Carte blanche partenaires

Note de Synthèse de 2 ème Années

INDEX

I.Introduction

Dans le cadre de ma Deuxième année de BTS Services Informatiques aux Organisations (SIO), formation proposée par le Lycée Geoffroy Saint-Hilaire (Essonne – 91), chaque étudiant est tenu d'effectuer un stage d'une durée minimale de six semaines en milieu professionnel.

Ce stage constitue une étape essentielle dans notre parcours, permettant à la fois l'acquisition de compétences techniques et le développement de savoir-être en entreprise. Il s'agit également d'une opportunité concrète de mettre en application les connaissances théoriques abordées durant l'année scolaire.

Présentation de l'entreprise
J'ai réalisé mon stage au sein de l'entreprise Carte Blanche Partenaire, où j'ai été intégré en tant que développeur. " a remplir et dire ce qu'ils font .

Objectif de la note de synthèse
Ce document a pour objectif de présenter les différentes missions qui m'ont été confiées tout au long de mon stage, ainsi que les compétences mobilisées et développées dans ce cadre.

Organisation du document

1. Présentation détaillée de l'entreprise
2. Description des missions réalisées
3. Bilan global de l'expérience professionnelle

II. Présentation de L'entreprise

1.Carte Blanche Partenaires

Carte Blanche Partenaires est une entreprise française créée en 2001, spécialisée dans les services de santé et la gestion de réseaux de soins. Elle a été fondée par des acteurs du secteur de l'assurance et des complémentaires santé afin de répondre aux difficultés croissantes d'accès aux soins et à la hausse des dépenses de santé. L'entreprise s'est ainsi positionnée comme un intermédiaire entre les organismes complémentaires santé, les professionnels de santé et les assurés, avec pour objectif de rendre le système de soins plus accessible, plus lisible et plus économique.

L'activité principale de Carte Blanche Partenaires consiste à créer, organiser et piloter des réseaux de professionnels de santé partenaires, notamment dans les domaines de l'optique, du dentaire et de l'audiologie. Ces professionnels sont sélectionnés selon des critères de qualité, de conformité réglementaire et de pratiques tarifaires maîtrisées. En contrepartie de leur adhésion au réseau, ils s'engagent à proposer des soins et équipements à des tarifs négociés, permettant ainsi de réduire le reste à charge pour les assurés. L'entreprise met également en place des services comme le tiers payant et des outils d'accompagnement pour faciliter le parcours de soins des bénéficiaires.

Le but principal de Carte Blanche Partenaires est de favoriser l'accès à des soins de qualité pour le plus grand nombre tout en contribuant à la maîtrise des coûts de santé. En proposant des services à forte valeur ajoutée aux complémentaires santé, elle leur permet d'améliorer leurs offres et de renforcer la satisfaction de leurs adhérents. Par son rôle d'intermédiaire et son expertise dans les réseaux de soins, Carte Blanche Partenaires participe à l'amélioration du système de santé en conciliant qualité des soins, accessibilité financière et efficacité économique.

II. Présentation de L'entreprise

2.Organisme

II. Présentation de L'entreprise

3.Environnement de travail

Mon stage s'est déroulé à Paris, au sein de l'entreprise Carte Blanche Partenaires, où j'ai été intégré au projet Récup-Achat. Je suis arrivé en plein milieu du projet et j'ai été accompagné par Tristan BOI et Asmaa NABIL. Les détails du projet seront expliqués plus tard.

Le langage de programmation principalement utilisé est **Angular**, un framework basé sur **TypeScript**, lui-même une surcouche de **JavaScript**. J'ai également utilisé du **HTML** et du **CSS**.

Afin d'assurer de bonnes conditions de travail, l'entreprise m'a mis à disposition un ordinateur portable et un environnement technique adapté à ses outils internes. Les locaux comprennent plusieurs open spaces, un espace pour manger avec les collègues, et l'accès au bâtiment se fait par badge fourni par l'entreprise.

L'ambiance de travail est agréable et professionnelle, et l'ensemble du personnel s'est montré accueillant et bienveillant tout au long de mon stage.

III. Gestion de Projet

4. Thème et Objectif

Le thème du projet est de développer des IHM (Interfaces Homme-Machine). Pour cela, nous disposons de maquettes réalisées sur Figma, qui servent de base au développement. Au total, le projet comprend quatre IHM : une IHM sous forme de formulaire et trois autres correspondant aux différents cas.

Le formulaire permet de saisir le numéro AMC, la clé AMC ainsi que le numéro PEC. Ce dernier est essentiel, car il permet de rediriger vers l'un des trois cas.

Avant de commencer le développement, il m'a été précisé que le projet se réaliserait en Angular, et donc en TypeScript, un framework que je n'avais jamais utilisé auparavant. Lors du premier jour, le matériel nécessaire m'a été fourni (ordinateur portable, souris, clavier et casque). Cette journée a été consacrée à la configuration de mon poste de travail, puis à une présentation globale du projet et de ce à quoi je devais m'attendre.

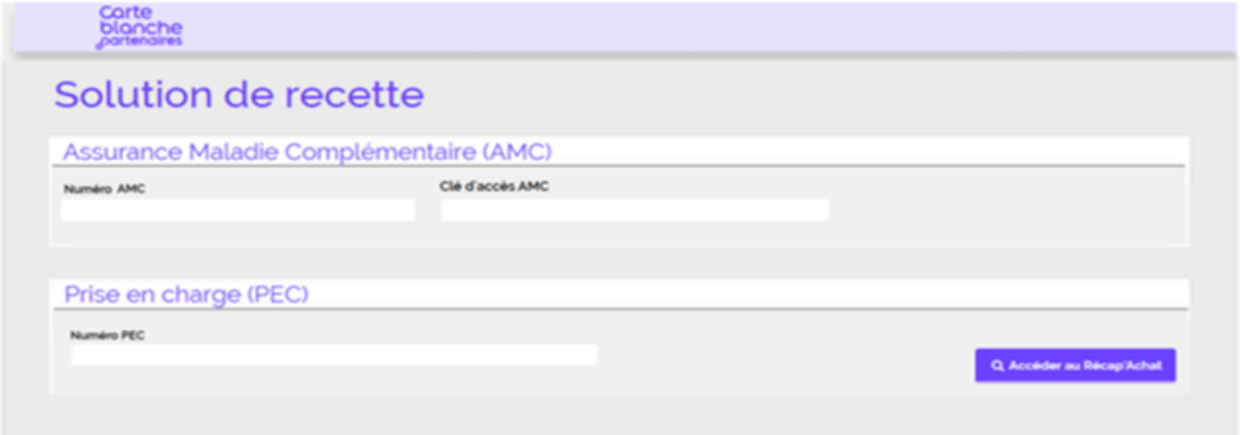
Le projet consiste, dans un premier temps, à développer la première IHM sous forme de formulaire, qui doit être identique à la maquette Figma. Ensuite, il faut mettre en place un système permettant d'analyser le numéro PEC saisi et de rediriger l'utilisateur vers le cas correspondant. Au total, trois cas sont prévus :

- **Cas 1** : remise
- **Cas 2** : remise + réduction
- **Cas 3** : aucune remise

L'objectif du projet est donc, à l'aide du numéro PEC, de déterminer automatiquement à quel cas celui-ci correspond. Cependant, pour y parvenir, il est nécessaire de mocker l'API. Le mocking consiste à simuler une API en attendant que l'entreprise chargée de la fournir nous la transmette. Cela permet de prendre de l'avance sur le développement, de démontrer les compétences en développement de l'équipe, et d'être prêt à intégrer rapidement l'API réelle dès sa réception.

III. Gestion de Projet

5. Détails du projet




```
<header class="bandeau">
  <div class="logo">
<img src="/img/logo.png" alt="Logo"/>
</div>
</header>

<main class="contenu-principal">
  <h1>Solution de recette</h1>


  <section class="bloc">
    <h2>Assurance Maladie Complémentaire (AMC)</h2>

    <div class="row">
      <div class="champ">
        <label>Numéro AMC</label>
        <input type="text" [(ngModel)]="numeroAMC"/>
      </div>

      <div class="champ">
        <label>Clé d'accès AMC</label>
        <input type="text" [(ngModel)]="cleAMC"/>
      </div>
    </div>
  </section>


  <section class="bloc">
    <h2>Prise en charge (PEC)</h2>

    <div class="row align-end">
      <div class="champ large">
        <label>Numéro PEC</label>
        <input type="text" [(ngModel)]="numeroPEC"/>
      </div>

 <button class="bouton" (click)="onAcceder()">Accéder au Récap Achat</button>
    </div>
  </section>


<div *ngIf="pec" class="recap-container">
  <h1>Récap Achat - Test des cas</h1>

  <!-- Affichage des valeurs reçues (pour debug) -->
  <p>Type de remise : <strong>{{ pec.typeRemise }}</strong></p>
  <p>Mention modif : {{ pec.mentionModifMontantHorsReseau }}</p>
```

Voici la maquette à reproduire. Cette partie du projet repose principalement sur du HTML et du CSS. Le bouton « Accéder au Récup-Achat » doit effectuer un appel à l'API afin d'identifier le cas correspondant et rediriger l'utilisateur vers l'IHM associée. C'est pour cette raison que l'utilisation d'une API est nécessaire. Cependant, celle-ci n'étant pas encore disponible, nous devons la mocker.

Ce code correspond au fichier **app.component.html** d'une application Angular. Il définit l'interface principale du formulaire affiché à l'utilisateur.

Tout d'abord, un conteneur principal est affiché grâce à la directive *ngIf="showFrom". Cela permet de contrôler l'affichage du formulaire : tant que la variable showFrom est vraie, le formulaire est visible.

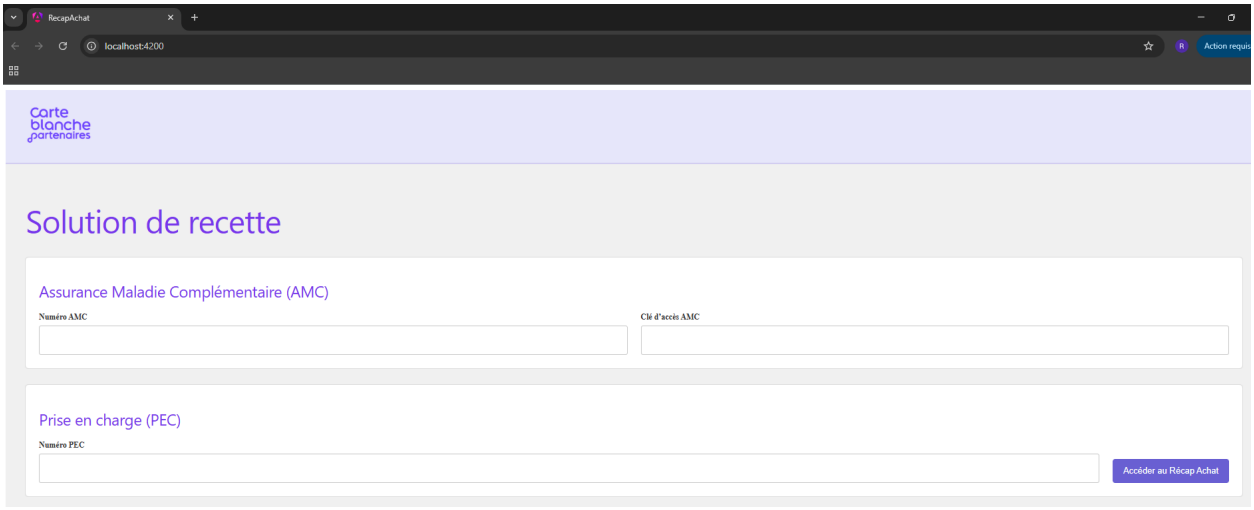
Le **header** contient un bandeau avec le logo de l'application, affiché à l'aide d'une balise <img>.

La partie principale (<main>) contient le contenu du formulaire. Un titre « Solution de recette » est affiché, suivi de deux sections distinctes :

- **Assurance Maladie Complémentaire (AMC)**
  Cette section contient deux champs de saisie :
    - le numéro AMC
    - la clé d'accès AMC
- Chaque champ est lié à une variable TypeScript grâce à la directive [(ngModel)]. Cela permet de récupérer automatiquement les valeurs saisies par l'utilisateur.
  En cas d'erreur, un message est affiché conditionnellement à l'aide de *ngIf, en utilisant l'objet errors.
- **Prise en charge (PEC)**
  Cette section contient un champ permettant de saisir le numéro PEC. Ce champ est limité à 16 chiffres grâce aux attributs maxlength et pattern. Un message d'erreur s'affiche également si la saisie est incorrecte.

Un bouton « Accéder au Récap Achat » est présent en bas du formulaire. Lorsqu'il est cliqué, il déclenche la fonction onAcceder() définie dans le fichier TypeScript. Cette fonction permet de traiter les données saisies et de lancer la logique de redirection vers le cas correspondant.

Enfin, la balise <router-outlet> permet d'afficher les autres pages de l'application. Lorsque le formulaire n'est plus visible (showFrom = false), ce sont les pages correspondant aux différents cas qui s'affichent à cet emplacement



Voici donc le résultat du code et son rendu final. Il existe quelques légères différences entre la maquette et la version que j'ai réalisée, mais l'ensemble est fonctionnel.

Une fois l'affichage de l'IHM du formulaire terminé, nous nous sommes attaqués à la mise en place du mock. J'avais déjà un exemple qui m'avait été fourni par un collègue. À partir de cet exemple, je devais faire en sorte que, lorsque les champs sont remplis et plus particulièrement le champ **NumPEC** et que l'utilisateur clique sur le bouton **« Accéder au Récap'Achat »**, le code reconnaisse le numéro PEC et redirige vers la page correspondant au bon cas.

```
// simule l'appel d'une API
getRecapAchat(numPEC: string): Observable<any> {

  return of({
    pec: {
      typeRemise: 'avecReduction',
      mentionModifMontantHorsReseau: true,

      montantTotalRemise: 25.00,
      pourcentageRemise: 10,
      montantHorsReseau: 500.00,
      montantReseau: 225.00,
      montantTotalRAC: 50.00,
      beneficiaire: {
        prenom: 'Jean',
        nom: 'Dupont',
        adresse: "12 rue des Lilias 37000 Tours",
      },
    }
  })
}
```

Ce code correspond à une fonction TypeScript utilisée pour simuler l'appel à une API (mock).

La fonction getRecapAchat prend en paramètre un numéro PEC (numPEC) de type string. En situation réelle, ce numéro serait envoyé à une API afin de récupérer les informations associées à la prise en charge. Ici, l'API n'étant pas encore disponible, les données sont simulées.

La fonction retourne un Observable, grâce à of(). Cela permet de reproduire le comportement d'un

véritable appel HTTP dans Angular, sans avoir besoin d'un serveur distant.

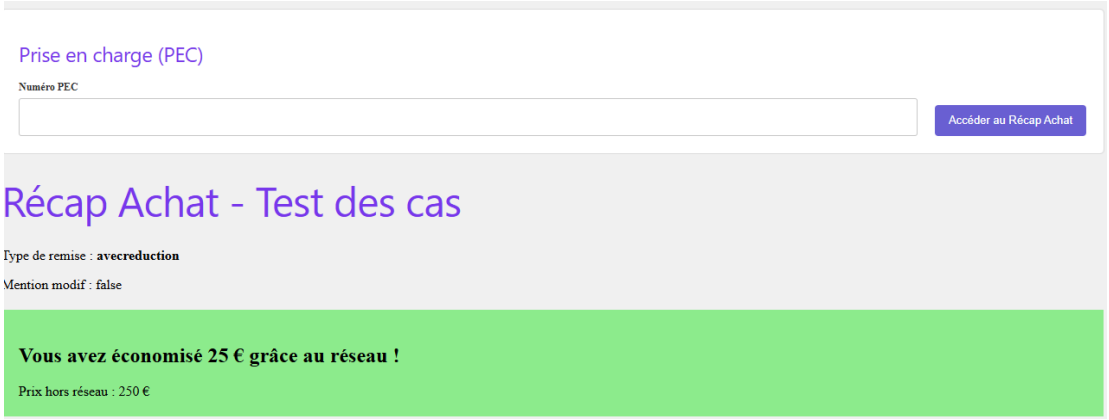
Les données retournées représentent un récapitulatif de prise en charge (PEC). Elles contiennent :

- le type de remise (typeRemise), ici avec réduction,
- des informations sur les montants (remise, pourcentage, reste à charge, montants réseau et hors réseau),
- des données concernant le bénéficiaire, telles que le prénom, le nom et l'adresse.

Ce mock permet donc de tester le fonctionnement de l'application, notamment la récupération des données et l'affichage du récapitulatif, en attendant la mise à disposition de l'API réelle. Une fois l'API fournie, cette fonction pourra être remplacée par un véritable appel HTTP sans modifier la logique globale de l'application.

Dans cette version du mock, le numéro PEC passé en paramètre n'est pas réellement utilisé. Cela signifie que, quel que soit le numéro PEC saisi, la fonction retourne toujours les mêmes données. Autrement dit, n'importe quel numéro PEC fonctionne et redirige vers le même cas.

Ce choix est volontaire à ce stade du projet, car l'objectif principal est de tester le fonctionnement global de l'application (appel de l'API, récupération des données et affichage des pages) sans dépendre de l'API réelle. Une fois l'API officielle disponible, la logique pourra être adaptée afin de traiter des numéros PEC différents et de retourner des résultats spécifiques en fonction de ceux-ci.

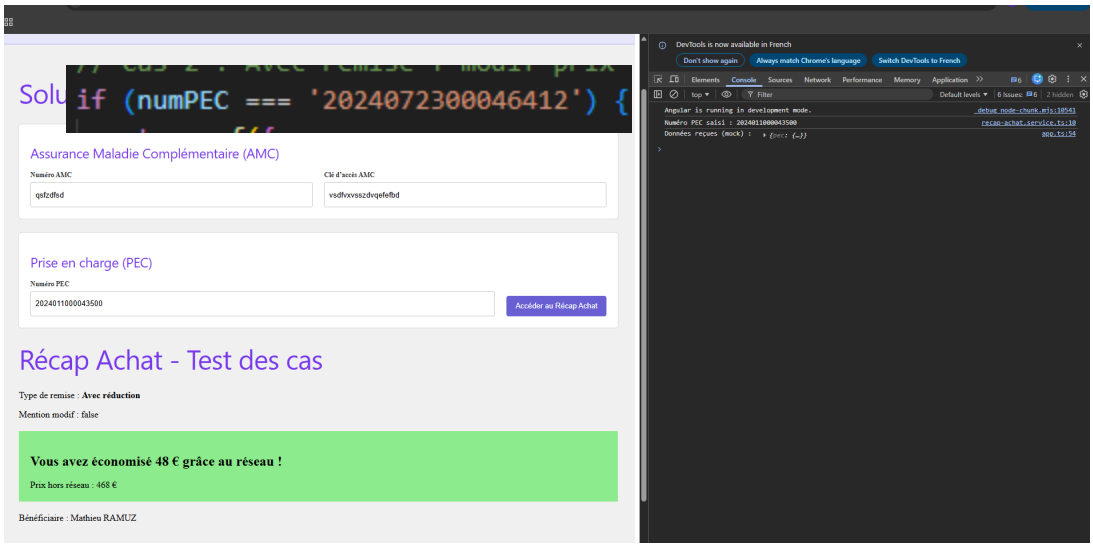


Voici le résultat qui s'affiche lorsque l'on clique sur « Accéder au Récap'Achat ». On peut constater que le code fonctionne, car il renvoie bien un cas. Cependant, il n'est pas encore totalement fonctionnel. Il est donc nécessaire de peaufiner le code afin qu'il se comporte comme attendu, c'est-à-dire qu'il reconnaisse correctement un cas et redirige vers l'IHM correspondante.

Comme on peut le voir sur la capture d'écran, il n'y a pas encore de redirection vers une page dédiée : le résultat s'affiche simplement en dessous du formulaire. Le développement se fait donc étape par étape. La première étape consiste à faire en sorte que le mock reconnaisse les différents cas, puis à créer des IHM spécifiques pour chaque cas, car chaque cas présente des différences visuelles.

Après plusieurs tentatives et essais, j'ai réussi à atteindre mes objectifs. Plusieurs cas fonctionnent correctement, et il est donc nécessaire de saisir le bon **NumPec** pour afficher le cas demandé dans la console. Étant très impliqué dans mon projet et satisfait du résultat, je n'ai malheureusement pas pris de captures d'écran. Je ne peux donc pas expliquer de manière détaillée à l'oral toutes les étapes, car j'ai souvent oublié de documenter mon travail.

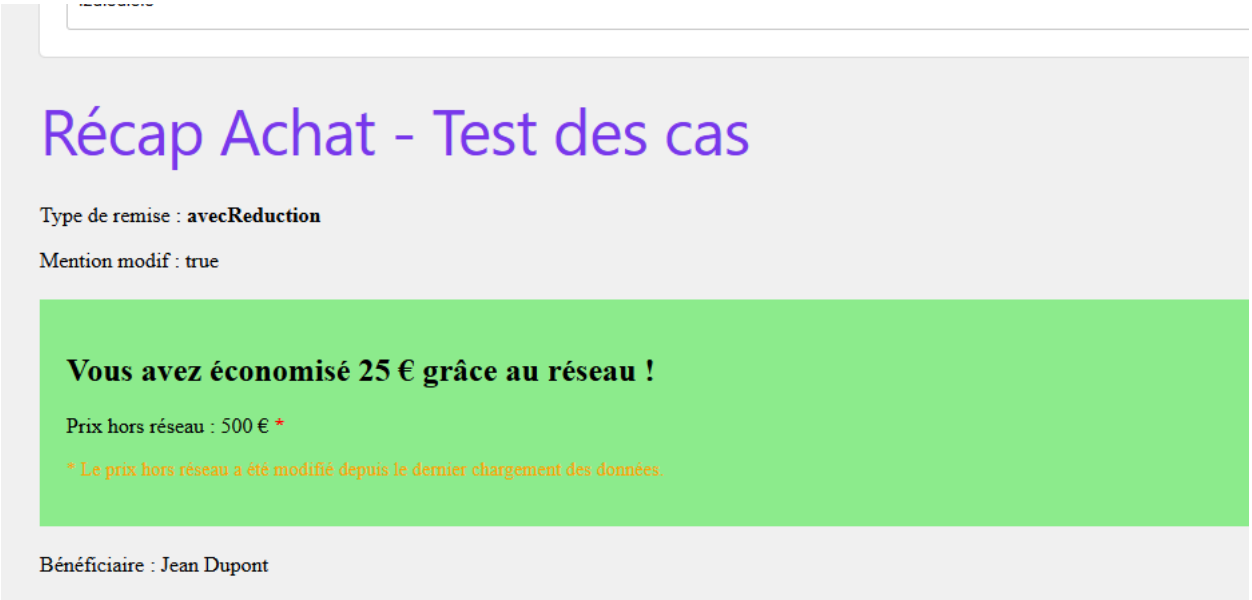
Puisque cette partie fonctionne maintenant dans la console et ne s'affiche pas encore à l'écran, avant de créer l'IHM de chaque cas, j'ai décidé de l'afficher juste en dessous du formulaire, comme le montre la capture d'écran ci-dessus. J'ai finalement réussi, et voici le résultat.



Comme on peut le constater, le numéro de PEC saisi fonctionne correctement et s'affiche bien en dessous du formulaire, comme prévu. La différence avec la première version, qui s'affiche également, est que dans celle-ci

je devais auparavant changer manuellement le cas en allant dans `récap-achat.service.ts` pour passer du cas 1 au cas 2, puis au cas 3. Désormais, il n'est plus nécessaire de le faire manuellement, car cette ligne de code me permet de vérifier le NumPEC et de le rediriger automatiquement vers le cas correspondant.

```
if (numPEC === '2024072300046412') {
```



Il s'agit d'un autre exemple visant à montrer que la fonctionnalité fonctionne correctement.

Maintenant que nous avons réussi à afficher les NumPEC sur l'IHM du formulaire et que le code vérifie correctement ces NumPEC pour rediriger vers le cas correspondant, nous allons nous concentrer sur le développement des IHM spécifiques à chaque cas.

À l'aide de Figma, où toutes les IHM sont présentes, nous devons les reproduire à peu près à l'identique. Je me suis donc référé à ce document (voir l'annexe) pour m'aider à construire mon IHM, qui sera principalement réalisée en HTML et CSS, tout en intégrant une grande part de TypeScript.

1. Structure HTML

Le HTML définit les blocs visibles de l'interface.

- Tout d'abord, on a un **en-tête (`header`)** contenant le **logo Carte Blanche**. Ce bandeau permet d'identifier immédiatement l'organisation et de maintenir une cohérence visuelle sur toute l'IHM. Le logo est dimensionné avec le CSS pour rester

proportionnel et agréable à l'œil.

- Ensuite, le contenu principal est contenu dans le **`<main>`**, lui-même divisé en plusieurs blocs :

    1. **Formulaire et saisie des données** : l'utilisateur saisit le **Numéro AMC**, la **Clé AMC** et le **Numéro PEC**. Ces champs sont liés au TypeScript via **`ngModel`** et `ViewChild`, ce qui permet de valider les données et d'interagir directement avec le DOM si nécessaire.

    2. **Synthèse des informations PEC** : une fois les données récupérées, la page affiche la **synthèse**. Les informations sont réparties dans plusieurs **cartes (`info-card`)** pour le bénéficiaire, la complémentaire santé et l'opticien. Chaque carte contient des informations détaillées : prénom, nom, adresse, téléphone, email, et logo éventuel. Ces cartes sont organisées en **grille responsive**, ce qui permet de s'adapter à différents écrans.

    3. **Bloc Économie et Équipement** : cette section centrale montre à la fois les **montants financiers** et la liste des **équipements optiques**.

        - Le graphique vertical représente visuellement les **totaux hors réseau**, le **reste à charge**, le **régime obligatoire** et le **régime complémentaire**.

        - Les **barres colorées** correspondent à ces montants, calculées dynamiquement via Angular (`[style.height.%]`) pour refléter les proportions exactes.

        - Une **légende** indique clairement la signification de chaque couleur.

        - La colonne des **avantages** met en avant les points positifs du réseau Carte Blanche : remises, proximité des professionnels, normes de qualité.

    4. **Liste des équipements** : les verres et montures sont affichés dans une **grille responsive**, avec des cartes contenant toutes les informations nécessaires :

fabricant, type, description, code EDI. Cette disposition permet à l'utilisateur de consulter facilement les détails de son équipement.

5. **Bouton de retour** : un bouton situé en bas de la page permet de revenir au formulaire initial, garantissant une navigation intuitive.

2. CSS – Mise en forme et ergonomie

Le CSS joue un rôle fondamental pour rendre l'interface **lisible, moderne et agréable** :

- Les **bandeaux (`.bandeau`, `.bandeau-card`, `.bandeau-title`)** définissent les titres et les séparations visuelles, avec des couleurs douces, des coins arrondis et des marges négatives pour superposer légèrement les titres sur les cartes.

- Les **cartes d'information** utilisent des **shadows**, du **padding** et des **bords arrondis** pour créer un relief et améliorer la lisibilité.

- Les **grilles (`.info-grille`, `.equipement-grille`)** garantissent un affichage **responsive**, où chaque carte s'ajuste automatiquement à la taille de l'écran.

- Le **graphique vertical** est stylisé pour afficher visuellement les proportions des différents montants financiers. Les couleurs sont choisies pour être distinctes et cohérentes avec l'identité visuelle du réseau Carte Blanche.

- La **typographie** est harmonisée : `Comfortaa` pour les titres, `Inter` pour les informations, et `Montserrat` pour certains textes spécifiques, assurant une **lecture agréable**.

- Les **listes d'avantages** utilisent des ✓ **colorés** pour attirer l'attention et guider l'œil de l'utilisateur sur les points importants.

- Enfin, les **boutons** sont stylisés avec un **fond violet**, du texte blanc et des coins arrondis pour indiquer leur interactivité et améliorer l'**expérience utilisateur**.

3. TypeScript – Logique et dynamique

Le TypeScript complète le HTML et le CSS en ajoutant **l'interactivité et la logique** :

1. **Validation du formulaire** : lorsque l'utilisateur clique sur "Accéder", la méthode `onAcceder()` vérifie que tous les champs sont remplis. Si un champ est vide, un message d'erreur est affiché, empêchant l'envoi du formulaire.

2. **Récupération des données PEC** : le service `RecapAchatService` est appelé pour récupérer les informations du PEC. Ces données sont ensuite stockées dans la variable `pec`, ce qui permet au HTML de les afficher dynamiquement grâce à Angular (`{{ pec.nom }}`, `{{ pec.montantTotalRemise }}`, etc.).

3. **Navigation conditionnelle** : selon les données récupérées (`typeRemise` et `mentionModifMontantHorsReseau`), l'utilisateur est redirigé vers l'un des trois cas (`cas1`, `cas2`, `cas3`). Cette logique garantit que chaque utilisateur voit uniquement la **page correspondant à sa situation**.

4. **Dynamisme de l'IHM** : les variables TypeScript (`showFrom`, `showRemise`, `pec`) contrôlent l'affichage des différents blocs, permettant à la page de **réagir immédiatement aux actions de l'utilisateur** sans recharger la page.

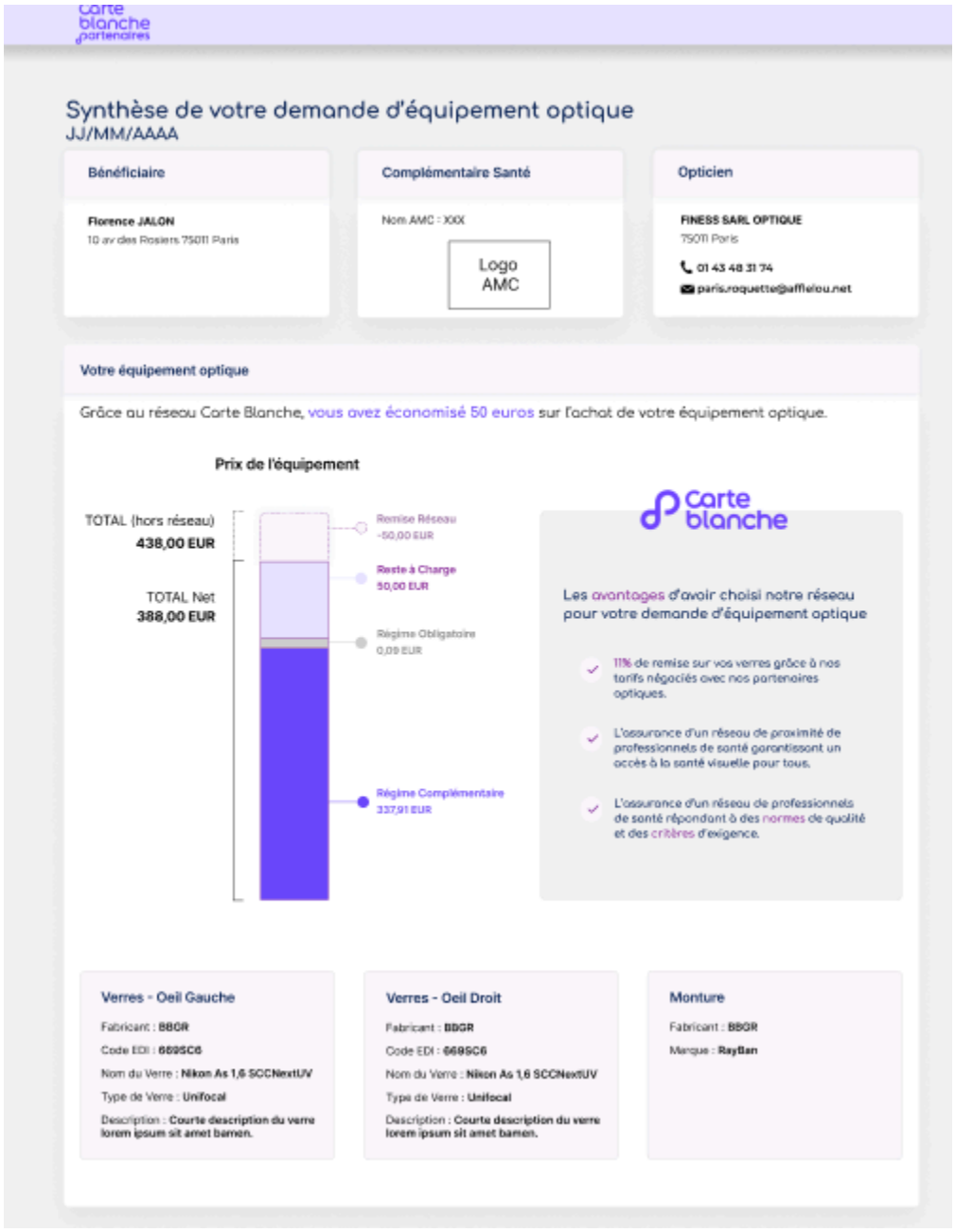
4. Interaction entre HTML, CSS et TypeScript

- Le **HTML** définit la structure et place les éléments sur la page.

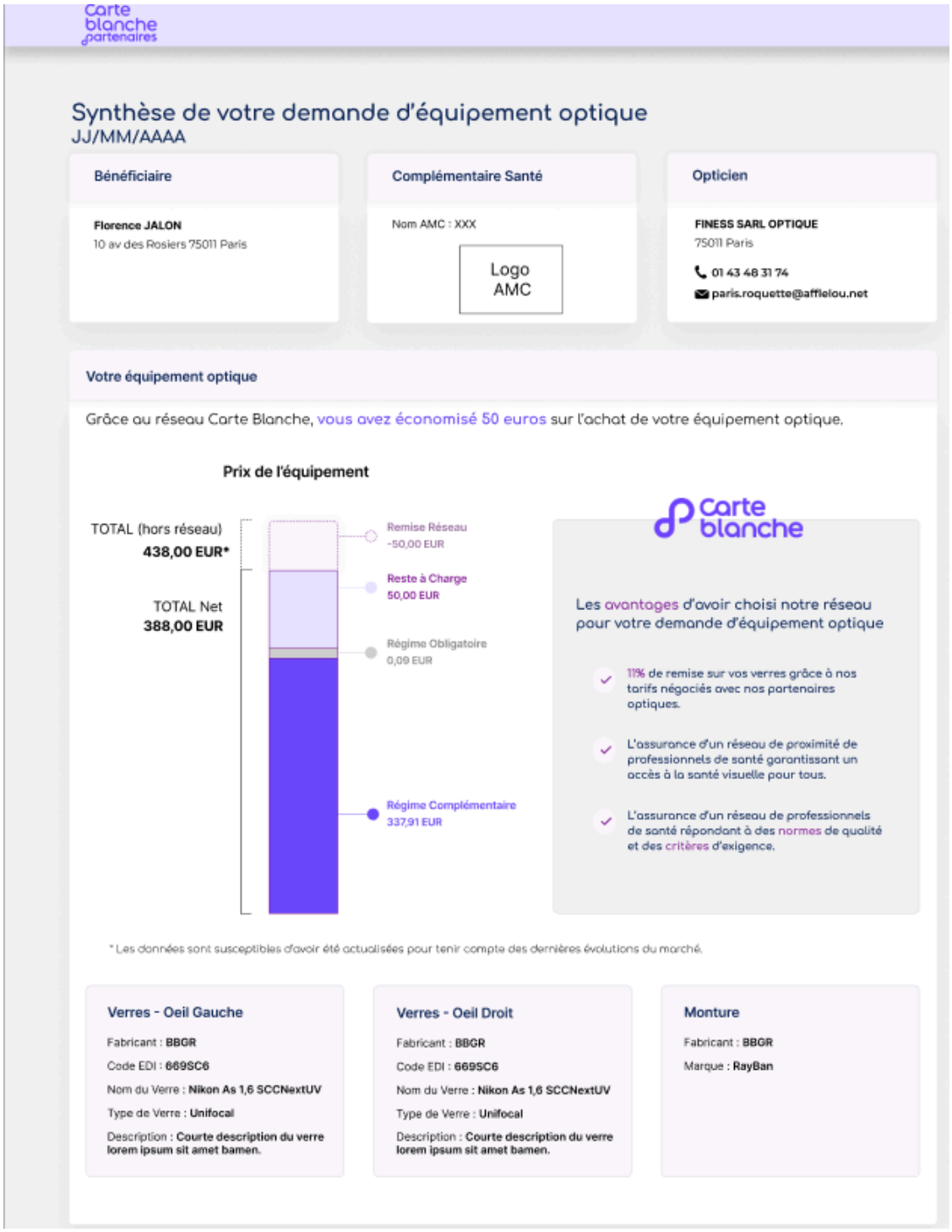
- Le **CSS** transforme cette structure en une interface agréable et lisible, en utilisant couleurs, polices, marges, grilles et flexbox.

- Le **TypeScript** rend cette interface **vivante et réactive** : il valide le formulaire, récupère les données, met à jour les informations affichées et redirige vers le bon cas.

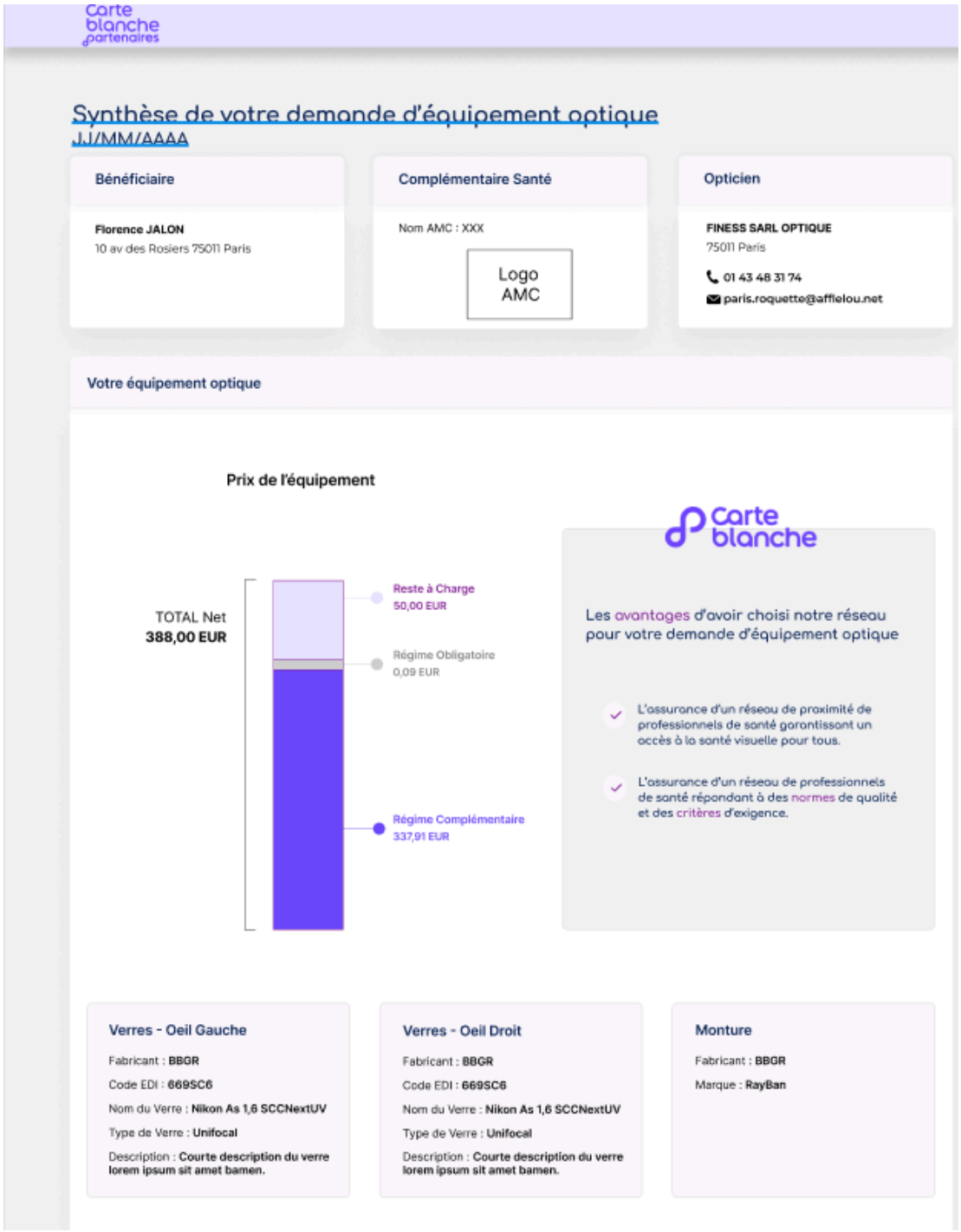
Ensemble, ces trois couches créent une **IHM moderne, responsive et fonctionnelle**, où l'utilisateur peut **saisir ses données, visualiser les informations clés et comprendre immédiatement les montants et économies liés à sa demande d'équipement optique**.



exemple IHM cas 1

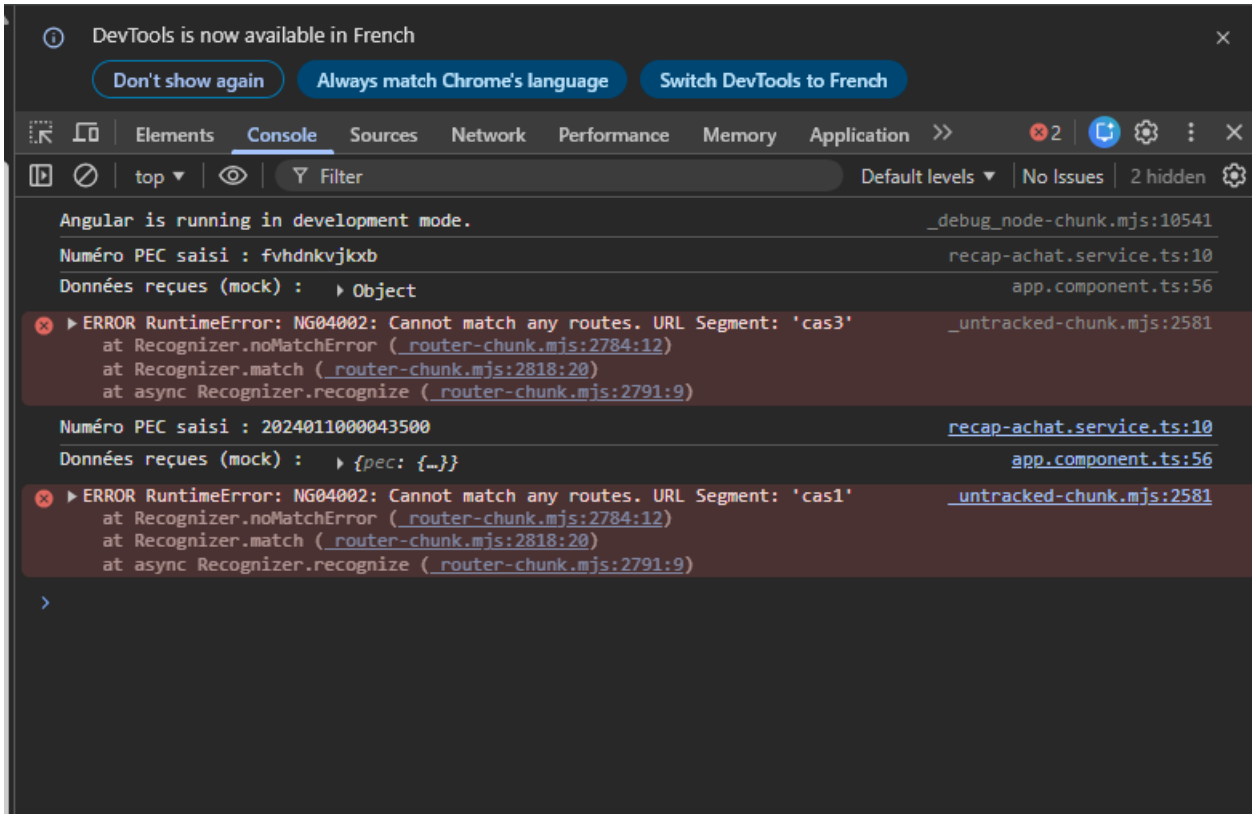


cas 2 avec remise

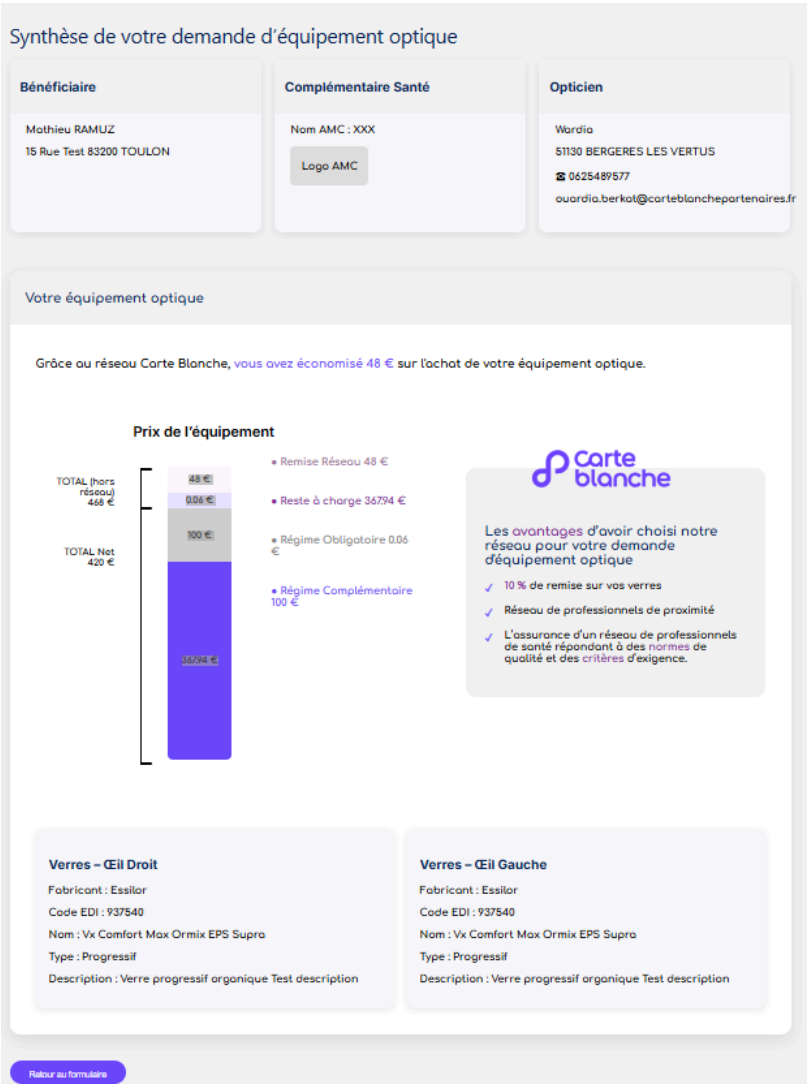

Carte blanche partenaires
Synthèse de votre demande d'équipement optique
JJ/MM/AAAA
Bénéficiaire
Florence JALON
10 av des Rosiers 75011 Paris
Complémentaire Santé
Nom AMC : XXX
Logo AMC
Opticien
FINESS SARL OPTIQUE
75011 Paris
01 43 48 31 74
paris.roquette@affielou.net
Votre équipement optique
Prix de l'équipement
TOTAL Net
388,00 EUR
Reste à Charge
50,00 EUR
Régime Obligatoire
0,09 EUR
Régime Complémentaire
337,91 EUR
Carte blanche
Les avantages d'avoir choisi notre réseau pour votre demande d'équipement optique
L'assurance d'un réseau de proximité de professionnels de santé garantissant un accès à la santé visuelle pour tous.
L'assurance d'un réseau de professionnels de santé répondant à des normes de qualité et des critères d'exigence.
Verres - Oeil Gauche
Fabricant : BBGR
Code EDI : 669SC6
Nom du Verre : Nikon As 1,6 SCCNextUV
Type de Verre : Unifocal
Description : Courte description du verre lorem ipsum sit amet bamen.
Verres - Oeil Droit
Fabricant : BBGR
Code EDI : 669SC6
Nom du Verre : Nikon As 1,6 SCCNextUV
Type de Verre : Unifocal
Description : Courte description du verre lorem ipsum sit amet bamen.
Monture
Fabricant : BBGR
Marque : RayBan

cas 3

il reconnaît les différents cas n’arrive pas à les rediriger vers l’IHM concerner



resultat final de l’ihm fait par moi meme :



le code est fonctionnele et on est même en avance donc maintenant que la 1 er partie du projet est terminé on s'attaque du coup au 2eme partie du projet donc adapter notre code a celui de l'api qui sera envoyer par une entreprise externe