

Rapport de stage de première année  
Sport Data Intelligence

Création de site web vitrine en Next.js



Du 24 mai au 2 juillet 2021

Tuteur pendant le stage: M. Camilo Coelho  
Tuteur académique: M. Alain Tarlowski

Stagiaire: Carl Fremault

Formation BTS SIO option SLAM au CNED  
Stage effectué en télétravail

# Sommaire

1. INTRODUCTION.....	3
2. CONTEXTE.....	3
2.1 Présentation de l'entreprise.....	3
2.2 Objectif du stage.....	3
2.3 Technologies.....	3
2.3.1 JavaScript.....	4
2.3.2 React.....	4
2.3.3 Next.js.....	6
2.3.4 Material UI.....	7
3. ORGANISATION.....	7
3.1 Git flow.....	7
3.2 GitHub, méthode Agile, Markdown.....	9
4. LE SITE.....	11
4.1 Les composants.....	11
4.1.1 La barre de navigation.....	11
4.1.2 Le "Hero".....	17
4.1.3 Les "Cards".....	18
4.1.4 Le "Footer".....	27
4.1.5 La carte sur la page "Contact".....	28
4.1.6 Le "Banner".....	29
4.1.7 Les titres des pages.....	30
4.1.8 TrustedBy.....	30
4.1.9 Le "Layout".....	31
4.1.10 La page "Connect".....	33
4.2 Google Analytics.....	40
4.3 Optimisation du site.....	41
5. Evolution / Axes d'amélioration.....	43
6. Conclusion.....	43
7. Remerciements.....	44

# **1. INTRODUCTION**

Dans le cadre de ma première année de formation au CNED, pour préparer un BTS Services Informatiques aux Organisations, option Solutions Logicielles et Applications, j'ai été amené à effectuer un premier stage en entreprise d'une durée de six semaines, afin de consolider mes acquis et d'avoir une première expérience du monde du travail en tant que développeur informatique.

L'entreprise qui a bien voulu m'accueillir pour ce projet est Sport Data Intelligence, une jeune start-up dans le domaine du Data Science basée à Annecy, en Haute-Savoie.

## **2. CONTEXTE**

### **2.1 Présentation de l'entreprise**

Sport Data Intelligence est une jeune start-up créée en 2020 spécialisée dans la Data Science. Elle propose des solutions pour des entreprises qui souhaitent mieux exploiter la masse de données qu'elles génèrent, à l'aide de nouvelles technologies tel que l'Intelligence Artificielle et le Revenue Management. Analyse de données, statistiques, tarification dynamique, ... les applications sont multiples pour aider les organisations à valoriser la quantité de données qui ne cesse de croître dans le paysage économique actuel, dominé par la technologie et le phénomène du Big Data.

### **2.2 Objectif du stage**

Lors de la création de l'entreprise une page vitrine a été faite avec le CMS Wordpress. Cette page était une solution temporaire, le but final étant d'avoir une page dynamique permettant aussi aux clients de se connecter à une partie back-office pour consulter leurs chiffres, graphiques, ... .

La mission qui m'a été donnée était de refaire la partie statique, la vitrine de l'entreprise.

### **2.3 Technologies**

La technologie choisie par M. Coelho pour la création du site est Next.js, un framework pour React, elle-même une librairie JavaScript. Pour l'interface graphique on utilisera Material UI.

### **2.3.1 JavaScript**

JavaScript est un langage de programmation né en 1995. Il est principalement utilisé pour créer des pages web interactives et est ainsi considéré comme un des piliers du "World Wide Web", à côté du HTML et du CSS. Si au début les fonctionnalités étaient limitées, et utilisées principalement pour des éléments dynamiques au niveau du front-end des pages web, des mises à jour ont progressivement ajouté de nouveaux éléments et des fonctionnalités et ont ainsi élargi le domaine d'application. On peut notamment citer l'introduction de AJAX (Asynchronous JavaScript and XML) en 2004, la librairie jQuery en 2006, et l'importante mise à jour ES2015/ES6.

Actuellement il est estimé que 97% des sites web utilisent JavaScript pour la partie front-end. Aussi, selon les statistiques des dépôts sur GitHub, dans le premier trimestre de 2021 JavaScript est le langage le plus populaire sur la plateforme. Le sondage annuelle de StackOverflow pour 2020 confirme ce statut du "langage le plus populaire".

Sources:

[https://madnight.github.io/githut/#/pull\\_requests/2021/1](https://madnight.github.io/githut/#/pull_requests/2021/1)

<https://insights.stackoverflow.com/survey/2020#technology>

### **2.3.2 React**

React est une librairie front-end Open Source pour JavaScript, créée en 2013 par Facebook. Elle est utilisé pour créer des interfaces utilisateur, des composants UI, et des applications web appelées "single-page application" (SPA).

En visitant un site web "classique", la page d'entrée est téléchargée et affichée dans le navigateur. Quand l'utilisateur navigue entre les différentes pages, le serveur web est interpellé et à chaque fois le contenu demandé est téléchargé.

Le principe des SPA est que, au lieu de faire une requête auprès du serveur web et de télécharger du contenu à chaque interaction du visiteur d'une page, le navigateur va récupérer tout le code et contenu nécessaire en bloc lors de l'ouverture de la page et écrire ou ré-écrire le contenu dynamiquement suivant les demandes des utilisateurs. (D'éventuelles ressources supplémentaires peuvent être téléchargés et ajoutés dynamiquement suite aux interactions des utilisateurs.)

Ceci est possible grâce à une des technologies utilisées par React: le "Virtual DOM" (Document Object Model). Le DOM classique est recalculé du début à chaque changement. React de son côté possède un DOM virtuel où l'application calcule et optimise pour ne modifier uniquement ce qui est strictement nécessaire avant de répercuter les changements sur le DOM.

Ainsi on constate un gain énorme en performance, et une expérience utilisateur beaucoup plus agréable.

```

return (
  <Provider store={store}>
    {/* <Fragment> */}
    <Head>
      <title>My page</title>
      <meta
        name="viewport"
        content="minimum-scale=1, initial-scale=1, width=device-width"
      />
    </Head>
    {/* Load custom theme and apply a base theme in the first request */}
    <ThemeProvider theme={theme}>
      <CssBaseline />
      <Layout>
        <Component {...pageProps} />
      </Layout>
    </ThemeProvider>
    {/* </Fragment> */}
  </Provider>
);

```

*Figure 1: Exemple: SPA. Le fichier \_App.js est le point d'entrée de l'application, le composant fonctionnel retourne une squelette de page web où le composant <Component {...pageProps}> retourne les différentes "pages" qui existent dans le projet.*

Pour les développeurs aussi il y a de nombreux avantages. React fonctionne avec des composants qu'on peut utiliser, modifier, et on peut en créer d'autres en fonction des besoins. Tous les composants peuvent être réutilisés permettant un gain de performance au niveau du développement des applications.

React permet aussi d'utiliser du JSX (JavaScript XML). Le JSX est une extension au syntaxe JavaScript qui permet d'écrire une variante du HTML au sein du code. Ainsi on peut incorporer du "mark-up", des fonctionnalités et de la logique tous ensemble dans un même contexte.

En 2020, toujours selon le sondage annuelle de StackOverflow, React est en deuxième position des frameworks web les plus utilisés par les développeurs (après jQuery, qui, par ailleurs, perd des points chaque année, au profit de React et Angular).

```

/**
 * Iterates over the NavigationItems array to generate a MenuLink for each item
 */
const Navigation = () => {
  const classes = Styles();
  const items = NavigationItems();

  return (
    <Grid width="50vw" className={classes.link}>
      {items.map((item) => (
        <MenuLink
          key={item.item}
          link={item.item}
          sublinks={item.subItems}
          path={item.path}
        />
      ))}
    </Grid>
  );
};

export default Navigation;

```

Figure 2: Exemple: JSX et logique coexistent au sein d'un même composant.

Malgré tous ces avantages, React possède aussi un désavantage: la technologie est basé sur du rendering dite "client side": tout le code, y compris le JSX, est "traduit" en JavaScript avant d'être stocké sur le serveur web. Quand un client visite le site, le code et contenu nécessaire sont téléchargés et le site est "créé" (ou: traduit en HTML) par le navigateur de l'internaute.

Comme spécifié au-dessus, cela permet un gain de performance pour l'utilisateur, toutefois, comme le code présent sur le serveur n'est pas du HTML il ne peut pas être interprété par les robots des moteurs de recherche, qui du coup dépendent des métadonnées insérés dans les "header" des fichiers "index" pour connaître et pouvoir indexer le contenu du site web.

### 2.3.3 Next.js

Pour résoudre ce problème Next.js entre en jeu. Ce framework front-end Open Source pour React créé en 2016 permet de faire du "Server side rendering" pour toute ou partie du site. Le contenu est stocké sur le serveur web sous forme de HTML et peut être lu par les robots, et on obtient des meilleures résultats sur les moteurs de recherche.

## 2.3.4 Material UI

Dernier élément des technologies utilisées, Material UI est un framework front-end Open Source de composants pour React basé sur Material Design, un ensemble de règles de design d'interfaces utilisateur proposé par Google en 2014. En tant que extension du langage CSS il propose un ensemble de composants cohérents au niveau du design et du choix de couleurs permettant une implémentation rapide et esthétique, tout en étant modifiable.

# 3. ORGANISATION

## 3.1 Git flow

Le modèle de développement utilisé est nommé "Git flow" qui s'appuie sur le système de gestion de versions (Version Control System) Git.

Le dépôt ("repository") central est hébergé sur GitHub. Ce dépôt contient la branche "master", où se trouve une version stable du projet, prête à être mise en production.

Une deuxième branche parallèle, "develop" est l'endroit où les dernières changements ou les nouvelles fonctionnalités sont intégrés dès qu'ils ont été approuvés. Quand le code de cette branche est jugé stable et prêt à être publié, on fusionne ("merge") le contenu avec la branche "master".

En début de stage, j'ai créé mon dépôt local en effectuant un "clone" de la branche "develop", puis j'ai créé une nouvelle branche à chaque fois que je commençais à travailler sur une nouvelle fonctionnalité: par exemple une branche pour le développement de la barre de navigation, une branche pour travailler sur la page "Contact", ...

Quand j'estimais que la fonctionnalité était implémentée, testée et documentée je faisais un "push" de la branche sur laquelle je travaillais vers la branche "develop" suivi d'un "pull request". Le responsable du projet révisait le code et, s'il décidait que le travail fait répondait aux attentes, il pouvait l'intégrer avec la branche "develop".

Entre-temps je pouvais créer une autre branche pour travailler sur une autre nouvelle fonctionnalité, ou retourner sur une branche existante pour faire des modifications, éventuellement demandées en réponse aux "pull request".

Ainsi j'ai été amené à travailler sur plusieurs branches simultanément et à utiliser un "stash" (un endroit virtuel où on peut "ranger" des fichiers et des changements faits et qui facilite l'utilisation de plusieurs branches en parallèle).

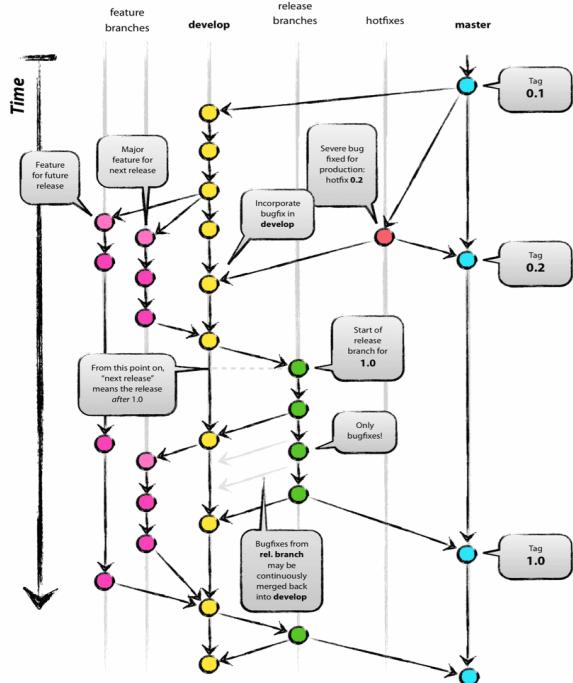


Figure 3: Schématisation du "Git flow" par Vincent Driessen, <https://nvie.com>

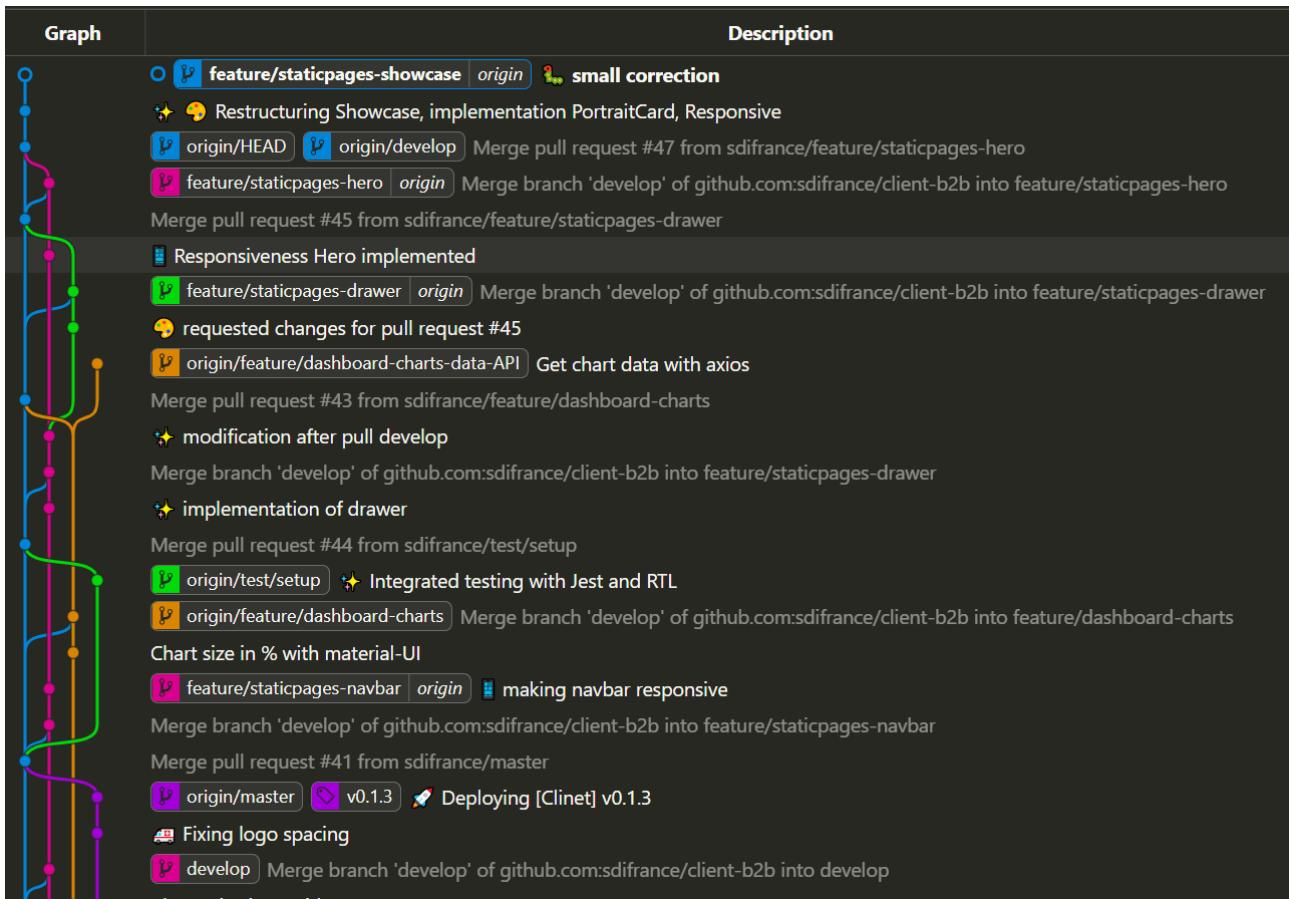


Figure 4: Schématisation des différentes branches du projet à un moment spécifique.

## 3.2 GitHub, méthode Agile, Markdown

Comme mentionné auparavant, le VCS utilisé est Git, par intermédiaire de la plateforme GitHub, qui permet une gestion visuelle du processus. Non seulement GitHub permet une gestion facile des branches et des "pull requests" (par exemple en utilisant une présentation très claire pour comparer les changements faits sur un fichier), la plateforme intègre également des outils pour gérer le projet selon la méthode AGILE.

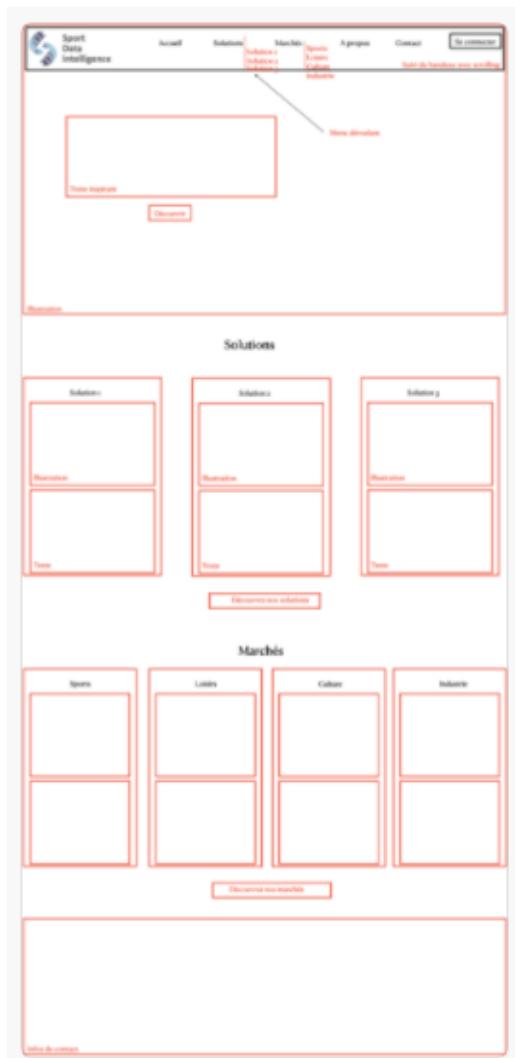


Figure 5: Maquette de la page d'accueil

Ainsi, en début du projet et en s'appuyant sur les maquettes fournis par les gérants de l'entreprise, nous avons entamé un premier "sprint" qui avait pour but de faire la première page, "vitrine" du site. Nous avons commencé par découper le projet en tâches, par exemple: la création de la barre de navigation, la création du héro, ...

Chaque tâche était inscrit sur une carte ("kanban") et classé selon le niveau d'achèvement: "idée", "à faire", "en cours", "fait". Les cartes peuvent contenir des souhaits, fonctionnalités, contraintes et explications, et sont complétées au fur et à mesure de l'avancement, tel qu'elles documentent le processus du développement de l'application.

Un atout de la plateforme GitHub est qu'elle accepte le "Markdown" partout (création de cartes, commentaires, "issues", fichier "readme", ...). Le Markdown est un langage de balisage léger qui permet une présentation claire et agréable tout en ayant un syntaxe facile à lire et à écrire. Ainsi on peut faire des cases à cocher, insérer des images, des extraits de code formatés tel quel, etc.

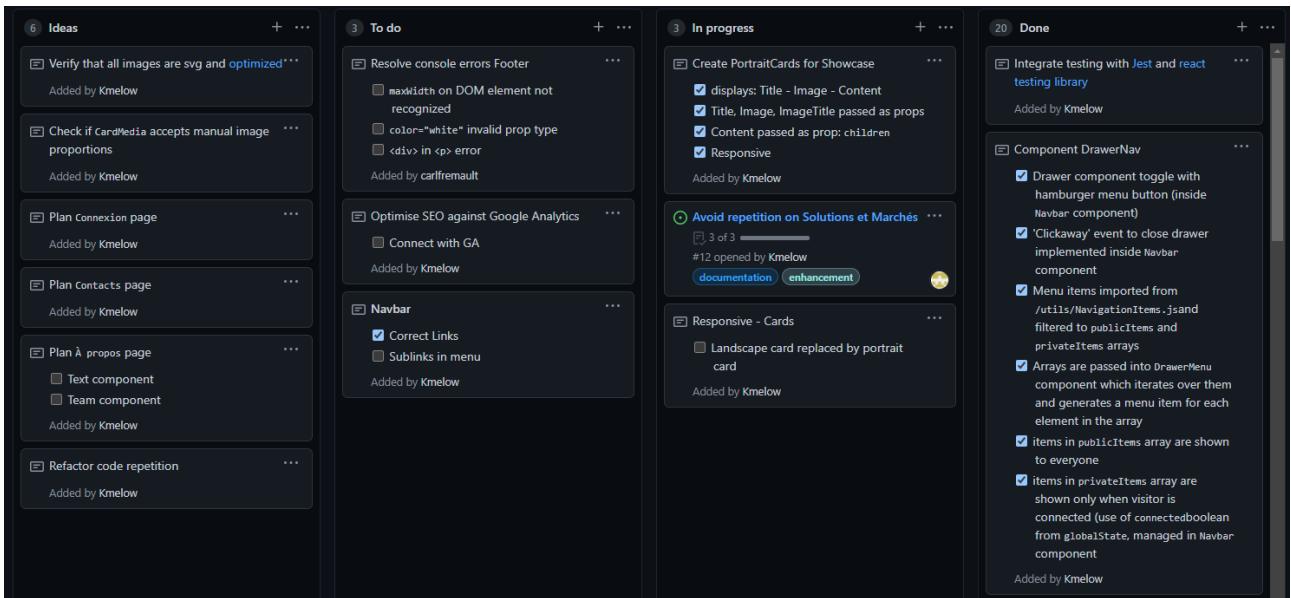


Figure 6: Les "kanban"

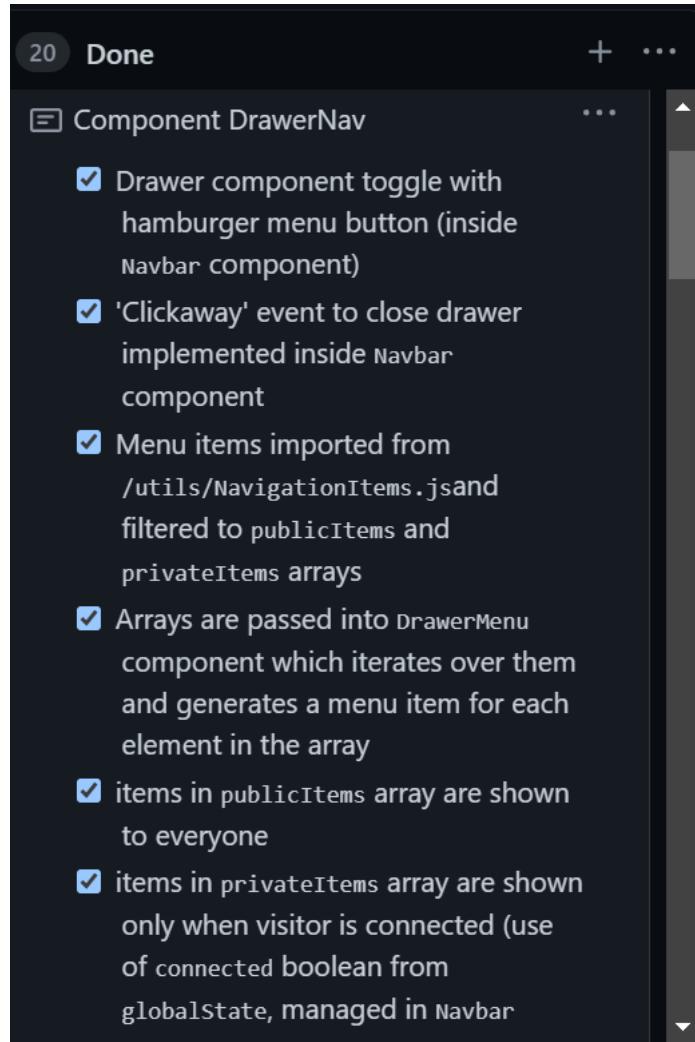


Figure 7: Exemple d'un "kanban" avec utilisation de Markdown

Lorsqu'on rencontre une difficulté technique, des messages d'erreur, ou encore tout simplement pour documenter certains aspects, on peut aussi créer des "issues" qui par la suite peuvent être liés avec les "kanban" ou encore avec les "pull request".

Toujours dans la démarche AGILE, nous avons commencé chaque journée par une "Daily Scrum Meeting" (en visioconférence comme, en raison de la situation sanitaire, le stage se déroulait en télétravail). Lors du DSM chaque membre de l'équipe prenait le temps de détailler les progrès faits le jour avant, et expliquait ce qu'il comptait faire pendant la journée. C'était également le moment d'aborder des questions techniques et de demander des avis ou de l'aide en cas de blocages.

## 4. LE SITE

Si le but principal du stage était de créer la page web vitrine de l'entreprise, un deuxième objectif était étroitement lié. Non seulement il y aura une partie back-office dans le futur, l'entreprise travaille aussi sur le site web de leur produit principal, un site comparateur pour l'achat de forfaits de ski.

Ainsi, il était nécessaire que les composants créés soient réutilisables dans les autres projets. Dans une optique "functional programming" chaque composant devrait être indépendant de façon à pouvoir l'intégrer ailleurs sans modifications nécessaires.

### 4.1 Les composants

#### 4.1.1 La barre de navigation

Le premier élément sur lequel j'ai travaillé n'était pas le plus simple: la barre de navigation.

Elle est faite avec le composant "Grid" de Material UI, qui est en fait une implémentation de la technologie CSS "Flexbox" (et non pas Grid....) permettant une gestion facile pour l'affichage et pour le côté "responsive".

Il y a trois parties principales:

- À gauche, le logo et le nom de la société (uniquement le logo pour les petits écrans), qui fonctionnent également comme lien pour retourner à la page d'accueil.
- Au milieu la section "navigation" avec les liens vers les différentes parties du site.
- À droite un bouton qui permet aux utilisateurs de se connecter au back-office.



Figure 8: La barre de navigation



Figure 9: La section "navigation" avec menu déroulant. Le "tab" "Accueil" est actif.

```

return (
  <>
    <AppBar position="sticky" elevation={1} className={classes.appbar}>
      <div ref={node}>
        <Toolbar className={classes.toolbar}>
          {/* Full navigation menu: only visible for lg, xl */}
          <Hidden only={["xs", "sm", "md"]}>
            <Navigation connected={connected} />
          </Hidden>
          {/* Hamburger icon: only visible for xs, sm, md */}
          <Hidden only={["lg", "xl"]}>
            <Logo />
            <IconButton color="primary" onClick={toggleDrawer}>
              {openDrawer ? (
                <MenuOpenIcon className={classes.iconButton} />
              ) : (
                <MenuIcon className={classes.iconButton} />
              )}
            </IconButton>
          </Hidden>
          <Connection />
        </Toolbar>
      </div>
    <AppBar>
      <DrawerNav drawerOpen={openDrawer} variant="persistent" />
    </AppBar>
  );
);
  
```

Figure 10: La structure JSX de la barre de navigation

Pour les petits écrans la partie "Navigation" est transformée en bouton "hamburger" qui ouvre un menu sur le côté de l'écran.



Figure 12: La page d'accueil "version mobile" ...

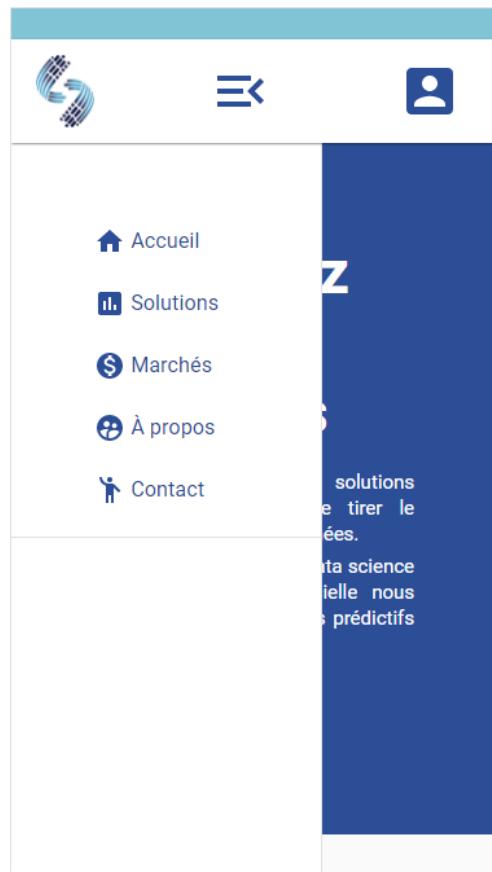


Figure 11: ... avec le menu de navigation

Tout le contenu de la partie "navigation" (les intitulés ainsi que les chemins d'accès ("path" )) se trouvent dans un fichier séparé contenant une fonction qui retourne un vecteur avec des objets qui à leur tour contiennent toutes les informations nécessaires. Ainsi il est très facile de modifier le contenu, ajouter des liens, .... , et le composant peut facilement être intégré dans un autre projet.

Chaque entrée contient son nom ("item"), une icône pour le menu version "mobile" et le chemin d'accès. Certains entrées contiennent un autre vecteur "subitems" s'il y a un menu déroulant avec d'autres options à afficher.

Pour chaque entrée il y a également un booléen "public". Effectivement, le contenu de la barre de navigation sera différent pour les clients connectés à la partie back-office.

```

const NavigationItems = () => [
  {
    item: "Accueil",
    public: true,
    icon: <HomeIcon color="primary" />,
    path: "/",
    activeIndex: 0,
  },
  {
    item: "Solutions",
    public: true,
    icon: <AssessmentIcon color="primary" />,
    path: "/solutions",
    activeIndex: 1,
    subItems: [
      {
        item: "Big Data & Analytics",
        path: "/solutions#bigdata",
      },
      {
        item: "IA & Modélisation",
        path: "/solutions#ia",
      },
      {
        item: "Revenue Management",
        path: "/solutions#revmgmt",
      },
    ],
  },
]

```

*Figure 13: Extrait de la fonction "NavigationItems"*

La propriété "activeIndex" est utilisée pour pouvoir déterminer le "tab" (i.e. l'entrée du menu) qui est actif. A chaque nouveau rendering de page le "hook" "useEffect" se déclenche pour déterminer sur quelle page on se trouve, et adapter le style du "tab" correspondant en conséquence (souligné et couleur plus foncé) avec le prop "value" de "Tabs" (fonctionnalité inhérente à Material UI).

```

// Verifies pathname and sets value accordingly, in order to style active tab
useEffect(() => {
  navItems.forEach((item) => {
    switch (window.location.pathname) {
      case `${item.path}`:
        if (value !== item.activeIndex) {
          setValue(item.activeIndex);
        }
        break;
      case "/connect":
        setValue(false);
        break;
      default:
        break;
    }
  });
}

return (
  <>
  <Link href="/" className={classes.logo} underline="none"> ...
  <Tabs
    value={value}
    onChange={handleChange}
    TabIndicatorProps={{
      style: { backgroundColor: "#81C4D6" },
    }}
  >

```

Figure 14: Utilisation du hook "useEffect" pour déterminer sur quelle page on se trouve et adapter le style du "tab" correspondant.

Le composant "Navigation" à l'intérieur de la barre de navigation reçoit le vecteur "NavigationItems" et fait une itération sur son contenu.

```

const publicItems = NavigationItems().filter((item) => item.public);
const privateItems = NavigationItems().filter((item) => !item.public);
const navItems = connected ? privateItems : publicItems;

```

Figure 15: Le contenu est filtré dans les vecteurs "publicItems" et "privateItems" depuis "NavigationItems". Le vecteur utilisé pour générer les liens de la barre de navigation dépend si l'utilisateur est connecté ou non.

```
return (
  <>
    <Link href="/" className={classes.logo} underline="none"> ...
    <Tabs
      value={value}
      onChange={handleChange}
      TabIndicatorProps={{
        style: { backgroundColor: "#81C4D6" },
      }}
    >
      {navItems.map((item) =>
        item.subItems ? (
          <SubMenu
            key={item.item}
            item={item.item}
            path={item.path}
            subItems={item.subItems}
          />
        ) : (
          <Tab
            key={item.item}
            style={{
              color: "#2B4E96",
              fontSize: 18,
              marginLeft: "1rem",
              textDecoration: "none",
            }}
            label={item.item}
            component={Link}
            href={item.path}
          />
        )
      )}
    </Tabs>
  </>
);

```

Figure 16: Ensuite on fait une itération sur le contenu de "navItems" pour générer les différentes entrées du menu. Si des "subItems" sont présents, le composant "SubMenu" est appelé...

```

return (
  <>
  <Tab
    className={classes.tab}
    aria-owns={anchorEl ? "simple-menu" : undefined}
    aria-haspopup={anchorEl ? "true" : undefined}
    onMouseOver={(event) => handleClick(event)}
    label={item}
  />
  <Menu
    id="simple-menu"
    classes={{ paper: classes.subMenu }}
    anchorEl={anchorEl}
    open={open}
    onClose={handleClose}
    elevation={2}
    MenuListProps={{ onMouseLeave: handleClose }}
  >
    <MenuItem onClick={handleClose}>
      <Link href={path}>
        <Tab className={classes.subTab} label={item} />
      </Link>
    </MenuItem>
    {subItems.map((subItem) => (
      <MenuItem key={subItem.item} onClick={handleClose}>
        <Link href={subItem.path} passHref>
          <Tab className={classes.subTab} label={subItem.item} />
        </Link>
      </MenuItem>
    ))}
  </Menu>
  </>
);

```

*Figure 17: ... qui itère de la même façon sur le vecteur des "subItems".*

Un clic sur le bouton "Se connecter" amènera l'utilisateur à la page "Connect" qui sera implémentée ultérieurement.

#### 4.1.2 Le "Hero"

Le "Hero Header" (ou "Hero Image", ou tout simplement "Hero") est l'image en haut d'une page web qui prends la totalité de la largeur d'écran. Il a pour rôle de capturer l'attention de l'internaute qui arrive sur la page web.

Dans le monde du "front-end" on parle du "fifteen seconds rule" (la règle des 15 secondes): on estime que à peu près 80% des visiteurs d'un site web quittent le site dans les 15 premières secondes, pour des raisons variées: ils ne comprennent pas où ils sont arrivés, ils ne se sentent pas concernés, ... . Il est donc crucial que le "Hero" répond aux questions fondamentales suivantes afin de ne pas perdre des clients potentiels:

- Qu'est-ce ce site web? Qui est cette entreprise?
- Qu'est qui est proposé? Que fait l'entreprise? Quel est le but de ce site?
- Qu'est-ce que j'ai à gagner si je prends le temps de visiter ce site web?

Ainsi le "Hero" doit être visuellement attractif et informatif en même temps.

Afin de bien pouvoir gérer l'affichage des différents éléments, et ce sur toutes les tailles d'écran (mobile et desktop) j'ai à nouveau utilisé l'élément "Grid" de Material UI.

Le composant "Hero" contient quatre éléments:

- un image de fond d'écran
- un "HeaderTag": le "slogan" du site web / de l'entreprise
- un "ActionText": une ou deux phrases d'introduction
- un "Call to action": un bouton qui invite l'internaute à visiter davantage le site web

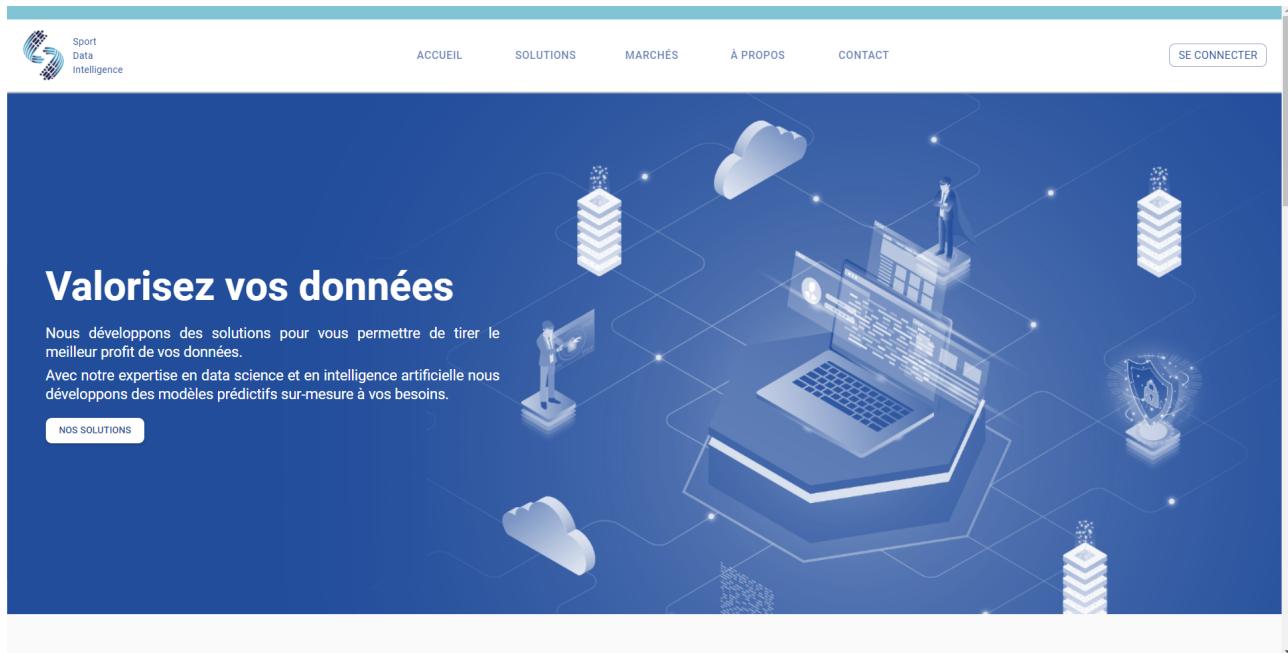


Figure 18: La page d'accueil avec la barre de navigation et le "Hero", version desktop

#### 4.1.3 Les "Cards"

Pour présenter le contenu du site de façon claire et attractive nous avons choisi d'utiliser le composant "Card" de Material UI. Une "Card" contient un titre, une image, et le contenu à afficher: paragraphes de texte, sous-titres, icônes, listes énumératives, ... .

Afin de répondre aux différents besoins j'ai créé deux composants, respectivement les "PortraitCards" et les "LandscapeCards" (selon leur orientation) qui reçoivent le contenu depuis un autre composant (spécifique à chaque "Card") et s'occupent de la mise en forme et du côté "responsive".

De cette façon les composants "PortraitCard" et "LandscapeCard" fonctionnent comme un "wrapper" autour du contenu qui est créé séparément. Ainsi il est très facile de faire évoluer le contenu de chaque "Card", de rajouter des cartes, ... sans devoir s'occuper du côté fonctionnel à chaque fois. La "PortraitCard" reçoit en paramètre le titre de la carte, l'image à afficher (et le titre de l'image pour l'attribut HTML "alt"), le nombre de colonnes du "Grid" qu'elle peut utiliser (pour le côté "responsive") et le prop "children" qui représente

les éléments à insérer à l'intérieur de la carte (concrètement: tous les éléments qui se trouvent entre les balises ouvrantes et fermantes de l'élément parent).

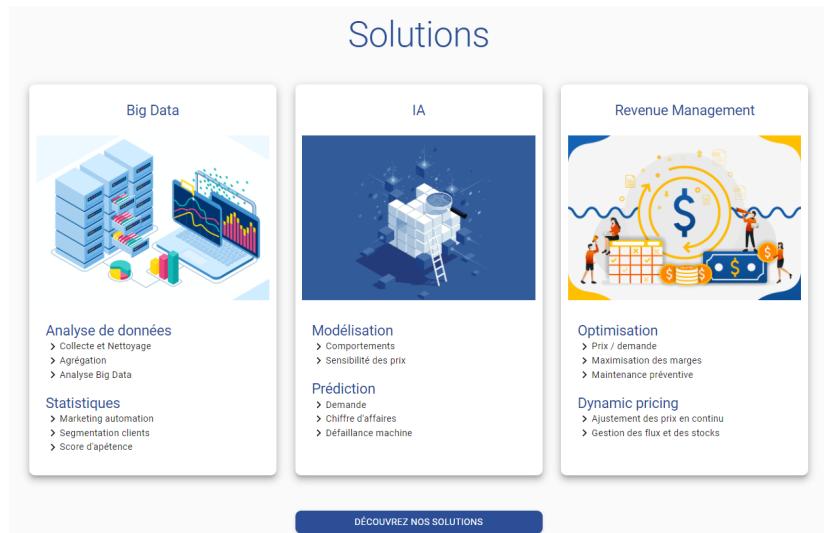


Figure 19: La section "Solutions" de la page d'accueil utilise le composant "PortraitCard".

```
return (
  <section id="solutions">
    <Container maxWidth={false} className={classes.solutionsBg}>
      <Grid>
        <Container ...>
      </Grid>
      <Grid item xs={12}>
        <Typography ...>
      </Grid>
      <PortraitCard
        img="/imgs/SolutionsBigData.png"
        imgTitle="Solutions Big Data"
        cardTitle="Big Data"
        cardColumns={4}
      >
        <SolutionsBigDataSC />
      </PortraitCard>
      <PortraitCard
        img="/imgs/SolutionsIA.png"
        imgTitle="Solutions IA"
        cardTitle="IA"
        cardColumns={4}
      >
        <SolutionsIASC />
      </PortraitCard>
    </Container>
  </Container>
)
```

Figure 20: Extrait du code de la section "Solutions" de la page d'accueil qui implémente les "PortraitCard" comme "wrapper" autour des composants spécifiques avec le contenu de chaque "Card".

```

/**
 * Renders portrait-oriented Card with title, image (passed as props) and children elements
 *
 * @param {string} img - image path. E.g. "/imgs/ImagePath.png"
 * @param {string} imgTitle - alt title for the image
 * @param {string} cardTitle - title for the card
 * @param {number} cardColumns - number of grid columns card should occupy
 * @param {node} children - text content of the card
 */
const PortraitCard = ({ children, img, imgTitle, cardTitle, cardColumns }) => {
  const classes = Styles();

  return (
    <Grid
      item
      className={classes.root}
      xs={12}
      md={cardColumns}
      component={Card}
      raised
    >
      <CardContent>
        <Typography variant="h5" align="center" color="primary">
          {cardTitle}
        </Typography>
      </CardContent>
      <CardMedia className={classes.media} image={img} title={imgTitle} />
      <br />
      {children}
    </Grid>
  );
};

```

Figure 21: La structure du composant "PortraitCard"

```

const analyseDeDonneesEntries = [
  "Collecte et Nettoyage",
  "Agrégation",
  "Analyse Big Data",
];
const statistiquesEntries = [
  "Marketing automation",
  "Segmentation clients",
  "Score d'apétence",
];
/* Content to be inserted in 'Solutions Big Data' PortraitCard */
const SolutionsBigDataSC = () => (
  <CardContent>
    <Typography variant="h5" color="primary" my={2}>
      Analyse de données
    </Typography>
    {analyseDeDonneesEntries.map((entry) => (
      <ListEntry key={entry} listText={entry} />
    ))}
    <br />
    <Typography variant="h5" color="primary" my={2}>
      Statistiques
    </Typography>
    {statistiquesEntries.map((entry) => (
      <ListEntry key={entry} listText={entry} />
    ))}
  </CardContent>
);

```

Figure 22: Exemple du contenu, ici pour la carte "Big Data". Pour les listes énumératives les éléments sont stockés dans des vecteurs. Avec la fonction JavaScript "map" on itère sur ces éléments et une "ListEntry" est créé pour chaque élément.

Afin de gérer correctement l'affichage des entrées des listes énumératives, présentes sur différents endroits dans le site, j'ai créé un composant fonctionnel "ListEntry" qui reçoit en paramètre le texte à afficher et retourne un icône (par exemple ici: ">") et le texte, alignés correctement. Un paramètre optionnel "responsive" peut être spécifié, à ce moment la classe "responsiveText" est ajoutée à la liste des classes et la taille de police (spécifiée dans les paramètres CSS de la classe) s'adapte en fonction des tailles d'écran.

```

/**
 * Takes a string and returns a list entry with icon + text perfectly aligned
 * @param {string} listText - text for list entry
 * @param {boolean} responsive - activates responsiveText className
 */
const ListEntry = ({ listText, responsive }) => {
  const classes = useStyles();

  return (
    <Typography
      className={`${classes.iconTextAlign} ${{
        responsive && classes.responsiveText
      }}`}
      variant="body1"
    >
      <ChevronRight />
      {listText}
    </Typography>
  );
};

ListEntry.defaultProps = {
  responsive: false,
};

ListEntry.propTypes = {
  listText: PropTypes.string.isRequired,
  responsive: PropTypes.bool,
};

export default ListEntry;

```

*Figure 23: Le composant "ListEntry"*

La plupart du temps, quand il y avait des "styles" CSS à insérer, le choix a été fait d'utiliser un fichier séparé "Styles.js" qui contient toutes les classes et leurs règles respectives, puis d'insérer ce fichier dans le composant où on souhaite appliquer ces classes.

Pour le composant "ListEntry" j'ai choisi d'incorporer les "styles" à l'intérieur du composant. Ainsi il devient complètement indépendant, le fichier peut être "copié-collé" tel quel pour ajouter cette fonctionnalité dans un composant sans devoir faire des modifications dans le composant récepteur.

```

    /**
     * Custom styles
     */
    const useStyles = makeStyles((theme) => ({
      iconTextAlign: {
        alignItems: "center",
        display: "flex",
        flexWrap: "wrap",
      },
      responsiveText: [
        [theme.breakpoints.only("xs")]: {
          fontSize: 20,
        },
        [theme.breakpoints.up("sm")]: {
          fontSize: 24,
        },
      ],
    }),
  );

```

Figure 24: Extrait du composant "ListEntry" avec utilisation du "hook" "useStyles"

Le composant "LandscapeCard" fonctionne de la même façon, avec toutefois l'ajout d'un booléen "inverted" qui permet d'alterner entre un affichage "image à gauche texte à droite" ou l'inverse.

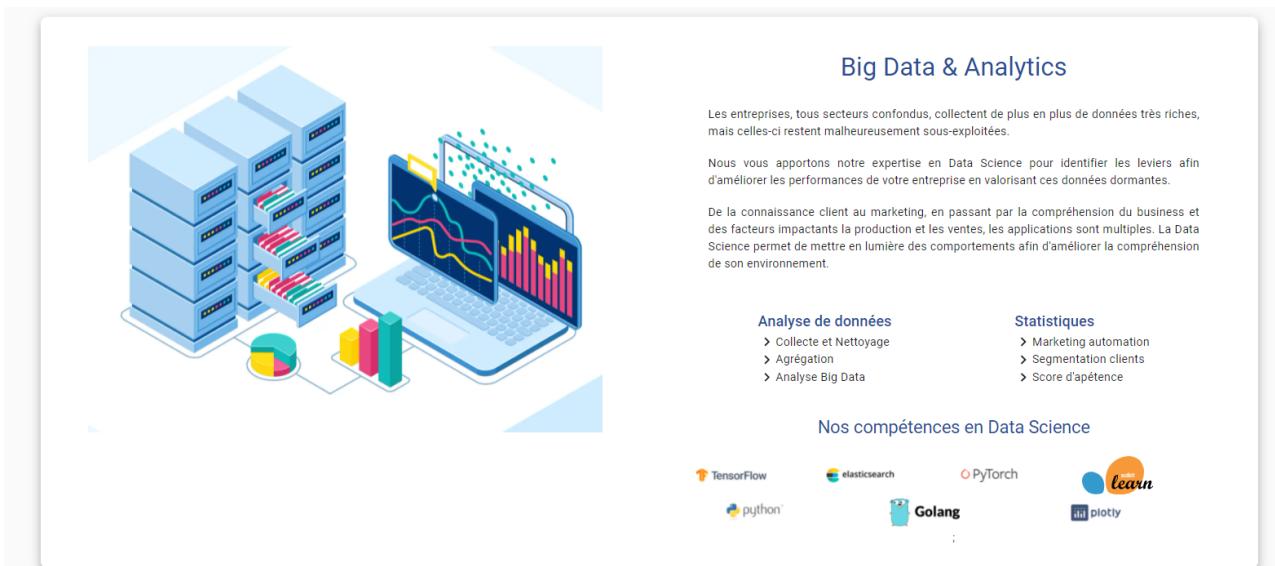


Figure 25: Utilisation du composant "LandscapeCard" sur la page "Solutions"

### IA & Modélisation

Nos analyses de données historiques permettent de modéliser les comportements (saisonnalité, impact météo, sensibilité au prix...).

Associé à nos outils d'Intelligence Artificielle nous développons des solutions sur-mesure pour prédire les comportements futurs et ainsi optimiser les différents services de l'entreprise (personnel, achats, ventes, capacité de production...).

<b>Modélisation</b> <ul style="list-style-type: none"> <li>&gt; Comportements</li> <li>&gt; Sensibilité des prix</li> </ul>	<b>Prédiction</b> <ul style="list-style-type: none"> <li>&gt; Demande</li> <li>&gt; Chiffre d'affaires</li> <li>&gt; Défaillance machine</li> </ul>
-----------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------

**Nos compétences en IA**

Machine learning (random forest, classification, regression), predictive modelling



*Figure 26: Le même composant "LandscapeCard", cette fois avec le booléen "inverted" avec valeur "true"*

En cas d'affichage sur mobile la fonctionnalité d'inverser l'affichage des images et du contenu doit être désactivée cependant, afin de toujours afficher l'image en premier.

```
return (
  <Grid
    container
    className={classes.root}
    direction="row"
    component={Card}
    raised
  >
  {inverted && widescreen ? (
    <>
    {content} {image}
    </>
  ) : (
    <>
    {image} {content}
    </>
  )}
  </Grid>
);
```

*Figure 28: La logique qui décide de l'ordre d'affichage en fonction des booléens "inverted" et "widescreen"*

≡
≡
≡

### IA & Modélisation

Nos analyses de données historiques permettent de modéliser les comportements (saisonnalité, impact météo, sensibilité au prix...).

Associé à nos outils d'Intelligence Artificielle nous développons des solutions sur-mesure pour prédire les comportements futurs et ainsi optimiser les différents services de l'entreprise (personnel, achats, ventes, capacité de production...).

```
const widescreen = useMediaQuery(theme.breakpoints.up("lg"));
```

Figure 29: Définition de la constante "widescreen"

Afin d'optimiser la lecture du code et éviter des répétitions inutiles, j'ai inséré les éléments "image" et "content" dans des constantes.

```
const image = (
  <Grid item xs={12} lg={imageWidth} className={classes.imageSide}>
    <Box className={classes.imageContainer}>
      <Image
        className={classes.image}
        alt={title}
        src={img}
        layout="fill"
        objectFit="contain"
      />
    </Box>
  </Grid>
);

const content = (
  <Grid
    item
    container
    direction="column"
    xs={12}
    lg={textWidth}
    className={classes.textSide}
    justify="center"
    alignItems="center"
  >
    {children}
  </Grid>
);
```

Figure 30: Les constantes "image" et "content"

Une dernière fonctionnalité est l'option de définir la répartition entre l'espace d'écran occupé par l'image et par le contenu. Cette fonctionnalité est obtenue en ajoutant un "prop" (paramètre) "imageWidth" lors de l'appel du composant "LandscapeCard", avec une valeur maximale de 12. En effet, le composant "Grid" de Material UI divise la largeur de l'écran en 12 colonnes. Un élément qui se voit attribuer 12 colonnes occupera la totalité de la largeur d'écran.

```

<section id="sports" className={classes.pageSection}>
  <LandscapeCard
    img="/imgs/MarchesSports.png"
    title="Marchés Sports"
    imageWidth={8}
  >
    <MarchesSports />
  </LandscapeCard>
</section>

```

Figure 31: Lors de l'appel de la "LandscapeCard" où on insère le contenu pour le "Marché Sports" de la page "Marchés", on attribue 8 colonnes pour l'image.

```

const textWidth = 12 - imageWidth;

```

Figure 32: La largeur attribué pour la partie "texte" est calculé à partir du paramètre "imageWidth". Ensuite les valeurs sont utilisés dans les constantes respectives (Figure 30)

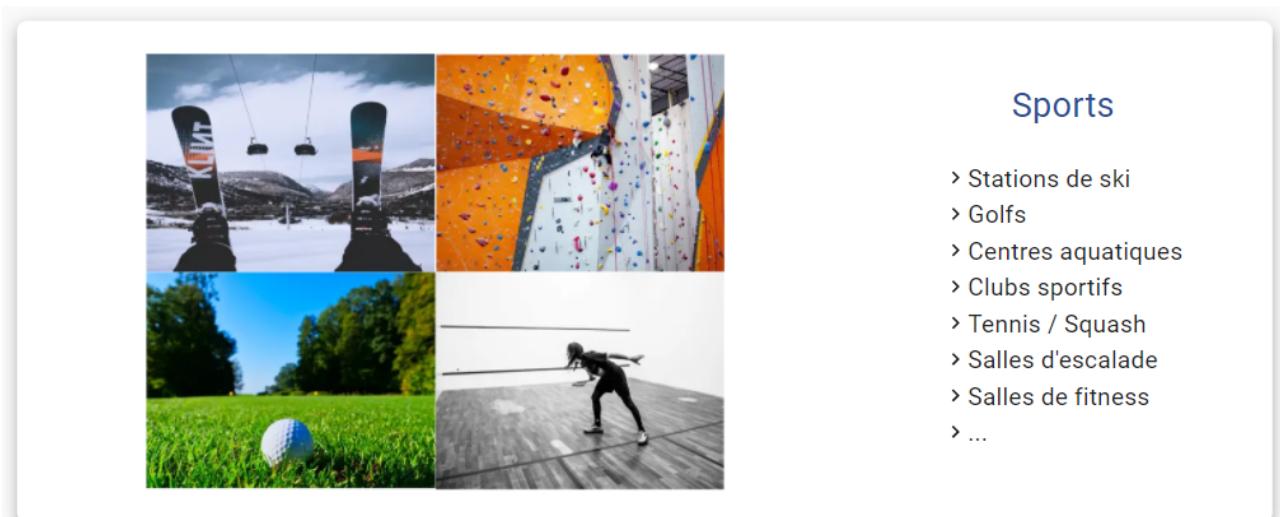


Figure 33: Le résultat.

#### 4.1.4 Le "Footer"

La partie du bas du page, le "footer" est un composant très simple qui utilise toujours ce même composant "Grid" de Material UI pour partager l'espace en trois parties :

- Une première partie à gauche qui fait office de "site map" avec des liens vers les différentes pages du site
- Le logo et le nom de l'entreprise au milieu
- Les coordonnées de l'entreprise à droite

```
/**  
 * Component filters NavigationItems.jsx and creates a link for each public entry  
 */  
  
const FooterNav = () => {  
  const classes = Styles();  
  const navItems = NavigationItems().filter((item) => item.public);  
  
  return (  
    <Box style={{ textAlign: "center" }}>  
      <Box style={{ display: "inline-block", textAlign: "left" }}>  
        {navItems.map((item) => (  
          <Link href={item.path} key={item.item} passHref>  
            <a href={item.path} className={classes.linkText}>  
              <Typography style={{ margin: "0.2rem 0" }}>  
                {item.item}  
              </Typography>  
            </a>  
          </Link>  
        ))}  
        <Link href="/cgu" passHref>  
          <a href="/cgu" className={classes.linkText}>  
            <Typography style={{ margin: "0.2rem 0" }}>CGU</Typography>  
          </a>  
        </Link>  
      </Box>  
    </Box>  
  );  
};
```

Figure 34: À nouveau, pour la partie "site map" le composant "FooterNav" itère sur le vecteur "navItems", lui même le résultat d'un filtrage des éléments publics de "NavigationItems". Le lien pour les Conditions Générales d'Utilisation est ajouté séparément comme on ne souhaite pas que cette page s'affiche dans la barre de navigation.

#### 4.1.5 La carte sur la page "Contact"

Afin de pouvoir insérer une carte interactive sur la page "Contact" j'ai installé les "packages" "leaflet" et "react-leaflet". Ces packages permettent d'insérer des "tiles" (sections de carte) depuis un fournisseur externe (dans ce cas ci: OpenStreetMap). On peut également y insérer des points de repère ("marker") éventuellement avec des bulles explicatives ("popup").

```
/**
 * Returns a page-wide OpenStreetMap component
 */
const ContactMap = () => {
  const classes = Styles();
  const { latitude, longitude } = ContactDetails;

  return (
    <MapContainer
      center={[latitude, longitude]}
      zoom={13}
      scrollWheelZoom={false}
      className={classes.mapContainer}
    >
      <TileLayer
        attribution='&copy; <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
      />
      <Marker position={[latitude, longitude]}>
        <Popup>
          <Typography variant="body1">
            <strong>Sport Data Intelligence</strong>
            <br />
            Coordonnées GPS:
            <br />
            Latitude: {latitude},
            <br />
            Longitude: {longitude}
          </Typography>
        </Popup>
      </Marker>
    </MapContainer>
  );
}
```

Figure 36: Le composant "ContactMap" qui affiche une carte sur la largeur entière de l'écran, et permet de spécifier un point central, niveau de zoom, et d'y insérer des points de repère.

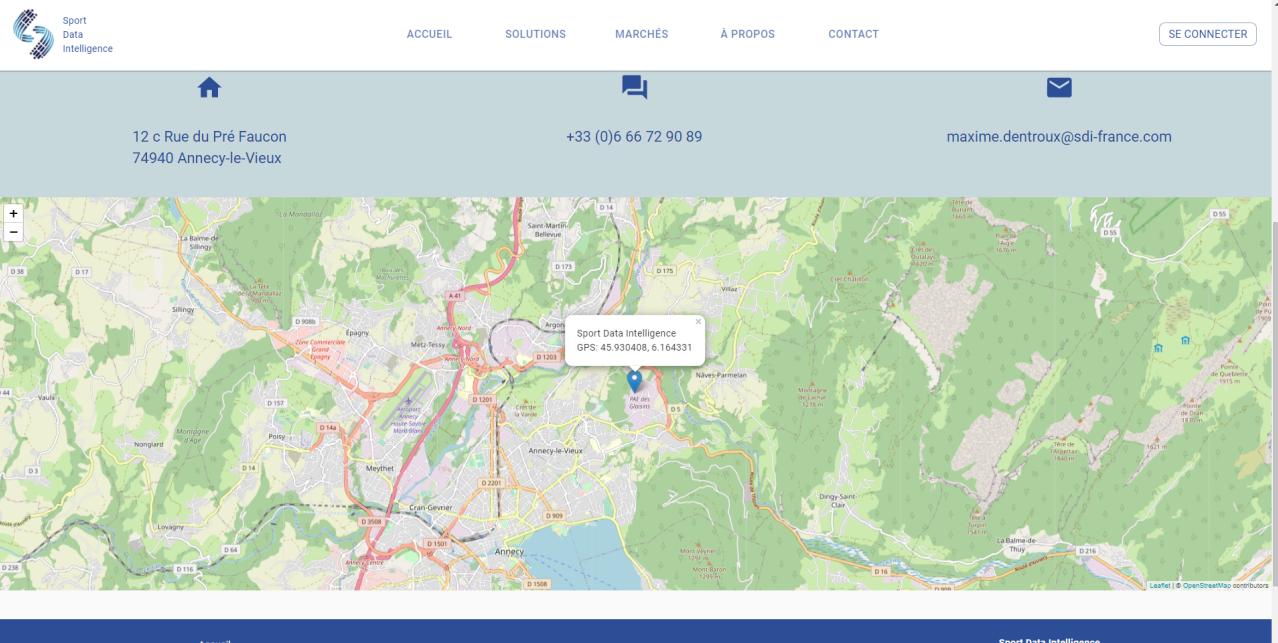


Figure 37: Le résultat

#### 4.1.6 Le "Banner"

En haut de chaque page (à l'exception de la page d'accueil, où figure le "hero") une image est affichée sur la totalité de la largeur de la page. Au lieu de ré-écrire ce code à chaque fois, j'ai créé un petit composant "Banner" qui prends en paramètre le chemin (path) d'une image (optionnelle) et/ou du texte (également optionnel) puis affiche les deux en haut de la page, en dessous de la barre de navigation, et en respectant le côté "responsive" (modification de la hauteur d'affichage selon la largeur de l'écran des utilisateurs).

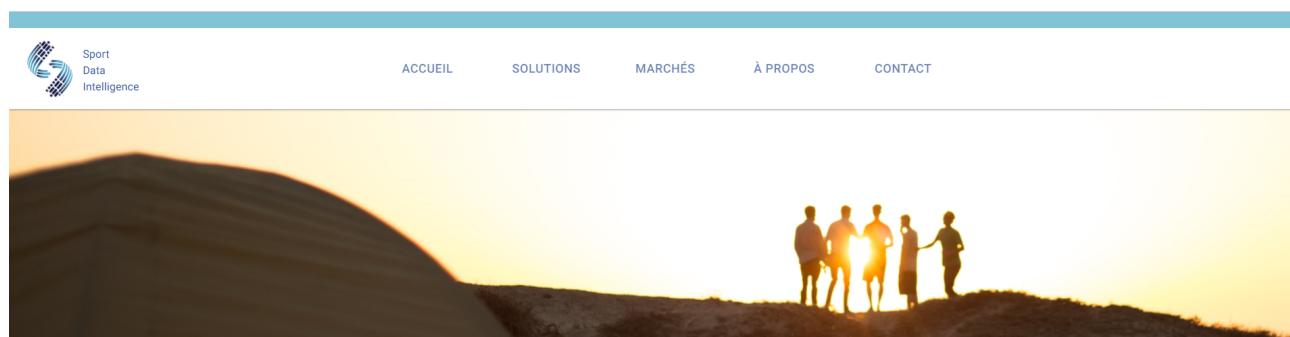


Figure 38: Haut de page avec barre de navigation et banner

## 4.1.7 Les titres des pages

Dans la même démarche, au lieu de coder à chaque fois l'affichage des titres des pages et gérer l'affichage et le côté responsive (identique à chaque fois), j'ai extrait cette fonctionnalité dans un petit composant séparé réutilisable.

```
/**  
 * Renders formatted page title component  
 * @param {string} pageTitle - Title of the page  
 */  
  
const PageTitle = ({ pageTitle }) => {  
  const classes = Styles();  
  
  return (  
    <Typography  
      className={classes.pageTitle}  
      variant="h2"  
      color="primary"  
      align="center"  
    >  
      {pageTitle}  
    </Typography>  
  );  
};  
  
PageTitle.propTypes = {  
  pageTitle: PropTypes.string.isRequired,  
};  
  
export default PageTitle;
```

Figure 39: Le composant "PageTitle" ...

```
<PageTitle pageTitle="Marchés" />
```

Figure 40: ... et son implémentation

## 4.1.8 Les partenaires

En bas de la page d'accueil, ainsi que sur la page "à propos", on affiche les logos des différents partenaires de l'entreprise. Toujours dans la même optique "Dont Repeat Yourself", j'ai codé cet affichage dans un composant séparé et responsive, ensuite on peut l'insérer partout où l'on veut.



Figure 41: Composant qui affiche les partenaires de l'entreprise

## 4.1.9 Le "Layout"

Afin d'éviter la répétition inutile de code, et en vue de la fonctionnalité "Single Page Application" et le fonctionnement du routeur Next.js, j'ai créé un composant "Layout" qui fonctionne comme "wrapper" et entoure le contenu de chaque page avec la barre de navigation en haut, et le footer en bas de la page.

Pour créer une nouvelle page du site on n'a qu'à insérer un nouveau fichier dans le dossier "Pages" du site et y ajouter son contenu, le fonctionnement et affichage de la barre de navigation et footer sont automatiquement présents sur toutes les pages.

```
return (
  <Provider store={store}>
    {/* <Fragment> */}
    <Head>
      <title>`Sport Data Intelligence - ${router.pathname.replace(
        '/',
        ''
      )}`</title>
      <link rel="icon" type="image/png" href="/imgs/LogoSDIfondblanc.png" />
      <meta
        name="viewport"
        content="minimum-scale=1, initial-scale=1, width=device-width"
      />
      <link
        rel="stylesheet"
        href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
        integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A=="
        crossorigin=""
      />
    </Head>
    {/* Load custom theme and apply a base theme in the first request */}
    <ThemeProvider theme={theme}>
      <CssBaseline />
      <Layout>
        <Component {...pageProps} />
      </Layout>
    </ThemeProvider>
    {/* <Fragment> */}
  </Provider>
);
```

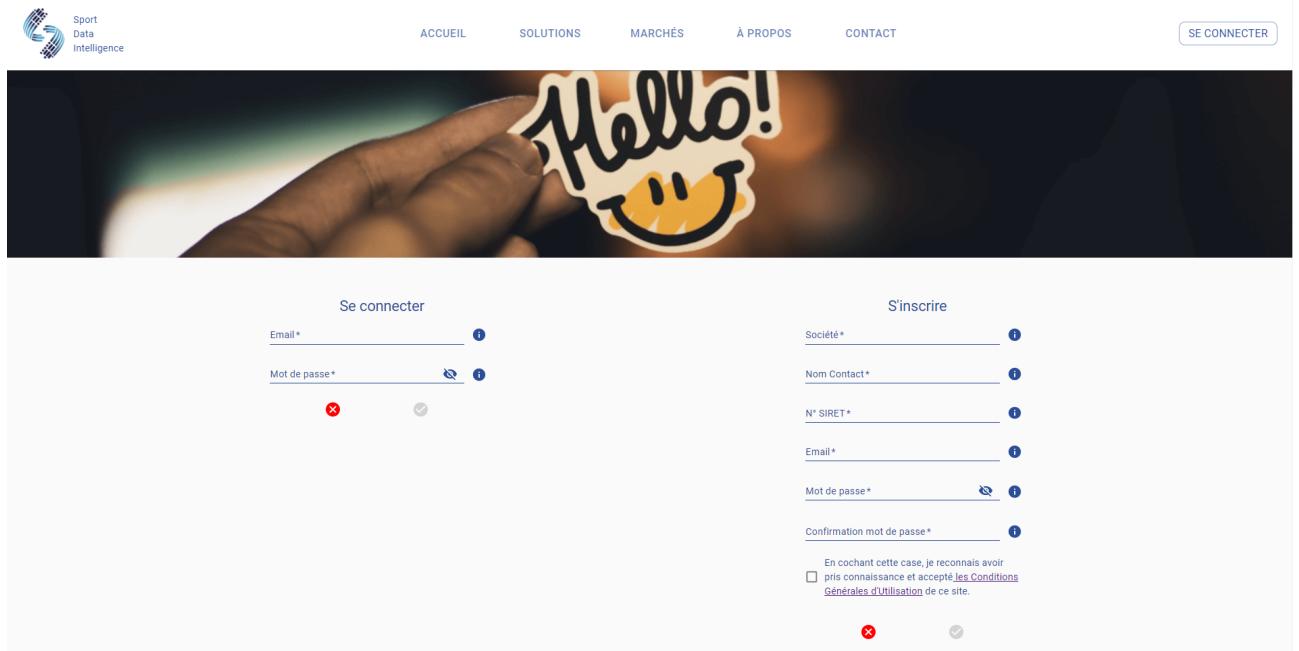
Figure 42: La page "`_app.jsx`", point d'entrée de l'application. On voit que le composant "Layout" entoure "`<Component {...pageProps} />`", l'élément dont le router NextJs se sert pour faire un "rendering" des différentes pages du site.

```
/**  
 * Wraps Navbar and Footer around children  
 */  
const Layout = ({ children }) => {  
  const classes = Styles();  
  
  return (  
    <>  
      <div className={classes.topborder} />  
      <Navbar />  
      {children}  
      <Footer />  
    </>  
  );  
};  
  
Layout.propTypes = {  
  children: PropTypes.node.isRequired,  
};  
  
export default Layout;
```

*Figure 43: Le composant "Layout" reçoit le "prop" "children" pour l'entourer avec la barre de navigation et le footer. Le "div" au dessus rajoute un petit élément visuel, une bande colorée qui s'affiche au dessus de la barre de navigation quand on est en haut de la page, et qui disparaît lorsqu'on descend.*

#### 4.1.10 La page "Connect"

Vers la fin du stage, la partie statique étant finie, j'ai fait une première proposition pour la page où les visiteurs sont amenés après avoir cliqué sur le bouton "Se connecter" de la barre de navigation.



The screenshot shows a website header with a logo for 'Sport Data Intelligence' and navigation links for ACCUEIL, SOLUTIONS, MARCHÉS, À PROPOS, and CONTACT. A 'SE CONNECTER' button is also present. Below the header is a large banner featuring a hand holding a yellow smiley face sticker with the word 'Hello!' written on it. The main content area contains two forms: 'Se connecter' (left) and 'S'inscrire' (right). The 'Se connecter' form has fields for 'Email\*' and 'Mot de passe\*', with a password strength indicator showing a red 'X'. The 'S'inscrire' form has fields for 'Société\*', 'Nom Contact\*', 'N° SIRET\*', 'Email\*', 'Mot de passe\*', and 'Confirmation mot de passe\*'. It also includes a checkbox for accepting terms and conditions, with a note in small print below it. Both forms include red 'X' and green checkmark icons at the bottom right.

Figure 44: La page "Connect"

Les utilisateurs ont le choix: ils peuvent se connecter s'ils possèdent un nom d'utilisateur et un mot de passe, ou bien ils peuvent s'inscrire pour recevoir des identifiants par la suite.

Pour le confort des utilisateurs, mais aussi et surtout pour sécuriser le site, les entrées de chaque champ sont contrôlées avant d'être validées. Ainsi j'ai créé un composant "FormInputField" qui peut être "configuré" à l'aide des paramètres requis suivants:

- le label, nom et "id" du champ
- une variable qui reçoit la valeur saisie
- le type du champ (optionnel, "texte" par défaut)
- le "helperText" qui est affiché quand la valeur entrée ne correspond pas au critères
- le texte qui est affiché dans une bulle qui s'affiche quand on positionne le curseur au dessus d'une icône "information" à côté du champ, et qui informe les utilisateurs par rapport aux caractères autorisés
- la méthode qui sera exécutée quand l'utilisateur saisit une valeur dans le champ

```

const FormInputField = ({  
    fieldLabel,  
    fieldId,  
    fieldName,  
    fieldValue,  
    fieldType,  
    fieldHelper,  
    fieldChange,  
    fieldCharacters,  
}) => {  
    const classes = Styles();  
  
    return (  
        <Grid item>  
            <Box className={classes.iconTextAlign}>  
                <TextField  
                    label={fieldLabel}  
                    id={fieldId}  
                    name={fieldName}  
                    value={fieldValue}  
                    type={fieldType}  
                    required  
                    fullWidth  
                    error={fieldHelper.length !== 0}  
                    helperText={fieldHelper}  
                    onChange={fieldChange}  
                />  
                <Tooltip disableFocusListener title={fieldCharacters}>  
                    <IconButton aria-label="Info caractères autorisés">  
                        <InfoIcon color="primary" />  
                    </IconButton>  
                </Tooltip>  
            </Box>  
        </Grid>  
    );  
};

```

*Figure 45: Le composant "FormInputField"*

Le champ pour la saisie du mot de passe est le seul champ qui n'utilise pas le "FormInputField", comme il a besoin de la fonctionnalité supplémentaire d'affichage (ou non) en clair du mot de passe.

```

/* Password field has extra functionality and cannot use 'FormInputField' */
<Grid item>
  <Box className={classes.iconTextAlign}>
    <FormControl>
      <InputLabel required htmlFor="password">
        Mot de passe
      </InputLabel>
      <Input
        id="sign-in-password"
        name="password"
        value={formData.password}
        type={showPassword ? "text" : "password"}
        error={fieldHelper.password.length !== 0}
        onChange={onChange}
        style={{ width: "20rem" }}
        endAdornment={

          <InputAdornment position="end">
            <IconButton
              aria-label="toggle password visibility"
              onClick={handleClickShowPassword}
              onMouseDown={handleMouseDownPassword}
            >
              {showPassword ? (
                <Visibility color="primary" />
              ) : (
                <VisibilityOff color="primary" />
              )}
            </IconButton>
          </InputAdornment>
        }
      />
      <FormHelperText id="password" style={{ color: "#f44336" }}>
        {fieldHelper.password}
      </FormHelperText>
    </FormControl>
    <Tooltip disableFocusListener title={passwordFieldCharacters}>
      <IconButton aria-label="Info caractères autorisés">
        <InfoIcon color="primary" />
      </IconButton>
    </Tooltip>
  </Box>
</Grid>

```

Figure 46: Le champ de saisie de mot de passe

Pour gérer les saisies j'ai créé trois pièces de "state" :

- "formData" mémorise les valeurs entrées et est initialisé avec l'objet "initialFormState" qui contient des strings vides :

```

const initialFormState = {
  company: "",
  contactName: "",
  siret: "",
  email: "",
  registerPassword: "",
  confirmPassword: "",
};
const [formData, setFormData] = useState(initialFormState);

```

Figure 47: "formData" et "initialFormState"

- "fieldValid" contient des booléens, initialisés à "false" et qui recevront la valeur "true" quand l'entrée du champ est validé :

```
const initialFieldValidState = {
  company: false,
  contactName: false,
  siret: false,
  email: false,
  registerPassword: false,
  confirmPassword: false,
};
const [fieldValid, setFieldValid] = useState(initialFieldValidState);
```

Figure 48: "fieldValid" et "initialFieldValidState"

- "fieldHelper", initialisé avec des strings vides depuis "initialFieldHelperState", contiendra les textes explicatifs à afficher quand l'utilisateur saisit une valeur qui ne peut être validée :

```
const initialFieldHelperState = {
  company: "",
  contactName: "",
  siret: "",
  email: "",
  registerPassword: "",
  confirmPassword: "",
};
const [fieldHelper, setFieldHelper] = useState(initialFieldHelperState);
```

Figure 49: "fieldHelper" et "initialFieldHelperState"

Lors de l'insertion d'un champ dans le formulaire, en appelant le composant "FormInputField", les liaisons avec les différents pièces de "state" sont faites avec les paramètres évoqués précédemment :

```
<FormInputField
  fieldLabel="Société"
  fieldId="company"
  fieldName="company"
  fieldValue={formData.company}
  fieldHelper={fieldHelper.company}
  fieldChange={onChange}
  fieldCharacters={nameFieldCharacters}
/>
```

Figure 50: Le champ  
"Société" de la partie  
"S'inscrire"

A chaque fois qu'un utilisateur saisit un caractère dans un des champs la fonction "onChange" est exécuté. Cette fonction commence par mettre à jour la valeur dans "formData" afin que le caractère saisi s'affiche bien dans le champ.

Ensuite elle exécute la fonction "ValidateField" qui retourne le booléen "valid" en fonction de la validité de la saisie, ainsi qu'un string contenant du "helperText" au cas où la saisie ne serait pas valide. Ces valeurs sont ensuite insérées dans les objets "fieldValid" et "fieldHelper". Pour le champ "mot de passe" et "confirmation de mot de passe" de la partie "S'inscrire" une deuxième vérification est effectuée afin de s'assurer que les deux valeurs correspondent.

```
const onChange = (event) => {
  setFormData({ ...formData, [event.target.name]: event.target.value });
  let [valid, helper] = ValidateField(event.target.name, event.target.value);
  if (event.target.name === "registerPassword") {
    if (
      event.target.value !== formData.confirmPassword &&
      formData.confirmPassword !== ""
    ) {
      [valid, helper] = [true, "Les mots de passe ne sont pas identiques"];
      setPasswordsMatch(false);
    } else {
      setPasswordsMatch(true);
    }
  }
  if (event.target.name === "confirmPassword") {
    if (
      event.target.value !== formData.registerPassword &&
      formData.registerPassword !== ""
    ) {
      [valid, helper] = [true, "Les mots de passe ne sont pas identiques"];

      setPasswordsMatch(false);
    } else {
      setPasswordsMatch(true);
    }
  }
  setFieldValid({ ...fieldValid, [event.target.name]: valid });
  setFieldHelper({ ...fieldHelper, [event.target.name]: helper });
};
```

Figure 51: La fonction "onChange" de la partie "S'inscrire"

```
const ValidateField = (fieldName, value) => {
  // If the field is empty, validity is returned as 'false' and helpertext empty
  if (value === "") {
    return [false, ""];
  }
  // Otherwise validity of value is checked against corresponding regular expression.
  switch (fieldName) {
    case "company":
    {
      const validCompany = reName.test(value);
      if (!validCompany && value !== "") {
        return [false, "Entrez un nom de société valide."];
      }
    }
  }
};
```

Figure 52: Extrait de la fonction "ValidateField" qui vérifie les valeurs par rapport à une "Regular Expression" correspondante et retourne un vecteur contenant un booléen et un string avec le "helperText" nécessaire

Pour tester la validité des valeurs saisies j'ai mis en place quelques "regular expressions" à l'aide des "experts" sur StackOverflow :

```
// Valid email addresses allowed
const reEmail = /^[^@\w+(\.-\.)?\w+]@[^@\w+(\.-\.)?\w+]*(\.\w{2,3})+$/;
// Minimum 2 characters: letters, numbers, space, comma, dot, apostrophe and hyphen allowed
const reName =
  /^[a-zA-Z0-9àáâãäçééííñóòôõúûýæåâââçééííñóòôõúûýæ ,.-]{2,}$/i;
// Numbers only, 14 digits only
const reNumbers = /^[0-9]{14}$/;
// Minimum 10, maximum 20 characters, at least one uppercase letter, one lowercase letter, one number and one special character:
const rePassword =
  /^(?=.*[a-z])(?=.*[A-Z])(?=.*[\d])(?=.*[$!%*?&#+_-])[A-Za-z\d@$!%*?&#+_-]{10,20}$/;
```

Figure 53: Les différents "regular expressions" utilisées

Quand tous les champs sont "validés", quand le mot de passe est identique à la "confirmation de mot de passe" et l'utilisateur à coché la case d'acceptation des "Conditions Générales d'utilisation" le bouton de validation s'active.

```
// Confirmation button only enabled when all required fields are filled out, box is checked,
// email is a valid address, and password and password confirmation are identical
const confirmationDisabled =
  !fieldValid.company ||
  !fieldValid.contactName ||
  !fieldValid.siret ||
  !fieldValid.email ||
  !fieldValid.registerPassword ||
  !fieldValid.confirmPassword ||
  !passwordsMatch ||
  !checked;
```

Figure 54: Vérification de tous les booléens...

```
<IconButton
  onClick={submitForm}
  aria-label="Confirmer"
  className={classes.connectButton}
  style={{
    color: `${confirmationDisabled ? "lightgrey" : "green"}`,
    margin: "0 1rem",
  }}
  disabled={confirmationDisabled}
>
  <CheckCircleIcon />
</IconButton>
```

Figure 55: ... afin d'activer le bouton de confirmation...

```
const submitForm = (event) => {
  event.preventDefault();
  dispatch(login(formData));
};
```

Figure 56: ... et envoyer les valeurs du formulaire (formData) au serveur à l'aide du "dispatch" de React-redux

J'ai appris les bases de React-Redux, toutefois afin de l'intégrer dans la partie back-end existante et aussi pour des raisons de manque de temps (dernière semaine du stage) le "store", les "reducers" et la gestion des "tokens" de connexion ont été mises en place par le CTO, M. Coelho.

Ainsi on obtient un formulaire fonctionnel, facile à remplir pour les utilisateurs, et sécurisé.

N° SIRET\*

sdfsdf

Entrez un numéro de SIRET valide.

Figure 57: Contrôle des saisies

Mot de passe\*

Confirmation mot de passe

Le mot de passe doit être composé de 10 à 20 caractères et contenir au moins une majuscule, une minuscule, un chiffre et un caractère spécial (@!\$%^&\*-). Les lettres accentuées et la cédille ne sont pas acceptées.

Figure 58: Des bulles informatives pour aider les utilisateurs

Se connecter

Email\*

carl@adressemail.com

Mot de passe\*

Motdepasse\$ecre

Entrez un mot de passe valide.

✗ ✓

Figure 59: Le bouton pour valider ne se valide ...

Se connecter

Email\*

carl@adressemail.com

Mot de passe\*

Motdepasse\$ecre1

✗ ✓

Figure 60: ... qu'après avoir saisi tous les champs. Possibilité d'afficher le mot de passe en clair.

## 4.2 Google Analytics

Afin de pouvoir vérifier le comportement des visiteurs du site il est important de faire le lien avec Google Analytics. Cet outil donne plein d'informations tel le nombre de visiteurs, quelles parties du site ils ont visités, combien de temps ils y restent, quel type d'appareil ils utilisent (mobile, desktop), ...

Dans ce but j'ai installé le "package" "react-ga" puis j'ai utilisé le hook "useEffect" pour ajouter l'initialisation et un "event listener" lorsqu'une page est chargée.

```
import { initGA, logPageView } from "../utils/Analytics";

export default function MyApp(props) {
  const { Component, pageProps } = props;
  const store = useStore(pageProps.initialReduxState);
  const router = useRouter();

  useEffect(() => {
    // Remove the server-side injected CSS.
    const jssStyles = document.querySelector("#jss-server-side");
    if (jssStyles) {
      jssStyles.parentElement.removeChild(jssStyles);
    }
    // Initialize react-ga
    initGA();
    if (!router.asPath.includes("?")) {
      logPageView();
    }
    /* eslint-disable-next-line react-hooks/exhaustive-deps */
  }, []);

  useEffect(() => {
    // Listen for page changes after a navigation or when the query changes
    router.events.on("routeChangeComplete", logPageView);
    return () => {
      router.events.off("routeChangeComplete", logPageView);
    };
  }, [router.events]);
}
```

Figure 61: Extrait du code de la page "`_app.js`" avec les hooks "useEffect" pour initialiser "react-ga" et rajouter un "event listener"

Évidemment, pour des raisons de sécurité, le code d'accès pour le compte Google Analytics de l'entreprise ne doit pas être codé en dur dans l'application. Le code sera stocké au niveau du serveur Web et est appelé depuis la fonction d'initialisation.

```

import ReactGA from "react-ga";

export const initGA = () => {
  ReactGA.initialize(process.env.NEXT_PUBLIC_GA_ID);
};

export const logPageView = () => {
  ReactGA.set({ page: window.location.pathname });
  ReactGA.pageview(window.location.pathname);
};

```

Figure 62: Les fonctions utilisés pour l'initialisation et pour le "event listener"

Bien entendu je n'ai pas encore le niveau pour écrire ces fonctionnalités moi-même. Toutefois c'était une bonne occasion pour apprendre à me servir de la documentation fournie avec les différents "packages" qui détaillent comment les utiliser et qui fournissent les extraits de code nécessaires.

## 4.3 Optimisation du site

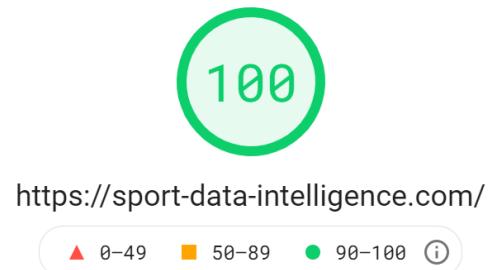
Au bout de 4 semaines de stage on approche la finition du site, tout le contenu "public" a été implémenté, le moment est venu pour publier une première version définitive en ligne. (Une version temporaire a été publiée au bout d'une semaine et demie en raison d'un événement auquel les dirigeants avaient participé. Comme cette version n'avait pas encore tout le contenu il a été décidé de la traiter tel quel - temporaire - et de ne pas s'occuper de l'optimisation de cette version).

Ainsi j'ai pu utiliser plusieurs sites qui "testent" les performances du site, sur différents niveaux:

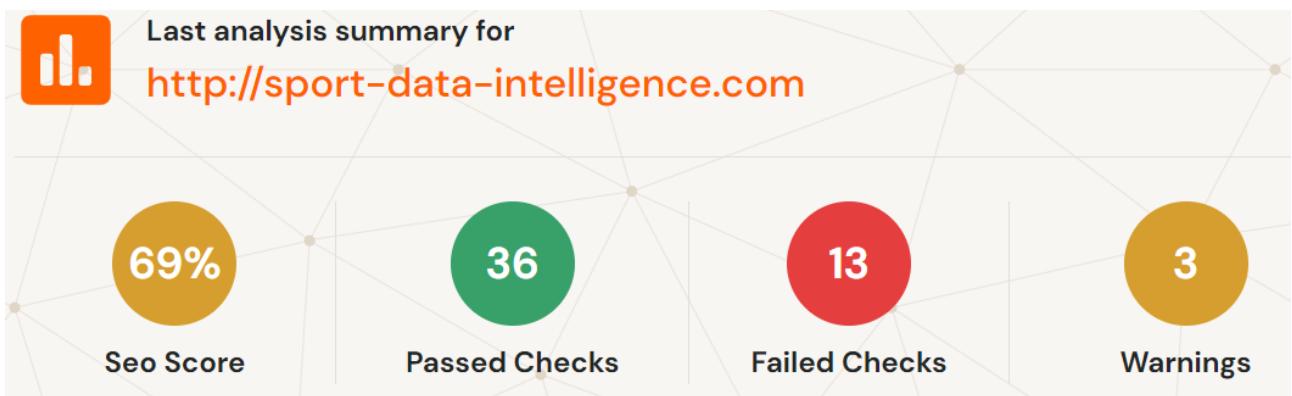
- le "Markup Validation Service" du W3C qui donne des indications par rapport à la structure HTML d'une page. Les résultats sont à prendre avec un grain de sel comme le HTML de la page est "généré" par Next.js, toutefois cela permet de remédier à certains manquements ou oublis (par exemple: la présence de l'attribut "alt" pour les images)
- Google PageSpeed Insights qui donne des informations par rapport à la vitesse de chargement de la page, et des indications et astuces pour l'améliorer
- le site "SEO Site Checkup", un des nombreux outils qui existent et qui donne des informations, astuces, et indications pour améliorer le SEO ("Search Engine Optimisation") du site



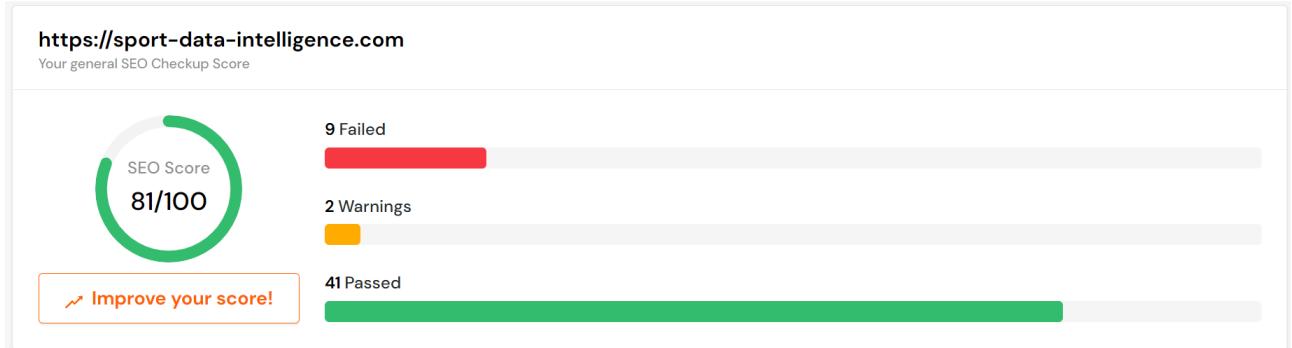
*Figure 64: Avant toute optimisation, un score de 95% pour le site version "desktop" sur Google PageSpeed Insights*



*Figure 63: ... puis un score de 100% après l'optimisation*



*Figure 65: Avant toute optimisation, un score "SEO" de 69% sur SEO Site Checkup*



*Figure 66: ... puis une nette amélioration après optimisation, même s'ils restent toujours quelques points à vérifier et à améliorer*

Suite à ces vérifications, j'ai commencé avec l'optimisation des différentes images utilisées sur le site pour réduire leurs tailles et ainsi améliorer la vitesse de chargement.

Le "validateur" W3C indiquait quelques attributs inutiles qui étaient restés dans le code et que j'ai pu supprimer par la suite (un point auquel je dois faire attention).

Pour optimiser les résultats sur les moteurs de recherche, j'ai ajouté les tags "Meta" nécessaires (leur absence était indiqué par SEO Site Checkup) et j'ai ajouté le fichier Robots.txt ainsi qu'un sitemap XML.

## 5. Evolution / Axes d'amélioration

A la fin de ce stage l'objectif initial a été atteint, à savoir de délivrer le site vitrine de l'entreprise. Le résultat correspond aux attentes des dirigeants, toutefois, en rétrospectif, deux axes d'amélioration se présentent à moi:

- Tout d'abord, je serais tenté de centraliser encore plus le contenu du site. Une grande partie est déjà faite : les éléments de la barre de navigation, l'utilisation des "portraitCard" et "landscapeCard", les liste énumératives, ... Toutefois j'aurais bien aimé aller encore plus loin dans la démarche et avoir, par exemple, quelques fichiers JSON avec la totalité du contenu (titres, paragraphes, listes, ...). Cependant, je dois avouer que l'utilité serait limitée, comme le contenu du site ne sera pas amené à changer régulièrement.
- Je me suis beaucoup amélioré au fur et à mesure de l'avancement du stage, notamment en ce qui est la maîtrise de Material UI. Heureusement, en vue des contraintes (par exemple: la nécessité d'avoir une version basique au bout d'une semaine et demie) je suis passé au moins deux fois sur chaque composant, ce qui m'a permis de faire des "refactoring" et d'améliorer les premières implémentations.

Par exemple, en ce qui est l'application des éléments de "style" je ne suis pas sûr si mon implémentation finale est la meilleure. Beaucoup de ces éléments sont incorporés dans les composants respectives ce qui améliore certainement leur "disponibilité" comme cela permet de les utiliser directement dans une autre application.

Toutefois, en rétrospectif, j'aurais préféré établir un "thème" personnalisé pour le site pour ainsi centraliser toute définition des éléments du style et ne garder que les "fonctionnalités" dans les différents composants.

## 6. Conclusion

Ce stage a été une expérience très enrichissante. Bien sûr, en ce qui concerne le développement informatique, on peut beaucoup apprendre en suivant des cours, en regardant des vidéos et en pratiquant chez soi avec des projets personnels.

Toutefois, coder chez soi n'est qu'une (petite) partie du parcours d'apprentissage. Les six semaines passées chez Sport Data Intelligence m'ont permis d'appliquer mes acquis dans un cadre professionnel : créer un site web d'entreprise, destiné au grand public, et ce au sein d'une équipe, avec des contraintes, des responsabilités, des délais à respecter, des attentes à satisfaire.

Or il est évident qu'en travaillant avec React, Next.js et Material UI pendant six semaines j'ai beaucoup progressé au niveau de ces technologies (principalement React et Material UI comme la partie statique du site ne s'appuie pas tant sur Next.js, peut être à l'exception du router, qui toutefois était déjà en place quand je suis arrivé).

Au moins aussi important à mes yeux cependant sont toutes les activités et compétences qui sont directement liées à la partie "code" et indispensables au bon déroulement d'un projet : la méthode Agile avec les "Daily Scrum Meetings", le "Storyboard" et les "Kanban", les revues de code, l'utilisation de Markdown, et les différentes "mancœuvres" avec Git.

Finalement, ce stage m'a confirmé que j'ai fait le bon choix en entamant ma reconversion professionnelle. C'était un énorme plaisir de pouvoir travailler sur un "vrai" projet, et les six semaines sont passées beaucoup trop vite.

Ce fut une première rencontre avec le monde du "développement web professionnel" que j'ai hâte de renouveler rapidement!

## 7. Remerciements

D'abord, je tiens à remercier les fondateurs de Sport Data Intelligence, Maxime Dentroux et Florian Gaté pour leur confiance et l'opportunité qu'ils m'ont donné.

Ensuite, je tiens à remercier tout particulièrement Camilo Coelho, CTO et maître de stage pour la qualité de son encadrement, ses bons conseils et sa patience.