

Antoine DUPUY
1A, rue Carnot
28 700 AUNEAU



RAPPORT DE STAGE

Développeur web et web mobile

Du 28/06/2021 au 17/09/2021

Tuteur : Arthur Robieux

ENTREPRISE



SportEasy
6 rue Claude Farrère
75 016 PARIS
www.sporteasy.net

FORMATION



AFPA
2 rue Gaston Planté
28 000 CHARTRES
www.afpa.fr

Sommaire

Avant-propos	3
1 - Introduction	4
1.1 - Présentation de SportEasy	4
1.2 - Pourquoi SportEasy ?	5
2 - Matériels & méthodes	6
2.1 - Technologies back-end	6
2.2 - Technologies front-end	7
2.3 - Environnement logiciel	9
2.4 - Focus sur React	10
2.5 - Le projet SportEasy : l'application web en détail ..	12
2.6 - Les outils à notre disposition	14
2.7 - Les méthodes : une organisation d'équipe	16
3 - Développement	18
3.1 - Objectifs, missions et contraintes	18
3.2 - Mes principaux travaux	19
3.2.1 - La « Home Team »	19
3.2.2 - La page des Présences	29
3.3 - Le Responsive Design	39
3.4 - D'autres travaux.....	43
4 - Conclusion	46
4.1 - Ce que le stage m'a apporté	46
4.2 - Avis final	46

Avant-propos

Je voudrais tout d'abord remercier SportEasy de m'avoir fait confiance et de m'avoir accueilli au cours de ces 12 semaines de stage.

Grâce à cette confiance, j'ai pu, durant cette période courte mais intense, découvrir tous les aspects du travail d'une équipe produit au quotidien, que ce soit à travers les réunions, les différentes réflexions ou les problématiques rencontrées.

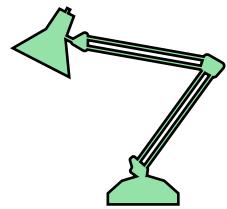
Considéré dès le départ comme un développeur à part entière avec les responsabilités qui vont avec, j'ai eu accès à tout le matériel et tous les outils utilisés par l'équipe produit, et j'ai pu proposer mes idées librement.

Merci d'avoir valorisé mon travail en me confiant des sujets importants et en permettant de les présenter moi-même lorsqu'ils furent menés à terme.

L'accueil réservé, la disponibilité de chacun, leur pédagogie et leur accompagnement m'a permis de progresser très vite, au-delà de mes espérances.

Et enfin, une pensée toute particulière à mon tuteur, Arthur Robieux, qui m'a accompagné à chaque étape de mon stage, avec bienveillance, encouragements et considération.

1 - Introduction



1.1 - Présentation de SportEasy

Créée en 2012 par deux ingénieurs passionnés de sport, Albin Egasse et Nizar Melki, SportEasy a pour mission de permettre à ceux qui font le sport amateur de passer plus de temps à vivre leur passion, moins à résoudre des problèmes.



En offrant aux dirigeants & entraîneurs des fonctionnalités de gestion administrative et sportive, et aux joueurs & parents des fonctionnalités sociales et ludiques, SportEasy s'est imposée comme l'application web et mobile de référence pour gérer une équipe ou un club amateur, quel que soit le sport.

Depuis 2018, SportEasy propose aux clubs des services supplémentaires et des avantages exclusifs en les connectant à un riche écosystème de partenaires : sponsors, centres sportifs, ligues ou autres applications.

SportEasy est basé à Paris, au Tremplin, incubateur de startups dédiés au sport.

Quelques chiffres :

SportEasy est utilisé par **1 500 000** personnes, en **8** langues et dans plus de **100** pays à travers le monde.

Cela représente plus de **1 000** clubs et **75 000** équipes, pour une vingtaine d'employés.

Les fonctionnalités clés de l'expérience SportEasy :

La gestion d'équipes, de groupes ou de membres, un calendrier partagé, une messagerie, la gestion des inscriptions, des cotisations et des collectes, la visualisation de statistiques et la mise en avant des sponsors.

1.2 - Pourquoi SportEasy ?

Au cours de ma formation, après différentes approches du développement web : back-end & front-end, gestion de base de données, frameworks, langages divers, etc... j'ai pu cerner mon domaine de prédilection : le développement front-end.

Apportant une importance particulière au visuel, étant rigoureux dans la mise en forme, soucieux des détails, et attiré par le domaine de l'UX (*User Experience*) c'est donc tout naturellement que j'ai souhaité basé mon stage autour de cet aspect du développement web et web mobile.

SportEasy m'offrait cette opportunité de travailler sur un projet front-end cohérent et moderne, développé dans une des technologies les plus utilisée et répandue dans ce domaine, React.

J'ai également eu la chance d'avoir pour tuteur le développeur front-end du projet, qui est à la base de la migration de l'application vers React, avec plusieurs années d'expériences, et une très bonne connaissance de son projet.

Mon intérêt pour le domaine sportif et le football ne fût donc qu'un bonus à cette opportunité de stage qui représentait déjà, en soit, un choix parfait par rapport à mes attentes.



2 - Matériels & méthodes

2.1 - Technologies back-end

BACK-END

Le projet back-end, nommé **deuzio**, est développé en **Python** et repose sur le framework **Django**.

Historiquement, toute l'application web de SportEasy était intégrée à ce projet via le langage de gabarit de Django. Depuis, la quasi-totalité de la partie front-end a été migrée vers React, et deuzio ne gère plus que l'API et la partie back-end en général.

Quelques définitions :

Django : Django est un cadre de développement web open source en Python. Il a pour but de rendre le développement web 2.0 simple et rapide. Plusieurs sites grand public sont désormais fondés sur Django, dont Pinterest et Instagram, ou encore Mozilla.

Python : Python est un langage de programmation interprété, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire et d'un système de gestion d'exceptions.

Le langage Python est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques.

Même si ce n'était pas le sujet principal de mon stage, j'ai pu parcourir et découvrir ce projet à plusieurs reprises, que ce soit pour ajouter une redirection vers l'application React, comprendre le fonctionnement de l'API en y ajoutant des routes très simples, ou utiliser son langage de gabarit pour le site vitrine (*toujours sous Django et pas encore migré vers React*).

2.2 - Technologies front-end

FRONT-END

Toute l'application web, nommée simplement **webapp**, est donc développé sous **React** avec une surcouche **TypeScript** et avec le moins de librairie possible, et au contraire le plus de composants « maisons », pour avoir un maximum de liberté sur chaque aspect du design et de l'UX.

La gestion du CSS se fait par l'intermédiaire de **Sass**.

La gestion des traductions se fait par l'intermédiaire de **POEditor**.

Des tests unitaires et fonctionnels sont régulièrement mis en place sur la plupart des composants importants grâce à **cypress** et **jest**.

Quelques définitions :

React : React est une bibliothèque JavaScript libre développée par Facebook depuis 2013. Le but principal de cette bibliothèque est de faciliter la création d'application web monopage, via la création de composants dépendant d'un état et générant une page HTML à chaque changement d'état. Je reviendrai sur les spécificités et avantages de React dans la partie dédiée.

TypeScript : TypeScript est un langage de programmation libre et open source développé par Microsoft qui a pour but d'améliorer et de sécuriser la production de code JavaScript. Il s'agit d'un sur-ensemble syntaxique strict de JavaScript.

Le code TypeScript est transcompilé en JavaScript, et peut ainsi être interprété par n'importe quel navigateur web ou moteur JavaScript.

TypeScript a été conçu pour pallier les lacunes de JavaScript pour le développement d'applications à grande échelle. Le langage ajoute des fonctionnalités à JavaScript telles que le typage statique ou générique, les interfaces, les classes ou les modules.

Sass : Sass est un langage de script préprocesseur qui est compilé ou interprété en CSS. SassScript est le langage de script en lui-même.

Sass se compose de deux syntaxes. La syntaxe originale, appelé « la syntaxe indentée » utilise l'indentation pour séparer les blocs de code et les sauts de ligne pour les séparer des règles. La nouvelle syntaxe : « SCSS » utilise les mêmes séparateurs de blocs que le CSS.

C'est la syntaxe SCSS qui est utilisée au sein de la **webapp**.

La grande force de SassScript repose sur les mécanismes suivants : variables, imbrication, mixins et héritage des sélecteurs.

L'application web est compilée grâce à **NodeJs**, une plateforme logicielle libre en JavaScript, orientée vers les applications réseau.

2.3 - Environnement logiciel

Pour travailler sur ces deux projets, SportEasy m'a confié un MacBook Pro durant toute la durée du stage.

N'étant pas familier avec l'environnement Apple, j'ai donc pu apprendre à travailler sur **macOs (macOs Big Sur version 11.4)** et avec sa suite logiciel (**Keynote** et **Pages** par exemple).



Éditeur de code ou IDE (*Integrated Development Environment*) :

J'ai principalement utilisé le célèbre **Visual Studio Code (VSC)** dans sa version **1.60.1**. J'ai également essayé PyCharm, dans sa version **2021.1.1**, lorsqu'il s'agissait de travailler sur le projet Django.



Extensions utilisées sur Visual Studio Code :

Prettier - Code Formatter -> pour formater et indenter le code à chaque sauvegarde et ainsi gagner en rapidité, productivité et confort d'utilisation.

ESLint -> outil d'analyse de code statique pour identifier les modèles problématiques trouvés dans le code JavaScript.

Paramétré au préalable pour harmoniser les pratiques de développement au sein de la webapp, garder un code clair et uniforme, et éviter les erreurs.

Live Share -> permet le peer-programming (*programmation en binôme*) en temps réel, en partageant son écran et son projet.
Idéal pour un stagiaire en télé-travail !

Jupyter -> Jupyter Notebook est un environnement de programmation interactif basé sur le Web permettant de créer des documents Jupyter Notebook.

2.4 - Focus sur React

On a vu précédemment que React était une librairie JavaScript facilitant la création d'application web. Voyons maintenant les grandes forces et les avantages de React.

Les forces de React :

React se démarque de ses concurrents par sa flexibilité et ses performances, en travaillant avec un DOM virtuel et en ne mettant à jour le rendu dans le navigateur qu'en cas de nécessité.

React est une bibliothèque qui ne gère que l'interface de l'application, considéré comme la vue dans le modèle MVC (Modèle - Vue - Contrôleur). Elle peut ainsi être utilisée avec une autre bibliothèque ou un autre framework MVC.

React n'utilise pas de système de templates et ne fonctionne qu'avec du JavaScript, permettant une encapsulation complète du composant au sein d'une unique classe.

Pour faciliter l'écriture de la vue, React possède son propre langage, JSX, qui permet de générer des objets Javascript avec une notation similaire à HTML (aussi simple d'accès et déjà bien connue des développeurs).

Développement de React Native depuis 2015, framework permettant de créer des applications cross plateformes iOS et Android, selon le même modèle.

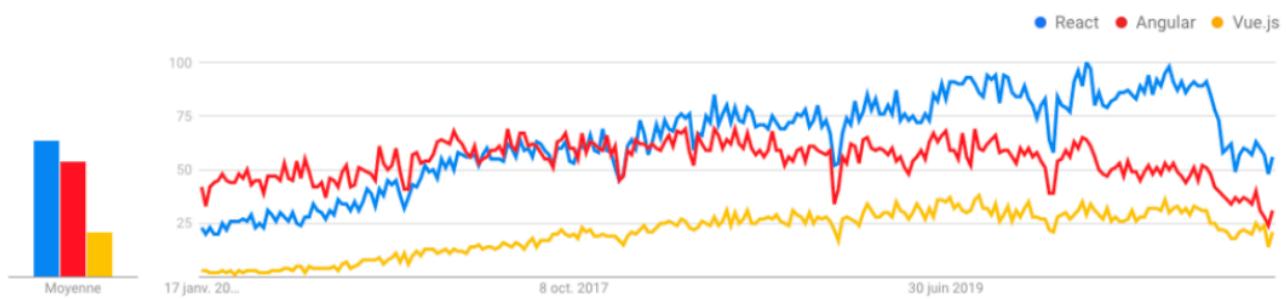
Les forces de React résumées en quelques mots clés :
SIMPLICITÉ - FLEXIBILITÉ - LÉGERETÉ - POPULARITÉ

Voyons maintenant quelques chiffres de l'évolution de React par rapport à d'autres frameworks front-end. (React est souvent considéré comme un framework par abus de langage, c'est en réalité, comme on l'a vu, une librairie JavaScript).

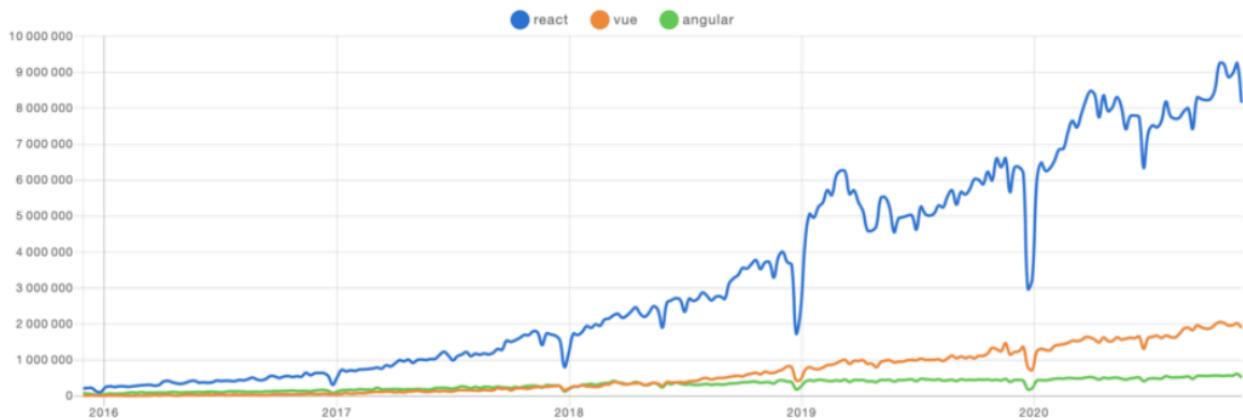
La popularité et l'évolution d'une technologie reflétant souvent les offres d'emploi et les perspectives d'embauche, je me suis penché sur ces deux critères pour faire un choix pertinent quant à la direction à donner à mon apprentissage, la finalité de ma formation étant de trouver un emploi et de pouvoir évoluer au sein d'un milieu passionnant, stimulant et porteur.

React face à ses concurrents en quelques statistiques :

Évolution des recherches Google dans le monde entier, sur les 5 dernières années (source : Google Trends) :



Nombres d'installations de React via l'outil NPM (source: thetribe.io):



Informations GitHub (source: GitHub) :

	stars 🌟	issues ⚡	updated ✅	created 🎉	size 🏋️
.react	159 882	667	Dec 1, 2020	May 24, 2013	minzipped size 2.8 KB
.vue	175 994	545	Nov 27, 2020	Jul 29, 2013	minzipped size 22.9 KB
.angular	59 527	465	Oct 25, 2020	Jan 6, 2010	minzipped size 62.3 KB

2.5 - Le projet SportEasy : l'application web en détail

Avant de présenter en détails les travaux effectués durant le stage, il me semblait pertinent d'introduire ici le fonctionnement global de l'application web, et la manière dont elle a été pensée. La compréhension de ces éléments a été essentielle et m'a permis d'appréhender le projet correctement.

Structure du projet :

- Le projet est construit sous forme de modules indépendants les uns des autres que l'on retrouve dans `src/modules`.
- Les composants réutilisables sont dans un module particulier : `common-ui`.

TypeScript :

- Le projet front-webapp est codé majoritairement en **TypeScript**, comme vu précédemment.
- Il faut veiller à bien **typer** la webapp, en particulier le client API afin de pouvoir faire des mocks pour réaliser les tests fonctionnels. Le type any est à bannir sauf cas obligatoire.

Hook et fonctions :

- Tous les composants de la webapp sont des fonctions :

```
export const Component = () => {
  return <div>Je suis un composant</div>;
};
```

- Les composants utilisent pour la plupart les **hooks** tels que : `useEffect()`, `useState()`, et `useTranslation()`.

Gestion des droits :

- La gestion des droits est effectuée grâce au hook `usePermissions()` défini dans le module `front-utils`.

Il s'utilise de la façon suivante :

Récupération des droits de l'équipe :

```
const [teamPermissions] = usePermissions('team');
```

Vérification du droit update_team_account :

```
teamPermissions.includes('update_team_account');
```

Client API :

Le module `sporteasy-api-client` est composé en plusieurs partie :

- des fichiers contenant les routes API pour chaque requête.
- le typage des payload.
- les fixtures pour les tests fonctionnels.

Feature toggling :

- L'url `/d**/*****` permet d'activer ou de désactiver les features en développement. Cela permet de voir et tester en preprod ou en prod des features pas encore actives.
- La gestion des features se fait dans `features.ts`.
- On active les features dans le fichiers `index.ts`.
- On vérifie qu'une feature est activée avec `isFeatureEnabled()`.

Traductions :

Les traductions sont stockées dans le projet front-webapp dans le dossier `front-webapp/src/translations`.

Pour les mettre à jour, il faut exécuter la commande :

```
front-webapp % npm run translations-update
```

Cela va alors pousser les labels inexistant sur **POEditor** et récupérer les traductions qui ont été faites sur PO. Ensuite il suffit de faire un commit avec ces traductions et le tour est joué.

2.6 - Les outils à notre disposition

Comme évoqué au début, SportEasy m'a dès le début débloqué l'accès à tous les logiciels utilisés par l'équipe produit. C'est l'utilisation combinée de tous ces outils au quotidien qui m'a permis de me familiariser avec les bonnes pratiques professionnelles attendues au sein d'une équipe web et de gagner en performance et en professionnalisme.



GitLab est un logiciel libre de forge basé sur git. L'utilisation d'un logiciel de ce type, indispensable à un niveau professionnel et pour le travail en équipe, m'a permis de me familiariser avec le langage git, les commit, les push et les différentes commandes possibles.



Zeplin est un logiciel qui permet aux graphistes et aux concepteurs (travaillant sur Sketch et/ou Photoshop) de collaborer simplement avec les développeurs. Zeplin m'a permis de développer la rigueur, l'exigence et la précision, des qualités essentielles pour bien reproduire une maquette.



Notion est une application de prise de notes, de bases de données, de tableaux, de wikis, de calendriers et de rappels. Idéal pour les prises de notes à plusieurs, partager de la documentation et effectuer des suivis de tâches. Je l'ai beaucoup utilisée au début afin de lire la documentation existante pour développer mon autonomie sur le projet.



Clubhouse (*renommé très récemment Shortcut*) est une plateforme de gestion de ticket et de projet, intuitive et puissante.

Sorte de to-do list évoluée, chaque tâche à accomplir est représentée par un ticket que l'on peut ensuite trier dans différents catégories (à faire, en développement, en prod, etc...).



Slack est une plateforme de communication collaborative, à destination des professionnels. Indispensable dans un contexte de télé-travail, Slack permettait de rester en contact, se partager des fichiers, et centraliser les rooms Google Meet pour les différentes réunions.



POEditor est un service de localisation en ligne, parfait pour gérer des projets de traduction collaborative. Utile pour traduire des sites Web, des applications, ou des jeux. Intégré à la webapp, il permet d'ajouter les labels directement sur la plateforme en ligne, prêts pour les traductions en 8 langues différentes.

2.7 - Les méthodes : une organisation d'équipe

L'équipe produit a en charge le développement et la maintenance de l'application web et mobile de SportEasy. Elle est composée de :

1 chargé de produit (Head of Product)

2 développeurs back-end

1 développeur front-end

1 développeur mobile (iOS et Android)

2 designers



Être intégré à cette équipe m'a permis de découvrir l'aspect collectif du développement web et les différents métiers s'y rattachant, dans un contexte professionnel. J'ai dû apprendre à travailler en cohérence par rapport au travail des graphistes, à répondre avec précision aux demandes d'un chef de produit, à cordonner mon travail avec celui des autres développeurs, et à gérer les priorités.

Sous la tutelle du développeur front-end, j'ai également été amené à travailler avec les développeurs back-end, notamment pour faire le lien entre les données attendues en front-end et les données renvoyées par l'API.

J'ai également collaboré avec les designers, pour demander des assets manquants ou échanger autour d'une maquette, de l'UX ou des composants existants.

Tout ce travail a été effectué sous la coordination du Chargé de produit qui fixait les orientations à prendre sur les sujets en cours.

Une organisation d'équipe bien rodée...

Plusieurs réunions rythment notre travail au sein de SportEasy et favorisent la cohésion, l'esprit d'équipe et l'innovation. J'ai pu participer à chacune d'entre elles, poser des questions, et proposer mes idées.

Réunions récurrentes :



Stand-up

Réunion quotidienne de ~15min avec toute l'équipe produit, sur les avancées de chacun.



Grooming

Réunion hebdomadaire de préparation à un nouveau projet, réflexions et brainstorming.



Réunion d'équipes

Réunion hebdomadaire regroupant toutes les équipes SportEasy (produit, ventes, etc...)



Sprint planning

Toutes les 2 semaines : répartition des tâches et des tickets, estimation du temps et du coût de chaque tâche.

3 - Développement

3.1 - Objectifs, missions, et contraintes



Principal objectif :

L'objectif principal consistait à migrer les templates et les vues du projet back-end historique de SportEasy vers le projet front-end : la webapp React.



Les contraintes :

- Reprendre toutes les features existantes sur les anciennes pages.
- Utiliser au maximum les routes déjà existantes de l'API et m'adapter aux données qu'elles renvoient pour utiliser le moins possible de ressource back-end.
- Respecter rigoureusement les maquettes des designers.
- Utiliser les composants déjà présents dans le projet React, en les adaptant si besoin, et en créer des nouveaux uniquement si nécessaires.
- Rendre mon code homogène par rapport au code de la webapp en utilisant les mêmes pratiques de nommage, d'import, etc...

Missions secondaires :

En parallèle de ce long travail, d'autres missions, plus secondaires, m'ont été confiées : résolution de divers bugs, fix de design, assimilation et utilisation des outils de l'équipe produit...

3.2 - Mes principaux travaux

3.2.1 - La « Home Team »

Le sujet :

Premier sujet à traiter : le cas de la page d'accueil quand un joueur, un coach ou l'administrateur d'une équipe se connecte.

Après connexion, l'utilisateur se voit alors redirigé automatiquement vers le dashboard de son équipe, tableau de bord centralisant plusieurs informations essentielles à la vie de son équipe comme : les coordonnées et les statistiques de la saison en cours, le dernier évènement en date et le prochain, la dernière conversation, etc...

C'est cette page d'accueil que l'on a appelée, au sein de l'application, la Home Team.

Les objectifs :

N'ayant pas été modifiée depuis les débuts de SportEasy, et n'ayant donc toujours pas été migrée vers l'application React, l'aspect de la Home Team actuelle était quelque peu austère et décalé par rapport au reste de l'expérience utilisateur. L'objectif était donc de remédier à cet état de fait en migrant cette page vers la webapp pour harmoniser le design avec le reste de l'application.

Le second objectif était de pouvoir proposer une expérience responsive en faisant en sorte que la Home Team s'adapte parfaitement à tous types de formats d'écrans (smartphone, tablette, etc...).

De plus, cette page a une importance particulière pour SportEasy. En effet, étant la première page visitée lors de la connexion d'un utilisateur, elle doit être agréable visuellement et donner envie de poursuivre l'utilisation de l'application, en reflétant une expérience positive. Cette page doit incarner et représenter SportEasy de la meilleure des manières.

Les contraintes spécifiques à la Home Team :

En plus des contraintes déjà évoquées précédemment et communes à tous les missions qui m'ont été confiées durant le stage, le sujet de la Home Team comportait aussi quelques difficultés supplémentaires, comme le fait de devoir adapter le comportement de la page selon plusieurs critères : l'utilisateur connecté a-t-il les droits pour effectuer des modifications sur l'équipe ? L'équipe est-elle en version gratuite ou en version payante ? (*certaines features étant réservées à la version payante dite « premium »*).

Les étapes préalables au développement :

Avant de commencer le développement à proprement parlé, il nous a fallu respecter une série d'étapes préalables et obligatoires au bon déroulement du projet. Ces étapes m'ont aidées à gagner du temps de développement en structurant le travail et les objectifs dès le départ, en accord avec les attentes de l'équipe produit.

1. Réunion avec toute l'équipe produit (*grooming*) pour valider les objectifs, les visuels et les différents comportements de la page, évaluer la faisabilité et la durée de la tâche, et évaluer les contraintes ou difficultés que je pourrais rencontrer.
2. Création du module « dashboard », réflexions autour de l'organisation par composants et de l'arborescence du dossier.
3. Mise en place de la route front-end et intégration du module au reste de l'application React.
4. Développement d'un feature toggle (*cf. la partie 2.5 - Le projet SportEasy : l'application web en détail, page 13*) qui permettra de « cacher » cette page durant son développement et de pouvoir effectuer des tests en préprod et en prod le moment venu.

Une fois ces étapes validées, tout est en place pour commencer à développer efficacement.

React, une organisation par composant :

Notre nouveau module `dashboard` s'articule donc selon le modèle des autres modules de la webapp, à savoir :

Deux dossiers se trouvent à la racine du module : un dossier `container` et un dossier `components`.



`src/modules/dashboard/container`

Le dossier `container` contient le composant parent, nommé selon le nom du module, soit ici `dashboard.tsx`. C'est ce composant parent qui gère toute l'intelligence de la page (les variables, les appels à l'API, les fonctions, etc...). Ce composant se contentera d'appeler les composants « enfants » en leur transmettant via des **Props** les données nécessaires à la bonne exécution de la page.

`src/modules/dashboard/components`

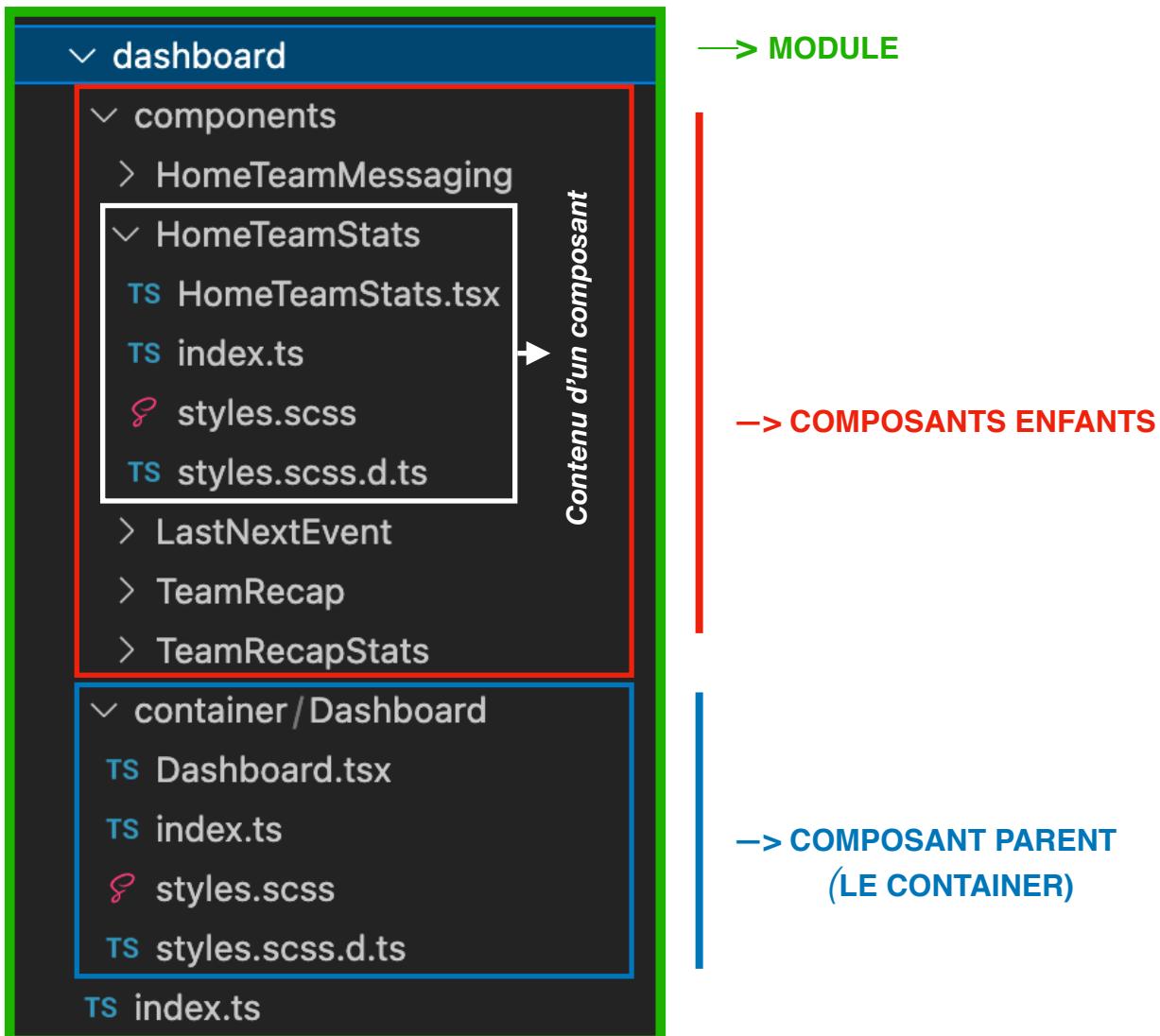
Le dossier `components` contient des sous-dossiers, un par composant. Ces composants « enfants » se contenteront de renvoyer de l'affichage et ne gère pas l'intelligence de la page (*regroupée au maximum dans le container*) mais peuvent contenir quelques fonctions secondaires si nécessaire.



Chaque dossier contenant un composant contient les mêmes fichiers :

- Le composant en lui-même, exemple : `HomeTeamStats.tsx`
- un fichier d'export : `index.ts`
- Un fichier pour le style du composant uniquement : `styles.scss`
- Un fichier pour synchroniser le scss : `styles.scss.d.ts`

Illustration de l'arborescence du module `dashboard` :



Cette découpe par composant, élément essentiel du développement en React, permet une **meilleur maintenabilité** du code et une **meilleure lisibilité** (*grâce à des fichiers séparés, ayant chacun leur rôle, et contenant moins de ligne*), une **grande facilité de refactorisation** (*on repère et réutilise facilement des fonctions ou des bouts de code qui se répètent*), et permet une **grande réutilisation des composants déjà existant** (*en appelant simplement un composant déjà construit*).

Le développement, exemples de code :

Pour illustrer mes pratiques de développement et mon travail concernant la réalisation de la Home Team, j'ai choisi de présenter le composant « parent » `dashboard.tsx`.

Pour ce faire, on verra, dans l'ordre, les 3 grandes parties formant un composant React classique :

1

Déclaration de l'état du composant
(variables et variables d'état)

2

Logique JavaScript
(appels API et fonctions)

3

JSX
(plus précisément du TSX car nous utilisons TypeScript)

1

Déclaration de l'état du composant :

```
/* -----
| DASHBOARD CONTAINER |
----- */

export const Dashboard = () => {
  const [t] = useTranslation();
  const store = useContext(StoreContext);
  const [loading, setLoading] = useState(true);
  const [team, setTeam] = useState(null as ApiTeam | null);
  const [teamStats, setTeamStats] = useState(null as ApiTeamRecap | null);
  const [lastNextEvents, setLastNextEvents] = useState(null as TeamApiLastNextEvents | null);
  const [selectedThread, setSelectedThread] = useState(null as number | string | null);
  const [socket, setSocket] = useState(null as Pusher | null);
  const [channel, setChannel] = useState(null as any);
  const [categories, setCategories] = useState([] as SelectInputOption[]);
  const [allCategories, setAllCategories] = useState([] as TeamApiListEventCategory[]);
  const [selectedCategory, setSelectedCategory] = useState(null as any);
  const [stats, setStats] = useState(null as ApiTeamStatsChartsData | null);

  const emptyCategories = [{ label: t('No category'), value: '1' }];
  const isGeneric = !(store.team && store.team.sport.slug_name === 'generic-sport');
```

On voit ici la déclaration du composant Dashboard et les variables d'état : déclarées au début du composant, elles permettent de stocker les réponses de l'API, de gérer le loading, ou de gérer des modules complémentaires (*comme Pusher pour la messagerie*). Leur état est initialisé grâce au hook `useState` et typé selon la syntaxe **TypeScript** pour éviter les erreurs de typage propres à JavaScript. On retrouve également la déclaration et l'initialisation de deux variables « classiques » (`emptyCategories` et `isGeneric`).

2 Logique JavaScript :

À noter : Pour avoir un chargement uniforme de la page, chaque call à l'API sera stocké dans une « promise », ces promises (*promesses*) s'exécuteront en même temps, une fois tous les appels effectués.

```
useEffect(() => {
  const promises: Promise<any>[] = [];

  // GET LAST/NEXT EVENTS DATA
  const promiseLastNext = apiClient.team.event
    .getLastNextEvents({
      id: TEAM_ID,
    })
    .then(response => setLastNextEvents(response));
  promises.push(promiseLastNext);

  // GET LAST THREAD
  const promiseMessaging = apiClient.team.messaging
    .getThreads({
      teamId: TEAM_ID,
    })
    .then(response => {
      if (response && response.results[0]) {
        setSelectedThread(response.results[0].id);
      }
    });
  promises.push(promiseMessaging);

  Promise.all(promises).then(() => setLoading(false));
}, []);
```

Je déclare mon tableau de promesses au début du hook `useEffect`. À l'intérieur de celui-ci, chaque appel à l'API sera stocké dans une nouvelle promesse.

Je récupère donc les données de l'API via la route correspondante, en utilisant la (ou les) clé(s) nécessaire(s), ici l'id de l'équipe.

Je stocke la réponse retournée par l'API (*en la filtrant si besoin*) dans une variable d'état et j'envoie cette promesse dans mon tableau de promesses déclaré au début du `useEffect`.

Une fois tous les appels à l'API effectués, j'exécute les promesses et réinitialise la variable d'état `loading` sur `false` une fois cette exécution terminée, afin de déclencher l'affichage de la page.

On a ainsi une page qui chargera d'un seul bloc, de manière homogène et fluide pour l'utilisateur.

3 Le TSX (TypeScript XML) :

Une fois l'état de la variable `loading` passé à `false`, on peut déclencher l'affichage de la page. Le composant `<Loader />` gère l'affichage du chargement en attendant ce changement d'état.

On retrouve à l'intérieur du TSX l'**organisation par composant** décrite précédemment. Chaque composant est appelé, et on lui passe **les Props nécessaires à son fonctionnement**, à l'aide de nos variables d'état déclarées au début du composant, et initialisé dans le `useEffect` (*où à travers des fonctions*).

On peut également voir l'utilisation de **Sass** avec le mot clé `className`. Le fichier de style pour ce composant a été importé au tout début et nommé simplement « `styles` » ce qui permet d'y faire référence à chaque nommage d'une classe CSS.

À noter également la réponses aux contraintes posées au début, sur le comportement de la page en fonction de divers critères. L'injection de code TypeScript dans le TSX permet de **conditionner l'affichage de certains composants** au contenu de variables ou à des fonctions déjà présentes dans le module `front-utils`.

Par exemple : le module `<HomeTeamStats />` ne s'affichera que si l'équipe est premium (via la fonction `isPremium()`) et si c'est un sport non générique (via la variable `isGeneric`).

```

if (loading) return <Loader />;
return (
  <>
    <Title>{t('Dashboard')}</Title>
    <div className={styles.dashboardSubTitle}>{t('You can find here all the news of your team')}</div>
    <div className={styles.homeTeamPage}>
      {/* --- LEFT SIDE --- */}
      <div className={isGeneric || !isPremium(store) ? styles.leftSideFree : styles.leftSide}>
        <TeamRecap team={team} teamStats={teamStats} />
        {/* PremiumBlock if not Premium */}
        {!isPremium(store) && (
          <div className={styles.homePremiumBanner}>
            <PremiumBanner />
          </div>
        )}
        {/* LastNextMatchBlock */}
        <div className={styles.lastNextMatchBlock}>
          <LastNextEvent event={lastNextEvents.last_event} isPast />
          <LastNextEvent event={lastNextEvents.next_event} isPast={false} />
        </div>
        {/* TeamMessagingBlock */}
        <div className={styles.thread}>
          <HomeTeamMessaging
            lastThreadId={selectedThread}
            socket={socket}
            channel={channel}
            onFetchThreads={onFetchThreads}
          />
        </div>
      </div>
      {/* --- RIGHT SIDE --- */}
      {/* TeamStatsBlock if is Premium and not Generic, with stats or not*/}
      {isPremium(store) && !isGeneric ? (
        <div className={styles.rightSide}>
          <div>
            <HomeTeamStats
              categories={stats ? categories : emptyCategories}
              selectedCategory={selectedCategory}
              setSelectedCategory={setSelectedCategory}
              allInfosCategories={allCategories}
              stats={!stats ? undefined : stats}
              isEmpty={!stats}
            />
          </div>
        </div>
      ) : null}
    </div>
  </>
);

```



Composants développés pour ce projet



Composants déjà existants

Dashboard

Vous pourrez retrouver ici toute l'actualité de votre équipe.



Les Olympiens

Sport : Football
Pays : France
Année de fondation : 2021

32 Membres | 4 Saison | 115 Matchs | 650 Points

Ajouter un numéro de téléphone
 olympiens_bureaux@gmail.com

Dernier événement

Mardi 7 avril 2021 - 17h00

FC Olympiens
FC Footix

[Voir le calendrier](#)

Prochain événement

Mardi 14 avril 2021 - 17h00
Ligue des champignons

FC Issy
FC Olympiens

[Voir le calendrier](#)

Dernier message

[Voir la messagerie](#)

Dimanche 23 avril 2019

Albin Egasse 07:44



Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Nicolas Potier Maintenant



Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod ?

[Écrire un message...](#)[Envoyer](#)[Créer un club](#)[Passer Premium !](#)[Gagnez 20€](#)[Bons plans](#)[Mes équipes](#)[Aurélien Bellis](#)[ACHETER](#)

MAÎTRISE L'ART DE LA VITESSE

SPEEDFLOW

Dashboard

Consultez l'actualité de votre équipe.



Bréal Fc

Sport : Football
Pays : France
Année de fondation : 1920

Ajouter un numéro de téléphone
 Ajouter une adresse e-mail

29 Joueurs | 1 Saison | 2 Matchs | 13 Buts

SPORTEASY PREMIUM

- ✓ Pas de publicité
- ✓ Nombre de membres illimité
- ✓ Statistiques des joueurs
- ✓ Et plein d'autres fonctionnalités...



Profitez des fonctionnalités en illimité !

[Passer Premium !](#)

Dernier événement



Prochain événement

[? Aide](#)

Conclusion concernant ce premier projet :

Sorte de « tutoriel géant », ce projet m'a permis de me familiariser avec la webapp de SportEasy et surtout d'aborder et d'appréhender toutes les notions essentielles au développement front-end et plus particulièrement avec React et TypeScript.

Il était pertinent et primordial de commencer avec ce sujet, qui m'a permis d'acquérir les connaissances nécessaires pour les futurs projets que l'on va me confier par la suite et pour être en mesure de corriger certains bugs ou d'appliquer certains quick fix.

Il m'a fallu 2 semaines pour mener à bien ce projet, mais j'ai continué à y apporter des modifications jusqu'à la quasi toute fin de mon stage, pour modifier un élément de design, pour améliorer le code et son optimisation, ou pour répondre à des demandes des designers ou de l'équipe produit.

À la fin du projet, avant la mise en prod, SportEasy m'a laissé la responsabilité de présenter mon travail et la nouvelle Home Team à toute les équipes, à travers une démonstration d'une quinzaine de minutes, filmée et retransmise sur les réseaux internes à SportEasy.

Les compétences apportées par ce projet :



Les notions essentielles de React
(*Composants, Hooks, Props, JSX*)



Le fonctionnement d'une API



Le typage et les particularités de TypeScript



Des notions de Sass et de CSS
(*Flexbox, pseudo-classes, variables, mixins, etc...*)

3.2.2 - La page des Présences

Le sujet :

Il s'agit cette fois de migrer toute la page des présences d'une équipe. Cette page comporte, comme son nom l'indique, le détail des présences, par joueur, sur chaque évènement passé de l'équipe (match de championnat, entraînement, match de coupe, etc...), sous forme de tableau.

La page contient 3 onglets : un onglet récapitulatif de toutes les présences, un onglet détaillant les présences par match, et un onglet concernant les tâches attribuées aux joueurs. Des filtres doivent permettre de trier les données par saison, par catégorie de match et par rôle de membre (joueur, coach, etc...), et chaque colonne des tableaux doit pouvoir être trier en ordre croissant ou décroissant. L'ensemble de la page doit bien sûr être parfaitement responsive.

Le contexte :

Les objectifs restent à peu près les mêmes que pour le sujet précédent : migrer la page vers React, améliorer le design et l'UX, et proposer une expérience responsive optimale.

Les contraintes spécifiques à cette page réside surtout dans le fait que l'API ne permettait pas de recevoir les données nécessaires à la réalisation de cette page. Il m'a donc fallu intégrer des fixtures et typer ces jeux de données moi-même.

Les étapes préalables de développement sont communes à celles de la Home Team, selon l'organisation de SportEasy.

L'**organisation par composant** et sa logique également, mais cette fois-ci, nous aurons 3 containers, un pour chaque onglet. Chaque tableau de chaque onglet sera décomposé en plus petit composant, avec des composants communs si possible.

Le développement, exemples de code :

Comme expliqué ci-dessus, faute de ressources back-end nécessaires, j'ai commencé le travail front-end avant la création des routes API nécessaires, pour gagner du temps.

J'ai donc créé un fichier de fixtures dans le répertoire approprié : [src/modules/sporteasy-api-client/fixtures/default/presence.ts](#), et documenté la page Notion dédié aux Présences avec le typage des données, à l'attention des développeurs back-end.

Extrait du Notion concernant le sujet des Présences :

Pour développer l'interface, nous avons créé des jeux de données avec le typage suivant, dans l'idéal, lorsque l'on développera les routes API il faudrait qu'elles respectent strictement ces types :

```
JavaScript ▾
Copy Caption ...
// Presences stats

GET v2.1/teams/${params.teamId}/stats/presences/

export type ApiPresencesStatsPayload = {
    season_slug: string;
    role: string;
    category_slug: string;
    presence_stats: [
        {
            total_events_count: number;
            ranking: {
                profile: ApiProfile;
                stats: {
                    injured: number;
                    not_excused: number;
                    waiting_list: number;
                    on_time: number;
                    total_selection: number;
                    not_selected: number;
                    rank: number;
                    late: number;
                    excused: number;
                    total_absent: number;
                    total_present: number;
                };
            }[];
        };
    ];
};
```

Une fois la réflexion autour du typage terminée, et une fois les fausses données créées, le développement peut réellement commencer.

NB : ces fixtures serviront également pour le Club de Découverte, démo en ligne de SportEasy à but commercial, disponible sur le site vitrine, permettant de tester l'application web avant de s'inscrire (nécessite donc des fausses données).

La mise en place des 3 onglets de la page :

```
<SegmentedControl
  tabs={[
    {
      label: t('Synthesis'),
      url: `/presences/synthesis/`,
      id: 'PresencesSynthesis',
    },
    {
      label: t('Per events'),
      url: '/presences/events/',
      id: 'PresencesEvents',
    },
    {
      label: t('Chores assignment'),
      url: '/presences/chores',
      id: 'PresencesChores',
    },
  ]}
/>
{isPremium(store) && TEAM_ID ? (
  <Switch>
    <Route path="/presences/synthesis/" render={() => <Synthesis />} />
    <Route path="/presences/events/" render={() => <Events />} />
    <Route path="/presences/chores/" render={() => <Chores />} />
    <Redirect from="/" to="/presences/synthesis/" />
  </Switch>
) : (
  <Block>
    <PremiumContent />
  </Block>
)}
```

Cette feature étant réservé aux équipes premium on note ici une condition ternaire permettant, soit d'afficher les différents onglets, soit le composant <PremiumContent />, invitant à passer premium.

Focus sur l'onglet Bilan (appelé ici Synthesis) :

Extraits du container **Synthesis.tsx** :

Pour fonctionner, le tableau de cet onglet a besoin de deux informations : le titre des colonnes (absent, à l'heure, blessé, etc...), et les données correspondantes, par membre de l'équipe.

```
export type ColumnLabelType =  
  | 'injured'  
  | 'not_excused'  
  | 'on_time'  
  | 'total_selection'  
  | 'not_selected'  
  | 'late'  
  | 'excused'  
  | 'total_absent'  
  | 'total_present'  
  | 'waiting_list'  
  | 'rank';  
  
export type PresenceStatsRankingRow = {  
  profile: ApiProfile;  
  stats: {[key in ColumnLabelType]: number};  
};  
  
export type PresenceStatsRanking = PresenceStatsRankingRow[];  
  
export type Headers = ColumnLabelType[];
```

Je prépare donc le type de mes variables en conséquences. **PresenceStatsRanking** et **Headers** étant deux tableaux, cela me permettra, via la méthode **.map()**, de boucler dessus et ainsi avoir un composant plus court et plus lisible.

Mes variables d'état, selon le typage ci-dessus :

```
const [sortedRanking, setSortedRanking] = useState([] as PresenceStatsRanking);
const [headers, setHeaders] = useState([] as Headers);
const [presencesLoading, setPresencesLoading] = useState(true);
```

Je récupère ensuite les données avec le hook `useEffect` :

```
useEffect(() => {
  apiClient.team.stats.getPresencesStats().then(response => {
    if (response) {
      setSortedRanking(response.presence_stats.ranking);
      const prepareHeaders = Object.keys(response.presence_stats.ranking[0].stats).filter(
        h => h !== 'rank' && h !== 'waiting_list'
      ) as ColumnLabelType[];
      setHeaders(prepareHeaders);
    }
    setPresencesLoading(false);
  });
}, []);
```

La méthode `.filter()` me permet de filtrer les données reçues pour garder que le nécessaire, et la méthode `.keys()` me permet de filtrer sur les clés de l'objet et non pas sur les valeurs de ces clés.

Je peux maintenant envoyés ces données via des Props à mon composant enfant `<SynthesisTable />` :

```
return (
  <Block noPaddingOnMobile>
    <CommonHeader tab="synthesis" />
    {!sortedRanking || presencesLoading ? (
      <Loader />
    ) : (
      <>
        <SynthesisTable sortedRanking={sortedRanking} headers={headers}>
        </SynthesisTable>
      </>
    )}
  </Block>
);
```

Bien sûr, j'ai typé les Props du composant enfant selon les données que je veux lui envoyer, en exportant les types déclarés ci-dessus, pour minimiser au maximum le risque d'erreur.

Extraits du composant `SynthesisTable.tsx` :

Ici, deux variables d'état seront nécessaires.

La première, `sortedRows`, est initialisé avec les données de l'API via la Props envoyée au composant et sera réinitialiser pour permettre les tris croissant et décroissant.

La deuxième, `sorting`, est un objet contenant le nom de la colonne à afficher et à trier, et un boolean permettant de switcher entre tri croissant et décroissant.

```
const [sortedRows, setSortedRows] = useState(sortedRanking);
const [sorting, setSorting] = useState({ column: headers[0], asc: true } as {
  column: ColumnLabelType;
  asc: boolean;
});
```

Fonction `sortStats()`, déclenchée au clic d'une colonne, permettant de trier les données :

```
const sortStats = (s: { column: ColumnLabelType; asc: boolean }) => {
  const r = [...sortedRanking];
  const columnToSort = s.column;
  if (s.asc) r.sort((a, b) => b.stats[columnToSort] - a.stats[columnToSort]);
  if (!s.asc) r.sort((a, b) => a.stats[columnToSort] - b.stats[columnToSort]);
  setSortedRows(r);
  setSorting(s);
};
```

Cette fonction prend en paramètre un objet du même type que la variable `sorting`. Grâce aux valeurs des clés `column` et `asc`, passées en paramètre, la fonction exécutera le tri adéquat (la bonne colonne, dans le bon ordre) et réinitialisera les variables `sortedRows` et `sorting` en conséquence.

Extrait du TSX du composant :

On peut voir qu'un autre composant, `<SynthesisHeader />`, est appelé. Il gère le header du tableau et prend donc en Props la variable `headers` contenant les titres de chaque colonne, la fonction `sortStats()` vu ci-dessus (car le tri s'effectue au clic sur le nom de la colonne) et la variable `sorting` (pour le symbole croissant ou décroissant).

Illustration du TSX :

```
<div className={styles.right}>
  <SynthesisHeader headers={headers} sortStats={sortStats} sorting={sorting} />
  {sortedRows.map(row => (
    <div className={styles.line}>
      <div className={[classNames(styles.categoryStats, styles.attendances)}>
        <div className={styles.column}>{row.stats.total_selection || '-'}</div>
      </div>
      <div className={[classNames(styles.categoryStats, styles.presences)}>
        <div className={[classNames(styles.column, styles.triple)}>{row.stats.on_time || '-'}</div>
        <div className={[classNames(styles.column, styles.triple)}>{row.stats.late || '-'}</div>
        <div className={[classNames(styles.column, styles.total)}>{row.stats.total_present || '-'}</div>
      </div>
    </div>
  ))}
```

On retrouve aussi la variable **sortedRows** sur laquelle j'applique la méthode **.map()**, pour lister les données selon le tri appliqué.

Les deux autres onglets de la page présences reprennent à peu près le même fonctionnement.

Exemple de problématique rencontrée : un scroll horizontal intelligent !

Constat :

Deux des tableaux de cette page pouvant contenir de nombreuses colonnes (si une équipe a 30 évènements sur la saison en cours par exemple, soit 30 colonnes), un scroll horizontal à l'intérieur du tableau devait être possible pour s'assurer de la lisibilité des données. Une mise en avant de ce scroll en terme de design devait être effectuée pour une UX agréable.

Or, selon les données de l'équipe, il se peut que le tableau ne comporte, au contraire, que 2 ou 3 colonnes... le scroll et le design qui va avec doivent donc être absent de la page.

Solution :

Pour répondre à cette problématique, j'ai choisi d'attribuer un `id` aux balises HTML : la `<div>` qui doit contenir la partie à scroller, et la `<div>` qui doit être scrollable, et d'utiliser un boolean en variable d'état :

```
const [enableScroll, setEnableScroll] = useState(false);
```

```
<div
  ref={divToScroll}
  id="CenterContainer"
  className={classNames(styles.center, { [styles.centerScroll]: enableScroll })}
>
  <div className={styles.line}>
    <div id="ChoresStatsValues" className={styles.valueContainer}>
```

Deux classes CSS sont attribuées à la `<div>` qui contient le scroll, afin que je puisse adapter son style en conséquence, selon la valeur de la variable `enableScroll`, initialiser dans le `useEffect` ci-dessous :

```
useEffect(() => {
  const centerContainer = document.getElementById('CenterContainer');
  const statsValues = document.getElementById('ChoresStatsValues');
  if (centerContainer && statsValues && centerContainer.offsetWidth < statsValues.offsetWidth) {
    const scroll = true;
    setEnableScroll(scroll);
  }
}, [choreContainer.current]);
```

Je récupère les `<div>` via leur `id`, je compare leur largeur via l'attribut `.offsetWidth` et initialise la variable `enableScroll` en conséquence.

J'ai choisi un `useEffect` et non une fonction classique, pour qu'à chaque changement d'onglet, ce `useEffect` s'exécute à nouveau (grâce au callback `choreContainer.current`)

On a donc un tableau qui propose un scroll horizontal avec un design adapté ou un tableau statique classique, en fonction du nombre de colonne, soit en fonction des données de l'équipe.

VISUELS PAGE DES PRÉSENCES

(Version en prod sur le Club de Découverte)

SportEasy

Gagnez 20€ | Bons plans | Mes équipes | Carlos Santana

Présences

Championnat Stratocast... | 2021-2022

Bilan | Par événements **Assignation des tâches**

Liste des status

- on time
- late
- excused
- not excused
- hurt
- not convened

Joueur (17)

15 Membres

	01/02	07/02	14/02	21/02	28/02	03/03	21/03	01/04	07/04
1 Nolan Alma	OK								
2 Rudy Diata	OK								
3 Saber Djamat Dubois	OK								
4 Souleymane Coulibaly	OK								
5 Theo Feuillade	OK								
6 Yabel Kouyouama Mothmora	OK								

15 Membres

Joueur (17)

Convocations	Présences			Absences		Autres		
	A l'heure	Retard	Total	Excusé	Non excusé	Total	Blessé	Non convoqué
26	24	1	25	1	-	1	5	1
30	19	1	20	9	1	10	-	-
19	15	3	18	-	1	1	9	3
10	10	-	10	-	-	-	-	-
18	10	1	11	2	5	7	6	9
20	9	1	10	1	9	10	-	-
18	9	8	17	1	9	10	4	3

Présences

Championnat Stratocast... | 2021-2022

Bilan | Par événements **Assignation des tâches**

Ouvrir le vestiaire

	Clé	Porte	Plaque	Boîte	Caméra	Musique	Tablette	Appareil photo	Carte	Total
1 Saber Djamat Dubois	0	0	0	0	0	0	0	0	1	9
2 Abdi El Kader El Manrouf	3	0	0	0	0	0	0	0	1	9
3 Arthur Robieux	3	0	0	0	0	0	0	0	1	9
4 Bilal Daoudi	3	0	0	0	0	0	0	0	1	9

Conclusion concernant ce deuxième projet :

Projet très intéressant, notamment au niveau du design et plus particulièrement du responsive design (sujet que j'aborderai dans une partie à part), où j'ai pu pousser plus loin les notions acquises lors de la réalisation du premier projet.

Ces notions et ces connaissances m'ont permis d'être plus efficace, plus pertinent dans mes échanges avec les développeurs back-end ou les designers, et plus confiant.

Comme pour le premier projet, il m'a fallu environ 2 semaines pour le terminer. Le projet a ensuite été déployé en prod sur le Club de Découverte (version démo de l'application, disponible sur le site vitrine), et j'ai pu continuer le travail en échangeant avec les développeurs back-end pour les orienter dans leur développement.

De la même manière qu'il m'avait été permis de présenter mon travail sur la Home Team à tous les salariés SportEasy, il en a été de même pour ce second projet, à travers là aussi une démonstration filmée et partagée.

Les compétences apportées par ce projet :

- ✓ La gestion d'inputs via des fonctions entre composants
(Tri de colonnes, filtres)
- ✓ Implémentation de **fixtures** et rédaction de documentation
(À l'attention des autres développeurs)
- ✓ Approfondissement en **syntaxe** et en méthodes JavaScript
(.keys() | .reduce() | .sort() | .find() | etc...)
- ✓ Responsive Design et approfondissement de **Sass**

3.3 - Le Responsive Design

Parce que c'est un élément essentiel du développement front-end, j'ai choisi d'en parler dans une partie dédiée, et de présenter ici quelques exemples de travaux réalisés, dans leur version responsive. J'évoquerai également les outils permettant la gestion du responsive design.

Le fonctionnement de la webapp pour la partie responsive :

Au niveau du CSS :

Il existe au sein de la webapp un fichier `vars.scss` qui regroupe toutes les variables récurrentes (*couleurs, marges, tailles de police, etc...*). C'est dans ce fichier que se trouve le `@mixin` (*morceau de code paramétrable et réutilisable n'importe où dans un fichier sass*) qui permet de gérer facilement le responsive design au sein de chaque fichier `.scss` :

```
@mixin for-mobile-only {  
  @media only screen and (max-width: $responsive-width) {  
    | @content;  
  }  
  @media only screen and (max-width: $responsive-width * 2) and (min-device-pixel-ratio: 2) {  
    | @content;  
  }  
}
```

Exemple d'utilisation du mixin `for-mobile-only` grâce au mot clé `@include` :

```
.rightSide {  
  width: calc(25% - #{$margin-1});  
  @include for-mobile-only {  
    | width: 100%;  
  }  
}
```

Au niveau du JavaScript :

Il peut également être intéressant de conditionner l'affichage d'un composant directement depuis le TSX, selon le retour d'une fonction par exemple.

C'est le rôle de la fonction **isMobile()** qui retourne un boolean selon la taille de l'écran, et que l'on peut appeler n'importe où dans la webapp :

```
export const isMobile = () => {
  return window.innerWidth <= 1024;
};
```

Exemple d'utilisation dans du TSX :

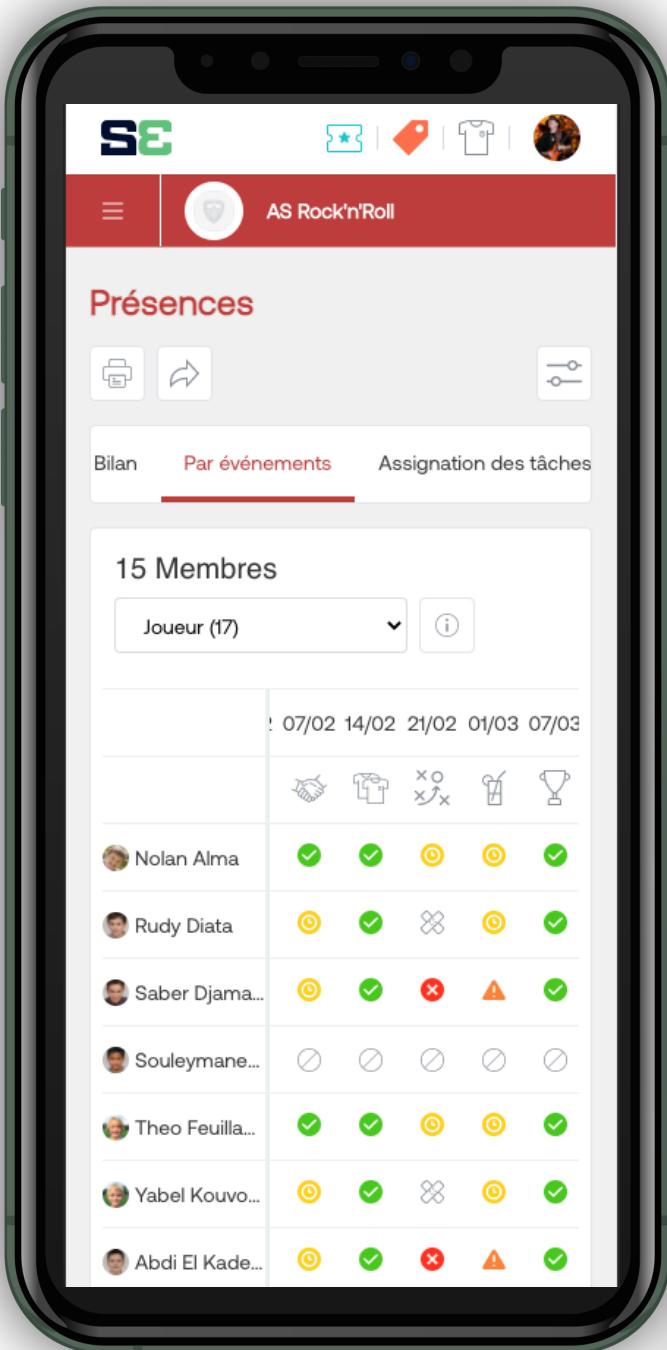
```
{isMobile() && (tab === 'event' || tab === 'synthesis') && (
  <>
    <div onClick={() => setModalLegendIsOpen(true)} role="button" tabIndex={0}>
      <ButtonIcon tooltip={t('Informations')} toolbar shadow={false}>
        <IconInformation size="16" color="gray" />
      </ButtonIcon>
    </div>
```

Cette fonction permettra par exemple d'afficher des modals responsives à la place de filtres classiques pour optimiser l'espace d'écran disponible en version mobile, ou au contraire de ne pas afficher des tooltips dont l'affichage ne correspondrait pas à un écran plus petit, etc...

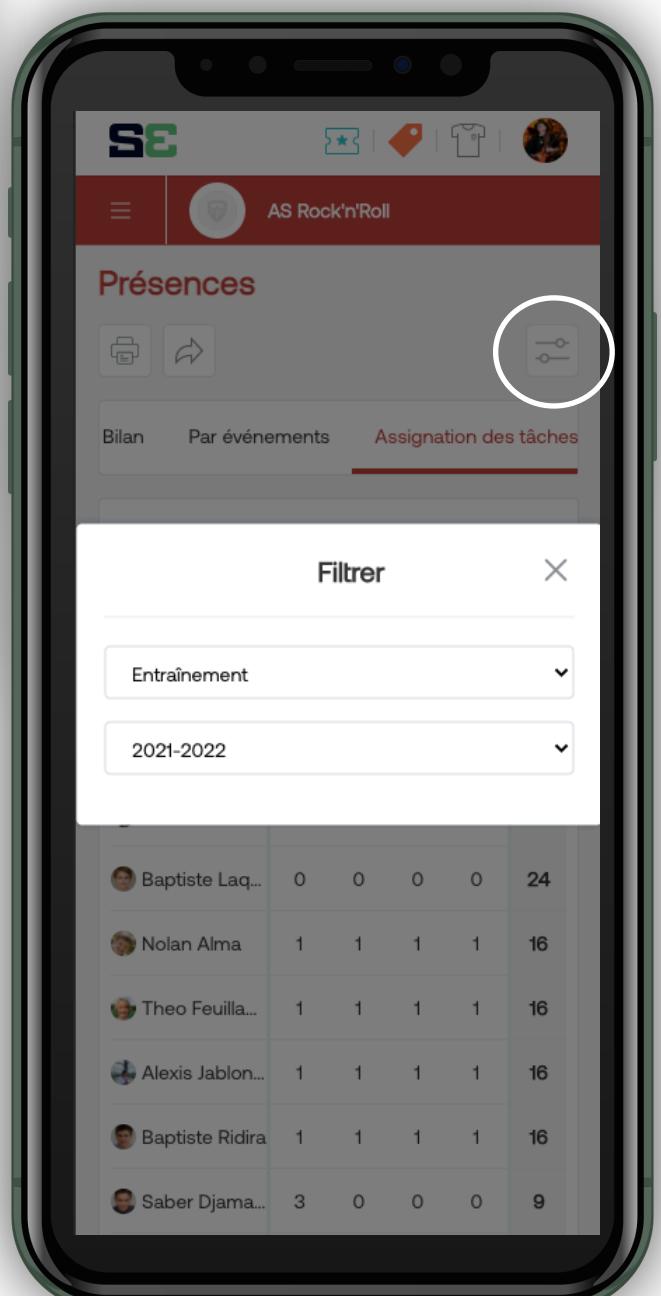
Toute la partie responsive de mes travaux a pu être gérer grâce à ces deux outils très simples, le **mixin Sass** et la **fonction JavaScript**.

Exemples :

Voyons maintenant quelques exemples tirés de mon travail sur la partie responsive sur divers sujets :

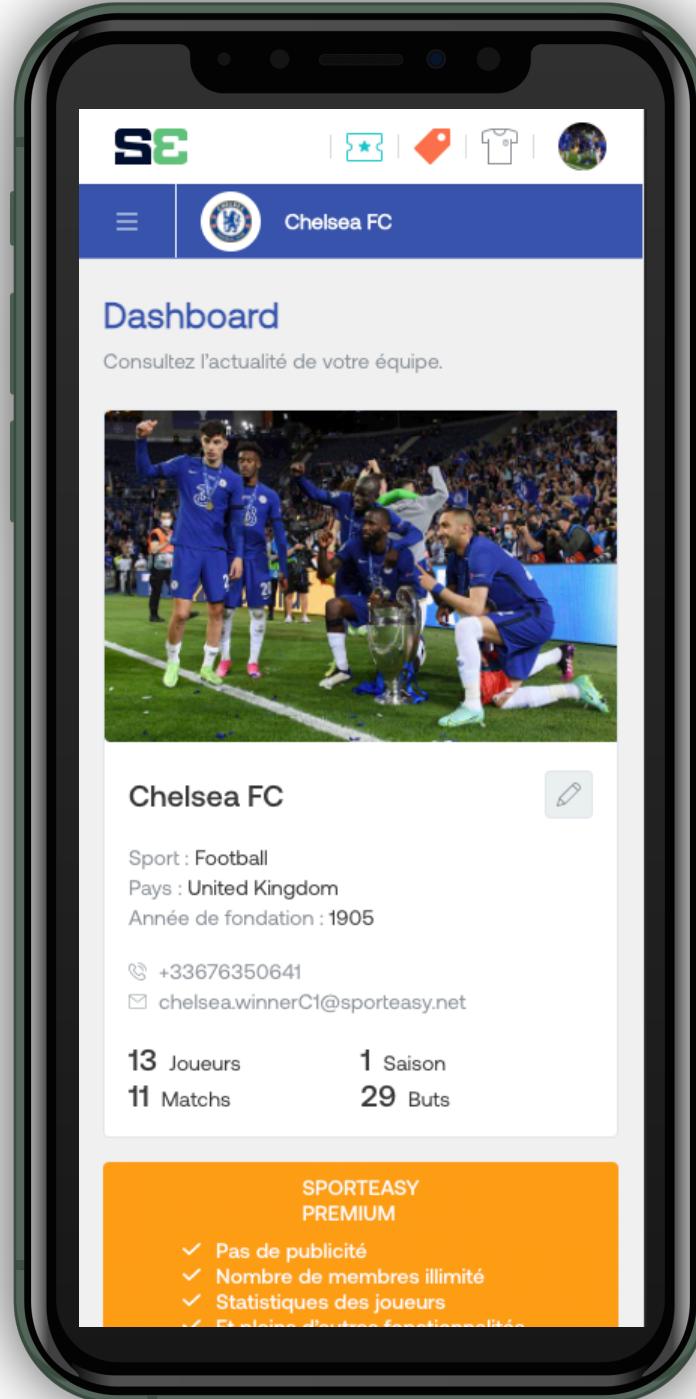


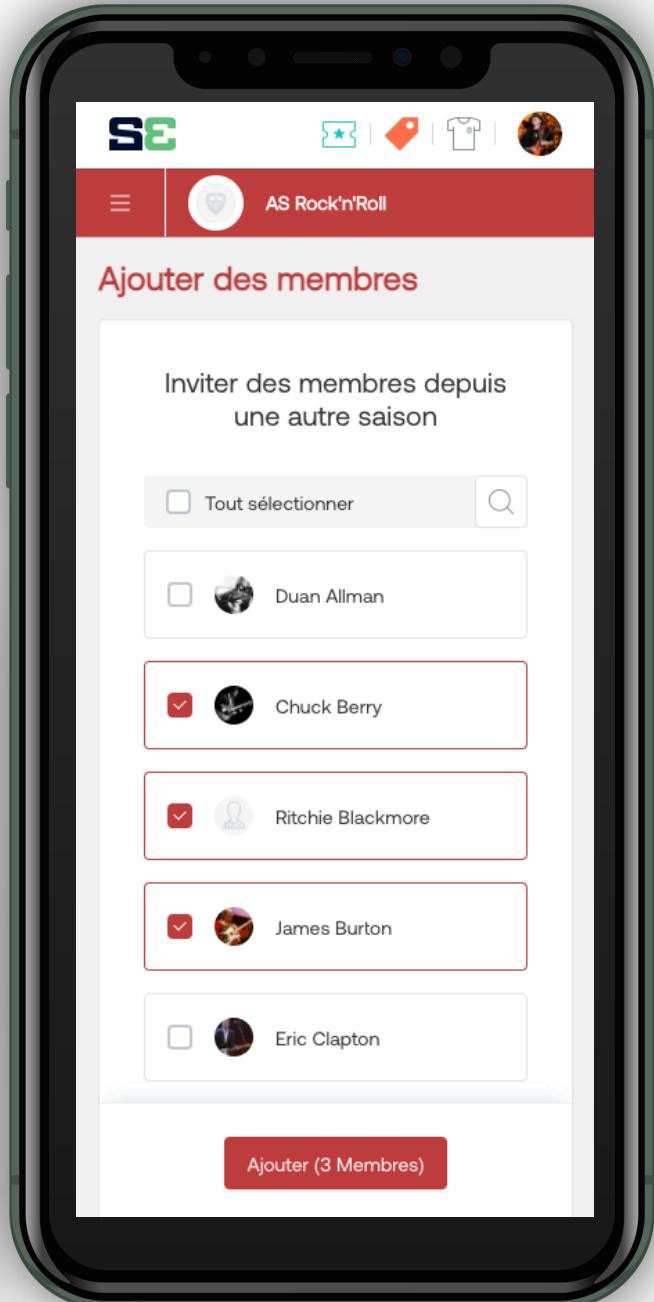
Extrait de la page des
Présences, tableau de l'onglet
« Par évènements »
(scroll horizontal du tableau,
icônes et cellules adaptées)



Extrait de la page des
Présences, ouverture de la
modal contenant les filtres
(apparent sur la version web)

Extrait de la page d'import de membres depuis une autre saison





Ajouter des membres

Inviter des membres depuis une autre saison

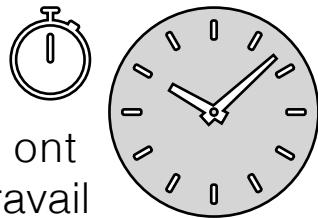
Tout sélectionner

- Duan Allman
- Chuck Berry
- Ritchie Blackmore
- James Burton
- Eric Clapton

Ajouter (3 Membres)

Extrait de la Home Team (*disposition des composants revue*)

3.4 - D'autres travaux...



Les deux sujets présentés précédemment ont représenté environ un tiers de mon temps de travail durant ce stage.

Un autre tiers, en parallèle de ces deux sujets, a été consacré à l'apprentissage théorique et aux actions du quotidien comme la résolution de bugs ou l'application de fix design.

Le dernier tiers, quant à lui, a été occupé à accomplir beaucoup d'autres sujets, certes de moindre envergure, mais tout aussi intéressants et formateurs.

Voici une liste non exhaustive de quelques exemples de sujets plus secondaires que j'ai été amené à traiter :

Gestion des coachs depuis une page dédiée :

The screenshots show the 'Gérer les coachs' (Manage Coaches) feature in the SportEasy app. The top screenshot shows two coaches assigned: Kurt Cobain and Carlos Santana. The bottom screenshot shows three coaches assigned: Chuck Berry, Ritchie Blackmore, and Carlos Santana. Both screens include a search bar and an 'Assigner ces coachs' (Assign these coaches) button.

Coachs assignés
Kurt Cobain, Carlos Santana
Chuck Berry, Ritchie Blackmore, Carlos Santana

Il s'agissait ici de proposer aux administrateurs d'équipes une page leur servant à assigner des coachs depuis un lien présent sur la page Effectif. C'est une feature qui existait déjà pour au sein d'un club, et que j'ai dû transposer à un contexte d'équipe (hors club).

Gestion des saisons depuis les paramètres d'une équipe :

The screenshot shows the 'Paramètres d'équipe' (Team Settings) page. On the left, there's a sidebar with links: 'Informations de l'équipe', 'Abonnement', 'Stades', 'Adversaires', 'Tâches', 'Gestion des saisons' (which is highlighted in red), and 'Supprimer l'équipe'. The main content area is titled 'Gérer mes saisons' and contains a table with two rows:

Nom	Du	Au	Statut	Nombre de joueurs	Nombre d'événements	Actions
2020-2021	01/08/2020	31/07/2021	Saison actuelle	-	-	
2019-2020	01/08/2019	31/07/2020	Archivée	-	-	

A note at the bottom states: 'Vous ne pouvez pas supprimer une saison lorsqu'elle contient des événements ou lorsqu'elle est comprise entre deux saisons.'

The screenshot shows the 'Editer mes saisons' (Edit Seasons) modal. It has a table with two rows, similar to the one above, but with different styling. The first row is highlighted in pink and labeled 'Saison actuelle'. The second row is white and labeled 'Archivée'. At the bottom of the modal, there are 'Enregistrer' (Save) and 'Annuler' (Cancel) buttons.

Depuis l'onglet Gestion des saisons, dans la page des paramètres d'une équipe, on peut éditer les différentes saisons : en ajouter, en supprimer ou en modifier.

Sujet réalisé en peer-programming avec mon tuteur de stage.

M'a permis d'aborder la notion de CRUD et de formulaire sous React.

Ajout de membres à une équipe depuis d'autres saisons :

Ajouter des membres

Inviter de nouveaux membres

Partager le lien ou le code permettant de rejoindre l'équipe

Inviter des membres depuis une autre saison

Ajouter des membres

Inviter des membres depuis une autre saison

Tout sélectionner Rechercher

<input checked="" type="checkbox"/> Duan Allman	<input checked="" type="checkbox"/> Chuck Berry	<input checked="" type="checkbox"/> Ritchie Blackmore
<input type="checkbox"/> James Burton	<input type="checkbox"/> Eric Clapton	<input type="checkbox"/> Kurt Cobain
<input type="checkbox"/> Ry Cooder	<input checked="" type="checkbox"/> Dick Dale	<input type="checkbox"/> Bo Diddley
<input checked="" type="checkbox"/> John Frusciante	<input type="checkbox"/> Jerry Garcia	<input type="checkbox"/> Kirk Hammett
<input type="checkbox"/> Jimi Hendrix	<input type="checkbox"/> Gérard Hérisson	<input type="checkbox"/> Robert Johnson

Ajouter (5 Membres)

Plusieurs choix sont possibles pour ajouter des membres au sein d'une équipe ou d'un club. L'un deux est d'inviter des membres déjà présents lors des saisons précédentes. La page permet de filtrer à la volée par nom et d'effectuer des choix multiples via une carte de membre et sa checkbox, pour ajouter autant de membres que l'on souhaite.

J'ai dû adapter cette page au contexte club, avec des filtres supplémentaires (choix de l'équipe et choix de la saison).

4 - Conclusion

4.1 - Ce que cette expérience m'a apporté

Cette expérience m'a conforté dans mon choix de reconversion professionnelle et m'a rassuré sur mes capacités à occuper un poste équivalent en m'apportant les compétences techniques et théoriques nécessaires.

De plus, travailler sur un projet complet et bien construit m'a permis d'analyser le code réutilisable sur d'autres projets, les bonnes pratiques de nommages, etc...

Toute un bagage que je pourrai apporter avec moi sur de nouveaux projets, ou mettre au profit d'autres entreprises.

Le fait de travailler en relation avec d'autres personnes et l'utilisation commune de différents outils m'a également amené à développer mon esprit d'équipe.

Au contraire, le fait d'être régulièrement en distanciel et de devoir analyser moi-même un projet conséquent et construit par d'autres a développé mon autonomie et ma capacité à agir sur un projet existant.

4.2 - Avis final

J'ai été convaincu par React et souhaite donc poursuivre sur cette voie, en me spécialisant sur cette technologie et sur l'aspect front-end du développement web en général.

En vous remerciant de m'avoir lu !