

RAPPORT DE PROJET

Corentin **SAN JUAN**

BTS SIO SLAM - 2023



BADG-HEURE

SOMMAIRE

- 01** Contexte
- 02** Expression du ou des besoins
- 03** Les objectifs
- 04** Analyse fonctionnelle
- 05** Tests
- 06** Conclusion



CONTEXTE

Description du laboratoire GSB

Le secteur d'activité :

L'industrie pharmaceutique est un secteur très lucratif dans lequel le mouvement de fusion acquisition est très fort. Les regroupements de laboratoires ces dernières années ont donné naissance à des entités gigantesques au sein desquelles le travail est longtemps resté organisé selon les anciennes structures. Des déboires divers récents autour de médicaments ou molécules ayant entraîné des complications médicales ont fait s'élever des voix contre une partie de l'activité des laboratoires : la visite médicale, réputée être le lieu d'arrangements entre l'industrie et les praticiens, et tout du moins un terrain d'influence opaque.

L'entreprise :

Le laboratoire Galaxy Swiss Bourdin (GSB) est issu de la fusion entre le géant américain Galaxy (spécialisé dans le secteur des maladies virales dont le SIDA et les hépatites) et le conglomérat européen Swiss Bourdin (travaillant sur des médicaments plus conventionnels), lui-même déjà union de trois petits laboratoires. En 2009, les deux géants pharmaceutiques ont uni leurs forces pour créer un leader de ce secteur industriel. L'entité Galaxy Swiss Bourdin Europe a établi son siège administratif à Paris. Le siège social de la multinationale est situé à Philadelphie, Pennsylvanie, aux Etats-Unis. La France a été choisie comme témoin pour l'amélioration du suivi de l'activité de visite.

Réorganisation :

Une conséquence de cette fusion, est la recherche d'une optimisation de l'activité du groupe ainsi constitué en réalisant des économies d'échelle dans la production et la distribution des médicaments (en passant par une nécessaire restructuration et vague de licenciement), tout en prenant le meilleur des deux laboratoires sur les produits concurrents. L'entreprise compte 480 visiteurs médicaux en France métropolitaine (Corse comprise), et 60 dans les départements et territoires d'outre-mer. Les territoires sont répartis en 6 secteurs géographiques (Paris-Centre, Sud, Nord, Ouest, Est, DTOM Caraïbes-Amériques, DTOM Asie-Afrique).



Problèmes du service des Ressources Humaines

Dans un premier temps, le service des Ressources Humaines a constaté des cas où certains employés ne respectaient pas leurs horaires de travail. À cette époque, une relation de confiance existait entre les employés et le service des Ressources Humaines. Les employés étaient simplement tenus d'informer le service RH s'ils avaient effectué des heures supplémentaires ou non. Ils n'étaient donc pas obligés de signaler leurs heures de présence "normales" et pouvaient ainsi travailler moins d'heures que ce qui était stipulé dans leur contrat. En conséquence, le service RH a décidé de mettre en place un moyen de tracer de manière plus précise les heures de travail des employés.

Dans un second temps, le service RH a identifié un besoin d'améliorer la communication avec les employés. Par exemple, la distribution des tickets restaurants se faisait de manière informelle, par le biais du bouche-à-oreille. Cela rendait difficile la communication officielle de la date de distribution.



EXPRESSION DES BESOINS

- Permettre aux utilisateurs de pointer leurs heures d'entrées et sortie
- Proposer aux utilisateurs une page de rétrospection afin d'avoir un aperçu des heures d'entrées sorties
- Permettre la diffusion de "News" afin de permettre une meilleure communication
- Permettre la connexion via des identifiants uniques
- Avoir la possibilité d'avoir des profils "d'administration" pour les employés RH.



LES OBJECTIFS

Grâce à ce projet, les employés du service des Ressources Humaines pourront facilement enregistrer et suivre les horaires de présence des autres employés. Cela leur permettra d'avoir une vue d'ensemble précise des horaires de travail et des absences, facilitant ainsi la gestion des ressources humaines au sein de l'entreprise.

De plus, ce projet offrira aux employés du service des Ressources Humaines un système de communication efficace. Ils pourront partager des informations importantes de manière simple et conviviale, tout en garantissant la fiabilité et la sécurité des échanges. Cela favorisera une meilleure collaboration au sein du service et une diffusion rapide des informations pertinentes.



ANALYSE FONCTIONNELLE

Cette Application Web présente plusieurs fonctionnalités :

- Connexion facile grâce à une combinaison identifiant/mot de passe.
- Interface différenciée entre les utilisateurs et les administrateurs.
- Interface simplifiée, rapide, optimisée et agréable.
- Barre de navigation déployante pour une meilleure navigation.
- Animations conçues pour être lues à une fréquence minimale de 60 images par seconde sur tous les navigateurs et machines.
- Architecture de site simple pour une utilisation intuitive.
- Possibilité de signaler les entrées/sorties et de les visualiser à l'aide d'un calendrier.
- Diffusion d'informations provenant des Ressources Humaines sur l'écran d'accueil.
- Déconnexion simple et rapide grâce à un bouton situé dans la barre de navigation.
- Visualisation facile des informations de compte.
- Gestion simple des utilisateurs pour les administrateurs.
- Possibilité pour les administrateurs de visualiser le calendrier d'entrées/sorties de l'employé sélectionné.

Langages, technologies et outils utilisés :

- MERN Stack (MongoDB, Express, React, Node.js)
- Vite & React TypeScript
- Axios
- JsonWebToken
- Mongoose
- SASS
- Tailwind
- Dotenv
- React Router DOM
- Git & Github & Visual Studio Code

Liens des répertoires Github :

- API : https://github.com/Crtnsj/BADG-HEURE_api
- Front : https://github.com/Crtnsj/BADG-HEURE_front



Choix des technologies :

- **MongoDB (Base de données) :** MongoDB a été choisi comme base de données pour cette application web en raison de sa nature orientée documents et de sa flexibilité. MongoDB permet de stocker des données sous forme de documents JSON, ce qui facilite la manipulation et la gestion des données. De plus, sa scalabilité horizontale et sa capacité à gérer de gros volumes de données en font un choix adapté pour une application nécessitant un suivi des entrées/sorties des employés.
- **Node.js :** Node.js a été utilisé comme framework backend en raison de sa rapidité et de sa capacité à gérer un grand nombre de connexions simultanées. Cela est particulièrement utile pour une application qui doit gérer les interactions en temps réel des utilisateurs, telles que le signalement des entrées/sorties et la diffusion d'informations sur l'écran d'accueil.
- **Vite & React avec TypeScript :** Vite a été choisi comme outil de développement rapide pour le frontend, combiné avec React et TypeScript. Vite offre des performances de développement améliorées grâce à sa capacité à démarrer rapidement et à recharger les modules de manière efficace. L'utilisation de React avec TypeScript permet de développer une interface utilisateur robuste et maintenable, en offrant des fonctionnalités avancées de typage statique et une meilleure lisibilité du code.
- **Express :** Express a été utilisé comme framework pour la gestion des routes et des requêtes HTTP côté serveur. Il est connu pour sa simplicité et sa flexibilité, ce qui en fait un choix populaire pour le développement d'applications web. Express facilite la création d'API RESTful et la gestion des demandes et des réponses.
- **SASS et Tailwind :** SASS a été choisi comme préprocesseur CSS pour sa puissance et sa facilité d'utilisation. Il permet d'écrire du code CSS plus propre et structuré, en offrant des fonctionnalités telles que les variables, les mixins et les fonctions. Tailwind a été utilisé comme framework CSS pour sa modularité et sa facilité de personnalisation. Il fournit une approche basée sur les classes pour le stylisme, offrant ainsi une plus grande flexibilité dans la conception de l'interface utilisateur.
- **Communication avec l'API :** Axios a été utilisé pour la communication avec l'API. Il s'agit d'une bibliothèque JavaScript populaire et facile à utiliser pour effectuer des requêtes HTTP. Axios offre des fonctionnalités telles que la gestion des erreurs, les intercepteurs et la facilité d'utilisation des méthodes HTTP, ce qui simplifie la communication entre le frontend et le backend de l'application.

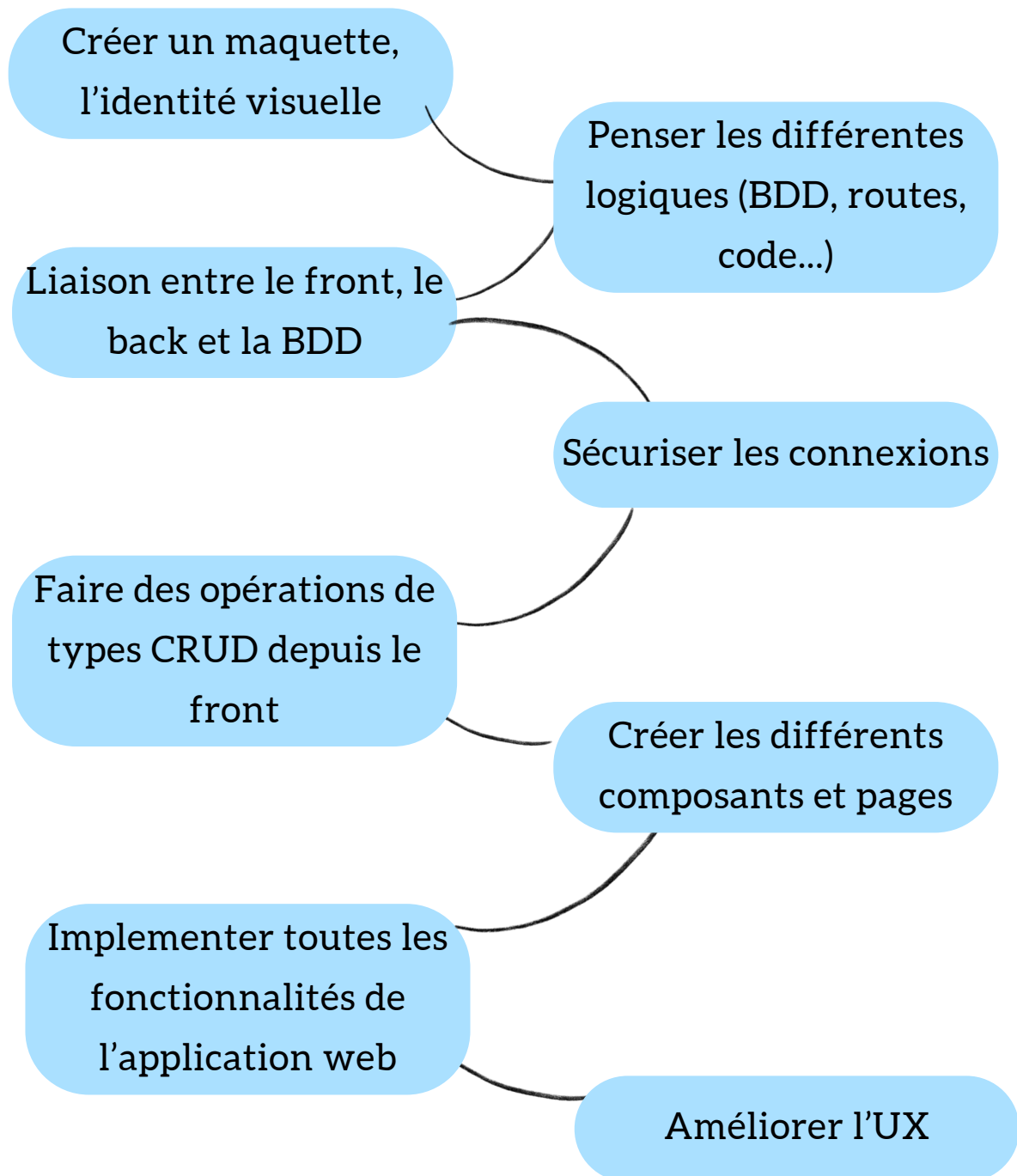


- **Gestion des sessions utilisateur** : JsonWebToken (JWT) a été choisi pour la gestion des sessions utilisateur. Il s'agit d'un standard ouvert qui permet de sécuriser les échanges d'informations entre les parties. JWT est utilisé pour générer des jetons d'authentification qui sont envoyés avec chaque requête pour vérifier l'identité de l'utilisateur. Cela permet de sécuriser les fonctionnalités sensibles de l'application, telles que la gestion des utilisateurs et les autorisations.
- **Gestion de BDD** : Mongoose est un ODM (Object-Document Mapping) conçu pour travailler avec MongoDB. Il facilite l'interaction avec la base de données MongoDB en fournissant des fonctionnalités de modélisation des données, de validation et de requêtes simplifiées. Mongoose permet de définir des schémas de données et d'effectuer des opérations CRUD (Création, Lecture, Mise à jour, Suppression) de manière plus intuitive.
- **Gestion des routes** : React Router DOM est une bibliothèque de routage pour les applications React. Elle permet de gérer la navigation entre les différentes vues de l'application en utilisant des routes définies. Cela facilite le développement d'une application à navigation fluide, en permettant aux utilisateurs d'accéder aux différentes fonctionnalités de manière structurée et intuitive.
- **Contrôle de version et collaboration** : Git & GitHub , Git a été utilisé comme système de contrôle de version pour le développement de l'application. Il permet de suivre les modifications du code source, de gérer les branches de développement et de collaborer efficacement avec d'autres développeurs. GitHub a été utilisé comme plateforme de gestion de projets et de collaboration, offrant des fonctionnalités telles que le suivi des problèmes, les demandes de fusion (pull requests) et la gestion des versions.
- **IDE** : Visual Studio Code a été utilisé comme environnement de développement intégré (IDE) pour la création de l'application. Il offre une interface conviviale, des fonctionnalités avancées d'édition de code, des extensions pour différentes technologies et une intégration étroite avec Git, ce qui facilite le développement et la collaboration sur le projet.

Ces choix technologiques ont été faits pour assurer une expérience utilisateur fluide, une architecture simple et une gestion efficace des fonctionnalités telles que la connexion, le suivi des entrées/sorties, la diffusion d'informations RH et la gestion des utilisateurs.



Les grandes étapes à réaliser :

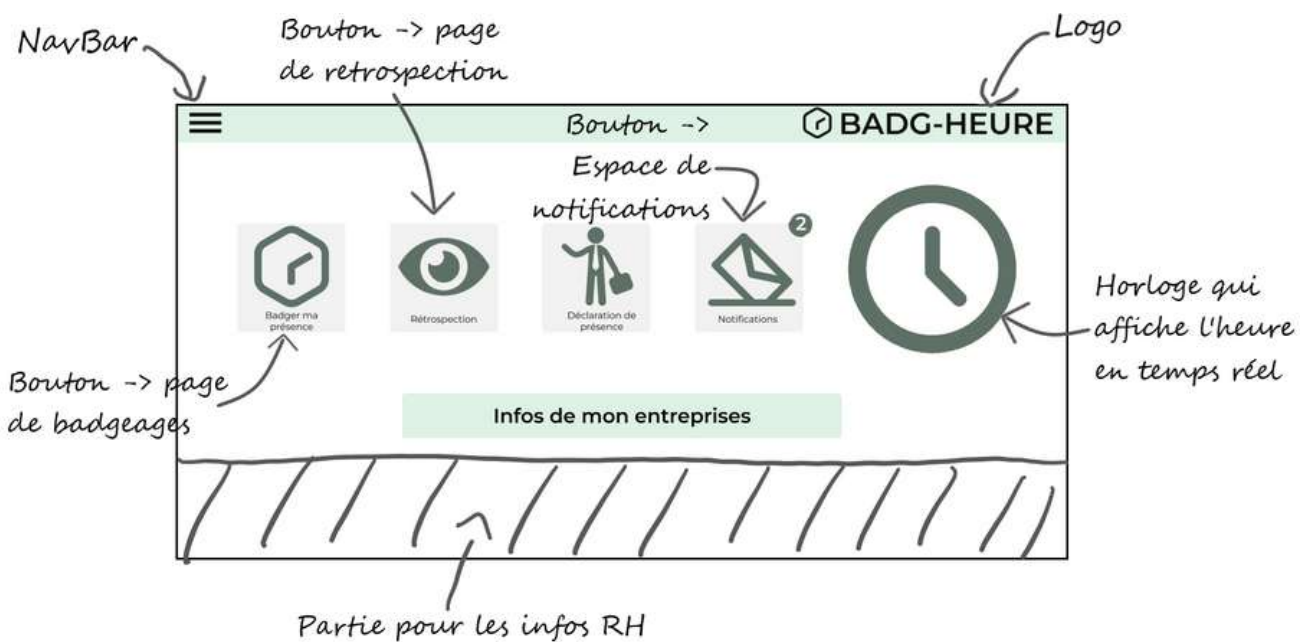


RÉALISATION DES GRANDES ÉTAPES

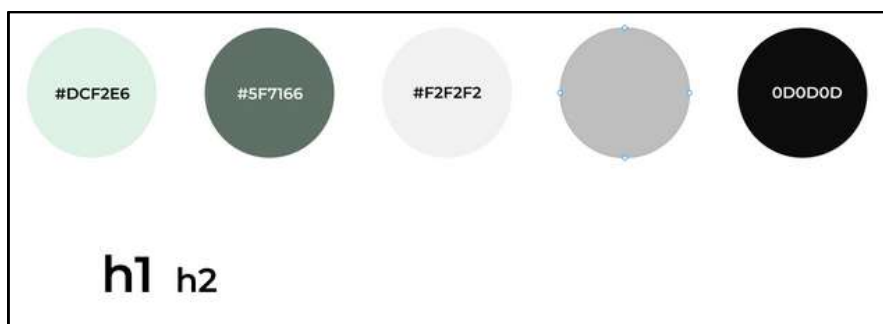
Création de la maquettes et de l'identité visuelle :

Pour créer la maquette, j'ai utilisé Figma. Figma est un outil convivial de conception d'interfaces utilisateur largement utilisé par les web designers. Grâce à son interface intuitive et ses fonctionnalités de collaboration en temps réel, il facilite la création d'interfaces attrayantes et la collaboration efficace au sein d'une équipe. De plus, Figma offre des outils de conception polyvalents et permet d'accéder à la maquette depuis n'importe quel endroit, favorisant ainsi la flexibilité et la mobilité lors du processus de conception. Une autre fonctionnalité intéressante de Figma est la possibilité de créer des bibliothèques de composants réutilisables, ce qui permet d'accélérer le processus de conception en utilisant des éléments prédéfinis.

Ci-dessous, je vais commenter un extrait de ma maquette :



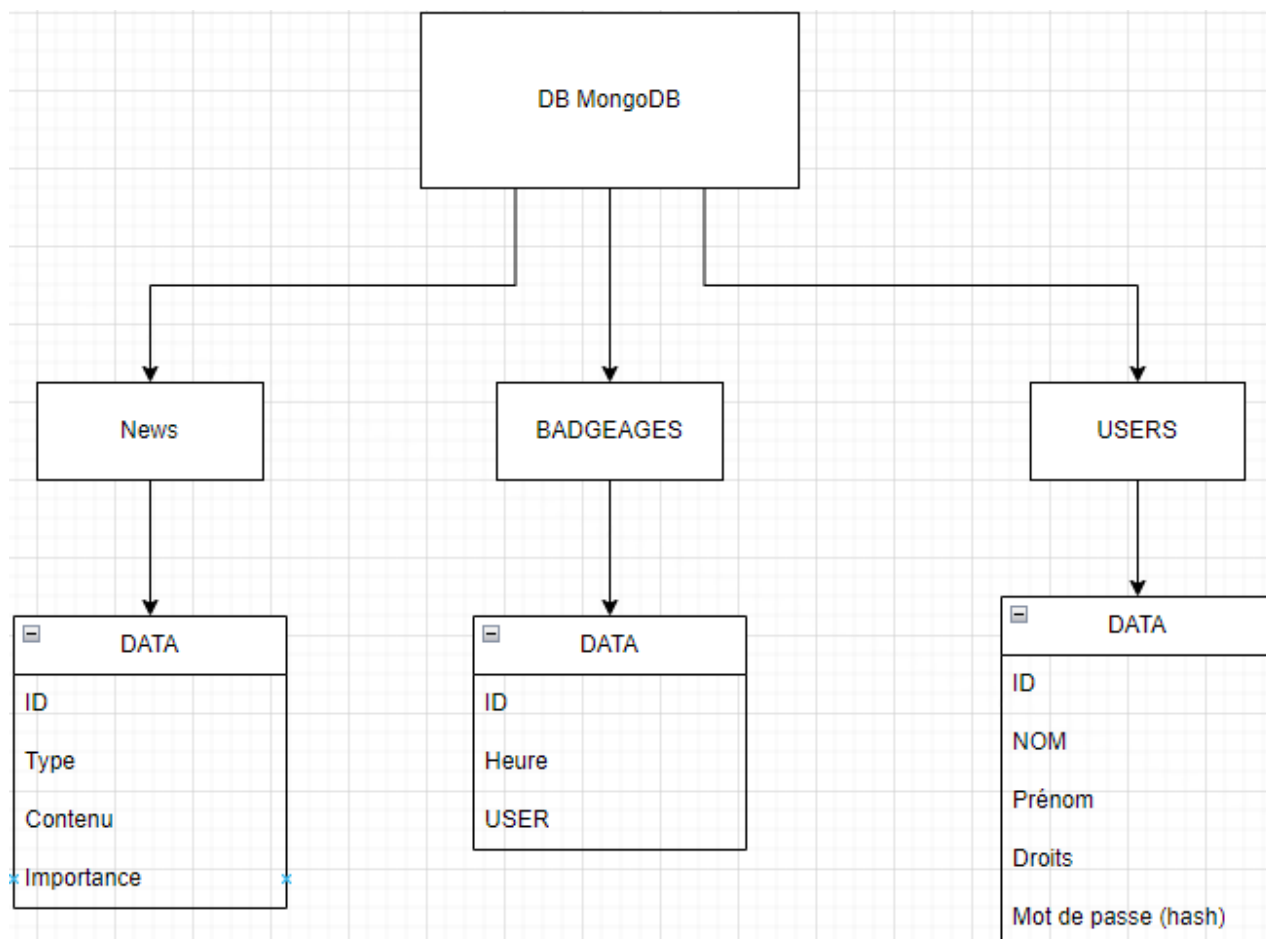
Avec Figma, j'ai également créé une identité visuelle. C'est à cet endroit que j'ai défini une première palette de couleurs et une police.



Penser les différentes logiques (BDD, routes, code...)

Avant d'écrire le code, j'ai réalisé des schémas afin de déterminer comment organiser mon code, ma base de données et ma logique de routage.

Voici un exemple de mon schéma pour comprendre la structure de ma base de données :



Je n'ai pas pu inclure mes autres schémas dans ce rapport car ils étaient trop volumineux pour être ajoutés.

Liaison entre le front, le serveur et la BDD

Pour rappel la dépendance choisie pour la liaison en le front et le serveur est **AXIOS**.

Voici un exemple de lignes de code servant à la communication entre les deux :

```
const fetchData = async (ID: string) => {
  const response = await axios.get<AccountData>(`http://localhost:3002/account/view/${ID}`, {
    headers: { Authorization: `Bearer ${localStorage.getItem('JWT')}` },
  });
  setAccountData(response.data);
};
```



Cette fonction fléchée asynchrone utilise Axios avec une requête de type GET (définie par le ".get"). La requête est envoyée à l'URL: "http://localhost:3002/account/view/\${ID}" où "\${ID}" correspond à l'ID de l'utilisateur passé en paramètre de la fonction. Cette requête comprend également un en-tête contenant le jeton de session de l'utilisateur. Ce jeton est stocké dans le "localStorage" sous le nom de JWT.

La connexion entre le serveur et la base de données est établie grâce à la dépendance Mongoose. Voici un extrait de code correspondant à la connexion :

```
//connexion à la base de donnée
connect(
  `mongodb+srv://${process.env.ACCESS_DB}@cluster0.qh5wykm.mongodb.net/BADG-HEURE?retryWrites=true&w=majority`,
  {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  }
)
```

Dans ce snippet nous pouvons retrouver la méthode "connect" de mongoose. Cette méthode sert à créer la connexion avec la base de donnée, elle se compose de cette manière : `connect(uri, options);` (source : documentation officiel de mongoose)

Dans l'exemple ci-dessus l'URI, en vert, est coupée par la variable d'environnement de la dépendance DOTENV qui correspond aux identifiants de la base de donnée.

Sécuriser les connexions

La sécurisation entre le front et le back-end est réalisée grâce à la dépendance Jsonwebtoken. Cette dépendance est utilisée pour générer des JSON Web Tokens (JWT). Le JSON Web Token est un standard ouvert (RFC 7519) utilisé pour l'échange sécurisé d'informations sous forme de jetons entre des parties de confiance. Il est couramment employé dans les systèmes d'authentification et d'autorisation pour sécuriser les communications entre un client et un serveur.

Un JWT est composé de trois parties distinctes, séparées par des points (".") :

- Header : Il contient des informations sur le type de token et l'algorithme de cryptage utilisé.
- Payload : Il contient des informations supplémentaires, telles que l'identité de l'utilisateur et les autorisations accordées. Il peut également contenir des données non sensibles.
- Signature : Elle est utilisée pour vérifier l'intégrité du token. La signature est générée à l'aide de la clé secrète du serveur, ce qui permet de vérifier si le token a été modifié ou falsifié.



Lorsqu'un client envoie une requête au serveur avec un JWT, le serveur peut vérifier la validité du token en vérifiant sa signature et en s'assurant que les informations qu'il contient sont valides et autorisées.

Lorsqu'un utilisateur se connecte pour la première fois à mon application web la requête est envoyée à l'URL "http://localhost:3002/login/signIn/". Après avoir comparé et vérifié les informations envoyées avec celles en base de données le code génère un JWT avec comme information l'ID de l'utilisateur. Voici le snippet qui correspond à cette création de JWT:

```
//constante qui définit la durée de vie d'un JWT
const maxAge = 2592000000; //30 jours;

//fonction pour créer un JWT
const createToken = async (id) => {
  return jwt.sign({ id }, process.env.SECRET_TOKEN, {
    expiresIn: maxAge,
  });
};
```

Ce snippet utilise la méthode "sign" de la dépendance jsonwebtoken. Le token généré comportera l'ID de l'utilisateur passé en paramètre de la fonction createToken et sera signé à l'aide de la clé secrète stockée dans une variable d'environnement DOTENV. Ce token aura une durée de vie d'environ 30 jours grâce à la constante "maxAge".

Lorsque le front-end récupère le token, il le stocke dans le localStorage en utilisant "JWT" comme nom de clé. Voici un exemple de snippet où la ligne encadrée en rouge "set" l'ensemble clé/valeur du JWT dans le localStorage :

```
axios
  .post(`http://localhost:3002/login/signIn/`, data)
  .then(async (response) => {
    localStorage.setItem('JWT', response.data.token);
    navigate('/home');
  })
  .catch((error) => {
    console.log(error);
  });
```



Ce token sera lu et utilisé à chaque échange entre le client et le serveur.

Pour réaliser la logique de connexion où un utilisateur n'a pas besoin de se connecter à chaque fois qu'il accède à l'application web, la première vérification effectuée consiste à vérifier si une valeur JWT est présente dans le localStorage. Si cette valeur est présente, une requête est envoyée à l'URL "http://localhost:3002/tokenValidator". Le middleware appelé "tokenValidator" est utilisé pour vérifier l'authenticité du token et renvoie un objet JSON contenant le résultat. Voici ce middleware :

```
pi > middleware > JS tokenValidator.js > ...
1  import jwt from "jsonwebtoken";
2
3  //middleware verifie l'authenticité du token et rends le résultat
4  const tokenValidator = async (req, res) => {
5    const authHeader = req.get("Authorization");
6    //si il n'y a aucun token -> renvoie "No token"
7    if (!authHeader) {
8      res.status(403).json({ message: "No token" });
9    } else {
10     const token = authHeader.split(" ")[1];
11     let decodedToken;
12     //tente de décoder le token grâce à la chaîne de caractères
13     //contenu dans la variable dotenv SECRET_TOKEN
14     try {
15       decodedToken = jwt.verify(token, process.env.SECRET_TOKEN);
16     } catch (err) {
17       console.error(err);
18     }
19     //Renvoyer le résultat du décodage
20     if (!decodedToken) {
21       res.status(200).json({ valid: false });
22     } else {
23       res.status(200).json({ valid: true });
24     }
25   }
26 };
27
28 export default tokenValidator;
```

Faire des opérations de types CRUD depuis le front

Faire des opérations de types CRUD veut dire être capable de créer, lire, modifier et supprimer un élément dans la base de données. Dans mon application les opérations de types CRUD seront utiles pour :

- Ajouter, lire un badgeage
- Ajouter, lire et supprimer une "News"
- Ajouter, modifier, lire les données d'un utilisateur



Afin d'expliquer chaque étapes d'une opération CRUD voici un petit schéma de l'opération d'ajout de badgeages.

*Composant qui
envoie les
datas à l'API*

```
axios
.post('http://localhost:3002/badg/badgIn', data, {
  headers: { Authorization: `Bearer ${localStorage.getItem('JWT')}` },
})
.catch((error) => {
  console.log(error);
});
```

*middleware qui
récupère les
données envoyés et
les crée dans la
base de donnée*

```
//middleware servant à la création d'un badgeage
export const badgIn = (req, res, next) => {
  let data = req.body;
  const newBadgeage = {
    date: data.date,
    userID: req.userId,
  };
  badgModel
    .create(newBadgeage)
    .then(() => {
      res.status(201).json({
        message: "badgeage pris en compte",
      });
    })
    .catch((error) => {
      console.log(error);
    });
};
```

Dans le snippet ci-dessus, la méthode "create" de Mongoose est utilisée pour écrire des données en base de données. Cette méthode utilise un modèle pour définir les types de données qui seront enregistrés en base. Utiliser cette méthode avec un modèle garantit une approche plus sûre, plus robuste et plus maintenable pour l'écriture des données.

La création d'un modèle passe par la méthode "schema" qui définit la structure et les propriétés des données à stocker dans la base de données. Elle permet également de définir le nom de la collection où les données seront écrites.



Voici le model qui correspond à la collection des badgeages :

nom du model

```
//model correspondant aux badgeages
const badgeageSchema = Schema({
  heure: { type: String, require: false },
  date: { type: String, require: true },
  userID: { type: String, require: true },
  valid: { type: Boolean, require: false },
  collection: "Badgeages"
});
```

nom du schéma

nom du schéma

nom de la collection

Au final voici le document qui sera écrit. Les champs “_id” et “__v” sont rajoutés automatiquement. Ils correspondent à l’id unique du document et sa version

```
{
  _id: ObjectId('649d7b99372aa62f8b58b647')
  date: "1688042393277"
  userID: "6492e134a5703050889d0d20"
  __v: 0
}
```

Créer les différents composants et pages

Dans le but de simplifier le code de mon application, j’ai souvent adopté une structure qui consiste à utiliser un composant pour afficher un autre composant. Pour simplifier, je crée un composant appelé "Display..." qui se charge du traitement des données et de leur transmission au composant principal responsable de la mise en page. Ce modèle me permet de séparer efficacement la logique métier de l’interface utilisateur, ce qui rend le code plus modulaire et facile à maintenir. Grâce à cette approche, je peux me concentrer sur la logique spécifique de chaque composant, tout en garantissant une interface conviviale et réutilisable pour les utilisateurs.

J’ai ajouté un exemple sur la page suivante de composants qui sont utilisés pour afficher les données du compte d’un utilisateur.



Affichage du composant
AccountCard avec en
props les informations
nécessaires au composant

```
client > src > Components > Account > TS DisplayMyAccounttsx > ...
1 import { useEffect, useState } from 'react';
2 import AccountCard from '../AccountCard';
3 import axios from 'axios';
4 import Title from '../Other/Title';
5
6 //Composant qui sert à l'affichage de la carte de compte
7 const DisplayMyAccount = () => {
8   type AccountData = {
9     name: string;
10    firstName: string;
11    email: string;
12    isAdmin: boolean;
13  };
14  //state qui stocke les informations du compte
15  const [accountData, setAccountData] = useState<AccountData>({
16    name: '',
17    firstName: '',
18    email: '',
19    isAdmin: false,
20  });
21  //Envoie de la requête à l'api afin de récupérer les informations du compte
22  useEffect(() => {
23    const fetchData = async () => {
24      const response = await axios.get('http://localhost:3002/account/view', {
25        headers: { Authorization: 'Bearer ${localStorage.getItem('JWT')}' },
26      });
27      setAccountData(response.data);
28      console.log(accountData);
29    };
30    fetchData();
31  }, []);
32
33  return (
34    <>
35    <div className="layoutPages">
36      <Title type="userPage" />
37    </div>
38    <AccountCard
39      nom={accountData.name}
40      prenom={accountData.firstName}
41      email={accountData.email}
42      droits={accountData.isAdmin ? 'Admin' : 'User'}
43    />
44  </>
45  );
46 };
47
48 export default DisplayMyAccount;
```

```
TS AccountCard.tsx M X TS DisplayMyAccounttsx M JS tailwind.config.js M
client > src > Components > Account > TS AccountCard.tsx > ...
1 type Props = {
2   nom: string;
3   prenom: string;
4   email: string;
5   droits: string;
6 };
7
8 //Composant de mise en forme de la carte de compte
9 const AccountCard = ({ nom, prenom, email, droits }: Props) => {
10   return (
11     <div className="flex justify-center">
12       <div className="accountCard">
13         <div className="accountCard_initials">
14           {prenom.charAt(0)}
15           {nom.charAt(0)}
16         </div>
17         <div className="flex flex-col gap-3">
18           <div className="flex gap-3">
19             <strong>{nom}</strong>
20             <p>{prenom}</p>
21           </div>
22           <p>{email}</p>
23           <p>{droits}</p>
24         </div>
25       </div>
26     </div>
27   );
28 };
29
30 export default AccountCard;
```

Page de rétrospection

Avant de clore cette présentation de ma méthode de création des composants j'aimerais mettre en avant ma page de rétrospection. Ce fut le fruit de presque deux semaines de réflexion, travail, débuge et responsivité.

Sa création fut réellement compliquée, car c'était la première fois que je manipulais des dates/heure, et aucune dépendance ne convenait à mes besoins. De plus, mes propres exigences étaient élevées, et je souhaitais vraiment mettre l'accent sur la simplicité de lecture et d'utilisation, notamment pour les administrateurs.

Mes objectifs pour cette page étaient les suivants :

- Afficher de manière chronologique les horaires de badgeage dans un tableau représentant la semaine courante, avec les jours correspondant aux dates.
- Assurer un affichage automatique et fiable de la semaine courante.
- Permettre la navigation en avant ou en arrière d'une semaine.
- Afficher le numéro de la semaine visualisée sur l'interface.
- Offrir à un utilisateur administrateur la possibilité de sélectionner un utilisateur spécifique afin de visualiser ses horaires.

J'ai travaillé pour atteindre ces objectifs et créer une page qui réponde à ces besoins, tout en veillant à ce que l'interface reste conviviale et facile à utiliser.

Voici le résultat de l'interface pour un administrateur, un utilisateur sans pouvoir spéciaux aura juste accès à son calendrier et ne verra pas la liste déroulante.

Page de Rétrospection				
Calendrier de : San Juan Corentin				
Semaine : 39				
Jour	1	2	3	4
Lundi - 25/09/23				
Mardi - 26/09/23				
Mercredi - 27/09/23				
Jeudi - 28/09/23				
Vendredi - 29/09/23				
Samedi - 30/09/23				
Dimanche - 01/10/23				



Voici le code du composant "DisplayRetrospectionAdmin" (page de rétrospection administrateur):

```
client > src > Components > Badgeage > TS DisplayRetrospectionAdmin.tsx > ...
1  import axios from 'axios';
2  import { useEffect, useState } from 'react';
3  import Retrospection from '../Retrospection';
4  import Title from '../Other/Title';
5
6  type User = {
7    _id: string;
8    name: string;
9    firstName: string;
10 };
11
12 const DisplayRetrospectionAdmin = () => {
13   const [badgeages, setBadgeages] = useState([]);
14   const [users, setUsers] = useState([]);
15
16   //hook qui sert à faire ressortir tous les utilisateurs de la base de données
17   //et les stocke dans le state "users"
18   useEffect(() => {
19     const fetchUsers = async () => {
20       try {
21         const response = await axios.get('http://localhost:3002/account/viewAll', {
22           headers: { Authorization: `Bearer ${localStorage.getItem('JWT')}` },
23         });
24         setUsers(response.data.result);
25         console.log(users);
26       } catch (error) {
27         console.error(error);
28       }
29     };
30     fetchUsers();
31   }, []);
32
33   // fonction servant a récupérer les badgeages d'un user grâce à son
34   // ID et stocke la réponse dans le state "badgeages"
35   const fetchData = async (ID: string) => {
36     try {
37       const response = await axios.get(`http://localhost:3002/badg/getBadgByID/${ID}`, {
38         headers: { Authorization: `Bearer ${localStorage.getItem('JWT')}` },
39       });
40       setBadgeages(response.data);
41     } catch (error) {
42       console.error(error);
43     }
44   };
45
46   //Handler qui ecoute s'il y a un changement dans la liste d'utilisateur
47   //et mets a jour les badgeages à visualiser
48   const handleChangeUser = (event: React.ChangeEvent<HTMLSelectElement>) => {
49     fetchData(event.target.value);
50   };
51 }
```




```

client > src > Components > Badgeage > TS DisplayRetrospectionAdmin.tsx > ...
51
52   return (
53     <div className="layoutPages">
54       {/* Composant qui sert aux titres */}
55       <Title type="retrospection" />
56       <div className="flex">
57         <p>Calendrier de :</p>
58         {/* Liste déroulante des utilisateurs */}
59         <select onChange={handleChangeUser} className="bg-color4 ml-2 rounded">
60           <option value="">Sélectionnez un utilisateur</option>
61           {users.map((user: User) => (
62             // Création à la volée de chaque option avec
63             // les noms, prénoms des utilisateurs
64             <option key={user._id} value={user._id}>
65               {user.name} {user.firstName}
66             </option>
67           ))}
68         </select>
69       </div>
70       {/* Si le nombre de badgeage est supérieur à 0 alors afficher le composant
71       retrospection avec les dates de badgeages correspondantes sinon écrire
72       "Aucune donnée de badgeage à afficher" */}
73       {badgeages.length > 0 ? (
74         <Retrospection dates={badgeages} />
75       ) : (
76         <div>Aucune donnée de badgeage à afficher</div>
77       )}
78     </div>
79   );
80 };
81
82 export default DisplayRetrospectionAdmin;

```

Pour développer cette page, je n'ai pas réussi à respecter pleinement mon pattern de composant qui traite les données et renvoie le composant de mise en forme. Cependant, j'ai réussi à séparer la logique de sélection des utilisateurs de celle de l'affichage des badgeages.

Pour la récupération des badgeages, le retour de l'API est composé de timestamps. Un timestamp est un système d'horodatage qui représente le nombre de secondes écoulées depuis le 1er janvier 1970 à 00h00. Il s'agit de la date de démarrage d'UNIX, le tout premier système d'exploitation moderne.

Par exemple, le "01/01/2022 à 00:00:00" correspond au timestamp de **1640991600**.



Voici la première partie du composant <Retrospection/>

```
client > src > Components > Badgeage > TS Retrospection.tsx > [e] Retrospection > actualWeek.map() callback
3  interface DateObject {
4      dayOfWeek: string;
5      date: string;
6      hour: string;
7  }
8
9  type Props = {
10     dates: number[];
11 };
12
13 const Retrospection = (props: Props) => {
14     // State pour : La semaine actuelle, L'offset de la semaine à visualiser, Le numéro de la semaine affichée
15     const [actualWeek, setActualWeek] = useState<string[]>([]);
16     const [weekOffset, setWeekOffset] = useState<number>(0);
17     const [weekNumber, setWeekNumber] = useState<number>(0);
18
19     // Tableau des jours de la semaine
20     const daysOfWeek = ['Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi', 'Dimanche'];
21
22     useEffect(() => {
23         // Date actuelle
24         let now = new Date();
25         // Calcul de la date en fonction de l'offset de la semaine
26         // En cas de changement de l'offset de la semaine now change +/- d'une semaine
27         now.setDate(now.getDate() + weekOffset * 7);
28
29         const calcNumWeek = () => {
30             // Premier jour de l'année
31             const firstDayOfYear = new Date(now.getFullYear(), 0, 1);
32             // Nombre de jours passés depuis le premier jour de l'année
33             const pastDaysOfYear = (now.getTime() - firstDayOfYear.getTime()) / 86400000;
34             // Calcul du numéro de la semaine grâce aux deux constantes précédentes
35             const currentWeekNumber = Math.ceil((pastDaysOfYear + firstDayOfYear.getDay() + 1) / 7);
36             setWeekNumber(currentWeekNumber);
37         };
38         calcNumWeek();
39
40         const otherDays: string[] = [];
41
42         const calcOtherDaysOfWeek = () => {
43             const today = () => {
44                 // Jour de la semaine (0 pour Dimanche, 1 pour Lundi, etc...)
45                 const dayOfWeek = now.getDay();
46                 // Jour du mois
47                 const day = now.getDate();
48                 return { dayOfWeek, day };
49             };
50             const todayInfo = today();
51             for (let i = 0; i < daysOfWeek.length; i++) {
52                 const day = todayInfo.day - todayInfo.dayOfWeek + i + 1;
53                 const date = new Date(now.getFullYear(), now.getMonth(), day);
54                 const formattedDate = `${date.getDate().toString().padStart(2, '0')}/${date.getMonth() + 1}
55                     .toString().padStart(2, '0')}/${date.getFullYear().toString().slice(-2)}`;
56                 otherDays.push(formattedDate);
57             }
58         };
59         calcOtherDaysOfWeek();
60
61         // Mise à jour de la semaine actuelle
62         setActualWeek(otherDays);
63     }, [weekOffset]);
64 }
65
```



La deuxième partie :

```
66 // Fonction qui traduit un timestamp et rend un objet avec le jour,
67 // la date au format : dd/mm/yy, et l'heure sous la forme hh:mm
68 const formatDate = (date: Date): DateObject => {
69   const daysOfWeek = ['Lun', 'Mar', 'Mer', 'Jeu', 'Ven', 'Sam', 'Diman'];
70   // Jour de la semaine au format court (Lun, Mar, etc.)
71   const dayOfWeek = daysOfWeek[date.getDay() + 1];
72   const day = date.getDate().toString().padStart(2, '0');
73   // Mois (ajout de 1 car les mois commencent à partir de 0)
74   const month = (date.getMonth() + 1).toString().padStart(2, '0');
75   // Année (obtention des deux derniers chiffres)
76   const year = date.getFullYear().toString().slice(-2);
77   const hours = date.getHours().toString().padStart(2, '0');
78   const minutes = date.getMinutes().toString().padStart(2, '0');
79   return {
80     dayOfWeek: dayOfWeek,
81     date: `${day}/${month}/${year}`,
82     hour: `${hours}:${minutes}`,
83   };
84 };
85
86 // Traduction des timestamps passés en props en dates lisibles
87 const traduction = props.dates.map((date) => formatDate(new Date(date)));
88
89 const handlePrevWeek = () => {
90   // Décrémente l'offset de la semaine pour revenir en arrière d'une semaine
91   setWeekOffset(weekOffset - 1);
92 };
93
94 const handleNextWeek = () => {
95   // Incrémente l'offset de la semaine pour avancer d'une semaine
96   setWeekOffset(weekOffset + 1);
97 };
98
99 return (
100   <div className="bg-color4 p-4 rounded">
101     /* Gestion des semaines */
102     <div className="flex justify-between mb-4">
103       <button
104         className="bg-prev bg-cover bg-center py-2 px-4 rounded"
105         onClick={handlePrevWeek}>
106       </button>
107       <p>Semaine : {weekNumber}</p>
108       <button
109         className="bg-next bg-cover bg-center py-2 px-4 rounded"
110         onClick={handleNextWeek}>
111       </button>
112     </div>
113     /* tableau de retrospection */
114     <table className="border-collapse border border-gray-400">
115       /* Définition des colonnes */
116       <thead className="bg-color1">
117         <tr>
118           <th className="border border-gray-400 px-4 py-2">Jour</th>
119           {[1, 2, 3, 4].map((numero) => (
120             <th key={numero} className="border border-gray-400 px-4 py-2 w-20">
121               {numero}
122             </th>
123           ))}
124         </tr>
125       </thead>

```



Et la dernière :

```
126 <tbody>
127 {actualWeek.map((jour, index) => {
128   const cells = [];
129   // filtre les dates traduites (traduction) pour ne conserver que celles
130   // qui correspondent à la date (jour) actuellement traitée
131   const matchingDates = traduction.filter((date: DateObject) => date.date === jour);
132
133   for (let i = 0; i < 4; i++) {
134     // Si une date correspondante est trouvée à l'index i, une cellule contenant
135     // l'heure est ajoutée à cells. Sinon, une cellule vide est ajoutée.
136     if (matchingDates[i]) {
137       cells.push(
138         <td key={i} className="border ■ border-gray-400 px-4 py-2">
139           {matchingDates[i].hour}
140         </td>,
141       );
142     } else {
143       cells.push(<td key={i} className="border ■ border-gray-400 px-4 py-2"></td>);
144     }
145   }
146   return (
147     // Créer une cellule contenant le jour de la semaine et la date,
148     // suivie des cellules générées dans l'étape précédente
149     <tr key={index}>
150       <td className="border ■ border-gray-400 px-4 py-2">
151         {daysOfWeek[index]} - {jour}
152       </td>
153       {cells}
154     </tr>
155   );
156   })
157 </tbody>
158 </table>
159 </div>
160 </div>
161 </div>
162 </div>
163
164 export default Retrospection;
```

Le composant Retrospection génère un élément visuel qui affiche une rétrospective des dates passées en prop. Voici concrètement ce que fait ce composant :

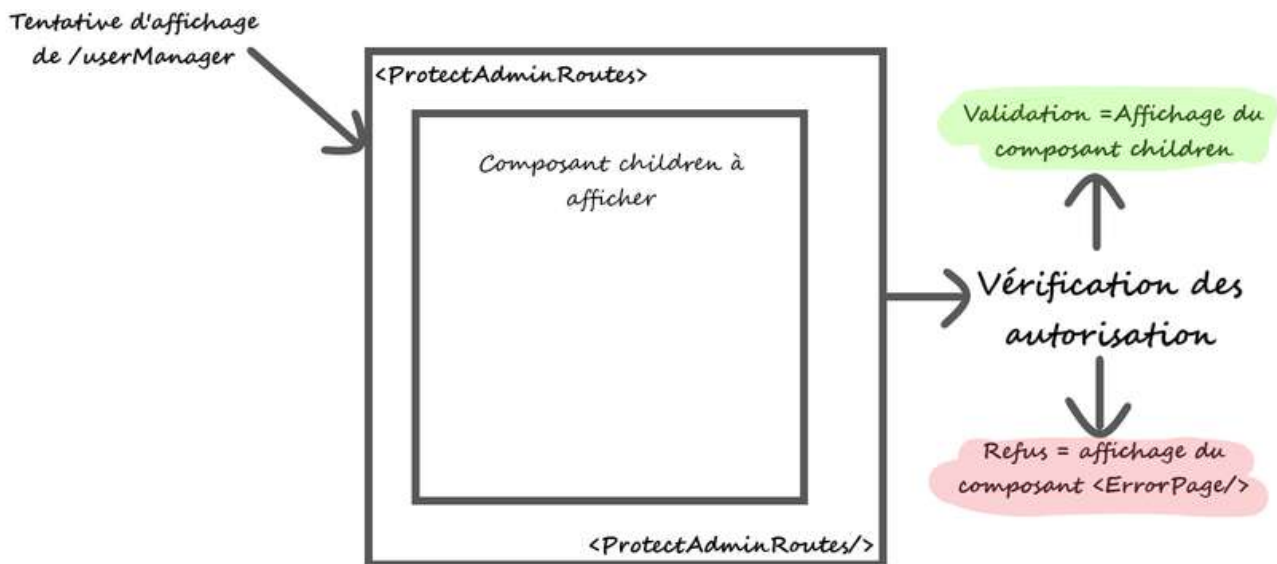
1. Il initialise les états actualWeek, weekOffset et weekNumber.
2. Dans le hook useEffect, le composant effectue des calculs pour déterminer la semaine actuelle en utilisant l'offset de la semaine. Il récupère la date actuelle, calcule le numéro de la semaine en utilisant des opérations sur les dates, et détermine les autres jours de la semaine en fonction de la date actuelle.
3. Une fonction formatDate est définie pour traduire les timestamps passés en dates lisibles.
4. Les timestamps passés en prop "dates" sont traduits en dates lisibles à l'aide de la fonction formatDate, et les résultats sont stockés dans le tableau "traduction".
5. Le composant rend un élément visuel qui affiche la semaine actuelle, les jours de la semaine avec leurs dates correspondantes, ainsi que les dates traduites.
6. Le rendu inclut également des boutons permettant de naviguer entre les semaines précédentes et suivantes, en modifiant l'offset de la semaine.



Gestion des droits et accès aux différentes pages

Afin de gérer les droits et donner accès aux pages administrateurs, j'ai créé une logique qui consiste à envelopper une page protégée dans un composant qui vérifie les droits.

Voici un schéma qui permet de mieux comprendre ma logique :



- 1 Lorsque'un utilisateur tente d'accéder à "/userManager", le routeur affiche le composant `<ProtectAdminRoutes/>` et lui transmet `<UserManager/>` en tant que composant enfant.
- 2 Le composant `ProtectAdminRoutes` vérifie si le demandeur est administrateur en envoyant une requête à l'API.
- 3 Si l'utilisateur est administrateur, le composant `ProtectAdminRoutes` rend le composant enfant. Sinon, il renvoie la page d'erreur.

Cette logique m'a également permis de bloquer l'accès à mes routes pour les utilisateurs non authentifiés ou dont le JWT n'est pas valide. Le composant qui permet cela est `<ProtectRoutes/>`. Son fonctionnement est similaire, à l'exception qu'en cas d'invalidité du token présent dans le localStorage, il supprime la paire clé/valeur et redirige vers la page d'accueil.

Voici un exemple de snippet de code qui illustre la protection des pages administrateurs :

```
client > src > TS Routes.tsx > ...
69
70     path: '/home/userManager',
71     element: (
72       <ProtectAdminRoutes>
73         <DisplayUserManager />
74       </ProtectAdminRoutes>
75     ),
76   },
```

Routeur

ProtectAdminRoutes

```
client > src > Components > TS ProtectAdminRoutes.tsx > [?] ProtectAdminRoutes
1  import { useEffect, useState } from 'react';
2  import axios from 'axios';
3  import ErrorPage from '../Pages/ErrorPage';
4
5  type Props = {
6    children: any;
7  };
8
9  // Composant ProtectAdminRoutes qui vérifie si l'utilisateur
10 //est un administrateur avant de rendre le contenu de l'enfant (children)
11 //passé en tant que prop.
12 const ProtectAdminRoutes = ({ children }: Props) => {
13   const [isAdmin, setIsAdmin] = useState(false);
14
15   // Verifie si l'utilisateur est admin ou non est met à jour le
16   //useState isAdmin avec la valeur de la reponse de l'API
17   useEffect(() => {
18     const fetchIsAdmin = async () => {
19       const response = await axios.get('http://localhost:3002/adminValidator', {
20         headers: { Authorization: `Bearer ${localStorage.getItem('JWT')}` },
21       });
22       setIsAdmin(response.data);
23     };
24     fetchIsAdmin();
25   }, []);
26
27   // Si l'utilisateur est admin -> rendre children, sinon rendre le composant ErrorPage
28   return isAdmin ? children : <ErrorPage />;
29 };
30
31 export default ProtectAdminRoutes;
```

```
api > JS server.js > ...
26 app.use(isAdmin);
27 app.use("/adminValidator", adminValidator);
```

Chemin coté serveur

```
import userModel from "../models/userModel.mjs";

// fonction qui rends les informations de l'utilisateur en fonction de son ID
const findAccount = async (userID) => {
  const finder = await userModel.findById(userID);
  return finder;
};

// middleware qui ajoute à la requête si l'utilisateur est admin ou non
const isAdmin = async (req, res, next) => {
  const data = await findAccount(req.userId);
  req.isAdmin = data.isAdmin;
  next();
};

export default isAdmin;
```

Middleware isAdmin

Middleware
adminValidator

```
api > middleware > JS adminValidator.js > ...
1 //middleware qui renvoie si l'utilisateur est admin ou non
2 //grâce au contenu ajouté précédemment à la requête (req.isAdmin)
3 const adminValidator = async (req, res) => {
4   res.status(200).json(req.isAdmin);
5 };
6
7 export default adminValidator;
```



Amélioration de l'UX

L'expérience utilisateur occupe une place primordiale dans le développement d'une application. En effet, elle constitue l'un des aspects les plus cruciaux pour garantir le succès et la satisfaction des utilisateurs. Une application dotée d'une expérience utilisateur soignée et intuitive permet aux utilisateurs de naviguer facilement, d'accomplir leurs tâches de manière fluide et d'obtenir rapidement les informations recherchées. En mettant l'accent sur l'ergonomie, la simplicité d'utilisation et la réactivité, une application peut offrir une expérience utilisateur positive, ce qui favorise l'adoption et fidélise les utilisateurs. Par conséquent, il est essentiel de consacrer du temps et des ressources à la conception et à l'amélioration continue de l'expérience utilisateur afin de proposer une application qui répond aux besoins et aux attentes des utilisateurs.

Dans ce chapitre, je vais principalement aborder le sujet des animations CSS. Elles sont essentielles pour rendre mon application plus fluide et attrayante. Afin d'obtenir des animations web fluides, il est important de proposer des animations qui peuvent être lues par tous les navigateurs sur chaque machine à une fréquence de 60 images par seconde (ips). Les animations que nous voyons sur nos écrans ne sont pas de réels mouvements. Il s'agit en réalité de séquences d'images qui défilent suffisamment rapidement pour être interprétées par notre cerveau comme des objets en mouvement. Plus une animation a un nombre d'ips élevé, plus elle sera fluide, et 60 ips est la norme pour que l'utilisateur ne perçoive pas de saccades.

De plus, la plupart des écrans actuels ont une fréquence de rafraîchissement de 60 Hz. Les Hz mesurent le nombre de fois où un événement se produit par seconde, donc 60 Hz équivaut à 60 ips pour un écran. Par conséquent, il est inutile de proposer des animations plus fluides pour ce type d'écran.

Pour expliquer comment atteindre les 60 ips sur le maximum de machine j'ai besoin d'expliquer les étapes qui sont réalisées par un navigateur pour afficher une page.

Pour passer des codes CSS et HTML... à une page web, le navigateur passe par plusieurs étapes pour afficher une page. Voici les étapes qui nous intéressent pour le rendu de nos animations :

- **Style** : le navigateur reçoit le code HTML. Il va l'interpréter pour comprendre la structure du DOM (Document Object Model). Ainsi, pour chaque balise HTML, il crée un élément du DOM, un peu comme un arbre de nœuds. Il parcourt ensuite le CSS, et détermine quelles règles s'appliquent à quels éléments. À partir de là, il va créer la structure qui s'affichera.



- Layout (mise en page) : maintenant que le navigateur connaît les styles et les éléments à afficher, il détermine la taille des éléments et où les placer.
- Paint (peinture) : le navigateur transforme les éléments en pixels en utilisant les styles de l'étape 1, et les positions et dimensions déduites de l'étape 2.
- Composition : le navigateur combine tous les éléments pour composer la page qui s'affiche dans le navigateur.

Chaque étape nécessite un temps de calcul au navigateur. Il est donc plus judicieux de passer par le moins d'étapes possible si on veut respecter la norme des 60 ips. Il est préférable aussi de modifier des propriété CSS qui correspondent aux étapes 3 et 4 afin d'éviter le recalcul des étapes suivantes. Par exemple la propriété "width" fait partie de l'étape 2 , si nous la modifier il faudra donc recalculer les étapes Layout, Paint et Composition ce qui n'est pas très efficient.

Le site <https://csstriggers.com/> permet de connaitre pour chaque propriétés les étapes utilisées par le navigateur.

Les meilleures propriétés à utiliser pour animer un site web sont : transform et opacity. Avec seulement ces deux propriétés nous pouvons animer à peu près tout et n'importe quoi. En plus d'utiliser ces deux classes j'ai également utilisé principalement des @keyframes car elles permettent de gérer précisément les étapes d'une animation.

L'exemple qui illustre le mieux cette recherche de fluidité est le bouton "addElement":

```
client > src > Components > News > TS DisplayNews.tsx > [0] News
47 {isAdmin ? (
48 | // bouton add qui tourne si inAdd = true
49 | <button onClick={handleClickAddNews} className={`btnAddElem ${inAdd ? 'turn' : ''} `}>
50 |   /* partie gauche du bouton */
51 |   <div className="btnAddElem__left"></div>
52 |   /* icon plus qui ne disparaît plus si inAdd = true */
53 |   <div className={`btnAddElem__plus ${inAdd ? 'turn' : ''} `}></div>
54 |   /* Si inAdd = false alors le bouton peut se développer */
55 |   {inAdd ? (
56 |     <></>
57 |   ) : (
58 |     <div className="btnAddElem__anim">
59 |       <span className="btnAddElem__anim--text">Ajouter une news</span>
60 |     </div>
61 |   )}
62 |   /* Partie droite du bouton */
63 |   <div className="btnAddElem__right"></div>
64 | </button>
65 | ) : (
66 |   <></>
67 | )}
```



Voici le code SCSS qui correspond à l'animation:

```
client > sass > components > _buttons.scss > ...
78   @media (max-width: 1000px) {
79     @media (hover: hover) {
80       &:hover {
81         animation: vibrate infinite 0.3s ease-in-out;
82         &.turn {
83           animation: vibrate-turn infinite 0.3s ease-in-out;
84         }
85       }
86     }
87   }
88   @media (min-width: 1000px) {
89     &:hover {
90       &.turn {
91         animation: vibrate-turn infinite 0.3s ease-in-out;
92       }
93       & > .btnAddElem__plus {
94         animation: btnAdd_icon 91ms ease-in-out both;
95         &.turn {
96           animation: none;
97         }
98       }
99       & > .btnAddElem__anim {
100        animation: btnAdd 250ms ease-in-out both;
101      }
102    }
103  }
104 }
```

Dans le snippet ci-dessus l'utilisation du "@media(hover: hover)" permet de cibler unique les media qui prennent en charge la classe pseudoclasse :hover. Concrètement les téléphones mobiles par exemple ne seront pas concernés par cette animation.

La propriété animation est ici utilisé pour fournir le nom de la keyframe qui correspond à l'animation, sa durée, sa méthode d'avancement et son comportement avant et après son exécution.



Voici les @keyframes utilisés

```
client > sass > utils > _keyframes.scss > @keyframes btnAdd {
1 //extension du bouton
2 @keyframes btnAdd {
3   from {
4     transform: scaleX(0);
5     opacity: 0;
6   }
7   20% {
8     opacity: 1;
9   }
10  to {
11    transform: scaleX(1);
12  }
13 }
```

```
client > sass > utils > _keyframes.scss > @keyframes btnAdd_icon {
15 //dispartition de l'icone plus
16 @keyframes btnAdd_icon {
17   from {
18     opacity: 1;
19   }
20   to {
21     transform: scale(0);
22     opacity: 0;
23   }
24 }
```

```
client > sass > utils > _keyframes.scss > @keyframes vibrate {
26 //Keyframe de vibration
27 @keyframes vibrate {
28   0% {
29     transform: translate(0);
30   }
31   20% {
32     transform: translate(-1px, 1px);
33   }
34   40% {
35     transform: translate(-1px, -1px);
36   }
37   60% {
38     transform: translate(1px, 1px);
39   }
40   80% {
41     transform: translate(1px, -1px);
42   }
43   100% {
44     transform: translate(0);
45   }
46 }
```

```
client > sass > utils > _keyframes.scss > @keyframes vibrate-turn {
48 //vibration mais avec le bouton tourné à 45deg
49 @keyframes vibrate-turn {
50   0% {
51     transform: translate(0) rotate(45deg);
52   }
53   20% {
54     transform: translate(-1px, 1px) rotate(45deg);
55   }
56   40% {
57     transform: translate(-1px, -1px) rotate(45deg);
58   }
59   60% {
60     transform: translate(1px, 1px) rotate(45deg);
61   }
62   80% {
63     transform: translate(1px, -1px) rotate(45deg);
64   }
65   100% {
66     transform: translate(0) rotate(45deg);
67   }
68 }
```

Elles utilisent toutes seulement les propriétés transform et opacity afin de respecter les 60 ips.

Vous pourrez voir les visuels de l'application finale en annexes



TESTS

Pour réaliser des tests vous pouvez cloner les deux repository github en local et suivez les intructions des README.md. pour Rappel voic les liens des repository :

- API : https://github.com/Crtnsj/BADG-HEURE_api
- Front : https://github.com/Crtnsj/BADG-HEURE_front

Il y a des valeurs à remplacer dans le readme de l'api, veuillez les remplacé par :

- SECRET_TOKEN=n!V&O\$gwZKTPsv1ghU+1Rw8EJH3\$^X
- ACCESS_DB=badg-heure-test:qr0vap0P1RDrdNNVQ
- PORT=3002

J'ai créer un administrateur pour pouvoir faire des tests. Voici les identifiant à renseigner :

- nom d'utilisateur : **administrateur**
- mot de passe : **P3jYtX9n\$o#M?d?!**

J'ai également créer un utilisateur sans droits spéciaux :

- nom d'utilisateur : **testeur@badg-heure.fr**
- mot de passe : **BcMsX5\$tNEeozNL@**

En cas de problèmes vous pouvez me contacter à l'adresse suivante :

- **crtnsj.pro@gmail.com**



CONCLUSION

Pour conclure, ce projet a été extrêmement intéressant à réaliser, car il m'a permis de découvrir de nouvelles technologies telles que React, MongoDB, Axios, Dotenv, TailwindCSS, Vite, Jsonwebtoken, ainsi que d'approfondir mes connaissances en Sass, Express et Node.

J'ai dû me documenter et rechercher des réponses à mes questions pour résoudre les problèmes rencontrés. Cette expérience a été enrichissante et formatrice, car c'était la première fois que je réalisais un projet de développement de cette envergure, ce qui m'a obligé à adopter une rigueur dans l'écriture du code pour éviter de me perdre au fil du temps.

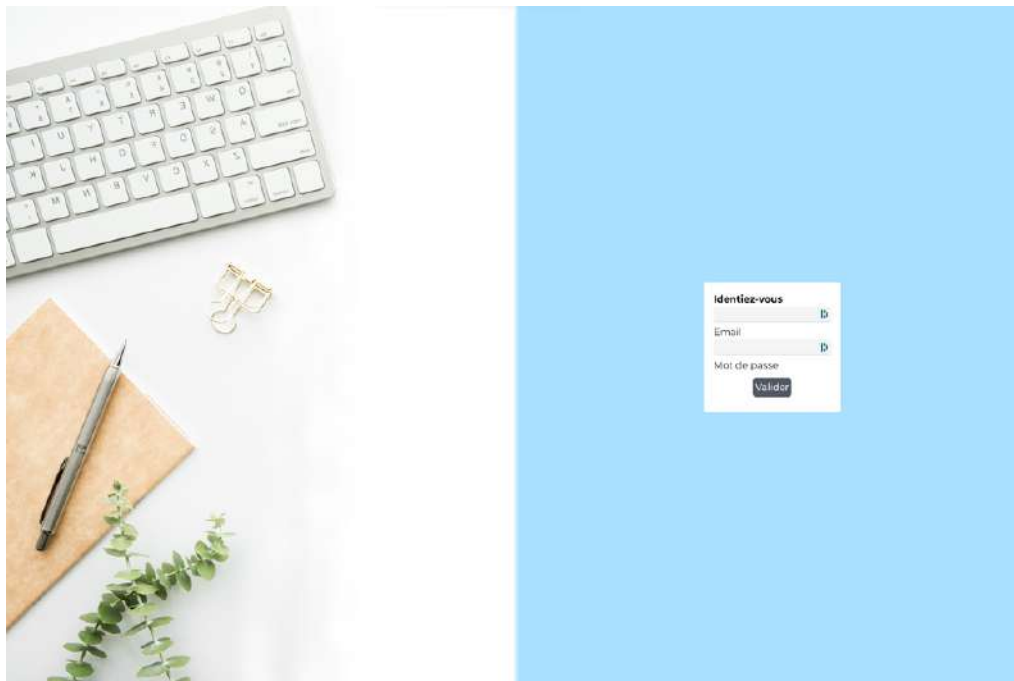
Cependant, je regrette de ne pas avoir réussi à implémenter toutes les fonctionnalités que je souhaitais. Par exemple, sur la maquette, il y avait une horloge, et j'aurais aimé la rendre entièrement fonctionnelle. Il y a aussi des pages "en construction", c'est-à-dire que je n'ai pas eu le temps d'implémenter ces fonctionnalités dans le délai imparti, mais elles feront l'objet d'une mise à jour ultérieure. Les prochaines fonctionnalités à ajouter seront probablement la possibilité de modifier les mots de passe, de demander des rectifications de badgeages en cas d'oubli, et également un compteur dans la page de rétrospection qui affichera clairement le nombre d'heures effectuées par jour.

Malgré tout, je suis très fier du résultat obtenu après environ 400 heures de travail, de découvertes, d'organisation, de réflexion et de documentation. J'ai hâte de proposer des versions futures de ce projet, avec pour objectifs une plus grande clarté dans le code, une meilleure sécurité et de nouvelles fonctionnalités. J'aimerais également rendre cette application accessible, c'est-à-dire permettre à tous de l'utiliser dans les meilleures conditions possibles.



ANNEXES

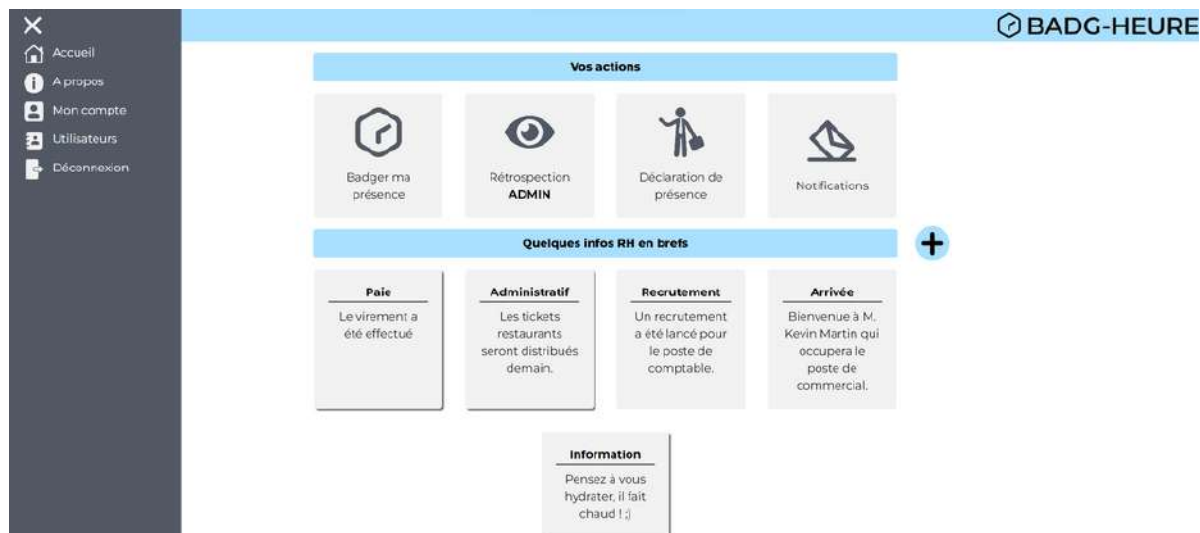
Page de connexion :



Page d'accueil Administrateur :



Barre de navigation administrateur



Bouton "Add" au :hover :

Ajouter une news

Page "En construction"



Cette page est en construction...

Le chef de chantier annonce une inauguration dans peu de temps 😊

Retourner à l'accueil

Page de Badgeage



Badger ma
présence

Valider mon entrée /
sortie

Voici vos informations personnelles



Administrateur administrateur
administrateur
Admin

Gestion des Utilisateurs 


Données personnelles de : BADG-HEURE Testeur 



BADG-HEURE Testeur
testeur@badg-heure.fr
User
Modifier

Gestion des Utilisateurs 

Données personnelles de : BADG-HEURE Testeur 



BADG-HEURE Testeur

testeur@badg-heure.fr

User 

Sauvegarder
Annuler

Gestion des Utilisateurs 

Ajouter un utilisateur

Nom

Prenom

Email

Mot de passe



Valider

