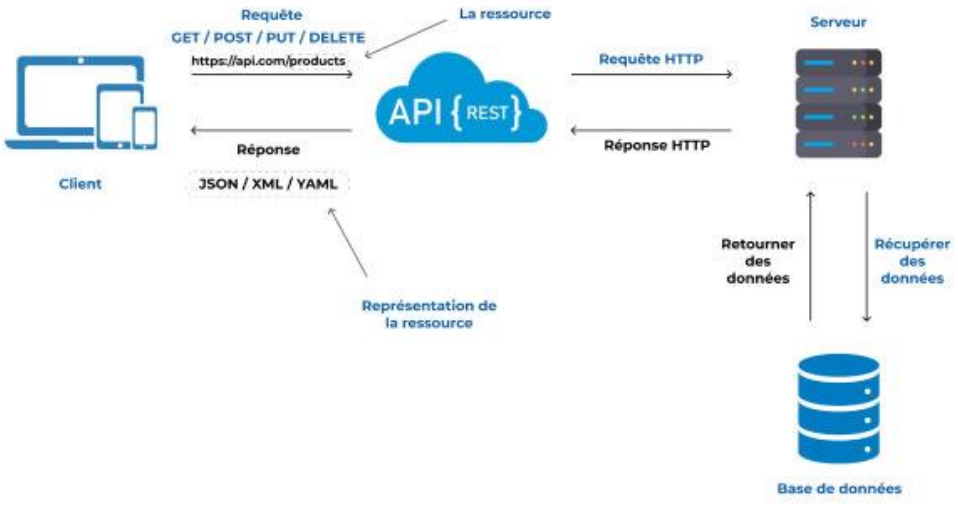


Figure 23 : Schéma explicatif API

Comprendre le fonctionnement des API m'a donc permis de réaliser qu'il fallait que je crée des routes permettant de pouvoir récupérer les informations de la base de données. Pour réaliser cela, je me suis à nouveau renseigné dans la documentation de NextJS, et j'ai pu voir que la création de route était automatisée. C'est-à-dire que je n'ai pas besoin de créer de route à proprement parlé, il faut

simplement créer dans le dossier « api », automatiquement généré lors de la création, un fichier « route.ts » ou « route.js ».

Dans mon cas, pour tester, j'ai pu simplement ajouter dans le fichier route.ts, une requête Prisma qui affichait tous les identifiants de la table « test », et retourner le résultat sous forme de JSON.

Grâce à cela, désormais, lorsque dans la barre de recherche de mon navigateur je rajoute « /api » à la suite de l'url de mon site, le résultat de ma requête au format JSON s'affiche dans une page.

D'ailleurs, cela ne fonctionne pas uniquement pour le résultat d'une requête, cela peut être n'importe quoi. C'est-à-dire que, si toujours dans ce même fichier route.ts, on indique de retourner « coucou », quand on accèdera toujours à cette même url, « url site + /api », on aura simplement le mot « coucou » d'affiché.

En plus de cette fonctionnalité, une autre permet de pouvoir bien organiser ses fichiers. Effectivement, on peut créer dans le dossier « /api », un dossier « /categories » (par exemple), contenant lui aussi un fichier route.ts. Cela nous permet donc d'avoir la route « /api/categories » qui est accessible de la même manière qu'avant.

Une autre des fonctionnalités très importantes que j'ai pu découvrir à ce moment-là, c'est la possibilité de récupérer une information passée dans l'url. Pour faire cela, on doit créer un dossier dont le nom est entre crochets. Donc par exemple, dans notre cas si on ne veut pas simplement récupérer toutes les catégories, mais bien une en particulier, on va créer, dans « /api/categories », un dossier « [id] », contenant toujours un fichier route.ts. Dans celui-ci, on va avoir la possibilité de récupérer la valeur qui sera passée dans la barre de recherche. Par exemple, si on saisit « /api/categories/4 », on va pouvoir récupérer ce « 4 » pour en faire ce que l'on veut comme dans une requête Prisma pour dire que l'on veut afficher la catégorie qui a pour id 4.

Il y a une autre façon, plus classique, pour récupérer une valeur. Par exemple, pour faire la même chose, on pourrait faire « /api/categorie?id=4 », et cette fois-ci on va avoir à récupérer ce 4 un peu différemment mais on pourra faire au final autant de choses.

Voilà pour les éléments les plus importants que j'ai pu découvrir par rapport aux API et aux routes gérés avec NextJS. J'espère que j'ai réussi à être assez clair, car ce n'est pas quelque chose de forcément très simple et évident à expliquer.

Pour revenir à mon objectif d'afficher les id de ma table test, j'ai créé un dossier « /essai » dans « /api ». J'ai créé un fichier route.ts, et j'ai codé quelques lignes pour récupérer avec Prisma tous les id de ma table. Je les ai retournés sous forme d'un JSON. Suite à cela dans mon menu, j'ai utilisé la fonction fetch, en lui donnant comme paramètres la route « /api/essai », afin de récupérer le contenu de la page retournée par mon fichier route.ts dans « /api/essai ». Après avoir fait cela, j'ai

affiché simplement tous les éléments. J'avais enfin bien les informations qui s'affichaient dans mon menu quand je cliquais sur le bouton.

Voici un exemple de requête que j'ai donc pu faire au début :

```
export async function GET() {
  const result = await prisma.datas_articles.findMany();
  return NextResponse.json({ result });
}
```

Figure 24 : Exemple requête API

Après avoir bien compris tout ce qu'il me fallait pour afficher des informations dans mon composant, j'ai réfléchi à comment je pouvais stocker toutes mes catégories. Donc j'ai repensé à mes maquettes et j'ai fait le point sur tout ce dont j'avais besoin. En termes de tables, j'en ai déduis qu'il en fallait trois : catégories_principales, sous_catégories et titres.

J'ai commencé alors à faire une liste pour chaque catégorie principale : J'avais besoin d'un nom, d'un nombre correspondant à l'emplacement dans la liste des catégories principales et d'un identifiant qui serait donc une clé primaire (voir glossaire), et serait défini à partir du nom, mais formalisé de manière à remplacer les espaces et apostrophe par un tiret, et à mettre tout en minuscule et sans accent.

Pour les sous catégories, il me fallait : le nom, l'ordre, l'identifiant comme pour les catégories principales. A cela, on ajouterait un URL, qui permettrait de rediriger vers une page interne ou externe, et pour finir, l'identifiant de la catégorie principale à laquelle serait rattachée la sous-catégorie, qui serait donc une clé étrangère (voir glossaire).

Pour les titres, on retrouve le nom, l'ordre, l'identifiant, l'url, et donc cette fois-ci, c'est l'id de la sous-catégorie correspondant au titre, qui sera une clé étrangère.

L'affichage est fait tel que toutes les catégories soient toujours affichées, que par défaut, la catégorie principale en première position soit sélectionnée. Les sous catégories correspondantes, ainsi que les titres correspondants à chaque sous-catégorie, seront affichés à côté.

Pour cette première partie, j'ai donc eu à construire l'architecture des dossiers du dossier « /api », ainsi qu'à veiller à ce que l'affichage soit correct : C'est-à-dire, réussir à bien afficher toutes les informations comme il le fallait, bien clairement, pas trop gros, pas trop petit, en occupant un maximum de place pour occuper l'espace, tout en s'adaptant au nombre de sous-catégories / titres pour ne pas dépasser.

A ce moment-là, j'étais assez content de mon travail, mais finalement, je n'avais pas encore commencé la partie la moins simple de mon projet. Effectivement le but est de pouvoir, en plus de consulter, modifier, créer, supprimer.

J'ai donc continué à créer toutes les types pour chaque type d'information. J'utilisais donc les méthodes POST PUT et DELETE. Celles-ci permettant respectivement, d'envoyer un formulaire donc d'ajouter, PUT pour mettre à jour et DELETE pour supprimer.

J'ai donc fait tout cela pour mes 3 tables, en utilisant correctement les fonctionnalités pour la gestion d'API. Au niveau de la structure des fichiers j'ai obtenu ce résultat :



Figure 25 : Architecture dossier API

Par la suite, j'ai ajouté des icônes pour pouvoir ajouter, modifier, supprimer des catégories principales, sous-catégories, titres. Pour ces fonctionnalités, j'ai fait une petite interface pour pouvoir gérer le nom, l'ordre, et l'url pour les sous-catégories et titres.

Ce travail a été très long et fastidieux. Malgré un maximum d'efforts pour faire le code le plus propre possible, le lendemain j'ai dû prendre la journée entière pour tout restructurer en plusieurs fichiers, car absolument tout ce que j'ai expliqué était réuni dans un fichier. Après modification, j'ai réussi à constituer 4 fichiers distincts :

Le principal, qui affiche tout ce qu'il faut dès le départ, et qui déclenche les premières fonctions lors de clics sur un icône.

Un deuxième, pour gérer l'interface qui doit s'afficher en fonction du type de la donnée et de l'action demandée. C'est lui qui va déclencher l'appel du troisième en lui transmettant les données qui ont été entrées (ou rien dans le cas de la suppression).

Ce troisième fichier permet de préparer toutes les requêtes API donc avec le type de méthode, l'url de la route correspondante à appeler, et potentiellement le contenu avant de le transmettre au dernier. Ce dernier permet donc d'exécuter les appels API avec les bonnes informations ainsi que d'autres petites choses.

Cette partie était donc assez complexe en raison du nombre d'actions possibles qui se ressemblent mais qui au final n'ont pas le même but. Au final pour résumer ce travail, je n'ai qu'un seul même composant qui est appelé pour afficher chaque interface, mais en fonction du type de données et d'action, des éléments différents sont affichés. Cela m'a permis de réduire de beaucoup de lignes la taille de mon

code, mais je pense que je pourrais encore réduire si j'avais le temps, en groupant mieux les données (les regroupant en un seul objet).

A ce moment-là, j'avais mon menu qui s'affichait correctement, et je pouvais le gérer facilement.

Mais la fin du projet est encore très loin, car mon menu ne sert pas à grand-chose, puisque l'on ne peut rediriger les sous-catégories et titres que vers des liens qui sont externes à l'intranet (google.com, youtube.com, etc.)

4.2 Création de pages

4.2.1 Découverte d'éditeurs

C'est à partir de là que je débute la partie la plus importante du projet, qui est donc la possibilité de créer, modifier, supprimer des pages qui seront accessibles depuis le menu.

Pour réaliser ce travail, je ne savais pas réellement comment m'y prendre, alors j'ai commencé à chercher ce qui pouvait se faire.

Pour pouvoir créer des pages, j'ai commencé par chercher comment est-ce que je pouvais créer un éditeur de texte, un peu comme Google Docs ou Microsoft Word. Je voulais pouvoir permettre aux utilisateurs de mettre en forme leur texte comme ils le souhaitaient.

Après réflexion, je me suis dit que cela serait très long, si je devais créer de toutes pièces un éditeur de texte. J'ai alors cherché ce qui pouvait exister, avec un maximum de fonctionnalités répondant à mes besoins, afin d'avoir à en rajouter le moins possible.

En faisant cette recherche, j'ai beaucoup retrouvé le terme WYSIWYG. Au début, je ne comprenais pas trop ce que cela était, mais après quelques recherches, j'ai découvert que c'était l'acronyme de « What You See Is What You Get », que l'on peut traduire par, « Ce que tu vois est ce que tu obtiens ». Ce terme est utilisé pour définir tous les outils permettant d'éditer du texte grâce à plusieurs fonctionnalités, puis de pouvoir en extraire le code HTML correspondant.

J'ai cherché des comparaisons afin de voir lequel pourrait être le meilleur, et après quelques sites consultés, j'en ai sélectionné deux.

Le premier se nomme Quill, il me paraît très simple, efficace, mais intègre des fonctionnalités avec un style pas réellement moderne.

Le deuxième, TipTap, a l'air tout de suite plus moderne, proposant plus de fonctionnalités, mais me faisait douter sur la difficulté d'intégration.

J'ai donc décidé d'essayer d'intégrer Quill, et effectivement, très vite, j'ai obtenu un résultat avec des options pour éditer mon texte. Mais après quelques essais, je n' étais pas très convaincu de l'esthétique de cet outil. En plus de cela, il manquait une fonctionnalité très importante, l'insertion de tableau (et donc tout ce qui va bien pour l'édition, comme insérer une colonne à droite, à gauche, une ligne en bas, en haut, supprimer une colonne, une ligne, un tableau). Toutes ces fonctionnalités m' auraient pris beaucoup de temps à développer, donc j'ai décidé d'abandonner Quill, et de tester TipTap.

4.2.2 Choix de TipTap et création du premier menu

Malheureusement, pour TipTap, cela n'a pas commencé aussi bien qu'avec Quill, car de base lorsque l'on intègre, l'éditeur n'affiche qu'une simple zone de texte, avec aucune fonctionnalité. Par conséquent, j'ai cherché dans la documentation, et j'ai découvert un site servant d'exemple, permettant de tester les fonctionnalités. TipTap met à disposition le code de ce site, afin de pouvoir permettre à tout le monde de tester. C'était donc une plutôt bonne nouvelle, sauf qu'après quelques heures à installer toutes les dépendances, et à supprimer tout ce qui ne fonctionnait pas (du fait que je n'avais pas accès aux fonctionnalités payantes), je n'arrivais pas à afficher les menus contenant toutes les fonctionnalités proposées.

Finalement, j'ai abandonné l'idée de pouvoir tester facilement, et j'ai commencé à récupérer petit à petit certaines fonctionnalités. J'ai commencé par une que je trouve très intéressante, qui consiste à afficher un menu dès que l'utilisateur commence à écrire par la commande « / ». Ce menu offre la possibilité de commencer à écrire du texte en gras, en italique, en différentes tailles, sous formes de listes numérotées, à puces et bien d'autres fonctionnalités, mais surtout, d'intégrer des tableaux.

Bien évidemment, j'ai récupéré toutes les fonctionnalités depuis le site exemple. J'ai mis à jour toutes les dépendances, et adapté tous les éléments dans ma structure de fichiers. Après avoir fait cela, j'ai testé et constaté que seulement quelques fonctionnalités simples fonctionnaient correctement. Après recherche, j'ai découvert que dans le code exemple, il y avait un dossier extension, qui modifiait les extensions. Ainsi, après avoir ajouté les modifications d'extensions, j'ai obtenu de plus en plus de fonctionnalités fonctionnelles, et après quelques petites heures supplémentaires, je suis arrivé à avoir presque tout de fonctionnel.

A ce moment-là, j'avais donc un menu, permettant de modifier le texte des différentes manières, mais l'intégration de tableau ne fonctionnait qu'à moitié, du fait que l'insertion fonctionnait, mais qu'après cela, on ne pouvait pas le modifier (ajouter colonnes etc.). J'ai testé ensuite sur le site exemple afin de voir comment cela pourrait fonctionner, mais j'ai constaté alors que cela n'allait vraiment pas être

simple à intégrer, et que cela ajoutait de la complexité pour pas grand-chose. Pour résumer, afin de modifier les lignes, on pouvait cliquer sur une petite zone, permettant d'afficher les fonctionnalités possibles, et la même chose pour les colonnes dans une autre zone.

Mon problème était donc que, lorsque j'utilisais la fonctionnalité pour insérer un tableau, je n'avais absolument pas cette petite zone pour modifier les lignes et colonnes. Donc j'ai mis de côté ce problème, et décidé que mes fonctionnalités ne s'afficheront pas comme cela : C'est-à-dire que, simplement le fait d'avoir le curseur dans une cellule d'un tableau, affichera les fonctionnalités : insérer colonne à droite, à gauche, ligne en bas, en haut, supprimer colonne, ligne, tableau, et insérer tableau (pour permettre d'avoir des tableaux dans des tableaux). Ce n'est pas si évident, car il faut bien faire attention à ce que le menu s'affiche uniquement quand le curseur est dans un tableau et pas n'importe où. Pour cela, après avoir passé un certain temps à essayer de comprendre le code, j'ai pu voir que le menu devait s'afficher en fonction de l'attribut « name » dans lequel le curseur est positionné. Effectivement, pour résumer très simplement le fonctionnement, il faut savoir que dès que l'on écrit du texte dans la zone de l'éditeur, cela créé un paragraphe donc : , et par exemple, si on choisit la fonctionnalité pour mettre en gras, on va modifier cette balise en lui attribuant des attributs de style pour le mettre en gras. Mais pour ce qui concerne des plus grosses fonctionnalités, comme l'insertion de tableau, cela va créer un nœud contenant tous les attributs nécessaires ainsi que l'attribut « name » à la valeur « table ». Grâce à cela on va pouvoir gérer l'affichage de notre menu en utilisant la fonction « isActive » qui va donc permettre de retourner vrai ou faux en fonction du paramètre. Ainsi, si le curseur est dans le tableau, si l'on a editor.isActive('table'), cela va retourner vrai, et nous allons donc pouvoir afficher notre menu contenant les fonctionnalités d'ajout, suppression etc.

J'ai donc dû recréer un menu avec les fonctionnalités, gérer l'affichage, et au final, j'ai réussi à avoir toutes les fonctionnalités nécessaires pour un tableau.

4.2.3 Deuxième menu

A ce moment-là, j'avais mon menu qui s'affichait bien lorsque l'éditeur utilisait la commande « / », mais il manquait très logiquement des fonctionnalités telles que : pouvoir modifier la couleur d'un texte, le surligner, l'aligner à gauche, à droite, au centre, le justifier, insérer un lien, et quelques autres fonctionnalités (la liste complète des fonctionnalités sera listée dans la section résultats de ce rapport).

Par chance, ce menu existait déjà dans le site exemple, donc j'ai fait la même chose que pour le précédent menu. J'ai à nouveau eu à peu près les mêmes problèmes, que je ne vais pas détailler, mais pour résumer, je suis passé par ces différentes étapes : récupération du code, adaptation, constatation que la moitié des fonctionnalités fonctionnent, ajout d'extensions, découverte de nouveaux fichiers nécessaires, réorganisation des fichiers, difficulté pour intégrer les sous menu pour l'intégration de liens et autres.

A cette étape, je suis très fier du résultat ! Les fonctionnalités sont presque toutes présentes, elles sont simples à utiliser, les menus sont modernes, et je considère que la proposition des deux menu (menu « / » et celui qui s'affiche lors de la sélection de texte), fonctionne très bien.

Cependant, je savais pertinemment qu'il me manquait une fonctionnalité essentielle, mais j'ai décidé de faire une petite pause avec TipTap. Effectivement cette première partie n'a vraiment pas été simple, même si le fait d'avoir du code proposant des fonctionnalités est très pratique, ne pas pouvoir tester « qu'est ce qui correspond à quoi », n'est pas évident, notamment quand il n'y a aucun commentaire pour expliquer, et que certains fichiers ont les mêmes noms, mais ne font pas la même chose. Tout ça pour dire, que même si je n'ai pas eu à créer toutes les fonctionnalités, cette partie là, n'a pas été facile.

A partir de ce moment-là, je me suis demandé : comment est-ce que je vais pouvoir récupérer le texte écrit, le stocker, l'afficher et pouvoir le modifier à nouveau ? J'ai commencé par chercher comment l'extraire, et j'ai constaté qu'il existait une fonction très simple, permettant de récupérer tout le contenu de la zone de l'éditeur de texte, au format HTML, en conservant tous les attributs de style etc.

Ensuite, j'ai testé si cela fonctionnait correctement, en affichant dans la console le contenu de l'éditeur au format HTML, et j'ai ajouté ce code dans une autre page pour voir ce qui s'affichait et de quelle manière. Je me suis aperçu que tout s'affichait correctement sauf les retours à la ligne. Après vérification dans le code HTML, c'était à cause du fait que, dès que l'on fait un retour à la ligne, cela ajoute des balises de paragraphe <p><p/>. Après quelques recherches, il m'a donc simplement fallu définir que dès qu'il y a des balises « p » qui sont vides, cela crée un retour à la ligne.

4.2.4 Stockage des pages

A ce moment-là, j'avais donc mon texte qui pouvait s'afficher comme il le fallait. Du coup, j'ai décidé alors de réfléchir à comment j'allais stocker ces pages. Pour cela, deux idées me sont venues en tête : Soit, créer des fichiers HTML à chaque fois qu'un utilisateur éditeur voulait créer une page, et donc modifier ou supprimer ce fichier html en fonction des actions ; soit, gérer tout en base de données, donc stocker tout l'html dans une table et en fonction de l'url à laquelle accède l'utilisateur, récupérer en base de donnée et afficher, modifier, supprimer le texte.

Après quelques minutes de réflexion, il me semblait bien que cela serait plus simple de tout stocker en base de données. J'ai décidé de créer une table en base de données : datas_articles. Cette table devait donc contenir un titre, un id (construit à partir du titre, avec la même fonction utilisée pour les id des catégories, sous catégories et titres), et un contenu.

Pour commencer à peupler ma base de données, j'ai rajouté une zone pour que l'utilisateur puisse ajouter un titre pour la page. J'ai créé un formulaire qui contiendrait, un titre et le contenu de la zone d'édition. J'ai rajouté un bouton pour envoyer ce formulaire en base de donnée, crée la requête API POST, afin d'ajouter dans la table correspondante de ma bd, le contenu du formulaire, sans oublier d'ajouter l'id, créé comme expliqué plus tôt.

Ainsi, j'ai configuré mon bouton pour qu'il appelle ma route API, et testé directement si tout fonctionnait. J'ai écrit un nom, utilisé quelques fonctionnalités dans mon éditeur et appuyé sur le bouton pour envoyer les données en BD.

En utilisant PgAdmin4, j'ai visualisé que mes données avaient bien été rajoutées. J'ai donc décidé d'enchaîner avec la visualisation de pages. En accord avec mon choix réalisé plus tôt, j'ai créé seulement un fichier « .tsx », qui permettra d'afficher n'importe quelle page, en fonction de l'url.

Pour faire cela, j'ai conçu directement la requête API, qui devra donc, en fonction de l'url, récupérer les données de la page correspondante. C'est-à-dire que j'ai définis que : si l'on va à l'url : « urlDuSite/articles/annuaire », ma page devra récupérer le contenu dans la base de données qui a pour id, « annuaire ».

J'ai donc fait le nécessaire dans mon nouveau fichier : j'ai ajouté une zone pour le titre, une pour le contenu, appelé l'API, extrait les données et les attributs pour chaque élément. Après cela, j'ai essayé avec l'id de la page créé plus tôt, et cela fonctionnait parfaitement.

A ce moment précis, je pouvais créer une page, puis y accéder depuis l'url. J'ai ajouté alors une redirection vers la page créée, afin qu'après la création d'une page, l'utilisateur puisse constater directement sa création.

Une fois cela fait, j'ai rajouté des boutons, afin de pouvoir revenir à l'accueil, ou pouvoir modifier, ou supprimer la page. J'ai joint la redirection vers l'accueil et j'ai

enchaîné avec la modification. Après réflexion, j' ai pensé que ce n'était pas forcément une bonne idée de créer à nouveau un fichier « tsx » ayant exactement les mêmes fonctionnalités que la création, mais faisant des actions différentes.

4.2.5 Modification et redirection vers une page

En conséquence, j'ai repris mon fichier, anciennement utilisé uniquement pour la création de page, l'ai renommé, et rajouté quelques éléments. J'ai commencé par récupérer dans l'url, l'identifiant de la page à modifier : C'est-à-dire que pour modifier une page il faut accéder à l'url : « urlDuSite/articles/createorupdate?id=idDeLaPage ». Ainsi, si nous avons un attribut id dans l'url, nous sommes en modification, sinon, nous sommes en création. En fonction de cela, si nous sommes en modification, nous allons donc récupérer l'id de la page, et récupérer son contenu, à l'aide de la même requête API créée plus tôt pour afficher le contenu de la page. En plus de cela, nous allons attribuer à la zone de titre, le titre correspondant, et dans la zone d'édition, nous allons pouvoir attribuer une valeur par défaut, correspondant donc au contenu de la page.

Nous pouvions alors afficher et modifier le texte récupéré. Il allait donc falloir faire le nécessaire pour pouvoir enregistrer ces modifications en base de données. Pour cela, j'ai rajouté une requête API PUT, en passant l'identifiant de la page à modifier accompagné du contenu et du titre.

A ce moment-là, je pouvais modifier facilement mon texte. Il restait désormais seulement la fonctionnalité de suppression à installer. Dans ce but, j'ai rajouté une requête DELETE prenant l'id de la page ainsi qu' un message de confirmation avant de déclencher la suppression. S'il y avait confirmation, l'action se faisait alors correctement et l'utilisateur était directement redirigé vers l'accueil.

Il était maintenant temps de pouvoir lier le premier élément réalisé, qui est le menu, avec les pages que j'avais pu créer.

Pour faire cela, dans l'interface créé auparavant pour modifier une sous-catégorie ou un titre, j'ai modifié la zone qui consistait à ajouter une url externe : C'est-à-dire que j'ai rajouté, avant la zone de texte, un élément permettant de sélectionner parmi les pages créées, celle sur laquelle je voulais être redirigé lors d'un click. J'ai géré cela de manière à ce que l'éditeur puisse sélectionner une option pour ajouter un lien externe s'il le souhaitait. Dans ce cas-là, il pourrait rajouter le lien dans la zone de texte juste après. Si l'éditeur sélectionnait une page précédemment créée, l'url s'affichait dans la zone juste après, qui n'est dans ce cas-là pas modifiable.

Le fait d'afficher l'url de la page sélectionnée, permet à l'utilisateur de vérifier que c'est bien la page à laquelle il souhaite accéder, mais permet également de ne pas avoir à modifier la requête, puisque ce sera cette url qui sera stockée en BD.

Ensuite, j'ai aussi rapidement modifié le menu, afin que si l'utilisateur survole une sous-catégorie ou un titre qui contient une url, le curseur indique un texte cliquable.

Désormais, nous pouvons choisir sur quelle page rediriger un titre ou une sous-catégorie, et y accéder en cliquant dessus.

4.2.6 Gestion de fichiers

Après avoir réalisé cela, j'ai décidé de passer à une autre fonctionnalité importante qui doit être présente dans l'intranet : pouvoir ajouter des fichiers liés à une page qui pourront être téléchargés par les utilisateurs.

Pour y aboutir, j'ai commencé par réfléchir à comment j'allais pouvoir stocker ces fichiers. La première idée qui m'est venue en tête, était de créer un dossier, qui allait stocker tous les fichiers déposés, et par la suite, en fonction des fichiers déposés par chaque page, récupérer ceux correspondants. J'ai alors commencé à créer ce dossier dans le dossier public de mon arborescence de fichier, car il était accessible depuis le navigateur, qui allait donc devoir afficher les fichiers qui peuvent être téléchargés.

Après avoir avancé, créé l'interface, j'ai pensé que ce serait mieux de créer un dossier par page afin de plus facilement visualiser les fichiers de chacun. Pour faire cela, j'ai choisi de créer un dossier en fonction de l'id de la page créée.

Mais à cette étape là, j' ai été confronté a un problème… Effectivement, cela pourrait potentiellement fonctionner pour ajouter des fichiers lors de la modification d'une page, mais pas lors de la création car l'identifiant de la page est créé à partir du titre lorsqu'on l'envoie en base donnée. C'est-à-dire qu'avant cela, il n'y a pas d'identifiants. J'ai donc fait une pause sur la gestion de fichiers, et commencé à réfléchir à la création d'un identifiant temporaire.

Finalement, j'ai choisi de créer une fonction qui va générer un nombre aléatoire entre 1 et 999999999. J'ai décidé de modifier la fonction que j'utilisais pour générer un identifiant à partir du titre d'une page, afin que si aucune valeur n'est passée en paramètres, elle appelle la fonction qui génère le nombre aléatoire et retourne ce nombre.

J'ai donc modifié le fichier de création de page afin de générer un identifiant temporaire dès l'arrivée de l'utilisateur sur la page de création.

Suite à cela, j'ai repris la gestion de fichiers, et défini alors que le dossier qui accueillerait les fichiers ajoutés, aurait au départ pour nom, l'id temporaire, puis, lors de l'envoi des données, serait renommé avec le nouveau nom.

A ce niveau-là, je n'avais pas encore testé mais cela me paraissait correct. J'ai pensé qu'il serait intéressant de créer deux dossiers dans le dossier correspondant à la page. Le premier contiendrait les fichiers qui ont été enregistrés, et le deuxième, un dossier temporaire, qui contiendrait les fichiers ajoutés lors de la modification d'une page. En fait, dès que les modifications seraient enregistrées, le dossier temporaire écraserait le dossier des fichiers sauvegardés. Cela permettrait aux utilisateurs de voir les nouveaux fichiers seulement si la page modifiée a été enregistrée.

J'ai donc commencé à tester ma fonctionnalité. Résultat, j'arrivais bien à rajouter des fichiers, ils s'affichaient, mais lorsque j'envoyais les données, et que le dossier devait être renommé par l'id formalisé de la page, il y avait une erreur d'accès.

Au départ je ne comprenais pas pourquoi… Puis, après un bon nombre de recherches, j'ai découvert que c'est TypeScript qui bloquait certaines actions dans le dossier public lorsque l'application était exécutée. J'ai donc essayé de contourner ce problème, mais au bout d'un certain temps, je me suis rendu compte que le problème n'était pas réellement TypeScript, mais la manière donc je stockais mes fichiers.

Brutalement, je me suis rendu compte que ce que j'avais fait dès le départ n'allait pas du tout… Stocker toutes les données dans le répertoire n'avait aucun sens, car de un, c'est TRÈS loin d'être efficace de stocker tout dans ce répertoire, et de deux, ce n'est absolument pas le but du répertoire public, il n'est pas fait pour stocker et gérer les fichiers d'un site. Il sert simplement d'intermédiaire, entre le stockage, qui doit se faire dans un répertoire à part du site, et le navigateur qui doit récupérer ces fichiers, pour pouvoir par la suite, permettre de les télécharger. Cette relation est comparable à celle que j'ai pu expliquer plus tôt, entre le navigateur, la base de données et les API. Ici, c'est le dossier public qui sert d'intermédiaire entre le navigateur et le répertoire de stockage.

Je peux donc remercier TypeScript d'avoir bloqué ces opérations, car je ne m'étais pas rendu compte de la bêtise que je réalisais.

J'ai alors réglé cela en prenant le temps de bien réfléchir à tous les éléments, en reprenant depuis le départ la structure des fichiers.

Au final, j'ai donc fini par gérer le stockage de manière à ce que, dès qu'une page est créée, si un fichier est ajouté, cela crée un dossier dans public, ayant pour nom l'id temporaire de la page. Si l'éditeur décide de publier cette page, le dossier qui est dans public est copié vers le répertoire de stockage, puis renommé par le nouvel id. On termine en supprimant le dossier dans public.

Par contre si une page est modifiée, alors le dossier est récupéré depuis le stockage, dans le dossier public, puis, lorsqu'on valide les modifications, on copie le dossier public vers le stockage que l'on écrase.

On remarque que l'on a plus besoin de dossiers temporaires, du fait que les fichiers sont mis à jour uniquement lorsque l'on enregistre les modifications.

Et pour ce qui est de la consultation, comme pour l'édition, on va simplement copier le dossier correspondant dans public, mais on ne modifiera jamais le stockage.

Une fois que tout a été mis au clair et testé, j'ai listé les fichiers et rajouté la possibilité de pouvoir télécharger les fichiers déposés. Pour ce faire, j'ai créé une nouvelle route API GET afin de pouvoir récupérer tous les noms des fichiers correspondant à la page. En plus du téléchargement, j'ai créé une nouvelle requête API DELETE, contenant l'id de la page et le nom du fichier, afin de bien supprimer le fichier demandé.

Grâce à tout cela, nous pouvons donc ajouter, supprimer et télécharger des fichiers liés à une page.

J'ai finalisé cette partie du travail en adaptant l'affichage en fonction de l'action en cours. C'est-à-dire que pour la consultation on ne peut que télécharger, alors que pour la création et l'édition, en plus du téléchargement, on peut également ajouter et supprimer des fichiers.

Suite à cette fonctionnalité, j'ai décidé de recommencer à avancer sur les fonctionnalités qu'il manquait à l'éditeur de texte : l'insertion d'image, et de vidéos.

4.2.7 Finalisation des fonctionnalités d'un menu

J'ai choisi de commencer par l'insertion d'images car il y avait une base qui était fournie dans le site exemple. Tout le côté interface était déjà fait (donc le bouton pour insérer, l'affichage), mais pas le système de stockage. Effectivement, cela tombait vraiment bien, du fait que dans la dernière fonctionnalité réalisée, j'ai appris à gérer des fichiers.

J'ai alors repris la même idée, à la différence que j'ai décidé de modifier légèrement la structure, afin de ne pas avoir les images mélangées avec les fichiers à télécharger. J'ai donc modifier ma précédente fonctionnalité afin de générer un dossier « files », où seront déposés les fichiers. Pour l'ajout d'images, j'ai fait générer un dossier « images », pour les images insérées par l'éditeur. Ces deux dossiers sont donc stockés dans le même dossier correspondant à la page.

Comme pour les fichiers, j'ai repris le même système de route pour stocker les images, et j'ai géré l'affichage de celles-ci.

J'ai également rajouté une fonction, qui vérifie le contenu du code HTML, lors de l'envoie en base de données, dans le cas où un l'utilisateur ajoute une image,

mais plus tard, décide de l'effacer, l'image ne sera pas conservée dans le stockage lors de la publication.

Dans le site exemple de TipTap, on retrouve un petit menu permettant de modifier des éléments d'une image comme sa dimension et l'alignement. J'ai rajouté ce menu, qui m'a causé quelques soucis, mais au bout de quelques heures, j'ai pu bien manipuler mes images ajoutées, en les déplaçant en utilisant le « drag and drop », en les redimensionnant, ou en les alignant à droite ou à gauche ou au centre.

J'ai donc réussi à avoir un système de gestion d'images relativement complet, donc j'ai considéré que je pouvais passer aux vidéos.

Pour les vidéos, la demande était de pouvoir réellement déposer des vidéos, comme pour les images. Effectivement, intégrer simplement un lecteur YouTube aurait été bien plus simple, mais ce n'était pas la demande. J'ai donc repris l'interface du dépôt de photo, pour les vidéos, et recommencé le même principe pour ce qui est du stockage. J'ai rajouté la génération d'un dossier « videos », à côté de « files » et « images ». Effectivement, j'aurais probablement pu stocker les images et vidéos dans le même dossier, mais j'ai pris la décision d'en avoir deux bien distincts.

Une fois que j'ai pu réussir à stocker mes vidéos, j'ai commencé à chercher un lecteur de vidéo, qui permette facilement de faire avancer, mettre en pause, ou par exemple accélérer la vidéo. Lors de mes recherches, j'ai trouvé l'extension Next-video. Elle m'a paru parfaitement adaptée à mon environnement. Cependant, après avoir suivi un tutoriel (voir glossaire) sur YouTube, je n'ai pas exactement tout compris, mais j'ai eu l'impression qu'il fallait stocker les vidéos sur un site afin de pouvoir par la suite les récupérer et les afficher.

A ce moment-là, j'étais bien embêté, car je voulais simplement afficher les vidéos enregistrées en local dans le stockage de mon ordinateur / serveur. J'ai repris mes recherches en utilisant simplement les mots « video react », et en premier résultat, j'ai vu la documentation de la bibliothèque React. En ouvrant cette page, j'ai découvert simplement qu'il existe un élément <video> que l'on peut utiliser simplement en passant la source de notre vidéo. En plus de cela, il y a une option « controls » qui peut être ajoutée, afin de pouvoir faire avancer la vidéo, mettre en pause, et toutes les fonctionnalités classiques d'un lecteur. En voyant cela, je me suis senti « bête » d'avoir fait une recherche complexe sur internet du style « nextjs localhosting video mp4 controler ». Je ne saurais pas dire pourquoi, mais j'étais absolument persuadé qu'il n'existait pas de fonctionnalités natives dans React permettant l'affichage d'un lecteur vidéo.

Malgré ce temps perdu bêtement, je n'ai pas lâché l'affaire, et j'ai continué d'avancer sur le projet. J'ai alors décidé de commencer à gérer la connexion.

4.3 Gestion de la connexion

Pour faire cela, j’ai commencé par créer les interfaces pour l’inscription et la connexion, puis j’ai créé la table « users » qui contiendra tous les membres inscrits. J’ai continué en créant la requête API afin d’ajouter l’utilisateur en BD, en prenant le soin de hasher (voir glossaire) le mot de passe avant de le stocker en base de données. J’ai poursuivi en gérant la connexion, en commençant par la vérification de l’email dans la BD, et en effectuant une comparaison avec le mot de passe en base de données. J’ai enchaîné rapidement en gérant la connexion avec un token (voir glossaire) JWT(voir glossaire) plutôt qu’en session. Ce token me permet de contrôler différentes choses comme le temps avant expiration (temps avant de devoir se connecter à nouveau ; durée limitée afin de ne pas laisser une personne connectée, si elle oublie de se déconnecter) ou la quantité de temps que l’on repousse l’expiration du token (on déduit que si la page est rechargée ou si l’utilisateur change simplement de page, l’utilisateur est toujours actif sur le site, donc on rajoute du temps).

J’ai ainsi abouti à un système de connexion sécurisé et fiable. J'ai continué alors avec la gestion des rôles qui est très importante.

4.4 Gestion des rôles

Effectivement, si on veut pouvoir gérer l'accès à des pages, que ce soit pour consulter ou modifier, il faut avoir une bonne gestion des autorisations, donc des rôles.

Pour cela, j'ai ajouté un bouton « Administration » qui permettra seulement aux utilisateurs avec le rôle « Administrateur », de gérer les rôles et les accès.

Suite à cela, j'ai créé une nouvelle table « rôles » avec comme clé primaire un nom, un attribut « pages » qui listent toutes les sous-catégories, et un attribut « access », qui correspond au niveau d'accessibilité des sous catégories (0 caché, 1 consultation, 2 édition). Je m'explique : je n'ai pas trouvé mieux comme solution me permettant de gérer l'accès aux pages, avec le temps qu'il me restait pour finaliser le projet. Effectivement, en gérant les rôles de cette manière, je suis parti du principe que :

- Il faut absolument un rôle Administrateur, permettant de modifier toutes les pages, tout le menu, les rôles
- Il faut absolument un rôle Éditeur, permettant de modifier toutes les pages et tout le menu.
- Seules les personnes ayant le rôle Administrateur ou Éditeur peuvent ajouter, modifier, supprimer une catégorie principale ou une sous-catégorie
- Tout le monde peut voir les noms des catégories principales
- Par défaut, tout le monde voit tout mais ne peux rien modifier
- Les rôles qui ont la permission d'éditer ou de consulter des sous-catégories ont les mêmes autorisations pour tous les titres de cette sous-catégorie (c'est-à-dire que si un rôle peut éditer la sous-catégorie annuaire, il peut alors éditer aussi les titres Recherche par service, Recherche par mot clé etc.)

J'ai fait ce choix de manière à être cohérent avec la manière choisie pour organiser les informations (au plus proche de l'ancien intranet pour ne pas trop perturber), et également, car selon moi, des personnes au sein d'une même catégorie principale, ne peuvent pas avoir les autorisations pour toutes les sous-catégorie sous-jacentes (ex : Dans la catégorie principale « Services », la sous-catégorie « Administratif » ne devrait pas pouvoir être modifiée par des personnes du service Biologie).

Je ne sais pas exactement comment seront utilisés les rôles, mais je laisserai à disposition les rôles Administrateur, Editeur et Membre, au cas où les utilisateurs ne voudraient pas s'embêter à se répartir les tâches d'édition proprement, car cela peut être contraignant.

Pour réaliser cette gestion de rôles, j'ai donc commencé par ajouter une zone de texte pour le nom du rôle, puis j'ai affiché la liste de toutes les sous catégories en les regroupant chacune, en fonction de leur catégorie principale. A cela, j'ai rajouté deux colonnes de cases à cocher, une permettant de cacher la sous-catégorie et une permettant d'autoriser l'édition. Effectivement, j'ai eu à gérer le fait que seulement une des deux cases soit cochées, car si on modifie, on doit voir, et si on ne doit pas voir, on ne peut pas modifier. J'ai également rajouté pour chaque catégorie principale, un bouton « tout cocher » permettant de cocher rapidement toutes les catégories pour chacune des deux permissions.

Après avoir fait cela, j'ai rajouté le bouton qui permettra d'envoyer ces informations en base de données. J'ai créé une liste contenant toutes les sous-catégories. J'ai continué d'avancer mais effectivement, en base de données, cela m'aurait gêné d'avoir deux attributs pour gérer les accès. Je les ai donc regroupés en un seul. En choisissant cette idée, j'ai apporté des modifications à chaque sous-catégories : si la case « caché » est cochée j'ajoute à une liste le nombre 0, si la deuxième est cochée, j'ajoute 2, si aucune des deux ne sont sélectionnées, j'ajoute 1. J'ai donc fait une boucle pour le faire pour chaque sous-catégorie.

Au final, j'ai obtenu un nom de rôle, une liste de sous-catégories, et une liste contenant les accès à chaque sous-catégorie.

La relation entre ses deux listes n'est pas évidente à expliquer, et donc à comprendre donc je préfère directement l'illustrer par un exemple.

Si par exemple, la sous-catégorie « Annuaire » à un indice (voir glossaire) de 5 dans la liste des sous-catégories, alors, l'élément qui a pour indice 5, dans la liste des accès, correspond au niveau d'accès pour la sous-catégorie « Annuaire ».

Suite à cela, j'ai créé la requête API POST, qui me permettra d'enregistrer les rôles dans la BD, puis j'ai déclenché l'appel à cette même route dès que le bouton pour enregistrer est cliqué.

A ce stade, je pouvais maintenant créer des rôles comme je le souhaitais, mais je ne pouvais ni les attribuer, ni les modifier.

J'ai décidé de commencer par la modification. Pour ce faire, j'ai récupéré un paramètre dans l'url qui correspondait soit à « create », soit à « edit ». Dans les cas où ce serait une édition, j'aurai la liste des rôles affichés, et je pourrai donc choisir lequel modifier. En fonction du rôle sélectionné, je pourrai récupérer toutes les informations de celui-ci. Encore une fois, en utilisant une requête API, je récupérerai toutes les valeurs enregistrées en BD pour ce rôle. Avant d'afficher les informations, il a fallu rajouter pour chaque sous-catégories, le fait que si dans la liste des accès, la valeur est 2, cela signifie qu'il faut cocher la case « éditer », si c'est 1 ne rien cocher et si c'est 0, cocher la case « cacher ».

J'ai donc appliqué ces résultats et j'ai vu alors apparaître, en fonction du rôle sélectionné, les différentes autorisations. Suite à cela, une fois de plus, j'ai créé

une requête PUT pour mettre à jour mes données, et j'ai déclenché son appel lors du clic sur le bouton pour enregistrer les modifications.

Pour en finir avec cela, j'ai rajouté un bouton pour supprimer le rôle (j'ai créé la requête …)

Une fois que cela a été fait, il ne restait plus qu'à avoir la possibilité d'attribuer un rôle. De base, un utilisateur n'est pas obligé d'avoir un rôle, mais j'ai fini de manière à gérer l'inscription pour ajouter le rôle Membre par défaut.

Pour continuer, j'ai réorganisé un peu mieux ma page d'administration, afin d'avoir un bouton pour créer un rôle, qui redirige vers la page de création, un pour l'édition. En dessous de ceux-ci, j'ai créé une liste de « cartes » représentant les utilisateurs enregistrés avec leur prénom et leur adresse email. J'ai ajouté en plus de cela, le rôle qu'ils possèdent parmi la liste de ceux existants. Cela permettra donc aux administrateurs de pouvoir accorder des permissions aux utilisateurs grâce aux rôles créés précédemment. L'administrateur doit valider un par un la modification du rôle en cliquant sur le bouton dans le « carte » de l'utilisateur. Ce bouton appelant une requête API PUT créée comme il se doit.

Voilà donc pour le système de gestion de rôles, je peux donc créer, modifier, supprimer, attribuer chacun des rôles. Je vais donc devoir désormais mettre en place la sécurité permettant de faire appliquer ces autorisations.

4.5 Gestion de la sécurité

J’ai commencé directement par l’accès au menu d’administration, que j’ai défini uniquement accessible par les personnes ayant un rôle administrateur (il faut donc être connecté). Pour ce faire, j’ai créé une fonction qui récupère le token de connexion, qui le décrypte et qui le retourne, ce qui me permet donc d’avoir les informations de connexion (email, nom, rôle). Je peux donc grâce à cela, vérifier si l’utilisateur à un rôle correspondant à Administrateur ou non.

Par la suite, j’ai poursuivi avec la gestion de l’affichage du menu. Pour cela, j’ai commencé par vérifier si l’utilisateur était connecté ou non. Je suis parti du principe que si l’utilisateur n'était pas connecté, il aurait les mêmes autorisations qu’avec le rôle « membre », donc il pourrait consulter toutes les pages (évidemment le rôle membre pourra être modifié plus tard si les administrateurs considèrent que n’importe qui ne peut pas tout voir).

J’ai conditionné l’affichage des sous-catégories de manière à ne pas l’afficher si le rôle à pour accès 0 pour cette sous-catégorie. Par cascade, je n’ai pas affiché non plus les titres correspondant à la sous-catégorie. J’ai géré ensuite la modification et la suppression, en utilisant le même principe, donc si pour une sous-catégorie, l’accès correspond à 2, j’afficherai l’icône pour supprimer, et celui accéder à la page d’édition, sinon non. Par cascade, j’ai affiché les icônes pour modifier et supprimer les titres.

La sécurité au niveau du menu étant faite, j’ai continué à sécuriser, en conditionnant l’affichage des boutons de modifications, que j’avais affichés directement depuis la visualisation des pages.

Pour finir, j’ai conditionné l’accès au pages, en édition, en consultation ou en création, afin que les utilisateurs puissent accéder seulement s’ils ont un rôle le leur permettant.

4.6 Ajout de plusieurs nouvelles fonctionnalités

Je considère que ce rapport de stage est assez long pour ne pas expliquer dans les détails des petites fonctionnalités. Je vais donc les lister rapidement :

Après avoir géré la sécurité et le système de rôle j'ai pu rendre la page d'accueil presque entièrement modulable, afin de pouvoir permettre à une personne ayant le rôle Editeur, de pouvoir modifier les accès rapides, et le contenu affiché. Pour faire cela j'ai rajouté des tables en base de données, créé mes requêtes API et conditionné les accès à ces actions. Nous constatons que nous retrouvons le même pattern (voir glossaire) que pour le menu, mais encore plus simplifié.

Après cela, j'ai également rajouté la possibilité, toujours pour l'Editeur, de pouvoir ajouter un bandeau d'annonce, comme vu sur la maquette, qui permettra d'annoncer une information généralement temporaire. Une fois de plus, c'est toujours le même pattern : nouvelle table, requêtes API, conditionnement de l'accès.

Ensuite, j'ai pu passer à autre chose de plus intéressant, car totalement différent. J'ai rajouté un bouton dans la page d'administration permettant de pouvoir créer des sauvegardes de la base, et donc aussi de pouvoir restaurer la base de données à partir de ces sauvegardes. Pour pouvoir permettre cela, j'ai donc eu à créer une requête API, consistant à exécuter la commande : "pg_dump", afin de générer le fichier de sauvegarde de celle-ci.

Une fois cela testé et bien fonctionnel, j'ai pu afficher la liste des fichiers, afin de permettre à l'administrateur de les télécharger. Je ne réexplique pas cela, car c'est exactement le même système que j'ai pu utiliser lors de l'affichage des fichiers liés à une page.

Je suis donc passé à la partie consistant à restaurer la base de donnée à partir d'une sauvegarde. Pour faire cela, j'ai donc rajouté un bouton permettant d'ajouter un fichier, puis créer la requête API qui va exécuter cette fois-ci la fonction : "pg_restore" suivi des attributs qui vont bien, afin d'accéder à la bonne base de données et d'utiliser le bon fichier pour la restaurer.

En plus de cela, j'ai préparé tout l'affichage qui permettra de redémarrer le serveur, une fois que l'administrateur aura cliqué sur un certain bouton. J'aurais aimé pouvoir faire une page de chargement, permettant aux utilisateurs de comprendre que le serveur redémarre, mais cela prendrais encore beaucoup de temps que je n'ai pas.

A ce moment-là, j'aurais également aimé pouvoir ajouter des fonctionnalités permettant de mettre à jour les outils utilisés, mais il me restait trop peu de temps avant le rendu de mon rapport de stage. Ce sera fait durant les dernières semaines de stage, potentiellement avant la soutenance.

4.7 Phase de tests

Avant de passer au déploiement, j'ai démarré la phase de test. Pour cela j'ai utilisé Jest, qui m'a permis de faire différents types de tests.

J'ai donc testé seulement quelques comportements. J'ai pu réaliser des tests unitaires dans les fonctions, des tests d'affichage de composants, d'affichage d'éléments, ou bien des tests. J'ai pu en réaliser au niveau de la fonction pour générer un titre, pour tout ce qui touche l'inscription / connexion, l'affichage du menu, et de l'éditeur.

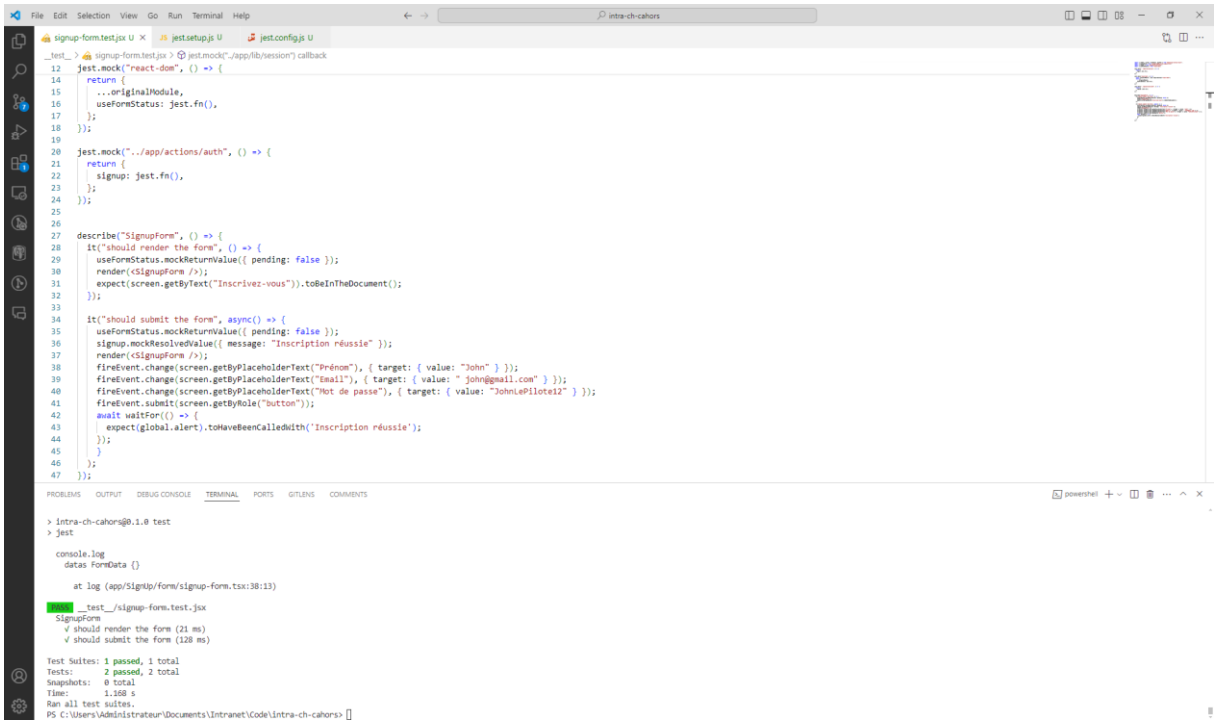


Figure 26 : Tests

4.8 Déploiement

En suivant, j'ai lancé l'étape la plus importante, qui est le déploiement. Pour pouvoir réaliser cela facilement, le personnel du service responsable de l'infrastructure ainsi que le support technique, m'ont configuré un poste, permettant d'accéder au nouveau serveur windows qui est actuellement vierge. Nous avons donc installé ce poste dans mon bureau et connecté au réseau afin de me permettre d'accéder à ce serveur dans les meilleures conditions.

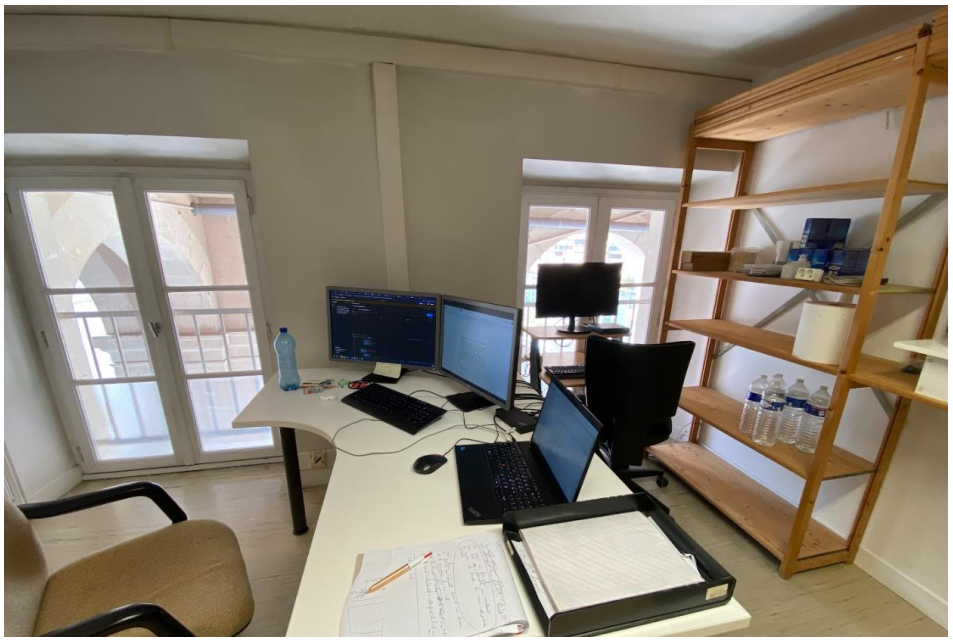


Figure 27 : Nouveau matériel à disposition

Avant de commencer à installer le nécessaire sur le serveur, j'ai commencé par nettoyer mon code, commenté tout le nécessaire, et testé si le déploiement sur Vercel fonctionnait bien. Une fois que je n'avais plus aucun warning ou erreur durant le build, j'ai pu commencer l'installation.

J'ai choisi de commencer avec une installation assez simple pour commencer, je verrais par la suite, si j'ai besoin réellement d'éléments supplémentaires.

Après recherche, j'ai choisi d'utiliser IIS (Internet Information Services), afin de pouvoir héberger l'intranet. Je ne pars pas de zéro avec cet outil, car j'ai pu l'utiliser lors de mon dernier stage lorsque je devais réaliser des actions sur des applications déployées sur IIS. Du fait que je n'ai pas de grandes compétences, j'ai décidé de suivre des exemples en vidéo ou sur des sites internets, et j'ai donc fini par réaliser ces différentes tâches.

J'ai commencé par activer IIS sur le serveur. Suite à cela, j'ai installé Node.js, puis Git, puis PostGreSql. J'ai commencé par récupérer mon code, en clonant mon répertoire qui est sur GitHub. Une fois que le code a bien été récupéré, j'ai installé toutes les dépendances. J'ai continué en restaurant ma base de données, à partir d'un fichier obtenu à partir d'un fichier de sauvegarde généré au préalable.

En voulant faire cela, j’ai décidé d’installer, au moins temporairement, Visual Studio Code et PGAdmin4. Avec Visual Studio Code j’ai modifié donc tous les éléments qui faisaient référence à la base de données hébergée sur Vercel, en remplaçant par ma nouvelle base de données sur le serveur. Tant que j’y pensais, à ce moment-là, j’en ai profité pour modifier tous les chemins que j’avais pu définir pour stocker les fichiers des différentes pages. Cette opération a été assez longue, afin de bien vérifier que tout fonctionnait comme il le faut.

J’ai continué à suivre les tutoriels. J'ai ajouté un fichier “server.js”, qui allait désormais servir de point d'entrée pour l’application. Pour ce faire, j’ai copié et collé le code dans ce nouveau fichier, puis j’ai continué à suivre. Après cela j’ai eu besoin de créer un fichier “web.config”, qui lui était un fichier lu par IIS, pour configurer cette application qui serait hébergée sur IIS. Suite à cela, il a fallu modifier la commande npm run start, afin qu’elle démarre le serveur grâce au fichier server.js.

Une fois cette étape réussie, j’ai donc pu construire notre application, puis m’attaquer à IIS. Il a été nécessaire de créer un nouveau site, définir le nom, et donner le chemin vers le répertoire de l’application NextJS. Finalement, il n’y a pas eu de problème, notre site est maintenant accessible !

Le déploiement étant fait, il faudra donc prochainement voir avec le reste de l’équipe, comment faire pour que le site soit intégré correctement au réseau, et que l’accès soit bien sécurisé pour éviter tout problème. Cela sera effectué dans la période après le rendu de ce rapport, tout comme la création des pages et le module de mise à jour expliqué plus tôt. J’espère vivement que j’aurais le temps pour pouvoir présenter cela lors de la soutenance, ou du moins, réussir à en faire le maximum.

V. Résultats

1. Ce qui a été fait

1.1 Utilisateur déconnecté ou Lecteur

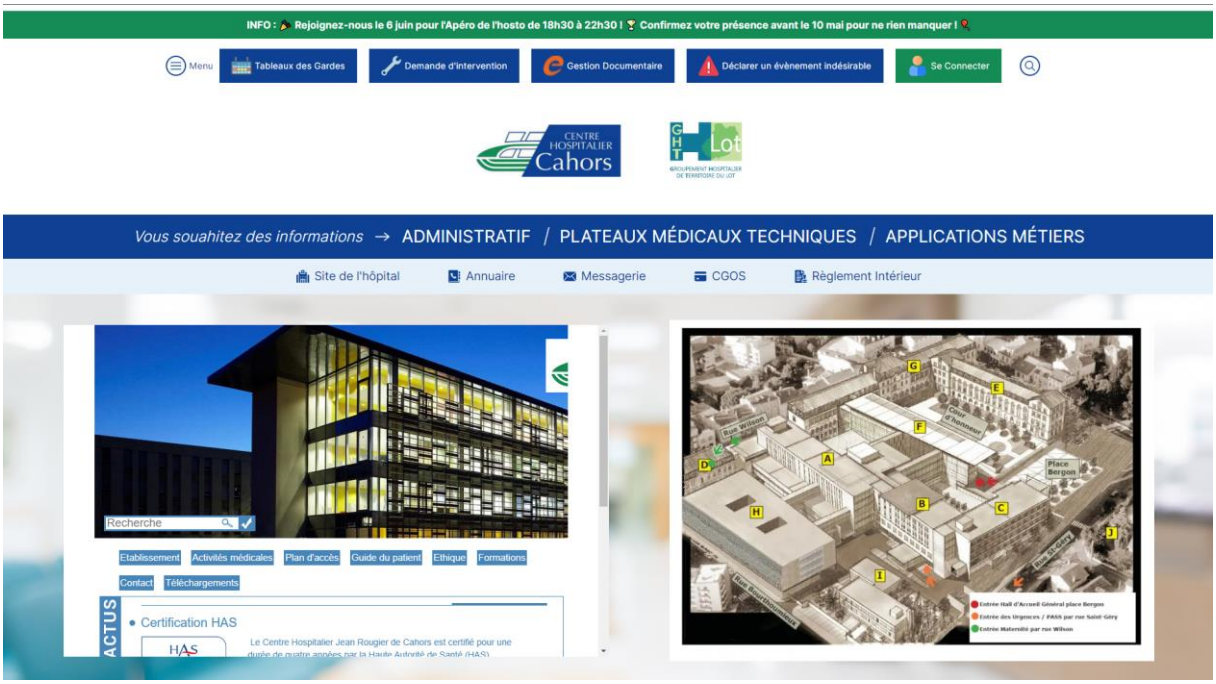


Figure 28 : Page d'accueil utilisateur non connecté

En ce qui concerne les fonctionnalités réalisées je vais donc lister celles qui sont présentes dans l'actuel intranet.

Nous avons donc ici la page d'accueil, sur laquelle nous arrivons quand nous accédons à l'intranet. D'ici, nous pouvons réaliser différentes actions :

- Consulter le bandeau d'annonce qui peut être utilisé pour diffuser des informations à durée temporaire.
- Accéder au menu en cliquant sur le bouton en haut à gauche
- Accéder à une page interne ou externe, depuis les raccourcis sur la partie haute
- Se connecter afin d'avoir des permissions particulières
- Rechercher un élément dans la barre de recherche en haut à droite
- Cliquer sur les deux logos pour revenir sur cette page
- Cliquer sur un des éléments du bandeau bleu foncé, pour ouvrir le menu directement dans la bonne catégorie
- Accéder à d'autres pages, internes ou externes

- Actuellement (ce n’est pas définitif), consulter le site internet de l'hôpital, directement depuis l’intranet
- Consulter le plan, ou accéder directement à l’annuaire quand nous cliquons dessus (fonctionnalité de l’ancien intranet reprise pour ne pas perturber) .

Nous allons donc commencer par parler de toutes ses fonctionnalités en tant que personne lambda, qui n’est pas connecté.

L’utilisateur va donc pouvoir ouvrir le menu :



Figure 29 : Menu utilisateur non connecté

L’utilisateur retrouve donc le menu, comme il avait été réalisé sur figma, avec les catégories principales sur la gauche, la première sélectionnée à l’ouverture du menu, permettant de consulter les sous-catégories et les titres. Il peut donc cliquer sur une sous catégorie ou un titre afin d’être redirigé vers une page.



Figure 30 : Exemple de page utilisateur non connecté

Quand il arrive sur cette page, il peut donc voir tout le contenu de la page, mais également, un bouton sur le côté qui est cliquable et qui permet de “voir les fichiers”.

Quand l’utilisateur (toujours non connecté) clique sur le bouton, alors un petit menu apparaît :



Figure 31 : Téléchargement de fichiers utilisateur non connecté

Nous avons ici deux fichiers qui ont été déposés par l’éditeur de la page, qui peuvent être téléchargés en cliquant sur le petit icône à côté du nom. Si nous ne

voulons plus voir ce menu, nous pouvons cliquer sur le bouton pour le réduire, et donc, revenir en position initiale.

Pour finir, un utilisateur qui n'est pas connecté, peut s'inscrire ou se connecter : pour cela il va appuyer sur le bouton "se connecter", qui va lui afficher l'interface suivante :

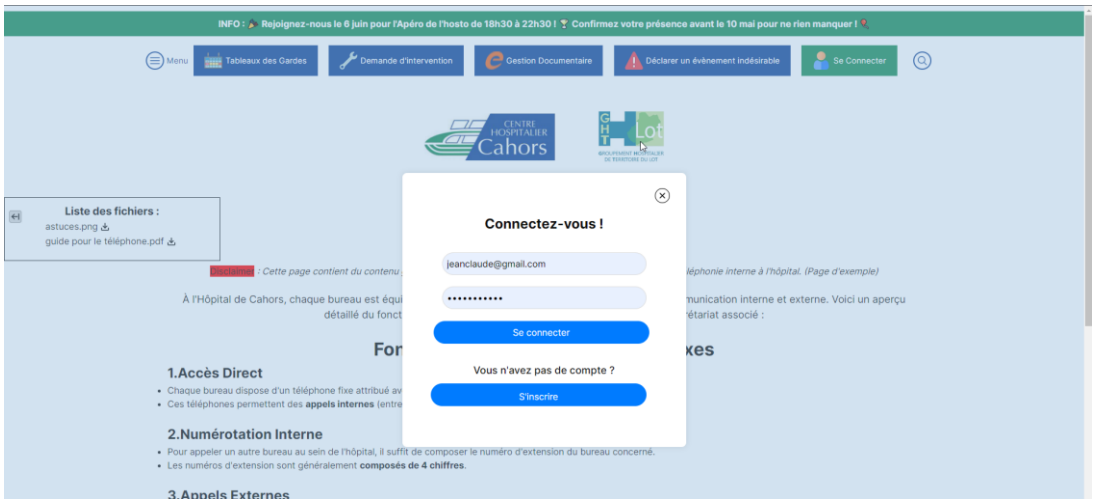


Figure 32 : Formulaire de connexion

Si l'utilisateur à un compte, il peut directement cliquer sur s'inscrire, sinon, il clique sur se connecter, et arrive sur la page suivante, qui est donc le formulaire d'inscription :

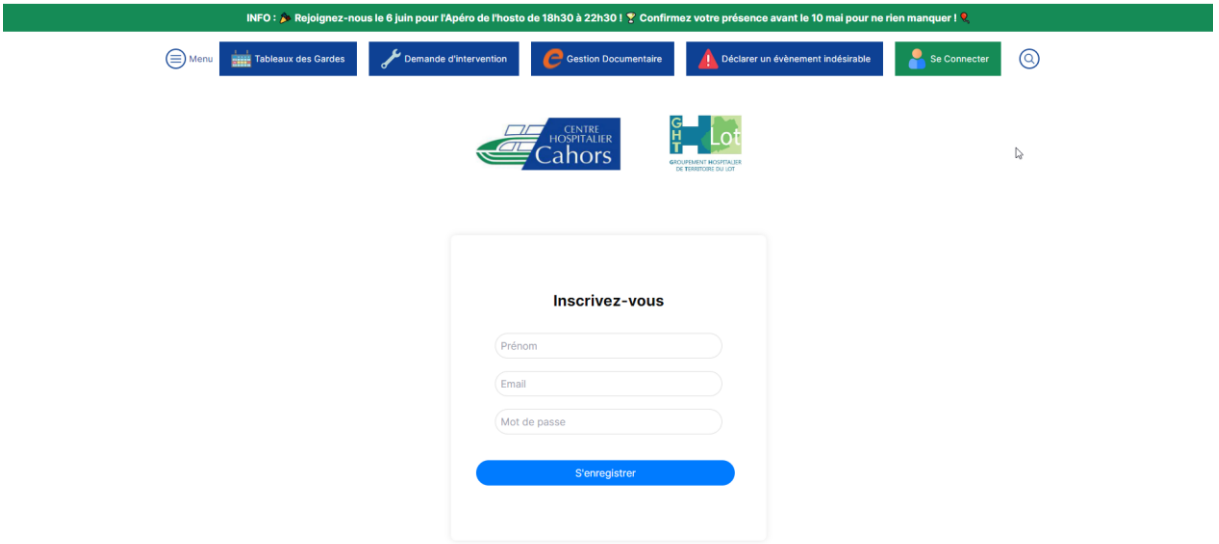


Figure 33 : Formulaire d'inscription

Le formulaire est très simple, seulement un email pour servir d'identifiant, un prénom et un mot de passe relativement complexe (huit caractères minimum, au moins un nombre, au moins un caractère spécial). Suite à cela, un message s'affiche pour lui confirmer son inscription et il est automatiquement connecté.

Voilà donc pour tout ce que peut faire un utilisateur qui n’est pas connecté (ou qui a seulement des permissions de lecture).

Nous allons donc passer maintenant au rôle d’Éditeur, qui lui, peut modifier tout le site. Pour cela, je me connecte donc avec un utilisateur ayant le rôle d’Éditeur.

1.2 Le rôle d’Éditeur

En tant qu’Éditeur, nous remarquons assez rapidement qu’il y a de nouveaux boutons dans la page d’accueil. Effectivement, l’éditeur va pouvoir modifier les neufs raccourcis au total, et le bandeau (afficher / modifier / supprimer).

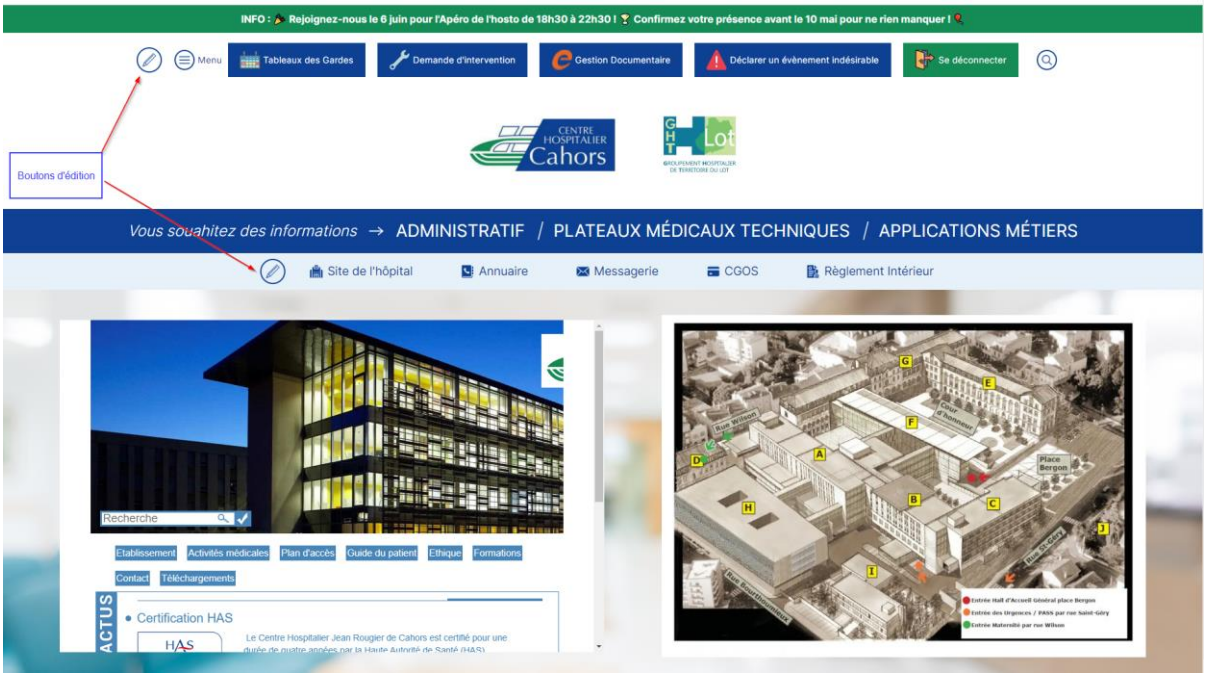


Figure 34 : Page d'accueil Éditeur connecté

Donc, quand nous cliquons sur le bouton d’édition dans l'en-tête, nous arrivons sur une interface permettant de paramétrer chaque raccourci. Le choix a été fait de pouvoir changer : l'icône, le nom et le lien vers lequel l'utilisateur est redirigé s’il clique, mais pas le nombre (c’est la même chose pour les cinq autres raccourcis plus bas).

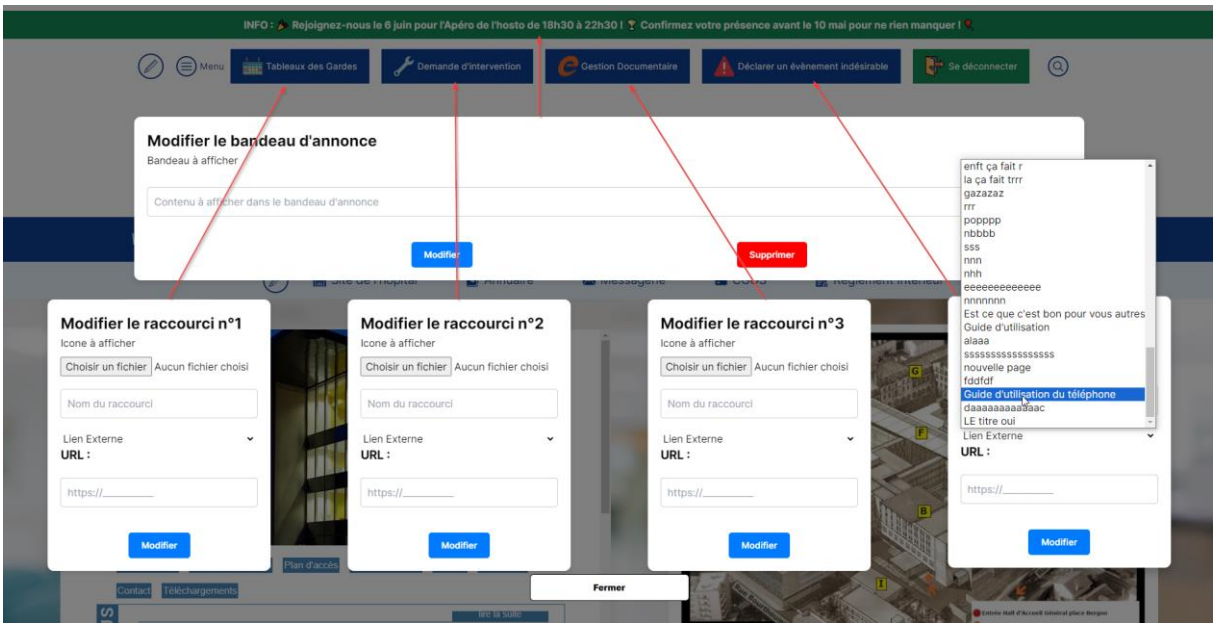


Figure 35 : Modification de la partie haute

Effectivement, il n'y a pas de valeur affichée par défaut, c'est simplement que, par manque de temps, cela n'a pas été fait mais je pense que ce n'est pas très dérangeant. Donc voilà, l'utilisateur peut choisir une image qui sera utilisée comme icône, donner un nom au raccourci, et choisir si lors du clic cela redirigera vers une page externe, ou une interne, parmi la liste de toutes les pages de l'intranet.

Pour les autres raccourcis, nous retrouvons presque la même interface :

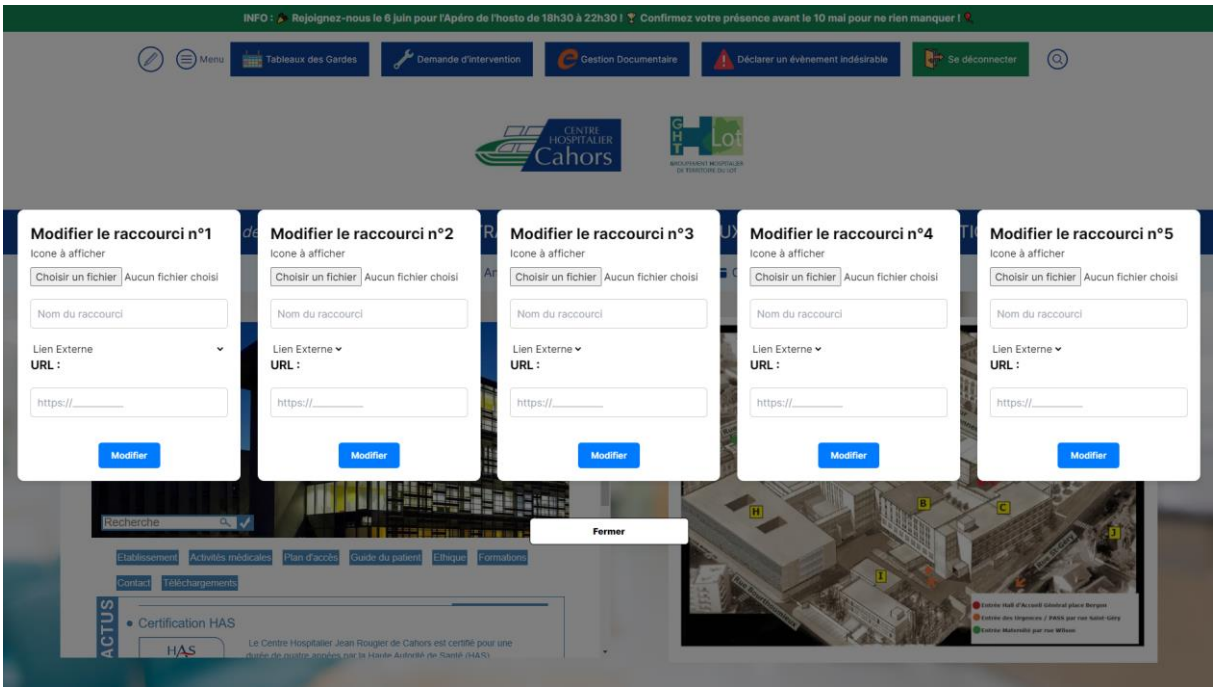


Figure 36 : Modification des autres raccourcis

Après avoir vu cela, nous allons pouvoir apercevoir plusieurs nouvelles fonctionnalités dans le menu. Effectivement, nous allons pouvoir ajouter / modifier / supprimer, des catégories principales / sous-catégories / titres :

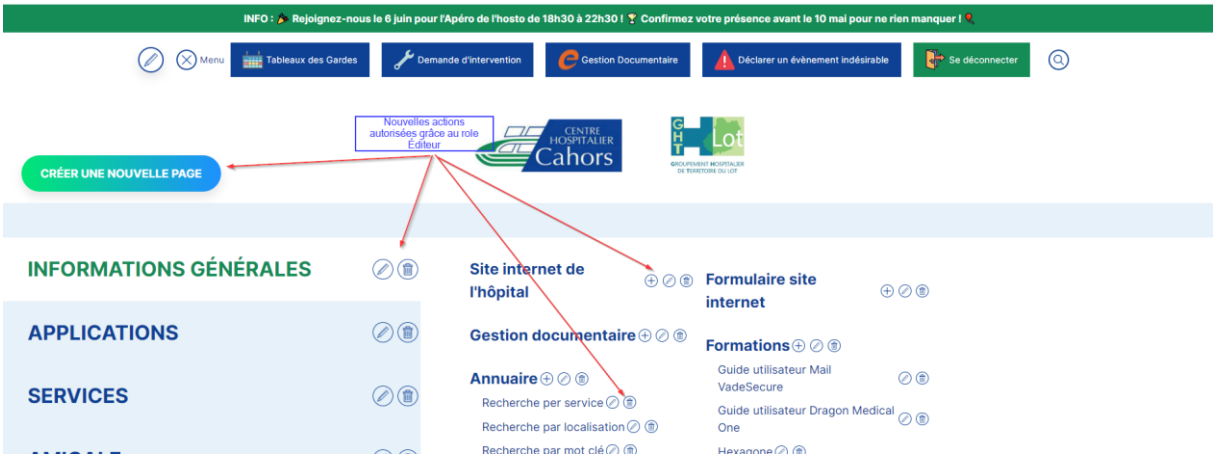


Figure 37 Menu Éditeur connecté

Je vais vous présenter seulement l'interface pour les catégories principales et les sous catégories, car c'est la même interface pour les sous-catégories et les titres.les trois interface de modifications pour chaque type d'information, car on retrouve à chaque fois presque la même interface :

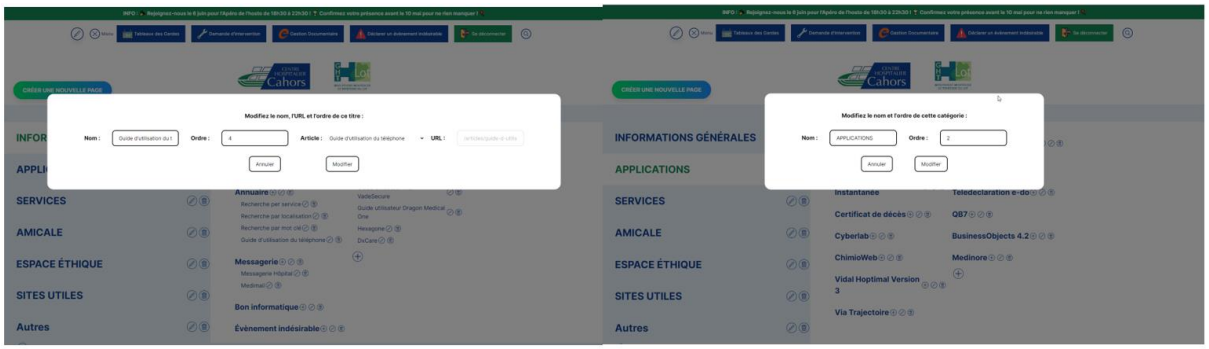


Figure 38 : Actions dans le menu

Pour la suppression, nous retrouvons simplement un message de confirmation :

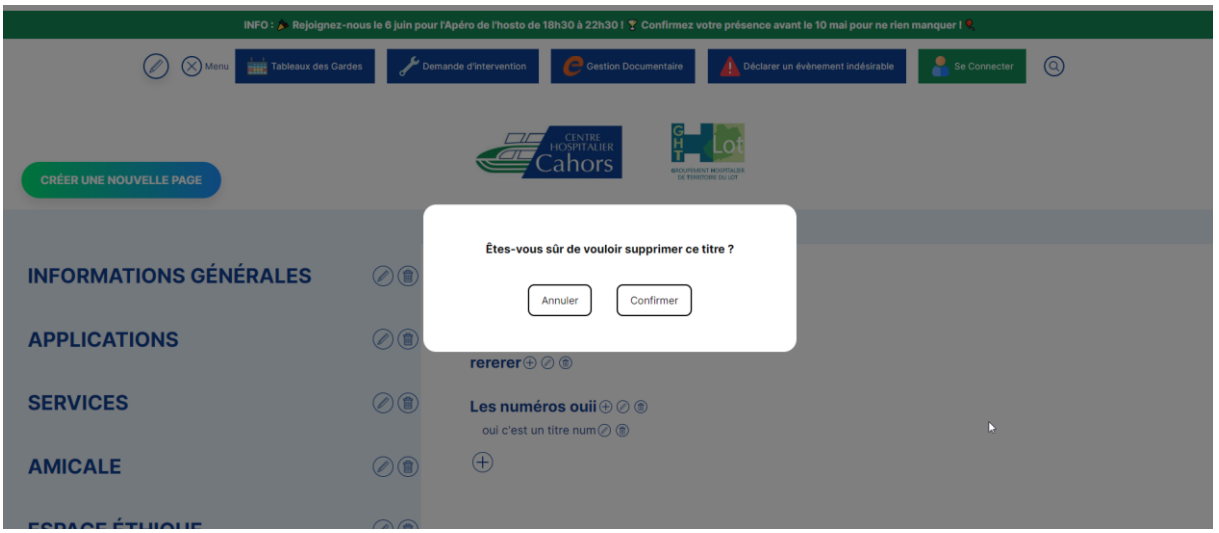


Figure 39 : Suppression dans le menu

A ce stade là, nous pouvons passer à la création de page, accessible depuis le menu comme vu plus tôt. Nous constatons de nombreuses actions possibles :



Figure 40 : Création d'une page

Nous allons commencer à nous intéresser au bouton “Gestion de fichiers”, que nous retrouvons comme lorsque nous consultons une page. Cette fois-ci, nous sommes en mode édition, donc il y a plus de fonctionnalités :

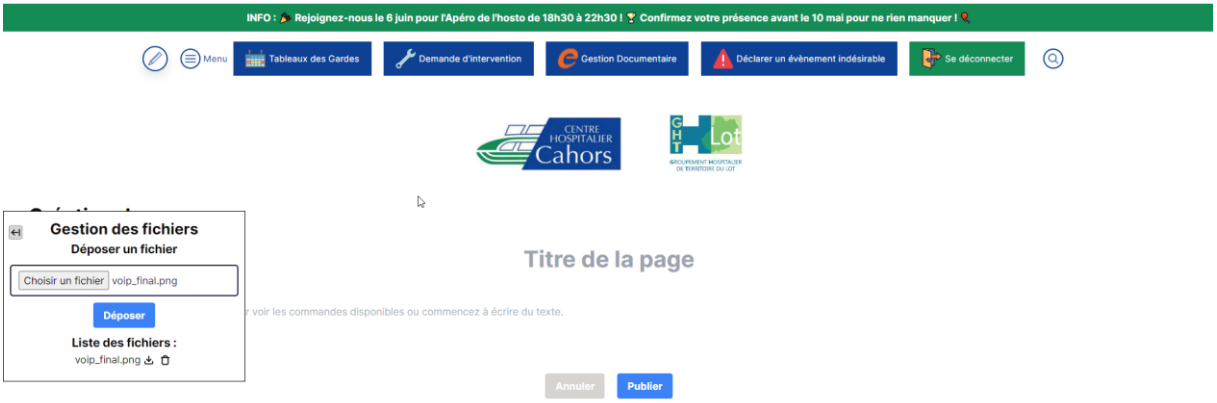


Figure 41: Gestionnaire de fichiers

Effectivement, pour l'exemple ici, j'ai cliqué sur le bouton "choisir un fichier", puis déposer, et mon fichier est apparu en dessous (tout le protocole détaillé et imagé étape par étape en annexe). Nous constatons, qu'en plus de retrouver le bouton pour télécharger, il y a le bouton pour supprimer ce fichier. Ce bouton comme tous les autres de suppression, va nécessiter une confirmation.

Nous allons donc passer à la partie rédaction, et donc pour cette exemple, nous allons commencer par le premier menu, qui, comme indiqué, apparaît quand nous écrivons '/' (pour l'exemple et pour limiter les captures d'écran, j'ai agrandi au maximum en hauteur le menu pour voir toutes les fonctionnalités, mais normalement, il est scrollable) :

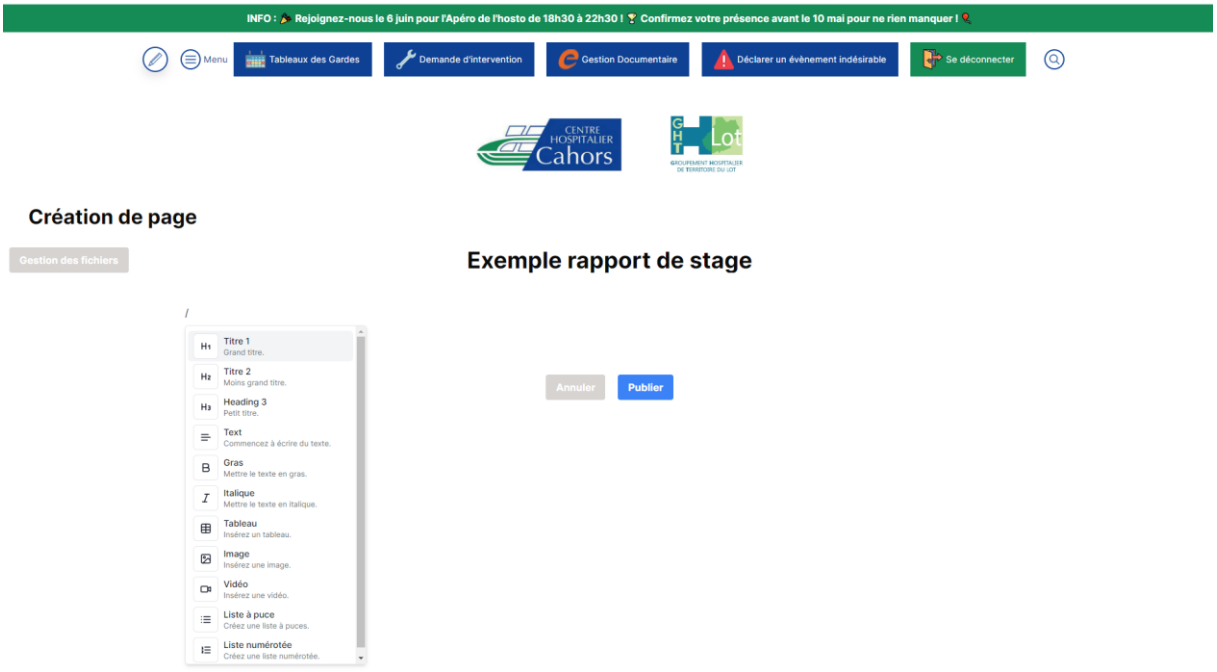


Figure 42 : Menu '/'

On retrouve donc ici toutes les fonctionnalités qui ont pu être intégrées en partie grâce à TipTap. Je vais simplement montrer ce qu'il se passe lorsque nous sélectionnons tableau, image ou vidéo (le reste du détail est dans l'annexe).



Figure 43 : Menu des tableaux

Je viens de sélectionner la fonctionnalité tableau, et nous nous apercevons que du fait que le curseur est dans le tableau, nous avons un petit menu pour pouvoir faire toutes les opérations possibles d'ajout et de suppression.

Pour ce qui est de l'insertion d'image, lorsque nous sélectionnons la fonctionnalité, une première interface s'affiche :

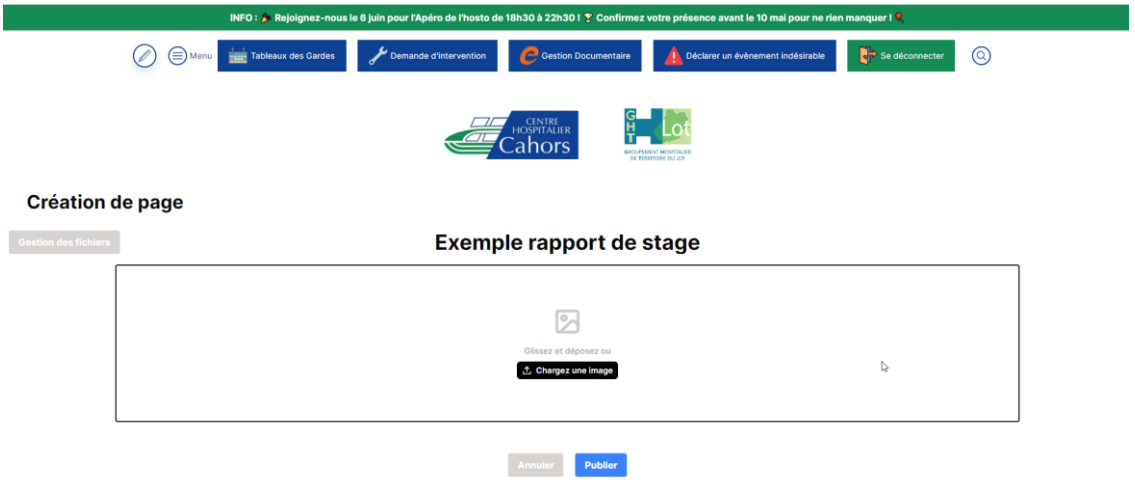


Figure 44 : Interface dépôt d'images

Lorsque nous cliquons, cela nous permet de parcourir notre stockage et de l'ajouter sur le site, donc je vais en ajouter une, qui va commencer à charger puis s'afficher comme ceci :

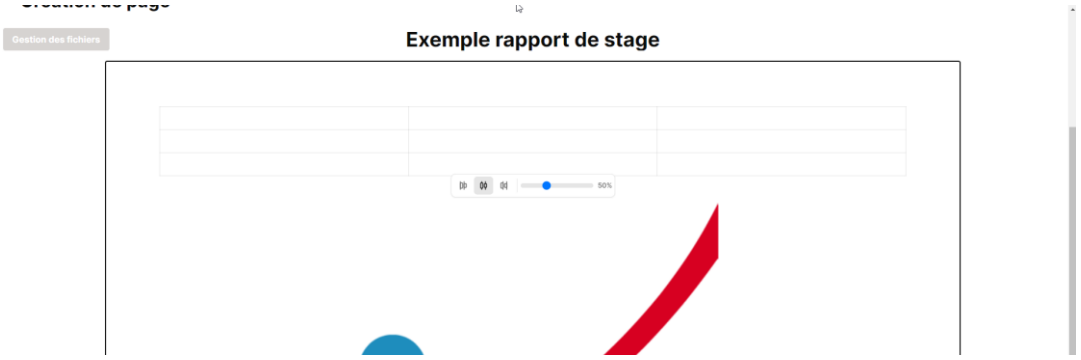


Figure 45 : Menu images

Ici comme pour le tableau, lorsque nous cliquons sur l'image, nous avons un petit menu, qui lui, permet de pouvoir aligner à gauche, au centre ou à droite, et de régler la taille sur 25% , 50% , 75% ou 100%.

Pour ce qui est de la vidéo, nous retrouvons la même interface pour sélectionner notre fichier, puis après le chargement, le lecteur s'affiche. (la vidéo d'exemple est de très mauvaise qualité, aucun lien avec le lecteur)

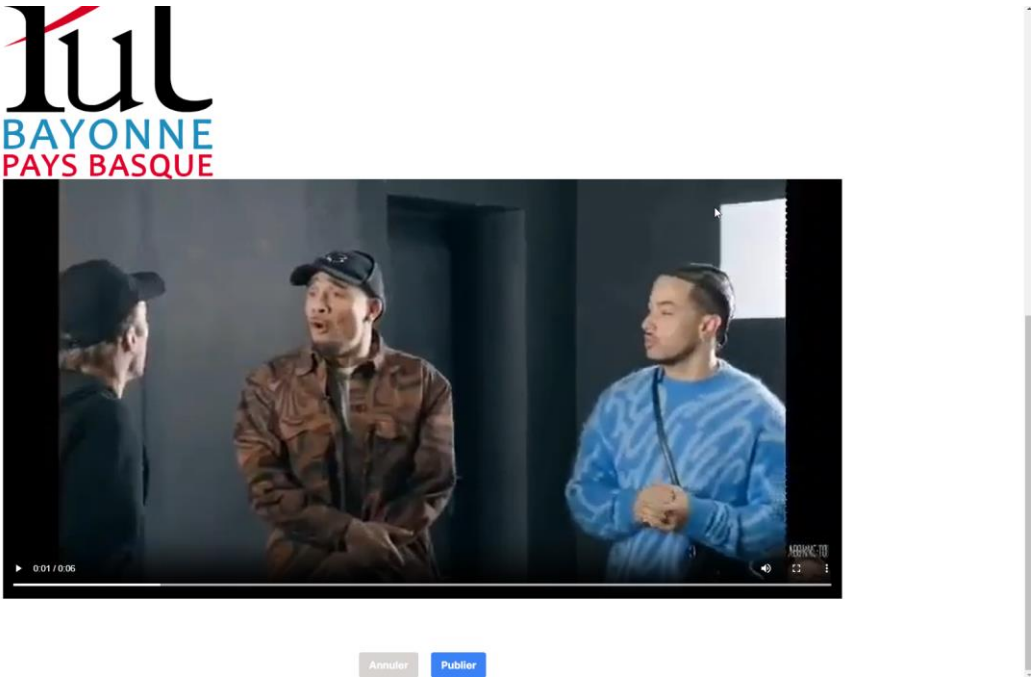


Figure 46 : Lecteur vidéos

Nous apercevons donc grâce à ce lecteur, le contenu du fichier vidéo sélectionné, qui permet de mettre pause / reprendre, avancer la vidéo, gérer le volume, mettre en plein écran, et dans les 3 petits points nous retrouvons, la possibilité de télécharger la vidéo et de la mettre en accéléré.

Nous avons vu tout ce que permettait ce menu, nous allons maintenant voir le deuxième, permettant de définir des choses, pour un certain texte. Pour pouvoir voir ce menu, il faut avoir déjà écrit, puis sélectionné du texte et ensuite, nous pouvons utiliser les fonctionnalités :

Pour l'exemple je vais regrouper chaque sous menu des fonctionnalités pour ne pas avoir à faire 7 captures d'écran.

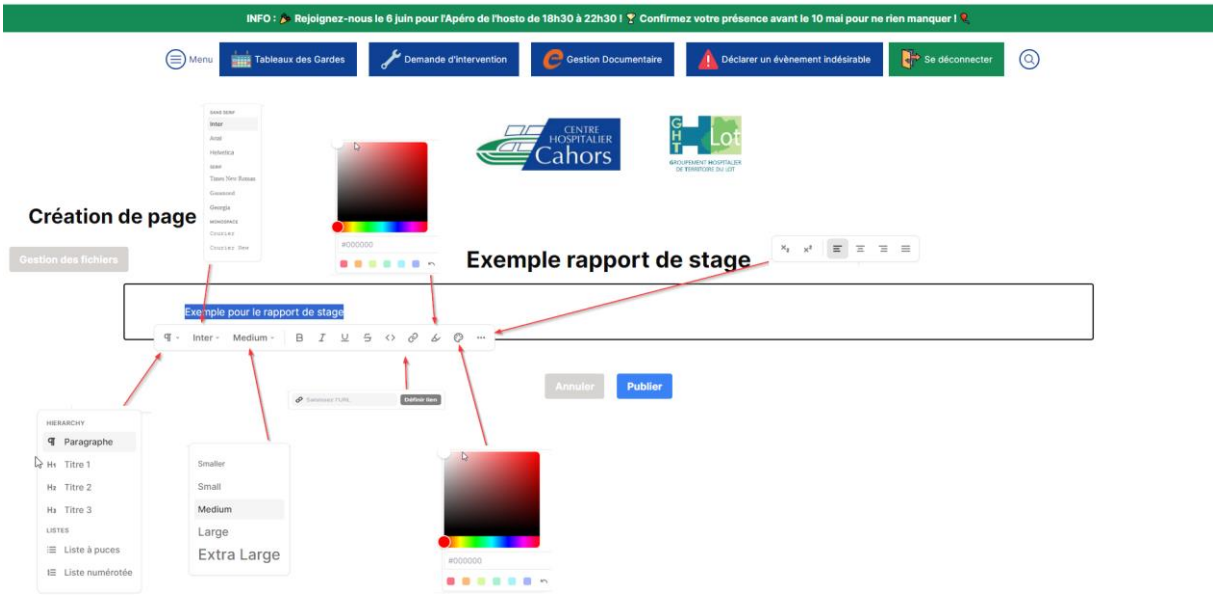


Figure 47 : Menu sélection

Voila tout ce que nous pouvons faire de plus avec ce menu. Ensuite, nous allons pouvoir cliquer sur publier et notre page va directement s'afficher :



Figure 48 : Affichage de la page

Nous avons donc bien notre texte qui s'affiche comme prévu, et en tant qu'éditeur, nous avons la possibilité de modifier cette page, de modifier à nouveau tout le contenu comme avant la publication, ou de supprimer cette page.

Je pense avoir fait le tour pour le rôle Editeur. Il est possible que j'ai oublié des éléments, mais nous passons à présent au rôle Administrateur.

1.3 Le rôle d'Administrateur

Ce rôle, en plus d'avoir toutes les permissions du rôle Éditeur, va pouvoir accéder à un menu d'administration :

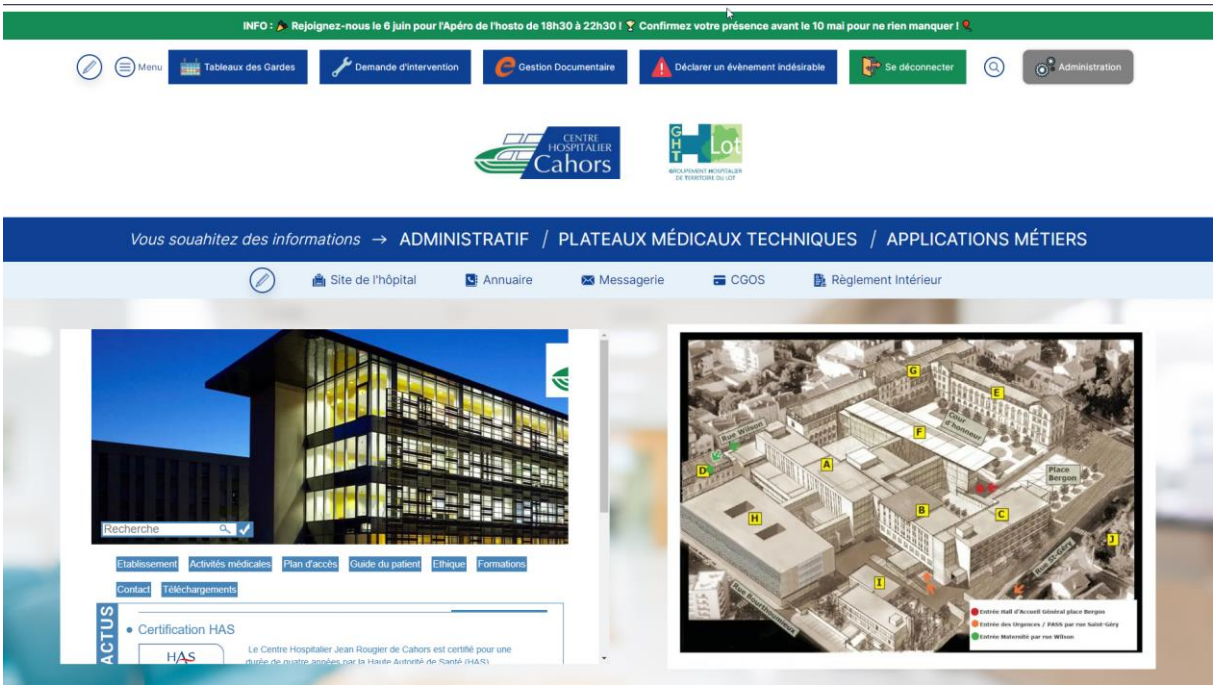


Figure 49 : Accueil Administrateur connecté

Nous retrouvons toutes les fonctionnalités autorisées par le rôle Éditeur, mais nous pouvons également voir en haut à droite un bouton "Administration". Lorsque nous y accédons, nous arrivons sur cette page :

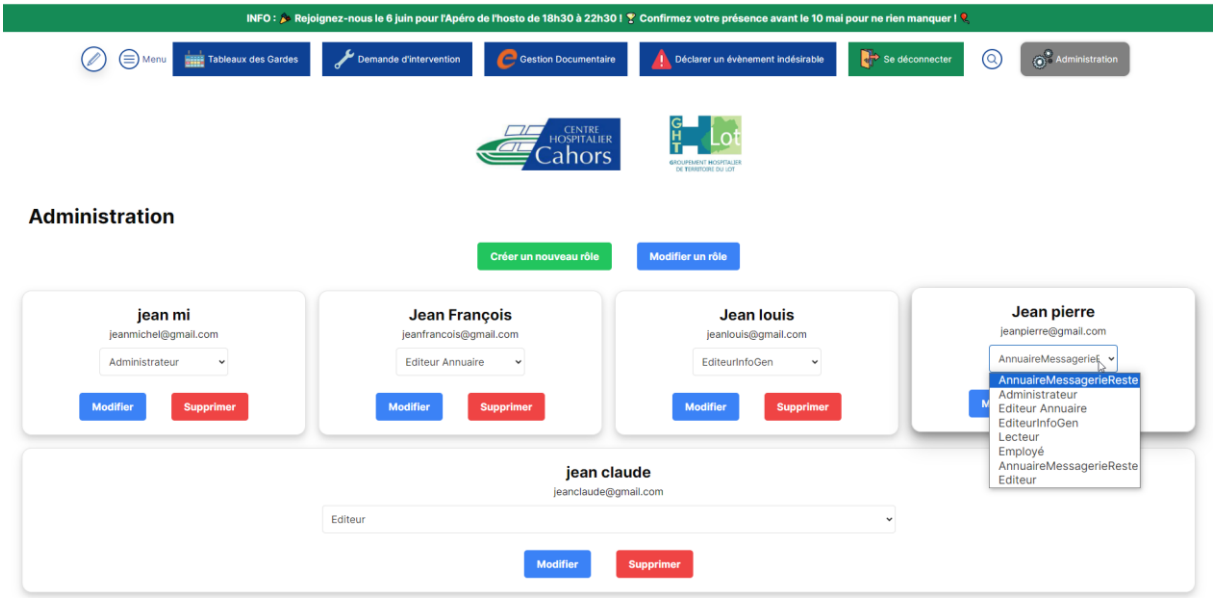


Figure 50 : Page d'administration

Sur cette page, nous observons dans un premier temps, tous les utilisateurs inscrits. Leurs informations sont affichées, et il est possible de définir un rôle à ceux-ci, en le sélectionnant dans la liste, puis en cliquant sur modifier. D'ici, nous pouvons également supprimer les utilisateurs. Grâce aux deux boutons au-dessus, nous pouvons créer et modifier les rôles. En cliquant sur "Créer un rôle", nous allons pouvoir commencer à le définir :



Figure 51: Création de rôles

Nous allons commencer par choisir le nom de notre rôle puis rapidement, choisir ce que nous voulons que notre rôle puisse modifier, ou ne pas voir. Pour rappel, par défaut, j'ai créé un rôle Administrateur, un Employé et un Éditeur. Ce dernier permet de tout modifier, mais si nous voulons limiter les accès, il faut donc créer de nouveaux rôles Editeurs pour permettre de modifier certaines sous-catégories.

Pour ce qui est de la modification, cela va être la même interface, sauf qu'au lieu de noter le nom du nouveau rôle, il va falloir le choisir dans la liste :

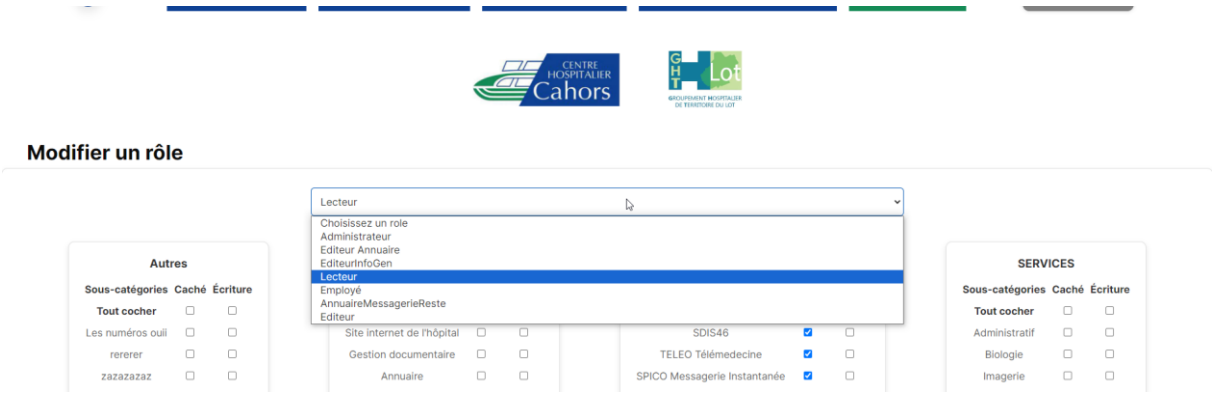


Figure 52 : Modification de rôles

En dessous, nous retrouvons les boutons pour mettre à jour, supprimer et retourner à l'accueil.

Voilà, nous avons terminé le long tour de toutes les fonctionnalités présentes sur l'intranet, nous allons donc passer à la partie correspondant aux fonctionnalités non encore faites, mais pour lesquelles il est probable qu'elles soient mise en place avant la soutenance.

2. Ce qui devra être fait

J'écris cette partie du rapport le vendredi 07/06, donc un peu plus d'une semaine avant le rendu du rapport de stage.

Maintenant, il est temps de parler de ce qu'il me reste à faire avant la soutenance. Pour rappel l'application est actuellement déployée sur le serveur mais accessible par tous les employés en saisissant "/intranet" dans le navigateur. Je vais donc devoir y consacrer un certain temps et je pense que je vais me faire épauler, principalement par mon tuteur, afin de réaliser cette étape, car je n'ai pas réellement confiance en mes compétences de paramétrage de pare-feu Windows entre autres choses. Je ne sais pas trop à l'heure actuelle, mais j'espère que cela pourra se mettre en place rapidement.

Après cela, je devrais créer les pages essentielles, avant de pouvoir faire tester les sites aux employés. Je pourrais continuer d'en rajouter pendant qu'ils testeront, mais il m'en faut un certain nombre de base. Ensuite, je développerai la fonctionnalité qui permettra de mettre à jour les différents éléments utilisés sur le site. Pour finir, je prendrais en compte les retours réalisés par les différents utilisateurs, et adapterai le site en conséquence (dans la limite du raisonnable évidemment)

Voilà pour ce qui devrait être fait, nous allons passer maintenant aux différents éléments que j'aurais aimé rajouter.

3. Ce qu'il aurait été bien de faire

Pour commencer, j'aurais aimé améliorer l'interface pour modifier le site, la rendre plus ergonomique, plus attrayante que ces simples icônes. J'aurais également aimé prendre plus de temps pour la gestion de rôle, car il me semble qu'elle pourrait être bien plus améliorée. Je n'ai pas eu le temps de créer la fonctionnalité de "Mot de passe oublié", un Administrateur doit donc directement supprimer l'utilisateur depuis le menu d'administration. J'aurais aimé rendre le site 100% modulable, en ayant la possibilité de faire du drag & drop, changer les couleurs principales, les logos, et pourquoi pas, créer des modèles utilisables.

Cela aurait pu permettre d'adapter l'intranet à d'autres hôpitaux comme celui de Gourdon. J'ai également plusieurs autres actions que j'aurais aimé mettre en place comme une page d'erreur 404 personnalisée, revoir les interfaces afin de les harmoniser parfaitement, intégrer de l'Intelligence Artificielle, créer un vrai espace

pour définir le profil des utilisateurs ou bien plusieurs autres éléments plus techniques autour de la base de données et l'architecture de fichier.

J'espère réellement que ce projet pourra être terminé proprement, car je sais que s'il y a des problèmes plus tard, le projet sera rapidement mis à la poubelle.

VI. Bilan

Pour conclure ce rapport je pense pouvoir dire que je n'aurais pas pu rêver mieux comme stage de fin d'étude. Effectivement, celui-ci m'a permis de pouvoir appliquer toutes les compétences que j'ai pu acquérir durant ces trois dernières années. J'ai donc pu constater que j'étais prêt à rentrer très sérieusement dans le monde professionnel. Je considère que toutes les compétences de cette troisième année de BUT en parcours A (Réalisation d'applications : conception, développement, validation), ont été vérifiées durant ce projet. Je vais donc justifier cette affirmation à travers différents éléments.

Par rapport à la compétence « C1 : Réaliser un développement d'application », je considère la compétence validée du fait que le projet que j'ai pu réaliser, contient une grande majorité de fonctionnalités que j'ai eu à développer moi-même. Celles que je n'ai pas eu à développer nécessitait pour la plupart des adaptations, et je considère que l'intégration de fonctionnalités est une tâche qui fait partie du quotidien de tout développeur. Pour ce qui est de la 3ème partie de cette compétence : Adapter des applications sur un ensemble de supports, effectivement, pour ce stage, l'objectif étant de créer un intranet, le seul support était donc un ordinateur. Je considère que le projet réalisé cette année pour la saé étant une application mobile sous Android et Ios, au final, cela fait un ensemble assez homogène.

Concernant la compétence « C2 : Optimiser des applications », j'ai eu à de nombreuses reprises à optimiser le site, car du fait que la plupart des données sont stockées dans une base de données, on peut rapidement surcharger la BD de requêtes. C'est pour cela qu'il n'a pas était évident, de bien évaluer quand et où appeler les requêtes, afin de limiter le surplus et les ralentissements.

Pour finir avec la compétence « C6 : Collaborer au sein d'une équipe informatique », cette compétence a été développée de manière assez particulière, du fait que j'étais le seul développeur de cette équipe. Au final, mon tuteur avait le rôle de chef de projet, car c'est à lui que j'ai pu présenter mon organisation, l'avancement, poser des questions, laisser faire les choix importants comme par exemple le choix du style global du site. Je considère que nous avons collaboré de la meilleure des manières, en partie grâce aux outils utilisés pour montrer l'avancement, mais également car nous nous entendions très bien.

Je peux affirmer que ce stage a été très bénéfique pour moi. Je me rends compte de la chance que j'ai pu avoir, de travailler sur un projet ambitieux comme

celui-ci, le tout sans avoir été mis sous pression durant ces quatorze semaines. Je considère avoir fait mon maximum avec le temps que j'avais à ma disposition, et je pense que dans tous les cas, les compétences de cette troisième année de BUT Informatique sont validées.

VII. Résumé en anglais

I did my internship in the Hospital Center of Cahors. The hospital is the biggest employer of the city with more than 1200 employees and 750 Computers. In the IT service there are 10 employees that are working for different things like the maintenance of the applications, or the support.

During this internship, my mission was to create a new version of the current intranet, which allows the hospital's employers to find information, like the phone number of doctors in specific service. The principal difficulty is to allow some people to edit the website, without developpement skills. To do it, I had two different choices possible. The first one was to start from the blank. The second one consists of using a tool like a CMS (Content Management System), to have a solid basis, and then, develop different modules. But this option was not really interesting. This, because I don't want to develop modules for CMS like WordPress in the future. But for the first, the problem was that I had only fourteen weeks to finish this project. So I decided to take this option, but with a doubt about the timing.

So for this project, I had to do absolutely everything, so I started with a phase of analysis, which allows me to understand the needs for this site. Then, I started to find ideas for the look of the final website. After this I started to design the final ones.

My tutor validated this, so I continued with the development phase. That was not really easy, because the principle was to allow people to edit the website, so when I would add a functionality, I immediately had to think about the way I would do it to make this updatable. This, for approximately like 90% of the elements in the page.

So the main functionality was to create pages, and add them to a menu to access them easily. To do this, I had to integrate a tool which could allow me to transform normal text, in title, or bold, or insert image. So I found TipTap, which was not really easy to integrate. All the functionalities needed were not already inside it, so I had to develop them.

I also add to manage roles, to allow people to update things, or not, to be administrator, or not. So this took me a lot of time, but it was really important to escape as many problems as possible.

Currently, the project is not hundred percent finished, there are things that I will do during the last weeks. I have to finish the deployment, let users try the intranet and update according to the feedback. During this last period, I also have to add a functionality, to allow administrators to update modules used, to have a durable tool.

So to conclude, I can say that I couldn’t imagine a project better than this one, because of the fact that I had the liberty to choose the technologies that I liked, and to do this alone (but helped if I needed). With this project, I have seen all the parts of a project, so thanks to that, I saw what I liked to do and what I didn’t.

VIII. Glossaire

Liens morts: Liens redirigeant vers une page qui n'existe plus, ou ne fonctionne plus.

Étude de l’existant: Phase d'analyse consistant à faire un état des lieux des éléments existants actuellement

Veille concurrentielle: Procédé consistant à rechercher, se renseigner, de manière à voir comment certains concurrents réalisent différentes choses. Cette partie permet de s'inspirer, mais également de comparer en quoi un projet est plus intéressant qu'un autre.

Templates: Traduit littéralement par modèle, différent d'un exemple car ne contient pas d'information

Microsoft SharePoint: Série de logiciels créé par Microsoft, utilisé principalement par les entreprises, qui permettent d'utiliser des outils comme la gestion de forums, de formulaire, ou bien de créer un portail web facilement grâce à un CMS et des templates fournies

Wireframe: Peut être traduit par esquisse, maquette fonctionnelle, armature, ou directement maquette fil-de-fer, il permet de donner un aperçu de la structure d'une maquette, mais sans afficher de contenu.

JavaScript (fichier .js): Langage principalement utilisé pour du développement web, orienté objet

TypeScript (fichier .ts): Langage permettant de définir des type dans JavaScript

JSX: Format créé par réact, pour utiliser de la syntaxe JSX, correspondant à de l'HTML, dans un fichier .js (javascript)

TSX: Format utilisé pour signaler qu'un fichier en TypeScript contient de la syntaxe JSX

BOOL: Booléen, type de variable pouvant indiquer Vrai ou faux, (true, false)

Clé primaire: Contrainte en base de données représentant la donnée permettant d'identifier de manière unique un enregistrement dans une table

Clé étrangère: Contrainte dans une base de données, permettant d'identifier une colonne d'une autre table, qui doit être elle même une clé primaire (A VERIFIER)

Tutoriel: Explication démontrée, afin de mieux comprendre des éléments

Hasher: Différent de chiffrer, mais permet de sécuriser des données comme un mot de passe afin de ne pas stocker directement le mot de passe d'un utilisateur. Un mot de passé hashé ne peux pas être “déhashé” comme un mot de passe crypté, qui lui peut être décrypté

Token: permet de sécuriser une connexion en contenant des informations comme un rôle afin de vérifier l'accès à des informations

JWT (JSON web token): Reprenant le principe de token mais structuré différemment afin de limiter les vérifications, donc plus efficace et plus scalable.

Scalabilité: Une application est dite scalable si elle supporte presque aussi bien 5 utilisateurs connectés que 50

Différence token et session: Différence principale est que l'information de connexion est gérée côté serveur pour les sessions et client pour le token (donc plus scalable).

Indice (d'une liste): Correspond à la position d'un élément dans une liste -1. Car le premier élément à 0 comme indice, donc le 12ème à 11 comme indice etc

Pattern: Traduit littéralement par “modèle”, correspond à une certaine manière de faire les choses. Donc lorsque je dis : “Toujours le même pattern”, cela signifie que je me retrouve à devoir refaire encore la même chose, en suivant les mêmes étapes, mais pour un autre contenu.

IX. Bibliographie

- Site internet de l'hôpital de Cahors (ch-cahors.fr)
- Wikipedia pour définir différents éléments comme la suite de logiciel Microsoft Sharepoint (fr.wikipedia.org)
- Les templates de Microsoft SharePoint (lookbook.microsoft.com)
- Comparaison NextJS à React (radixweb.com/blog/nextjs-vs-react)
- Explication très simple de la différence entre un token et un JWT (grafikart.fr/tutoriels/json-web-token-presentation-958)
- Youtube pour des explications en vidéos (youtube.com)
- StackOverflow pour des explications et problèmes plutôt techniques (stackoverflow.com)
- GitHub pour croiser les réponses avec StackOverflow
- HubVisory pour le schéma réprésentant le fonctionnement d'une API (hubvisory.com)*

X. Annexes

- Manuel d'utilisation :