

Page de garde

Sommaire

Introduction

1. Compétences couvertes

2. Contexte du projet

3. Front-end

4. Back-end

5. Sécurité

6. Jeu d'essai

7. Veille

Conclusion

8. Annexes



# ***O'Clock***

Dossier de projet réalisé dans le cadre de la présentation au

**Titre Professionnel Développeur web et web mobile**

Session Juin 2024

Présenté par Laure SENG

# Remerciements

Je tenais à remercier les personnes de l'école O'Clock qui m'ont accompagnée lors de ma formation en développement web et web mobile.

Merci aux différents professeurs ( Yannick, J-C, Ben, Pierrick, Baptiste et Greg) pour leurs réponses à toutes mes interrogations, à ma tutrice pédagogique Clara Molina pour son accompagnement si humain et ses éclaircissements techniques. Merci à tous les étudiants de la promo Jelly pour tous ces échanges constructifs tout au long de la formation.

Merci à ma famille qui a su s'adapter à mon investissement durant la formation.

Et merci à tous les développeurs qui ont contribué aux outils qui me servent tous les jours à coder et sans lesquels le web ne serait pas ce qu'il est.

# INTRODUCTION

## 1/ A propos de moi

Depuis l'adolescence, j'ai toujours été passionnée par l'informatique et la création sur ordinateur.

J'ai commencé par explorer Word, MS Publisher, Paint Shop Pro. Puis j'ai fait de la musique avec des trackers puis des séquenceurs MIDI et audio. Puis du montage vidéo avec Adobe Première. J'ai même fait un site web avec un éditeur WYSIWIG dont j'ai oublié le nom !

Puis j'ai découvert le monde du libre avec Linux et l'immensité des logiciels libres. C'est là que j'ai essayé de faire des scripts Shell, de faire des sons avec la librairie Csound, des scripts Python avec Gimp... Mais tout cela dans le cadre de mes loisirs et sans jamais arriver à surpasser mes déboires avec l'algorithmie.

Après le bac, je me suis orientée vers le métier de psychomotricienne. Il me permettait d'utiliser ma créativité au quotidien à travers les exercices que j'inventais pour rééduquer les patients tout en s'amusant. Mais bon, hormis pour faire des transmissions sur les dossiers patients et rédiger les bilans, je ne faisais plus trop d'ordinateur...

Durant mon congé maternité, j'ai replongé dans le monde du numérique à travers des MOOC sur l'impression 3D, sur l'Arduino et j'ai découvert OpenClassRooms et le célèbre cours de Mathieu Nebra sur le HTML et le CSS. Et je suis tombée sur la roadmap pour devenir développeur web qui m'a un peu effrayée...

J'ai compris que le développement web est un domaine très vaste et passionnant et que en apprendre les bases seule en vue d'une professionnalisation serait long. J'ai donc décidé de suivre une formation de développeur web et web mobile avec l'école O'Clock dont le programme me semblait complet.

Après avoir suivi cette formation, j'ai engrangé assez de connaissances pour mettre en place des petits projets personnels de mini-jeux en JavaScript, des petits sites statiques et dynamiques en PHP ou avec Symfony.

Et le dernier mois de formation, que l'on surnomme « l'Apothéose », a été l'occasion tant attendue de mettre à profit les connaissances accumulées, tout en s'adonnant aux joies et contraintes du travail en équipe. C'est ainsi que j'ai pu à la fois avoir le plaisir de coder tous les jours et de réfléchir et échanger avec d'autres développeurs autour de notre projet commun : O'Broderie.

## 2/ Présentation du projet

L'idée du projet O'Broderie provient d'un de mes collègues de promo dont la femme est gérante d'une **P.M.E. de broderie**. Cette idée découle d'un **besoin réel** et c'est cela qui m'a donné envie de participer à ce projet.

La patronne de l'entreprise de broderie réalisait la **gestion commerciale** de son entreprise avec un tableur Excel qui était pas assez adapté à ses besoins. L'idée de mon collègue a été de créer une application web dédiée qui desservirait cette gestion commerciale et proposerait des fonctionnalités supplémentaires. L'objectif principal étant **d'être plus pratique que le tableur et aussi plus sécurisé**.

# COMPÉTENCES COUVERTES

L'ensemble des compétences du REAC de développeur web et web mobile sont abordées dans ce projet. :

CP1 - Installer et configurer son environnement de travail en fonction du projet  
web ou web mobile

CP2 - Maquetter des interfaces utilisateur

CP3 - Réaliser des interfaces utilisateur statiques

CP4 - Développer la partie dynamique des interfaces utilisateur

CP5 - Mettre en place une base de données relationnelle

CP6 - Développer des composants d'accès aux données

CP7 - Développer des composants métier côté serveur

CP8 - Documenter le déploiement d'une application dynamique web ou web  
mobile

Je ne détaillerai pas l'installation de l'environnement de travail nécessaire au développement de l'application ni les procédures suivies pour le déploiement de celle-ci afin de me concentrer dans ce dossier sur les étapes de la réalisation du projet O'Broderie.

# Contexte du projet

## 1/ Objectifs du projet

Le projet O'Broderie a pour but de créer une **application de gestion commerciale pour une PME de broderie**. Cette entreprise vend des prestations de broderie à des particuliers ou des professionnels afin de leur permettre de personnaliser des textiles fournis par le client ou par l'entreprise. Le but de l'application est de permettre de **réaliser des devis à partir d'un premier contact avec le client et de suivre la transaction jusqu'à sa livraison**.

L'application sera principalement **utilisée par la gérante de la PME** de broderie mais sera aussi utilisée par un administrateur. Son **utilisation est optimale sur ordinateur de bureau** mais elle peut se faire aussi sur écrans plus réduits.

## 2/ Présentation de l'équipe

L'équipe est constituée de cinq personnes auxquelles nous avons attribué des rôles pour simuler un cadre de travail qui s'inspire de la méthodologie Agile:

- Fayis : **Product Owner**, c'est lui qui **connaît le produit et les attentes du client**
- Laure : **Scrum Master**, c'est elle qui organise les **daily** et s'occupe de la **continuité et de la fluidité des échanges** entre les membres de l'équipe
- Laurent : **Lead Back**, c'est le référent technique du **backend**
- Théophile: **Lead Front**, c'est le référent technique du **frontend**
- Eva : **Git Master**, c'est elle qui veille au respect du workflow choisi pour le **versioning**

## 3/ Cadre spatio-temporel

Le déroulement d'une journée-type était le suivant : **daily** de 9h à 9h15-30 selon les problématiques rencontrées, puis **travail par sous-groupes** jusqu'à 15h, puis travail de **documentation** de 15h à 17h ( voire plus...).

La principale contrainte temporelle a été de **respecter la date de livraison du projet** pour le 22 avril 2024, c'est à dire après **4 semaines de travail (sprints)** , dont la première a été consacrée à l'élaboration du projet, la deuxième et la troisième à la réalisation des fonctionnalités principales et la quatrième à la finalisation de ces fonctionnalités.

Tout le projet s'est réalisé en distanciel et s'est appuyé sur des outils de travail collaboratif.

# Conception du projet

Le premier sprint a été consacré à la réflexion autour du projet, en respectant à la fois les besoins du client et les contraintes liées au contexte du projet décrits précédemment.

Mettre à plat les idées sous forme de documents structurés a facilité amplement la phase de réalisation qui suivra, concrétisation de ces documents par le codage de l'application.

## 1/ Minimum Viable Product ( MVP)

Le MVP permet de circonscrire les **fonctionnalités minimales attendues** pour la livraison de notre application :

- Gérer le catalogue de clients
- Gérer le catalogue de textiles
- Gérer le catalogue de broderies
- Générer un devis suite à un contact client (mail, téléphone...)
- Établir une commande d'après ce devis
- Émettre une facture une fois la commande préparée
- Générer les pdf des devis, commandes, factures

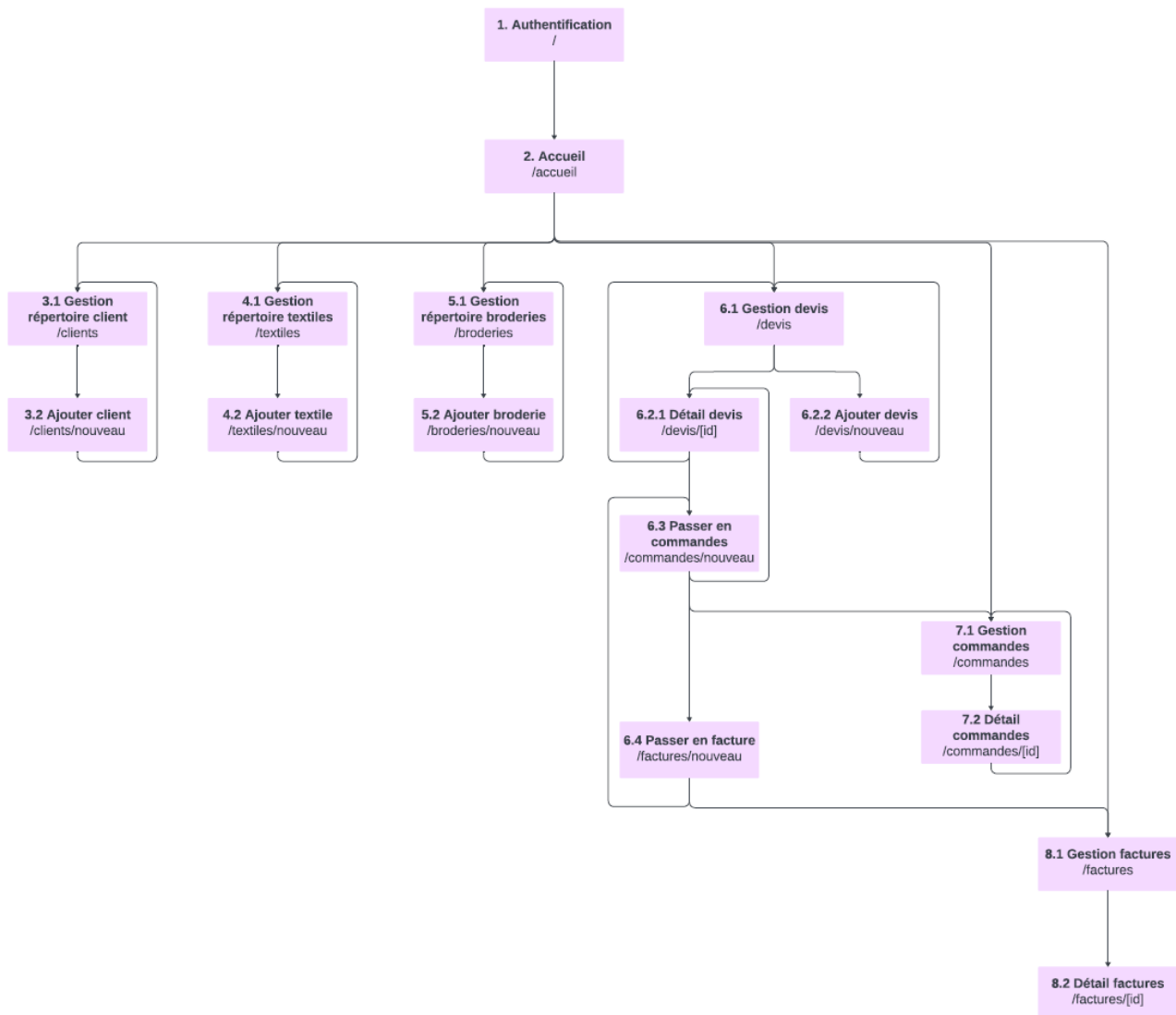
Nous avons évoqué la possibilité d'implémenter des **fonctionnalités supplémentaires**, selon le temps qu'il nous resterait et les possibilités techniques :

- Tableau de bord de suivi commercial (statistiques des devis, commandes, factures)
- Automatisation de l'envoi par mail des documents commerciaux
- Recherche avancée des différentes entités de manière ergonomique
- Authentification et mise en place du JSON Web Token (JWT)



## 2/ Arborescence des routes, user stories et scénario utilisateur

Voici le schéma que nous avons réalisé de l'arborescence du site :



Nous avons aussi listé l'ensemble des **user stories** (cf annexes) dont voici quelques exemples :

- Répertoire des clients: En tant que USER je veux consulter le répertoire client afin d'obtenir des informations sur les clients.
- Répertoire des textiles: En tant que USER je veux créer une fiche textile afin d'ajouter un textile dans le répertoire des textiles.
- Répertoire des broderies : En tant que USER je veux modifier une fiche broderie afin de mettre à jour des informations liées au broderie.
- Répertoire des devis : En tant que USER je veux créer un devis afin d'enregistrer une nouvelle demande client.
- Répertoire des devis : En tant que USER je veux modifier un devis afin de mettre à jour une demande client

- Répertoire des devis : En tant que USER je veux pouvoir rechercher une broderie afin de lier facilement la broderie au devis.

- Répertoire des commandes: En tant que USER je veux valider le devis afin de le passer en commande.

Nous avons aussi décrit un **scénario utilisateur** qui trace le cheminement lors d'une visite classique de la création d'un devis :

1. L'utilisateur s'authentifie et arrive sur la page d'accueil.
2. Il se rend sur la page gestion de devis, puis clique sur 'créer un devis'.
3. Il renseigne son client et les produits via des listes déroulantes puis renseigne les champs texte pour les autres informations.
4. Il génère le PDF du devis qu'il envoie au client.
5. Quand le client aura validé le devis, l'utilisateur sélectionne son devis dans la page de gestion des devis et arrive sur la page de modification du devis pour faire d'éventuelles modifications.
6. Après vérification des informations, il clique sur 'passer en commande'.
7. Il arrive sur la page de création de commande. Il renseigne les dates de livraisons contractuelles.
8. Il génère l'accusé de réception de commande qu'il envoie au client.
9. Lorsque la commande est prête à être expédiée, il clique sur 'passer en facture' pour générer le pdf de la facture qu'il envoie au client.
10. Après paiement de la facture par le client, il sélectionne la facture dans la page de gestion de facture.
11. Il passe la facture au statut 'archivé'.

## 4/ Technologies utilisées pour le projet :

Le frontend et le backend de l'application ont été réalisés avec des technologies différentes. La communication entre le frontend et la base de données a été faite par l'intermédiaire d'une **API** (Application Programming Interface) REST. Cette API fournira des **endpoints** au frontend qui pourra ainsi récupérer les données de la base de données.

Voici la liste des principales technologies utilisées pour chacun de ces domaines de notre application avec quelques uns de leurs intérêts. J'illustrerais plus en détail l'intérêt de ces technologies pour notre projet dans la partie suivante.

### a/ Technologies du front-end

Le frontend de l'application a utilisé les technologies suivantes :

- la librairie **React.js** qui, grâce notamment à ses concepts de **DOM virtuel**, **JSX composants**, **props** et **state**, va nous permettre de créer une **Single Page Application**, réactive et sans besoin de rechargement de la page.
- la librairie **Redux** qui va permettre la mise en place d'un state global via un **store**.
- des librairies complémentaires telles que **React-Router** et **React-Redux**.
- la librairie de composants **Material UI**.
- la librairie **SASS** qui utilise le SCSS, CSS avancé.
- la librairie **Axios** pour faire des requêtes http.
- **Vite.js**, bundler pour combiner les fichiers et optimiser l'application.

## **b/ Technologies du back-end**

Le backend de l'application a utilisé les technologies suivantes :

- Le framework **Symfony** qui optimise la création d'une application, notamment grâce à son outil en ligne de commande **bin/console**, à certains de ses composants comme le **MakerBundle** qui permet de créer en ligne de commande des éléments du code , ou encore le composant **HttpFoundation** qui permet notamment d'interagir avec les requêtes et les réponses HTTP via les classes Request et Response.
- **L'ORM** (Object Relational Mapping) **Doctrine** qui va faciliter les interactions avec la base de données en **transformant les entités en objets PHP** et en traduisant les modifications de ceux-ci en requêtes SQL. Il va **gérer aussi les relations** entre les entités et la **validation des données** et fournit un système de sauvegarde des changements avec les **migrations**.
- Des **composants** de Symfony spécifiques tels que **nelmio/alice** pour créer des fixtures, **LexikJWTAuthenticationBundle** pour générer un JWT, **KnpSnappyBundle** pour générer un pdf, **Mailer** pour l'envoi du pdf par email.

La **base de données** a été établie en utilisant le **SGBDR( Système de gestion de Base de Données Relationnelle ) MySQL**. L'interface **Adminer** a été utilisée pour visualiser la base de données directement dans le navigateur.

# FRONTEND

## 1/ Wireframes

Après la réflexion autour de l'arborescence du site, nous avons réfléchi sur la disposition des différents éléments des pages et comment les informations seront présentées pour que l'**interface utilisateur (UI)** soit à la fois claire et épurée, pour permettre une **expérience utilisateur (UX)** intuitive et fluide.

Nous avons donc utilisé le logiciel Figma pour créer des wireframes sur lesquels figurent les **différents éléments et les légendes** correspondant à leurs fonctions et contenus.

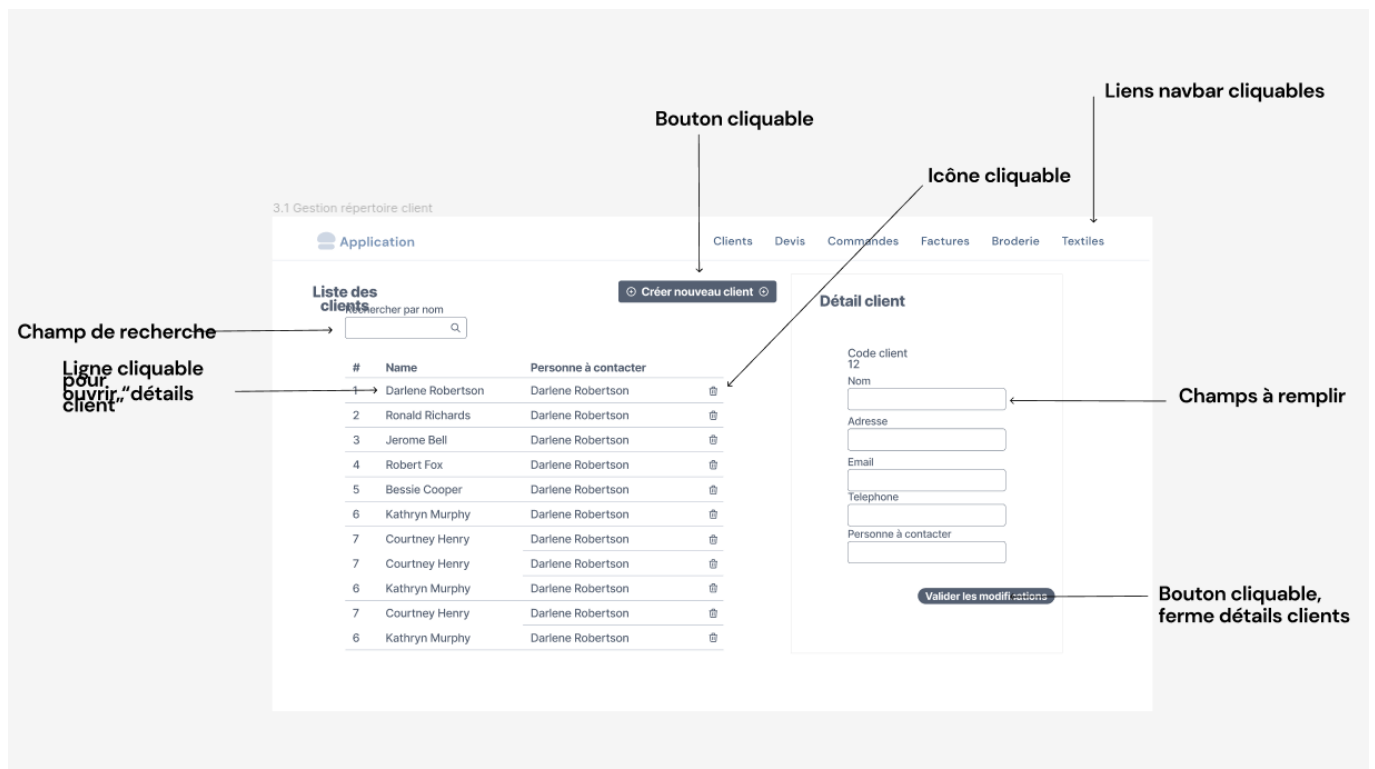
La structure des pages est sensiblement la même d'une catégories de données à l'autre, nous avons donc adopté les mêmes structure pour les pages concernant les textiles, les broderies, les clients et les devis.

Voici un exemple de wireframes définissant le **zoning**, découpage de la page en éléments, pour les pages rencontrées lors du parcours de l'utilisateur au cours de la création d'un nouveau client.

La **page listant les clients** a au début été pensée ainsi :

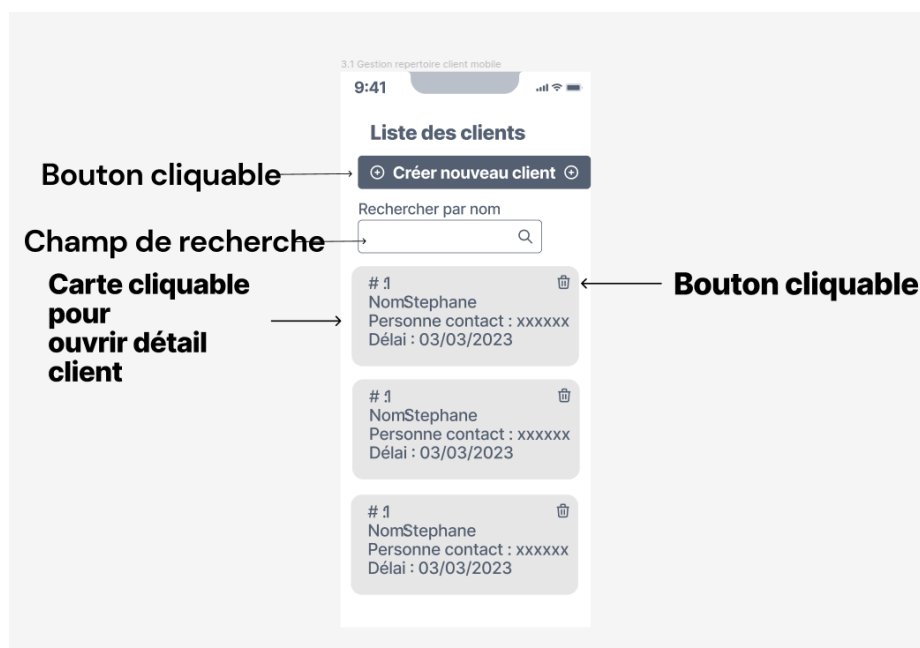
### a/ pour ordinateur de bureau :

La page est divisée en deux parties, une partie sur la gauche présentant la liste de tous les clients et une partie sur la droite qui se mettrait à jour quand l'utilisateur cliquerait sur un client de la liste et qui permettrait d'éditer les détails du client sur la même page.



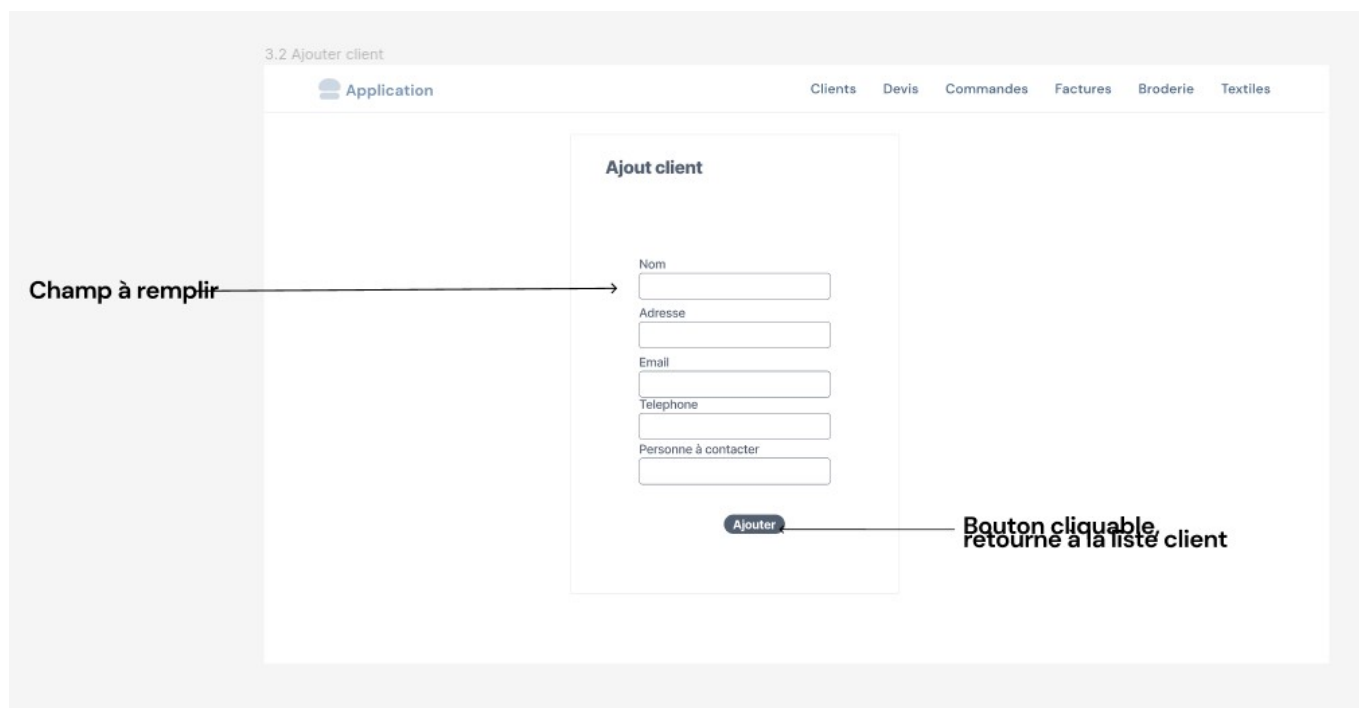
## b/ pour écrans de supports mobiles (téléphones et tablettes en mode portrait)

Nous avons opté pour un affichage des items en colonne.



La page de création d'un client est un formulaire assez classique :

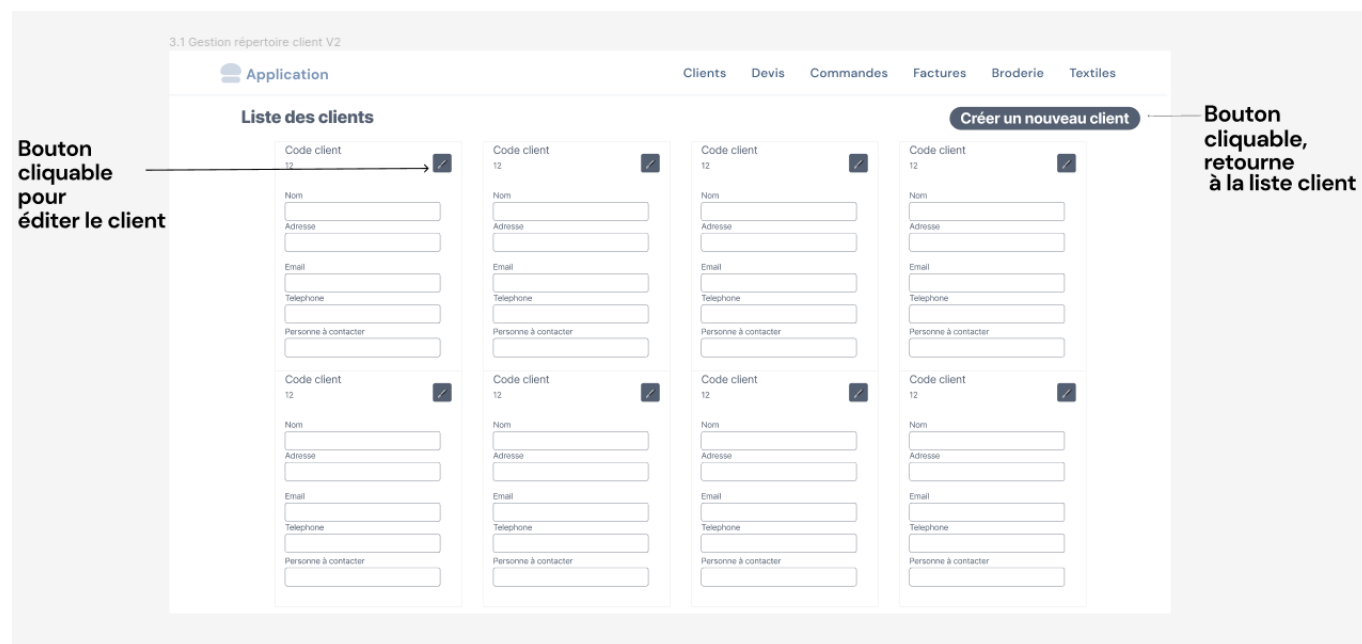
## a/ pour ordinateur de bureau :



**b/ pour écrans de supports mobiles (téléphones et tablettes en mode portrait) :**



Au cours de la création du frontend de l'application, nous avons préféré simplifier la page d'affichage de la liste des items version Desktop par une vue sous forme d'un cadre global qui contiendrait des cards correspondant à chaque item, ce qui ressemble à cela :



Les wireframes originaux conçus en début de projet sont consultables à cette adresse :

## 2/ Création des interfaces utilisateurs

Une fois la **structure des pages décidée**, nous avons pu mettre en place les interfaces utilisateurs correspondant.

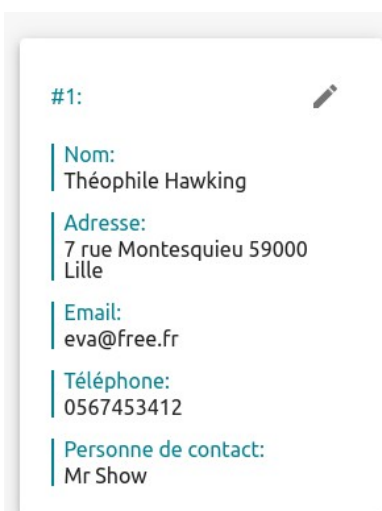
### a/ Intérêts de React pour la création de pages statiques et leur dynamisation

Les exemples de cette partie sont tirés des page de visualisation du répertoire « client ».

Un premier intérêt de React est qu'il induit un **découpage de notre interface en composants** ce qui va permettre notamment une réutilisation du code. Nous avons choisi d'utiliser une librairie de composants prêts à l'emploi et customisables : **Material UI (MUI)**.

Tout d'abord nous avons **créé les composants les plus petits**, tels que les labels des éléments, basés sur le composant `<Typography/>` de MUI, les champs des formulaires, basés sur le composant `<TextField/>` de MUI, et nous avons sélectionné des composants MUI à utiliser tels que les composants `<Paper/>`, `<Stack/>`, `<IconButton/>` .

Nous avons ensuite créé, en **combinant tous ces composants**, un composant `<Header/>` commun à toutes nos pages, des composants `<Card/>` configurés selon nos besoins, pour pouvoir ensuite les **réutiliser** dans les différentes pages.

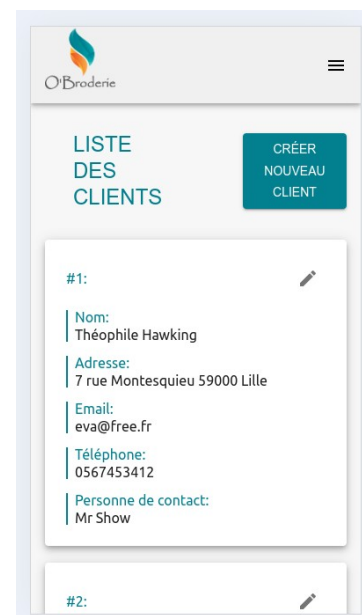
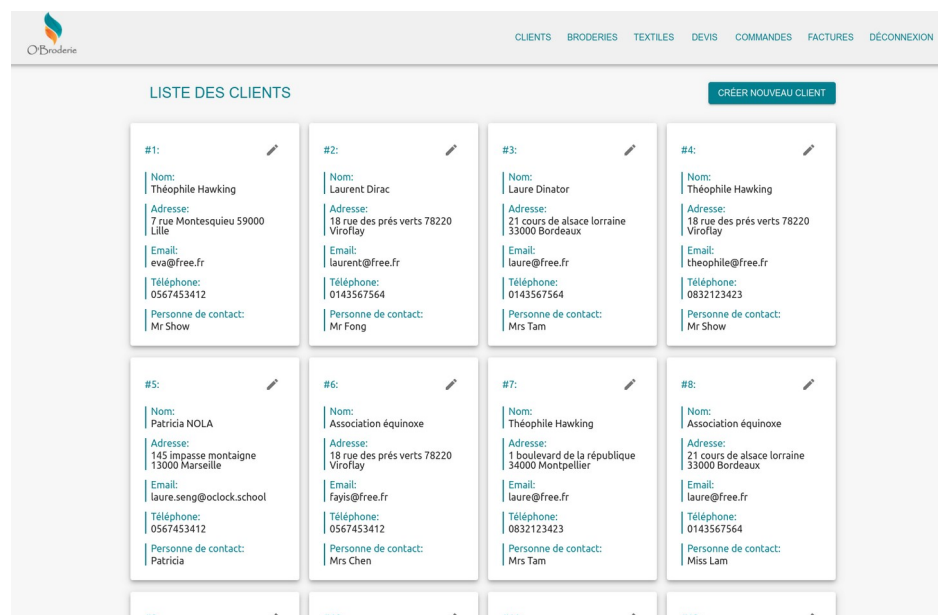


```
1  const CardCustomerShow = ({ customer, handleEdit }) => {
2    const onClickEdit = () => {
3      handleEdit(customer.id);
4    };
5    return (
6      <Paper
7        sx={{ padding: '1.5rem', margin: '0.1rem auto', width: '100%',
8          }} elevation={5}
9      >
10     <Stack direction="column" spacing={2}>
11       <Stack direction="row" justifyContent="space-between" alignItems="center">
12         <TypoLabel text={`#${customer.id}`} />
13         <IconButton aria-label="edit" onClick={onClickEdit}>
14           <EditIcon />
15         </IconButton>
16       </Stack>
17
18       <Stack spacing={2}>
19         <TypoLabelData label="Nom" data={customer.name} />
20         <TypoLabelData label="Adresse" data={customer.address} />
21         <TypoLabelData label="Email" data={customer.email} />
22         <TypoLabelData label="Téléphone" data={customer.phoneNumber} />
23         <TypoLabelData label="Personne de contact" data={customer.contact} />
24       </Stack>
25     </Stack>
26   </Paper>
27 );
28 };
```

Un autre avantage de React est la possibilité d'utiliser le **langage JSX** (JavaScript XML) qui permet d'écrire du **HTML tout en incluant du JavaScript** à l'intérieur ce qui facilite la dynamisation des pages, notamment les pages listant des items. En effet, pour chaque entité, nous avons une page qui **liste l'ensemble des enregistrements** pour cette entité, une page de **création d'un nouvel enregistrement** et une **page d'édition** d'un enregistrement existant. Ces pages ayant des similarités, nous avons pu tirer bénéfice du découpage en composants ainsi que des avantages du JSX pour être plus rapides dans l'implémentation des différentes pages.

Si on prend par exemple la page listant les clients, nous avons pu utiliser la **fonction map()** sur le tableau d'objets *customerList* pour créer la page affichant la liste des clients à partir du composant `<CardCustomerShow/>`.

```
1 <Grid container spacing={2} justifyContent="center" alignItems="stretch">
2   {customerList.map((customer) => {
3     return (
4       <Grid item key={customer.id} xs={12} sm={6} md={4} lg={3} display="flex">
5         <CardCustomerShow customer={customer}
6           handleEdit={() =>
7             navigate(`/clients/modifier/${customer.id}`)
8           }
9       </Grid>
10    )
11  }
12 }
13 </Grid>
```



Une dernière fonctionnalité essentielle de React qui facilite l'implémentation de nos pages est le principe des « **props** ». Ce sont des propriétés propres à chaque composant auxquelles on va pouvoir attribuer des valeurs différentes. Un même composant pourra donc être réutilisé en modifiant uniquement la valeur de ses props. Dans l'exemple ci-dessus,



*customer* est une prop de `<CardCustomerShow/>`. Cela nous permet d'avoir le contenu de la carte qui s'adapte aux données du client à afficher.

## B/ Mise en place du style

Pour la mise en place du style, nous avons eu recours à plusieurs procédés : le SCSS, la configuration du thème de MUI, le style à l'intérieur des composants. Nous avons utilisé le **fichier de reset fourni par MUI** pour mettre à plat les valeurs des propriétés CSS des éléments HTML.

Le **SCSS** permet d'avoir recours à des fonctionnalités de CSS avancé telles que le **nested CSS**, les **variables CSS**. Nous l'avons utilisé pour mettre en place le style pour le composant `<Header/>` et le composant `<App/>` et ainsi configurer la disposition des éléments, les couleurs, le texte dans des fichiers .scss...

```
1 .header-container {
2   background-color: #5a5c5e;
3 }
4
5 .logo {
6   display: flex;
7   align-items: center;
8   justify-content: space-between;
9 }
10
11 .logo-text {
12   font-family: monospace;
13   font-weight: 700;
14   letter-spacing: 0.3rem;
15   color: white;
16   text-decoration: none;
17 }
```

Nous avons aussi configuré le thème de MUI grâce au **ThemeProvider** et à la méthode **createTheme()** fournis par MUI, afin de configurer les couleurs et les polices de l'application. Ce thème est importé dans le composant `<App/>` et sera appliqué à toute l'application. C'est pratique si, ultérieurement, on décidait de changer la charte graphique du site.

```
1  const theme = createTheme({
2    palette: {
3      colors: {
4        editable: '#ffffff',
5        uneditable: '#e0e0e0',
6        white: '#ffffff',
7        grey: '#cccccc',
8      },
9      primary: {
10       main: '#03808F',
11     },
12     secondary: {
13       main: '#FAA744',
14     },
15     background: {
16       paper: '#FFFFFF',
17     },
18   },
19   typography: {
20     fontFamily: ['Segoe UI', 'Roboto', 'Arial', 'sans-serif'].join(','),
21     h1: { fontSize: '2.5rem' },
22     h2: { fontSize: '2.5rem' },
23     h3: { fontSize: '2rem' },
24     h4: { fontSize: '1.5rem' },
25     body3: { fontWeight: 400 },
26     body4: { fontWeight: 400 },
27     body5: { fontSize: '0.8rem', fontWeight: 400, color: '#898989' },
28   },
29 });
```

On peut aussi configurer le style des composants MUI via **des propriétés du système MUI**.

```
1  <Stack
2    direction="row"
3    justifyContent="space-between"
4    alignItems="center"
5  >
```

Ou encore via la **prop sx**. Ces deux techniques permettent d'écrire du style *à l'intérieur des balises HTML*.

```
1  <Paper
2    variant="outlined"
3    sx={{
4      padding: '1rem',
5      margin: '0.5rem auto',
6      maxWidth: '80rem',
7      cursor: 'auto',
8    }}
9  >
```

La demande du client est de pouvoir utiliser l'application sur un **ordinateur de bureau**. Nous avons tout de même implémenté une **version mobile** avec toutes les fonctionnalités de l'application, mais l'édition est plus pratique sur ordinateur.

Pour tout projet HTML, il est important de configurer le **viewport** avec cette balise meta :

```
1 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
2
```

Elle

garantie que la largeur de la surface visible du navigateur sera de la largeur de l'écran et que le zoom sera à 100 %.

MUI met à disposition un **composant <Grid/>** qui est basé sur une subdivision en 12 colonnes et utilise l'algorithme de mise en page **Flexbox**. Ce composant peut être de type **container** ou **item**. Les dimensions des *items* pourront être définies en **pourcentage** pour être fluides. Il y a par défaut cinq **breakpoints** : xs, sm, md, lg et xl.

```
1 <Grid item key={customer.id} xs={12} sm={6} md={4} lg={3} display="flex">
2
```

Légende :Exemple de card responsive

Voici un autre extrait de code pour illustrer l'ajustement de la mise en page en fonction de la taille de l'écran, pour la page listant les contrats , cette fois-ci en utilisant les propriétés MUI :

```
1 <Stack
2   direction={{ xs: 'column', sm: 'row' }}
3   justifyContent={{ xs: 'center', sm: 'space-between' }}
4   alignItems={{ xs: 'center', sm: 'center' }}
5   spacing={1}
6 >
```

Ces techniques sont des alternatives aux **Media Queries** qui ont été aussi utilisées.

Une fois que les éléments statiques sont implémentés, on peut rajouter de l'interactivité grâce aux fonctionnalités de JavaScript.

Dans React, le JavaScript est imbriqué avec le HTML dans le JSX. On va donc utiliser des **attributs d'évènement HTML** plutôt que la méthode `addEventListener()` pour appeler nos **handlers**, fonctions qui seront lancées au déclenchement de l'évènement. De nombreux évènements existent pour détecter des signaux venant du clavier, de la souris ou des changements dans un formulaire. Nous avons principalement utilisé les évènements **'click'**, **'change'** et **'submit'**. Voici trois exemples pour illustrer des interactions possibles, extraits de l'implémentation de la page de listings des broderies.

## A/ Naviguer dans l'application

Dans une application classique, quand l'utilisateur va cliquer sur un bouton envoyant vers une URL (Uniform Resource Locator), cela va l'envoyer vers une nouvelle page. Mais l'un des principes de React est de permettre de créer une **Single Page Application** (SPA). Pour cela, il crée un **DOM virtuel** auquel se rattachent les différents composants qui vont se mettre à jour via leurs **states, états de ceux-ci lors de leurs cycles de vie**.

Nous avons utilisé la librairie React-Router qui permet de simuler la navigation et d'afficher des composants selon une URL. Elle met à disposition des composants tels que **<Route/> pour créer des routes** vers lesquelles nous allons diriger l'utilisateur.

Par exemple, lorsqu'il veut éditer une carte, l'utilisateur va cliquer sur le bouton « Édition », ce qui va déclencher une fonction qui va le diriger vers la « page d'édition », qui est en fait la page régénérée avec les composants propres à la page d'édition.

```
1 // Example of user interactions implementation
2
3 // The user clicks on the edition button
4 // /src/components/CardEmbroidery/CardEmbroideryShow.jsx
5 <IconButton aria-label="edit" onClick={onClickEdit}> </IconButton>
6
7 // It runs the onClickEdit() function that refers to handleEdit() function
8 const CardEmbroideryShow = ({ embroidery, handleEdit }) => {
9   const onClickEdit = () => {
10     handleEdit(embroidery.id);
11   };
12
13 // handleEdit() function is a prop which value is the function navigate()
14 // from React-Router with the route aimed in argument
15 // /src/views/embroideries/Embroiderylist.jsx
16 <CardEmbroideryShow
17   embroidery={embroidery}
18   handleEdit={() =>
19     navigate(`/broderies/modifier/${embroidery.id}`)
20   }
21 />
22 // The routes are defined in the App.js file
23 <Route path="/broderies/modifier/:id" element={<EmbroideryEdit/>} />
```

Dans cet exemple, la fonction `handleEdit()` retourne la fonction `navigate()` de React-Router qui permet de simuler la redirection.

## B/ Mise à jour de l'interface

Notre application servant à visualiser, modifier et enregistrer des données en base de données, nous passons par des **formulaires** pour permettre à l'utilisateur d'exécuter ces opérations.

Pour une bonne **expérience utilisateur** (UX), il faut que **l'interface utilisateur** (UI) se mette à jour rapidement et soit le reflet le plus exact des données.

Un exemple est l'importance de la **mise à jour du formulaire lors de la saisie**. C'est là que l'évènement '**change**' rentre en jeu. A chaque changement détecté au niveau du champ de formulaire, le state va se mettre à jour pour que l'utilisateur ait un feedback en temps réel de ce qu'il saisit.

Pour poursuivre avec l'interaction vue ci-dessus, voici une explication avec le formulaire d'édition d'une broderie.

```
1 // Example of onchange event and user interface update
2
3 // The user changes a field in the form
4 // /src/components/CardEmbroidery/CardEmbroideryEdit.jsx
5 <FieldInput value={embroidery.name} name="name" label="Nom de la broderie" onChange={(e) => onChange(e)} required />
6
7 // It runs the onChange() function that refers to onChange prop which value is {(e) => handleChange(e)}
8 // /src/views/embroideries/EmbroideryEdit.jsx
9 <CardEmbroideryEdit
10   embroidery={embroideryCurrent}
11   onChange={(e) => handleChange(e)}
12 />
13
14 // The handleChange function updates the store with the dispatch method
15 // Method created from useDispatch() of react-redux
16 const handleChange = ({ name, value }) => {
17   dispatch(updateEmbroideryField(name, value));
18 };
19
20 // The updateEmbroideryField() function returns
21 // the action object with the values entered by the user
22 // /src/actions/embroidery.js
23 export const updateEmbroideryField = (name, value) => ({
24   type: UPDATE_EMBROIDERY_FIELD,
25   name,
26   value,
27 });
28
29 // The reducer chooses the right instructions to run according to the action
30 // /src/reducers/embroideriesReducer.js
31 const embroideriesReducer = (state = initialState, action = {}) => {
32   switch (action.type) {
33     //extract of the reducer
34     case UPDATE_EMBROIDERY_FIELD:
35       return {
36         ...state,
37         embroideryCurrent: {
38           ...state.embroideryCurrent,
39           [action.name]: action.value,
40         },
41       };
42     //extract of the reducer
43   }
44 }
```

Dans cet exemple, la fonction **dispatch()**, qui découle de la librairie React-redux, va permettre **d'émettre l'intention de lancer l'action** updateEmbroideryField(), le reducer concerné va **mettre à jour le state global** avec les valeurs saisies par l'utilisateur et donc l'interface.

## C/ Envoyer et recevoir des données

Un autre évènement que nous avons écouté dans notre application est l'évènement **'submit'**. Il est déclenché lors de la **soumission d'un formulaire** qui va déclencher l'envoi d'une requête http. Dans notre application, cela va concerner la création, la modification et la suppression d'enregistrements en base de données via l'API du backend, qui sera présentée tout à l'heure.

Pour **interagir avec l'API du backend**, nous avons utilisé **Axios** qui est une librairie qui permet de faire des **requêtes http de manière asynchrone**, sans nécessiter un rechargement de la page, en utilisant l'objet **XMLHttpRequest** fourni par les navigateurs. Cela nous a été indispensable pour récupérer les données pour l'affichage du site mais aussi pour la mise à jour de celles-ci suite aux modifications envoyées par l'utilisateur.

Pour en revenir à notre exemple de modification d'une broderie, l'interface va changer visuellement lors du remplissage des champs du formulaire, mais il va falloir aussi que les données modifiées soient envoyées à l'API. Pour cela, nous avons dû créer un intermédiaire, le **middleware**, qui va permettre de lancer les requêtes http sans bloquer l'application.

```
1 // Example of middleware implementation for embroidery update
2
3 // /src/middleware/embroideriesMiddleware.js
4
5 const embroideriesMiddleware = (store) => (next) => (action) => {
6   switch (action.type) {
7     //extract of embroideriesMiddleware
8     case PUT_EMBROIDERY:
9       axios
10        .put(`${API}/embroideries/update/${action.data.id}`, action.data)
11        .then(() => {
12          store.dispatch(updateEmbroidery(action.data));
13        })
14        .catch((error) => {
15          console.warn(error);
16        });
17       break;
18     }
19     //extract of embroideriesMiddleware
20     next(action);
21   };
```

Grâce à ce middleware, le JSON modifié sera envoyé à l'API du backend et le state sera mis à jour uniquement si la requête a réussi.





# BACKEND

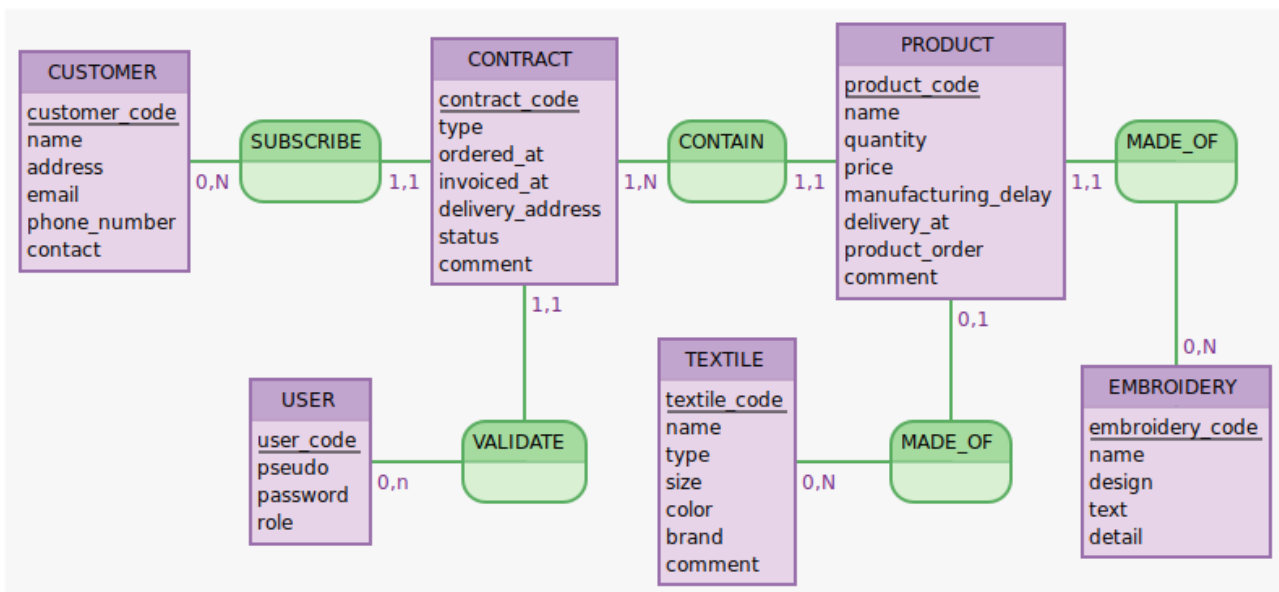
Le backend de l'application consiste en une API REST qui devra fournir des endpoints pour servir les données requises au fonctionnement de l'application.

Nous avons tout d'abord mis en place la base de données puis les composants permettant d'y accéder pour ensuite construire les composants spécifiques répondant à la logique métier tout en respectant les précautions de sécurité indispensables.

## 1/ Mise en place de la base de données

La mise en place de la base de données a été grandement facilitée par la **réflexion préalable lors de l'élaboration des modèles de données** avec le choix des entités, de leurs propriétés et le type de relations qui les lient.

Nous avons distingué les différentes **entités** et déterminé leurs **cardinalités** que nous avons représentées dans un **MCD** (Modèle Conceptuel de Données ), réalisé avec le logiciel en ligne Mocodo.



Nous avons ensuite construit le **MLD** (Modèle Logique de Données), en faisant apparaître les clés étrangères.

Customer ( <u>customer_code</u> , name, address, email, contact, phone_number)
Embroidery ( <u>embroidery_code</u> , name , design, text, detail)
Contract ( <u>contract_code</u> , type, ordered_at, invoiced_at,delivery_address, status, comment, #customer_code, #user_code)
Product ( <u>product_code</u> , name, quantity, price, delivery_at, manufacturing_delay, product_order, comment, #contract_code, #embroidery_code, #textile_code )
Textile ( <u>textile_code</u> , name, type, size, color, brand,comment)

Puis nous avons établi le **dictionnaire de données** (cf annexes cbien) qui va préciser pour chaque table ses **champs** et, pour chacun de ses champs, son **nom**, son **type**, ses **spécificités** et une **description**.



Le **MPD** (Modèle Physique de Données) va nous permettre d'avoir une **représentation visuelle complète de la base de données**, représentant les **tables** et les **relations** qui les lient. Nous avons utilisé PhpMyAdmin pour créer ce MPD.

Cette réflexion préalable sur la structure des données nous a avancé pour la création de la base de données. Nous avons opté pour le SGBDR (Système de gestion de Base de Données Relationnelle) **MySQL**.

Comme nous l'avons introduit dans la partie sur les technologies utilisées dans ce projet, nous utilisons l'**ORM Doctrine** qui fournit une couche d'abstraction entre le PHP et la base de données relationnelle. En effet, Doctrine permet de **transformer les tables de la base de données relationnelle en objets PHP**, ce qui facilite leur manipulation, notamment lors des opérations du CRUD.

De plus, Doctrine dispose de fonctionnalités, que nous allons utiliser plus tard, tels que :

- les **annotations**, pour définir le type de colonne des tables à créer en fonction des attributs des classes
- le **DQL** (Doctrine Querying Language) un langage dédié pour faire des requêtes SQL mais en style orienté objet et le **Query Builder**, une interface qui simplifie la création des requêtes SQL.
- les **migrations**, fichiers en PHP qui contiennent les modifications à appliquer en base de données, sous forme de requêtes SQL.
- les **events** Doctrine, qui vont permettre de lier des instructions à des événements survenant du côté de la base de données.

Pour créer la base de données, nous avons créé un utilisateur via l'interface Adminer.

Puis nous avons rempli le **.env.local** en affectant les valeurs adéquates aux variables d'environnement utilisées, notamment **DATABASE\_URL** qui contiendra les login et mot de passe de l'utilisateur de la base de données, le nom de la base de données et le driver utilisé.

Finalement on a lancé la création de la base de données via la commande du Maker de Symfony ``php bin/console doctrine:database:create`` .

## 2/ Accéder aux données

L'utilisation de Symfony implique l'adoption d'une architecture de projet qui suit le **design pattern MVC** (Modèle-View-Controller). Le projet respecte donc cette répartition du code en **trois couches** : une pour les **données** (le Model), une pour leur **traitement** ( le Controller) et une pour leur **présentation** (la View). Dans le contexte de notre API, la View n'a pas été utilisée puisque l'affichage des données a été mis en place côté frontend.

### AI Création et configuration des entités

Comme nous utilisons l'ORM Doctrine, il est pratique d'utiliser la commande du Maker de Symfony pour créer nos entités avec la commande `'php bin/console make:entity'` . Cette commande déclenche un script qui nous permet de créer un **fichier php avec la classe représentant notre table et de spécifier ses propriétés et leur type, ainsi que les getters et setters** correspondants à ces propriétés. C'est aussi lors de la création des entités qu'on va pouvoir préciser les **relations** éventuelles que l'entité a avec d'autres entités et le **type de ces relations** ( ManyToOne, OneToMany, OneToOne, ManyToMany) selon les cardinalités définies. Un fichier php dans le dossier ./src/Entity a été généré pour chaque entité du MCD.

On pourra compléter le fichier php généré par :

- des **validations de contraintes Symfony** qui vont permettre de respecter certains critères lors de la création des objets Doctrine au sein de l'application. Par exemple, dans notre extrait ci-dessous, l'annotation« `#[Assert\Length(max: 100)]` » va autoriser une chaîne de caractères de 100 caractères maximum.
- des **validations de contraintes Doctrine** qui assurent une cohérence des données enregistrées en base de données et un respect des règles métier. Par exemple, dans notre extrait ci-dessous, la contrainte « `#[ORM\Column(length: 100, nullable: true)]` » va autoriser une chaîne de caractères de 100 caractères maximum ou une valeur nulle.

Ces validations de contraintes ont été mises en place en s'appuyant sur les règles métier exprimées lors de la réalisation des modèles de données pendant le sprint 1.

Voici un extrait du fichier d'entité Textile.php créé pour l'entité « Textile » présentant les propriétés de cette entité.

```

1  #[Groups(['textile'])]
2  #[ORM\Entity(repositoryClass: TextileRepository::class)]
3  #[ORM\HasLifecycleCallbacks]
4
5  class Textile
6  {
7      #[Groups(['contract','textileLinked','textileLinkedId'])]
8      #[ORM\Id]
9      #[ORM\GeneratedValue]
10     #[ORM\Column]
11     private ?int $id = null;
12
13     #[Groups(['contract','textileLinked'])]
14     #[Assert\Length(max: 100)]
15     #[ORM\Column(length: 100, nullable: true)]
16     private ?string $name = null;
17
18     #[Groups(['contract','textileLinked'])]
19     #[Assert\NotBlank]
20     #[Assert\Length(max: 100)]
21     #[ORM\Column(length: 100)]
22     private ?string $type = null;
23
24     #[Groups(['contract','textileLinked'])]
25     #[Assert\Length(max: 100)]
26     #[ORM\Column(length: 100, nullable: true)]
27     private ?string $size = null;
28
29     #[Groups(['contract','textileLinked'])]
30     #[Assert\Length(max: 100)]
31     #[ORM\Column(length: 100, nullable: true)]
32     private ?string $color = null;
33
34     #[Groups(['contract','textileLinked'])]
35     #[Assert\Length(max: 100)]
36     #[ORM\Column(length: 100, nullable: true)]
37     private ?string $brand = null;
38
39     #[Groups(['contract','textileLinked'])]
40     #[ORM\Column(type: Types::TEXT, nullable: true)]
41     private ?string $comment = null;
42
43     #[Groups(['contract','textileLinked'])]
44     #[ORM\Column(type: 'datetime_immutable', options: ['default' => 'CURRENT_TIMESTAMP'])]
45     #[Assert\type(Types::DATE_IMMUTABLE)]
46     private ?\DateTimeImmutable $createdAt = null;
47
48     #[Groups(['contract','textileLinked'])]
49     #[ORM\Column(type: 'datetime_immutable', options: ['default' => 'CURRENT_TIMESTAMP'], nullable: true)]
50     #[Assert\type(Types::DATE_MUTABLE)]
51     private ?\DateTimeImmutable $updatedAt = null;
52
53     #[ORM\OneToMany(targetEntity: Product::class, mappedBy: 'textile')]
54     private Collection $products;

```

C'est aussi dans ces fichiers d'entités que l'on pourra notamment préciser les **groupes de sérialisation** et aussi les **événements Doctrine** liés au « Lifecycle callbacks » ou aux écouteurs d'entités.

```

1  #[ORM\PrePersist]
2  public function setCreatedAt(): static
3  {
4      $this->createdAt = new \DateTimeImmutable();
5
6      return $this;
7  }

```

```

1  #[ORM\PreUpdate]
2  public function setUpdatedAt(): static
3  {
4      $this->updatedAt = new \DateTimeImmutable();
5
6      return $this;
7  }

```

Les **groupes de sérialisation** nous serviront à sélectionner les données à intégrer dans le JSON transmis par la réponse HTTP. Par exemple, l'annotation « `#[Groups(['textile'])]` » va créer un groupe de sérialisation « textile » qu'on va pouvoir utiliser lors de la sérialisation, transformation des objets PHP en JSON.

L'évènement Doctrine « **PrePersist** » a été utilisé pour systématiser l'affectation de la date du jour à la propriété `CreatedAt()` lors de la création d'un enregistrement. L'évènement « **PreUpdate** » a concerné, lui, l'affectation de la date du jour à la propriété `UpdatedAt()` à la mise à jour d'un enregistrement.

Afin d'appliquer ces modifications en base de données, il faut **créer un fichier de migration** qui va contenir les requêtes SQL correspondant aux modifications à appliquer avec la commande ' `php bin/console make:migration` '. Puis on va exécuter la commande ' `php bin/console migration:migrate` ' pour **appliquer ces modifications à la base de données**.

## **B/ Création et configuration des contrôleurs**

Une fois les différentes entités créées, on va pouvoir créer des **contrôleurs** qui vont nous permettre d'accéder aux données. Dans Symfony, les contrôleurs vont faire intervenir les **Repositories** des différentes entités et **l'Entity Manager**, classes propres à Symfony qui vont faciliter la manipulation des données en fournissant des méthodes facilitant les opérations du **CRUD** (Create Read Update Delete).

En effet, les repositories vont fournir, entre autres, les méthodes ***find*** et ***findAll()*** qui vont nous être utiles pour afficher les données d'un enregistrement. L'Entity Manager va nous fournir, entre autres, les méthodes ***persist()*** et ***flush()*** qui vont nous servir à sauvegarder en base de données les changements effectués sur les objets.

La **création des contrôleurs** peut aussi être facilitée par le **Maker** de Symfony grâce à la commande ' `php bin/console make:controller` ' à laquelle nous rajoutons l'option ' `--no-template` ' pour ne pas générer de fichier HTML correspondant automatiquement. Un fichier php dans le dossier `./src/Controller` a été généré pour chaque entité.

Ensuite, pour chaque contrôleur, les **routes correspondant aux méthodes du CRUD** (Create Read Update Delete) ont été codées. Ces routes permettent de fournir les **endpoints** requis par le frontend pour accéder aux données. Elles renvoient une réponse qui contient :

- les données au format **JSON** ( JavaScript Object Notation), format généralement utilisé lors de la communication via des API REST.
- le statut, statut ou **code HTTP** correspondant au **statut de la requête** (succès ou erreur).
- les **headers** éventuels pour la redirection.
- le **contexte**, notamment les groupes de sérialisation.

Nous en verrons des exemples dans la partie suivante.

### 3/ Exemples de composants métier

Certaines spécificités liées aux contraintes métier ont orienté la réalisation de certains composants de l'application. Nous allons vous montrer ci-après comment ont été réalisés deux exemples de composants métier pour être en concordance avec les besoins du client.

#### A/ Envoyer et recevoir les données requises des enregistrements

Une fonctionnalité principale de l'application est de pouvoir effectuer les opérations du CRUD sur des différents répertoires (textiles, broderies, clients, produits, contrats). Il faut donc que l'API envoie les bonnes données dans la réponse HTTP que fournira le endpoint ciblé par le frontend.

Pour cela, deux composants de Symfony nous ont été indispensables : le **Serializer** et le **Validator** de Symfony.

Le Serializer est intervenu lors de la transformation des objets PHP en JSON pour leur envoi vers le frontend de l'application mais aussi lors de l'opération inverse lors de la réception des JSON pour l'enregistrement des modifications en base de données. Il a fallu utiliser les **groupes de sérialisation** pour sélectionner les propriétés des entités qu'on voulait convertir en JSON / objets et éviter les **relations circulaires**, évènement survenant lorsqu'on convertit un objet au sein duquel existe une propriété qui fait elle-même appel à l'objet lui-même et induit en quelque sorte une boucle infinie.

Pour l'affichage des textiles, le client ne souhaite pas avoir les informations des produits qui y sont liés donc nous n'avons affiché que les propriétés de l'entité Textile, propriétés regroupées dans le groupe de sérialisation ['textileLinked'] :

```
1 class TextileController extends AbstractController
2 {
3     #[Route('api/textiles', name: 'app_api_textiles', methods: ['GET'])]
4     public function index(TextileRepository $textileRepository): JsonResponse
5     {
6         $data = $textileRepository->findAll();
7         return $this->json($data, 200, [], ["groups"=>['textileLinked'] ]);
8     }
}
```

Par contre pour l'affichage des contrats, il était demandé de pouvoir voir les informations des produits et des textiles et broderies constituant ses produits. Nous avons donc dû créer plusieurs groupes de sérialisation :



```

1  #[Route('api/contracts', name: 'app_api_contracts', methods: ['GET'])]
2  public function index(ContractRepository $contractRepository): JsonResponse
3  {
4      $contracts = $contractRepository->findAll();
5      // Convert the objects' array into a JSON Response
6      return $this->json(
7          $contracts,
8          200,
9          [],
10         ["groups" => ['contract', 'userLinked', 'productLinked', 'customerLinked',
11             'contractTextile', 'contractEmbroidery']]
12     );
13 }

```

- un groupe 'contract' qui comprenait les propriétés propres à l'entité Contrat, sans les relations aux autres entités
- un groupe 'userLinked' qui comprenait les propriétés propres à l'entité User
- un groupe 'productLinked' qui comprenait les propriétés propres à l'entité Product
- un groupe 'customerLinked' qui comprenait les propriétés propres à l'entité Customer
- un groupe 'contractTextile' qui comprenait les propriétés propres à l'entité Textile
- un groupe 'contractEmbroidery' qui comprenait les propriétés propres à l'entité Embroidery


Le Validator de Symfony a été utilisé afin vérifier la concordance des données avec les contraintes mises en place selon les exigences métier. Il permet aussi de transmettre des messages d'erreur en cas de non-respect des contraintes de validation, dans la propriété 'message', au sein des annotations.

## **B/ Accéder aux différents types de documents**

Dans ce projet, les documents concernent **l'entité Contract qui est déclinée pour trois types** : « quotation » (devis) , « order » (commande), »invoice » (facture). En effet, **c'est le même document que le client veut pouvoir passer d'un type à l'autre selon l'avancée de la transaction.**

Nous avons donc, au début de la conception du projet, choisi un type SQL Enum pour la propriété Type mais, celle-ci n'étant pas prise en charge par défaut par l'ORM Doctrine, nous avons opté pour une solution plus simple qui a été de contraindre les entrées possibles sur un champ de type string en utilisant une **contrainte de validation de Symfony**.





```

1  #[Assert\ExpressionSyntax(
2      allowedVariables: ['quotation', 'order', 'invoice'],
3      message : 'You should provide a valid type of contract ! '
4  )]
5  private ?string $type = 'quotation';

```


Ensuite, pour pouvoir accéder aux devis, commandes ou factures, nous avons créé une **route paramétrique** qui a pour paramètre le type de contrat dans le ContractController.

Dans Symfony, les routes vont faire intervenir les **Repositories** des différentes **entités** et l'**Entity Manager**, classes propres à Symfony qui vont faciliter la manipulation des données en fournissant des méthodes facilitant les opérations du **CRUD** (Create Read Update Delete).

En effet, les repositories vont fournir, entre autres, les méthodes **find** et **findAll()** qui vont nous être utiles pour afficher les données d'un enregistrement. L'Entity Manager va nous fournir, entre autres, les méthodes **persist()** et **flush()** qui vont nous servir à sauvegarder en base de données les changements effectués sur les objets. *Mais pour des cas plus spécifiques, il faut créer des méthodes supplémentaires dans le repository de l'entité concernée.*

Dans notre exemple, il a fallu que nous réalisons une méthode **findByType()** dans le ContractRepository avec une requête customisée pour filtrer les contrats par type. On peut remarquer l'utilisation d'une requête paramétrée ce qui prévient des injections SQL, en plus de la contrainte sur le champ type.

La route dans le ContractController fera appel à cette méthode **findByType()** du ContractRepository pour fournir les JSON des contrats du type demandé en paramètre de l'URL.



```

1  public function findByType($value): array
2  {
3      return $this->createQueryBuilder('c')
4          ->andWhere('c.type = :val')
5          ->setParameter('val', $value)
6          ->orderBy('c.id', 'ASC')
7          ->setMaxResults(10)
8          ->getQuery()
9          ->getResult()
10
11  };

```



```
1  #[Route('api/contracts/type/{type}', name: 'app_api_contracts_type', methods: ['GET'],
2  requirements: ['type' => 'order|quotation|invoice'])]
3  public function findByType(ContractRepository $contractRepository,$type): JsonResponse
4  {
5      $data = $contractRepository->findByType($type);
6      // Convert the objects' array into a JSON Response
7      return $this->json(
8          $data,
9          200,
10         [],
11         ["groups" => ['contract','userLinked','productLinked','customerLinked',
12         'contractTextile','contractEmbroidery']]
13     );
14 }
```

# SECURITE

Dans cette partie, nous allons présenter certains éléments de sécurité utilisés dans l'application.

Puis nous ferons un rappel des précautions inhérentes à l'utilisation de React.js et Symfony.

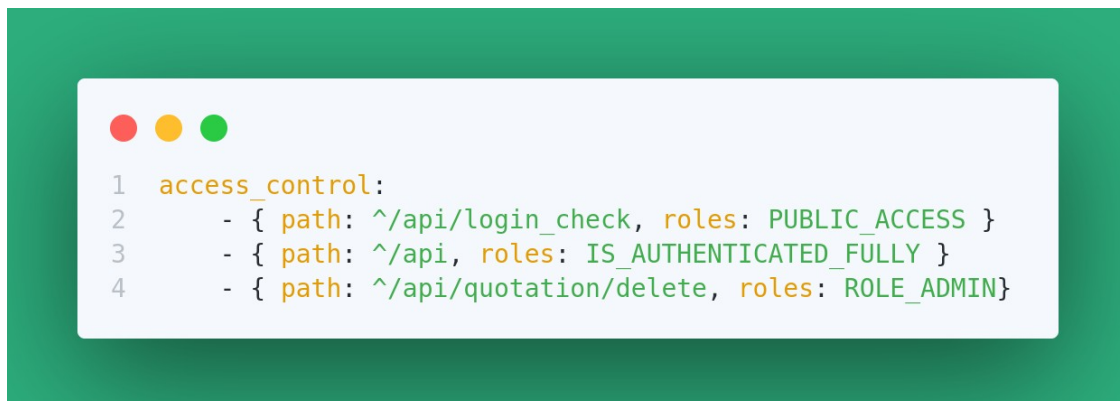
## 1/ Eléments de sécurité

### A/ Authentification et autorisations

L'authentification a été mise en place côté backend grâce au **composant SecurityBundle**. Il a permis la création d'une entité particulière User qui implémente les interfaces `UserInterface` et `PasswordAuthenticatedUserInterface`, interfaces qui vont permettre d'avoir accès aux fonctionnalités de **gestion des rôles** et de **hashage** de mot de passe de Symfony.

Ce bundle va aussi rentrer en jeu dans la création du fichier `/config/packages/security.yaml`, fichier qui va notamment être impliqué dans les autorisations d'accès.

Les **autorisations** d'accès aux différents endpoints se font grâce à la définition de **l'ACL** (Access Control List) dans le fichier `security.yaml`. Pour répondre à la user story « En tant que ADMIN, je veux pouvoir supprimer les factures afin de gérer un cas exceptionnel de suppression de facture », fonctionnalité à laquelle l'utilisateur classique n'a pas accès, on doit restreindre l'accès à la route concernée.



L'authentification et les autorisations ont utilisé la technologie du **JWT** (JSON Web Token) qui permet de transporter les informations de manière sécurisée grâce à un encodage et de manière **stateless**, ce qui est essentiel dans le cadre de notre API.

### B/ CORS

Afin de limiter l'accès de l'API au domaine autorisé, l'adresse du frontend a été spécifiée dans le fichier `/config/packages/nelmio_cors.yaml`.

```

1  nelmio_cors:
2      defaults:
3          origin_regex: true
4          allow_origin: ['%env(CORS_ALLOW_ORIGIN)%']
5          allow_methods: ['GET', 'OPTIONS', 'POST', 'PUT', 'PATCH', 'DELETE']
6          allow_headers: ['Content-Type', 'Authorization']
7          expose_headers: ['Link']
8          max_age: 3600
9      paths:
10         '^/':
11             allow_origin: ['https://obroderie.lorlor.site/']

```

## C/ Eléments de sécurité par défaut dans React et Symfony

Sources :

<https://owasp.org/www-project-bullet-proof-react/>

[https://medium.com/@smitha\\_ml/owasp-principles-for-secure-react-applications-f7f75e851d31](https://medium.com/@smitha_ml/owasp-principles-for-secure-react-applications-f7f75e851d31)

[https://cheatsheetseries.owasp.org/cheatsheets/Symfony\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Symfony_Cheat_Sheet.html)

Il est nécessaire de s'informer quant aux protections que l'utilisation de bibliothèques ou frameworks peut induire, ces éléments étant parfois occultés par l'« enveloppe » du framework. Nous nous sommes intéressés aux mesures de sécurité liées aux attaques les plus courantes: injections XSS(Cross-Site-Scripting), injections SQL et CSRF

**L'utilisation de React va prémunir des injections XSS et SQL** de par l'utilisation du JSX qui va permettre d'échapper automatiquement les entrées utilisateurs et les valeurs dynamiques. De plus, l'utilisation du DOM Virtuel empêche l'insertion malveillante de nouveaux éléments HTML. On évitera d'utiliser la propriété *dangerouslySetInnerHTML()*. **Au niveau des attaques CSRF, il n'y a pas de protection par défaut mais la configuration des CORS côté serveur en prémunit.**

Côté **Symfony**, la validation des données permet une protection supplémentaire et l'utilisation de Doctrine permet l'utilisation de **requêtes paramétrées** avec le Doctrine Query Language et le QueryBuilder de Doctrine.

```

1  public function findContractsByCustomerName($customerName): array
2      {
3          return $this->createQueryBuilder('contract')
4              ->join('contract.customer', 'customer')
5              ->andWhere('customer.name LIKE :customerName')
6              ->setParameter('customerName', '%' . $customerName . '%')
7              ->orderBy('contract.id', 'ASC')
8              ->setMaxResults(10)
9              ->getQuery()
10             ->getResult();
11     }

```

# JEU D'ESSAI

Dans ce jeu d'essai, nous allons **tester manuellement la création d'un client au niveau de l'API**.

Nous allons donc passer par le **endpoint `api/customers/create`** et vérifier que les différentes étapes de la création du client se passent comme prévu. Pour cela nous allons utiliser le **client HTTP PostMan** via son extension intégrée à VSCode.

Dans le corps de la requête HTTP (**body**), nous allons envoyer en POST le JSON de test et observer la réponse HTTP renvoyée en retour (cf. ci-contre).



```
1 {
2   "name": "Pierre Chambois",
3   "address": "321 rue des thermes 75012 PARIS",
4   "email": "pierre@free.fr",
5   "contact": "M. Chambois",
6   "phone_number": "0678956723"
7 }
```

## 1/ Réponse et code HTTP corrects et enregistrements en base de données

La réponse renvoyée doit correspondre au **JSON du nouvel enregistrement** en base de données. Il doit avoir en plus des propriétés du JSON envoyé des **propriétés remplies automatiquement** lors de la persistance de l'objet en base de données :

- - **l'id**, qui doit être auto-incrémenté
- - **la date de création**, qui correspond au datetime de la date du jour.
- - **la date de mise à jour**, qui doit être nulle puisque c'est une création d'un enregistrement.

Le code de la réponse HTTP est **201** ce qui correspond au status « created » ce qui est correct.

## 2/ Affichage cohérent des JSON d'erreurs et codes http dans la réponse HTTP

Nous allons maintenant vérifier que la route renvoie bien un JSON avec un message d'erreur si le JSON envoyé mal renseigné.

Si on reprend le dictionnaire de données rédigé lors du sprint 1, on peut constater qu'il faut que les champs « name », « address » et « email » ne soient pas nuls. En cas de champs non renseignés, on reçoit bien un message d'erreur. Si le champ est rempli de caractères vides aussi.

Mettre à jour toutes les propriétés de customer au cas o ù

name	VARCHAR(100)	NOT NULL	Le nom de l'entreprise ou du particulier
address	VARCHAR(500)	NOT NULL	L'adresse de la personne
email	VARCHAR(100)	NOT NULL	L'email de la personne
contact	VARCHAR(100)	NULL	Le nom de la personne à contacter

phone_number	VARCHAR (50)	NULL	Le numéro de téléphone de la personne sous forme de chiffres
--------------	--------------	------	--

**Parler d fait que la contrainte notblank ne prend pas en compte les string composés d'espaces et qu'i la fallut rajouter une contrainte regex**

# VEILLE

# CONCLUSION

parler des difficultés rencontrée et des améliorations

oagation

erginomie

amélioration sécurité



4. Back-end

5. Sécurité

6. Jeu d'essai

7. Veille

Conclusion

8. Annexes

Realisations les plus significatives

# ANNEXES

faire un sommaire des annexes

# User Stories complètes

## **Authentication**

- En tant que USER je veux m'authentifier afin d'accéder aux fonctionnalités qui me sont réservées

## **Répertoire client**

- En tant que USER je veux consulter le répertoire client afin d'obtenir des informations sur les clients.
- En tant que USER je veux créer une fiche client afin d'ajouter un client dans le répertoire client.
- En tant que USER je veux modifier une fiche client afin de mettre à jour des informations liées au client.
- En tant que USER je veux supprimer une fiche client afin de nettoyer le répertoire client

## **Répertoire textile**

- En tant que USER je veux consulter le répertoire textile afin d'obtenir des informations.
- En tant que USER je veux créer une fiche textile afin d'ajouter un textile dans le répertoire textile.
- En tant que USER je veux modifier une fiche textile afin de mettre à jour des informations liées au textile.
- En tant que USER je veux supprimer une fiche textile afin de nettoyer le répertoire textile

## **Répertoire broderie**

- En tant que USER je veux consulter le répertoire broderie afin d'obtenir des informations.
- En tant que USER je veux créer une fiche broderie afin d'ajouter une broderie dans le répertoire broderie.
- En tant que USER je veux modifier une fiche broderie afin de mettre à jour des informations liées au broderie.
- En tant que USER je veux supprimer une fiche broderie afin de nettoyer le répertoire broderie

## **Devis**

- En tant que USER je veux consulter les devis afin d'obtenir des informations.
- En tant que USER je veux créer un devis afin d'enregistrer une nouvelle demande client

- En tant que USER je veux modifier un devis afin de mettre à jour une demande client
- En tant que USER je veux supprimer un devis afin d'annuler une demande client
- En tant que USER je veux lier un client au devis afin d'assigner le client à la demande
- En tant que USER je veux créer un produit dans le devis afin de renseigner le contenu de la demande client.
- En tant que USER je veux modifier un produit du devis afin de mettre à jour le contenu de la demande client.
- En tant que USER je veux supprimer un produit du devis afin d'annuler le contenu de la demande client.
- En tant que USER je veux consulter le montant total du devis afin d'apprécier la pertinence de mon chiffrage
- En tant que USER je veux pouvoir rechercher un client afin de lier facilement le client au devis
- En tant que USER je veux pouvoir rechercher un textile afin de lier facilement le textile au devis
- En tant que USER je veux pouvoir rechercher une broderie afin de lier facilement la broderie au devis

### **Commande**

- En tant que USER je veux valider le devis afin de le passer en commande.
- En tant que USER je veux consulter les commandes afin d'obtenir des informations sur les demandes clients en cours de fabrication
- En tant que USER je veux renseigner les dates de livraisons de la commande afin de définir les délais contractuels.
- En tant que USER je veux rétrograder la commande au stade de devis afin de modifier la demande client en cours de fabrication

### **Facture**

- En tant que USER je veux valider la commande afin de la passer en facture.
- En tant que USER je veux consulter les factures afin d'obtenir des informations sur les dernières factures émises
- En tant que USER je veux archiver une facture afin de clôturer la demande du client après paiement

### **Rôle ADMIN**

- En tant que ADMIN je veux avoir les mêmes droits que le USER
- En tant que ADMIN, je veux pouvoir rétrograder une facture au stade de commande afin de permettre à USER de modifier la facture

- En tant que ADMIN, je veux pouvoir supprimer les factures afin de gérer un cas exceptionnel de suppression de facture
- En tant que ADMIN, je veux pouvoir consulter des utilisateurs afin d'obtenir des information sur tous les utilisateurs
- En tant que ADMIN, je veux pouvoir créer des utilisateurs afin d'intégrer une nouvelle personne à l'équipe commerciale
- En tant que ADMIN, je veux pouvoir modifier des utilisateurs afin de mettre à jour les informations des utilisateurs
- En tant que ADMIN, je veux pouvoir supprimer des utilisateurs afin de nettoyer le répertoire des utilisateurs en cas de départ d'un employé

# Dictionnaire de données

**Table Customer**

Champ	Type	Spécificités	Description
customer_code	INT	PRIMARY KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	L'identifiant de la personne
name	VARCHAR(100)	NOT NULL	Le nom de l'entreprise ou du particulier
address	VARCHAR(500)	NOT NULL	L'adresse de la personne
email	VARCHAR(100)	NOT NULL	L'email de la personne
contact	VARCHAR(100)	NULL	Le nom de la personne à contacter
phone_number	VARCHAR (50)	NULL	Le numéro de téléphone de la personne sous forme de chiffres
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Date de création du client
updated_at	TIMESTAMP	NULL, DEFAULT CURRENT_TIMESTAMP	Date de modification du client

**Table Embroidery**

Champ	Type	Spécificités	Description
embroidery_code	INT	PRIMARY KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	L'identifiant de la broderie
name	VARCHAR(100)	NULL	Nom personnalisé de la broderie
design	VARCHAR(100)	NOT NULL	Intitulé de la broderie
text	VARCHAR(100)	NULL	Texte personnalisé
detail	TEXT()	NULL	Options éventuelles
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAM P	Date de création de la broderie
updated_at	TIMESTAMP	NULL, DEFAULT CURRENT_TIMESTAM P	Date de modification de la broderie

### **Table Textile**

Champ	Type	Spécificités	Description
textile_code	INT	PRIMARY KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	L'identifiant du textile
name	VARCHAR(100)	NULL	Nom customisé du textile
type	VARCHAR(100)	NOT NULL	Le type de support (casquette, tee-shirt...)
size	VARCHAR(100)	NULL	La taille éventuelle du textile
color	VARCHAR(100)	NULL	La couleur éventuelle du textile
brand	VARCHAR(100)	NULL	La marque éventuelle du textile
comment	TEXT()	NULL	Commentaires sur le textile
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Date de création du textile
updated_at	TIMESTAMP	NULL, DEFAULT CURRENT_TIMESTAMP	Date de modification du textile

### **Table Contract**

Champ	Type	Spécificités	Description
contract_code	INT	PRIMARY KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	L'identifiant du contrat
type	ENUM (quotation, order,invoice)	DEFAULT quotation	Enum : (quotation, order,invoice)
ordered_at	DATE	NULL	Date du passage en commande
invoiced_at	DATE	NULL	Date de facturation
delivery_address	VARCHAR(500)	NOT NULL	Adresse de livraison
user_code	INT	FOREIGN KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	Personne qui suit le contrat
status	ENUM (created/ archived/obsolete/ deleted)	DEFAULT created	ENUM (created/archived/obsolete/deleted)
comment	TEXT()	NULL	Commentaires éventuels
customer_code	INT	FOREIGN KEY, NOT NULL	Client qui a souscrit le contrat
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Date de création du contrat
updated_at	TIMESTAMP	NULL, DEFAULT CURRENT_TIMESTAMP	Date de modification du contrat

**Table Product**

Champ	Type	Spécificités	Description
product_code	INT	PRIMARY KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	L'identifiant du produit
name	VARCHAR(100)	NOT NULL	Nom personnalisé du produit, s'affichera dans le devis
quantity	INT	NOT NULL	Nombre de produits
price	DECIMAL	NOT NULL	Prix du produit, limiter à deux chiffres après la virgule
delivery_at	DATE	NULL	Date de livraison du produit
manufacturing_delay	INT	NOT NULL	Délai de fabrication du produit, en jours
product_order	INT	NOT NULL, UNIQUE	Rang du produit dans le contrat
comment	TEXT()	NULL	Commentaires éventuels
contract_code	INT	FOREIGN KEY, UNSIGNED, NULL, AUTO_INCREMENT	L'identifiant du contrat
embroidery_code	INT	FOREIGN KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	L'identifiant de la broderie
textile_code	INT	FOREIGN KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	L'identifiant du textile
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP	Date de création du produit
updated_at	TIMESTAMP	NULL, DEFAULT CURRENT_TIMESTAMP	Date de modification du produit



## **Table User**

Champ	Type	Spécificités	Description
user_code	INT	PRIMARY KEY, UNSIGNED, NOT NULL, AUTO_INCREMENT	L'identifiant de l'utilisateur
pseudo	VARCHAR(100)	NOT NULL, UNIQUE	Le pseudo de l'utilisateur
password	VARCHAR(255)	NOT NULL	Le mot de passe de l'utilisateur
role	ENUM (admin, user)	NOT NULL, DEFAULT user	Le rôle de l'utilisateur,ENUM (admin,user)
created_at	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAM P	Date de création de l'utilisateur
updated_at	TIMESTAMP	NULL, DEFAULT CURRENT_TIMESTAM P	Date de modification de l'utilisateur