**Jade DOMAS-VASSEROT**

Option SLAM

Session 2022

**APPLICATION COMPTES-RENDUS**



Documentation gsb

|  |
| --- |
| **BTS Services informatiques aux organisations SESSION 2022**  **ANNEXE 7-1-B : Fiche descriptive de réalisation professionnelle (recto)**  **Épreuve E5 - Conception et développement d’applications (option SLAM) - Coefficient 4** |

|  |  |  |  |
| --- | --- | --- | --- |
| **DESCRIPTION D’UNE RÉALISATION PROFESSIONNELLE** | | | **N° réalisation : 2** |
| **Nom, prénom : DOMAS-VASSEROT Jade** | | **N° candidat : 02146788571** | |
| **Épreuve ponctuelle** | **Contrôle en cours de formation** | **Date :** 24 / 04 / 2022 | |
| **Contexte de la réalisation professionnelle**  L’organisation Galaxy Swiss Bourdin est un laboratoire médical qui entreprend des recherches à propos des soins médicaux. Les salariés sont amenés à se déplacer. A chaque visite, ils doivent faire un compte-rendu de la visite en renseignant le praticien, les informations et les produits. | | | |
| **Intitulé de la réalisation professionnelle**  Application pour la gestion des comptes rendus | | | |
| **Période de réalisation :** 2ème année  **Lieu :** Lyon  **Modalité :  Seul(e)**  **En équipe** | | | |
| **Compétences travaillées**  Concevoir et développer une solution applicative  Assurer la maintenance corrective ou évolutive d’une solution applicative  Gérer les données | | | |
| **Conditions de réalisation[[1]](#footnote-1) (ressources fournies, résultats attendus)**  Un cahier des charges concernant le projet.  Un aperçu d’une pré-version attendue sur Access.  Résultats attendus :  Développement d’une application web sécurisé et authentifié pour permettre la gestion des comptes-rendus  Création d’un espace visiteur où l’on peut voir ses comptes-rendus ainsi qu’en saisir (une page profil également). Un espace où l’on peut visualiser ou chercher les différentes informations concernant un visiteur, un praticien ou un médicament. | | | |
| **Description des ressources documentaires, matérielles et logicielles utilisées [[2]](#footnote-2)**  On a pu utiliser la documentation et les ressources données (Cahier des charges et prévisualisation de l’appli sur access).  Logiciels :  PhpMyAdmin (MySQL), WampServer, Visual Studio Code, Navigateur Chrome, Apache, Lucid.app  Langages :  Framework :  Symfony (Php, Twig) et Vue JS (HTML, CSS, JavaScript) | | | |
| **Modalités d’accès aux productions [[3]](#footnote-3) et à leur documentation [[4]](#footnote-4)**  Documentation : <https://jadedomasvasserot.fr/index.php/ppe-2/> et dans le dossier doc du projet sur GitHub  <https://github.com/JadeDomasVasserot/GSB_BTS_CR/tree/main/doc>  Accès aux productions : Versionning sur GitHub : <https://github.com/JadeDomasVasserot/GSB_BTS_CR> | | | |

**SOMMAIRE**

1. *Le contexte gsb de l’appli de gestion comptes-rendus*
2. *configuration de l’environnement Symfony + vue js*
3. *mise en place de la base de données*
4. *développement de la connexion*
5. *Développement de l’interface médicaments*
6. *Développement de l’interface Praticiens*
7. *Développement de l’interface visiteurs*
8. *Développement de l’interface du visiteur connecté profil et voir ses comptes-rendus*
9. *développement de la création de compte-rendu*

**CONTEXTE GSB – APPLI COMPTE-RENDU**

L’organisation Galaxy Swiss Bourdin (GSB) est un laboratoire médical.

Elle résultat de la fusion en Galaxy (spécialiste dans les maladies virales comme le SIDA) et Swiss Bourdin (plutôt sur les médicaments).

Ils ont équipe d’employés amenés à participer à des réunions et visiter d’autres laboratoires. Après chaque visite, il faut élaborer un compte rendu de la visite renseignant quel praticien on est allé voir, s’il était remplacé, la date de la visite et les informations complémentaires.

Pour cela, il nous a été demandé de développer une application web dans le but de centraliser les comptes-rendus.

**CONFIG DE SYMFONY/VUE JS**

Dans un premier temps, il nous fallut créer un projet Symfony grâce à :

Symfony new

Puis, j’ai téléchargé les packages Vue et Encore (package pour permettre au fichier twig de lire le Vue JS) grâce à npm et yarn.

J’ai activé dans le package Encore : Vue Loader



J’ai ensuite configuré mes fichiers twig.

Un fichier est présent dans le répertoire /doc du GitHub, j’ai suivi cet exemple.

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

2 commandes pour lancer le symfony et le vue :

symfony serve:start

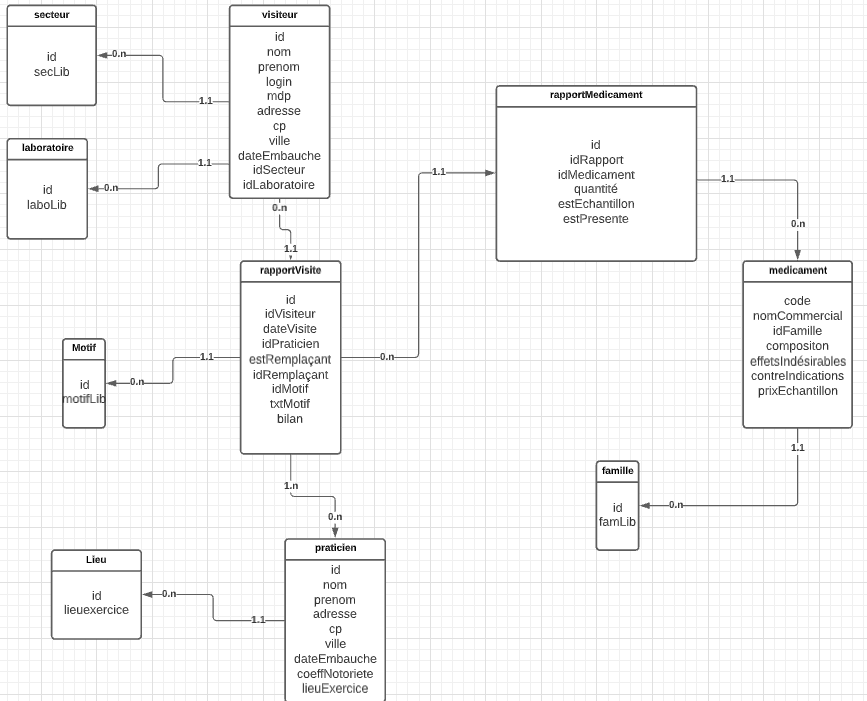
yarn encore dev-server --hot

J’ai décidé d’utiliser un outil que je connaissais qui permet de créer des API basiques. Il s’agit d’API Platform. Je l’ai donc installé via

Composer require api

**MISE EN PLACE DE LA BDD**

J’ai tout d’abord fait la conception de ma BDD grâce à Lucid.



J’ai ensuite créé mes entités avec :

Php bin/console make :entity

J’ai renseigné pour chaque les différents champs et type.

Une image contenant texte

Description générée automatiquement

Grâce à php bin/console make:entity - --regenerate App

J’ai généré les getters/setters

Il faut bien annoter

@ApiResource pour que

Api plateform puisse lire l’entité

Puis j’ai fait un php bin/console d doctrine:database:create

* php bin/console

doctrine:schema:create

Ce qui permet de créer la BDD

J’ai ensuite inséré des données.

**DEVELOPPEMENT CONNEXION**

J’ai créé le composant HomeController grâce à la commande php bin/console make:controller HomeController

Dans le dossier templates j’ai créé [home.html.twig](http://home.html.twig) dans le dossier /Home.

Il s’étend de base.html.twig où l’on ajoute notre config pour le vue JS. Il appelle le composant <home></home>

Une image contenant texte

Description générée automatiquement

Dans mon répertoire /assets qui comprend mes fichiers VueJS dans le répertoire assets/js.

Une image contenant texte, écran, capture d’écran

Description générée automatiquement

Dans le fichier app.js dans ce dernier j’ajoute mon composant VueJS Home.vue à app.js.



Une image contenant texte

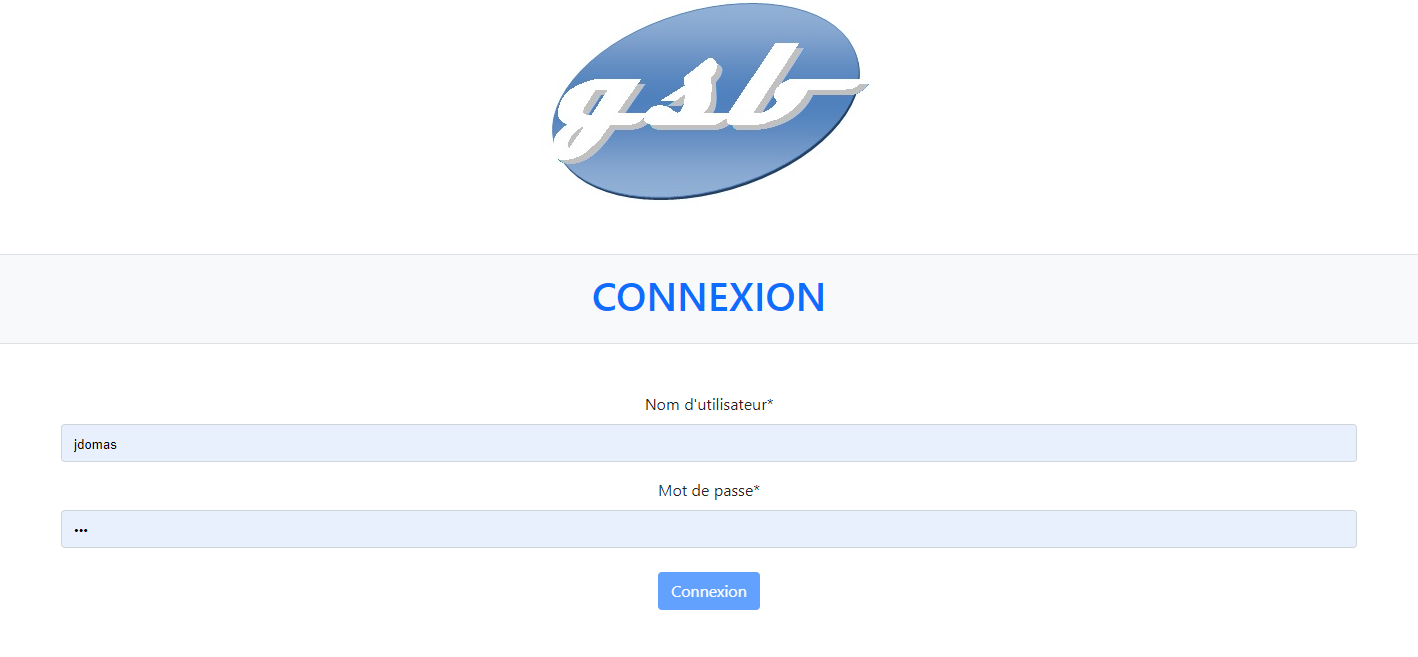
Description générée automatiquement

J’ai créé un répertoire js et components dans le répertoire assets/js où je créé un fichier Home.vue et Home.js.

Home.js correspond au javascript du component VueJS

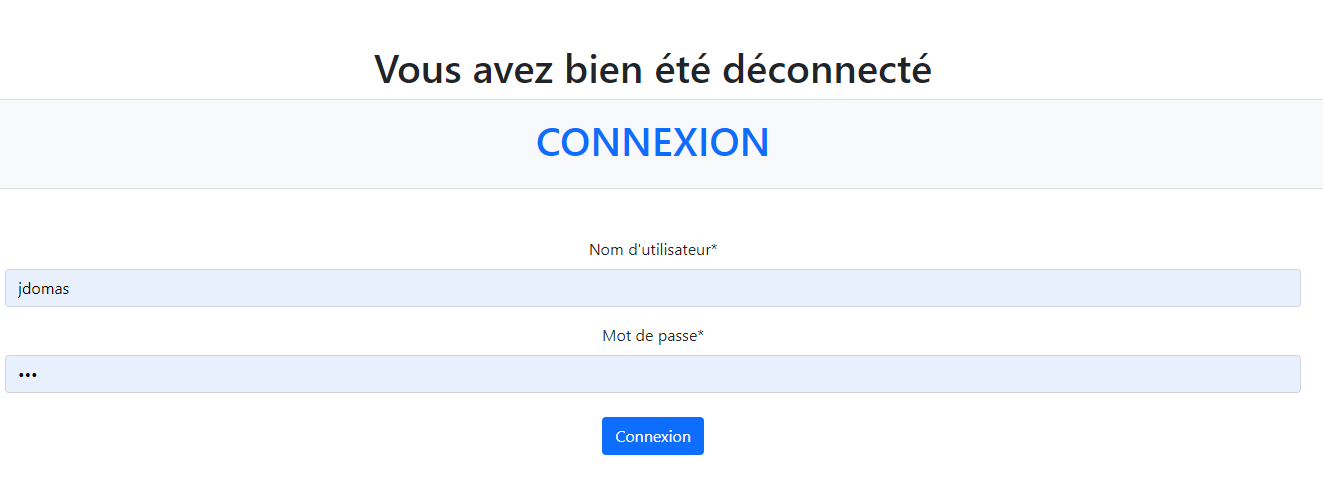


Tandis que Home.vue comprendra le code front de la page. C’est un formulaire avec 2 inputs text permettant à l’utilisateur de se connecter via l’action /connexion. J’ai créé des fonctions JS permettant à l’utilisateur d’envoyer le formulaire avec entrée ainsi que de gérer l’état disabled ou non du bouton se connecter.

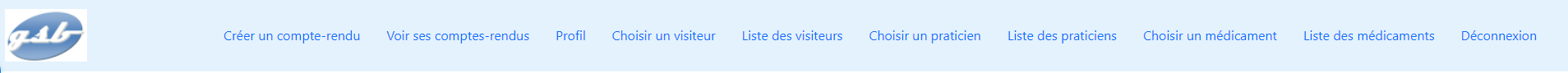


Du côté back, j’ai créé la route /connexion qui permet de récupérer les variables POST. J’effectue une recherche dans la BDD par le nom d’utilisateur entré. Puis, on vérifie que la password entré correspond bien à celui en rapport avec celui de l’user. Si c’est le cas on débute une session grâce à SessionInterface de Symfony et je set les variables dans la session. Je return la vue profilVisiteur.html.twig dont on parlera plus tard si ça marche sinon sur erreurConnexion.html.twig si le mot de passe ne correspond pas. S’il y a une erreur avec le serveur, on affiche la page erreur404.html.twig (chaque twig est relié à un composant Vue JS).

J’ai créé en parallèle ma route /deconnexion qui déconnecte l’utilisateur et clear la session. Elle amène sur la page twig deconnexion.html.twig et remet le formulaire pour se connecter.



J’ai aussi créé le composant sommaire qui est ajouté à chaque vue.



**DEVELOPPEMENT MEDICAMENTS**

Pour l’interface médicament, il y a la partie « Liste des médicaments » et l’autre partie qui est « Choisir un médicament ».

J’ai créé le Controller : MediacamentController qui définit les différentes routes pour nos pages twigs.

Une image contenant texte

Description générée automatiquement

Le fonctionnement sera le même pour les interfaces praticiens et visiteurs, je n’expliquerais donc que pour l’exemple de médicament.

1. Pour la vue « Liste des médicaments ».

J’ai créé un composant ListeMedicament.vue que j’ai ajouté dans mon app.js

Mon fichier twig appelle ce composant ainsi que le sommaire en haut.

Une image contenant texte

Description générée automatiquement

J’ai utilisé un tableau avec du css bootstrap pour faire la Vue.

Je fais un appel axios (j’ai installé le package axios avec npm avant) pour récupérer l'ensemble des données de l'item grâce à une API créée par API Platform.

Une image contenant texte

Description générée automatiquement

Avec le Vue JS j'affiche les informations grâce à un v-for

Une image contenant texte

Description générée automatiquement

Pour praticien et visiteur j'utilise le package Moment afin de formater la date.

Une image contenant texte

Description générée automatiquement

1. Pour la vue « Chosir un médicament ».

J’ai créé un composant IndexMedicament.vue que j’ai ajouté dans mon app.js

Mon fichier twig appelle ce composant ainsi que le sommaire en haut.

Une image contenant texte

Description générée automatiquement

Voici le rendu du template lorsque l’on n’a pas choisi de médicament.

Il y a donc un select pour choisir l’ensemble des médicaments dans la BDD grâce à Axios on va faire un appel API (avec API Plateform) afin de récupérer la liste des médicaments qu’on affiche par nom grâce à un v-for sur les options du select.

Ensuite, je récupère l’id passé en value du select ce qui me permettra de faire un autre appel Axios d’une Api de Api Platform qui me permettra d’obtenir les informations de 1 éléments grâce à son id.

    Explication des méthodes :

Je fais un appel Axios pour récupérer la liste de l'item en question.

J’affiche l'ensemble grâce à un v-for avec un keys et une value pour chaque item qui sera l'id.

La personne clique sur le son choix qu'il souhaite dans le select avec v-model une variable afin de récupérer la valeur choisie.

Ce qui va lancer la méthode choixPraticien() qui fait l'appel Axios afin de récupérer les informations de l'item choisi.

Puis on affiche les information en vérifiant que l'item a bien été choisi avec la méthode isPraticienChoisi()

Avec le Vue JS j'affiche les informations.

Une image contenant texte

Description générée automatiquement

**DEVELOPPEMENT PRATICIENS**

1. Liste des praticiens

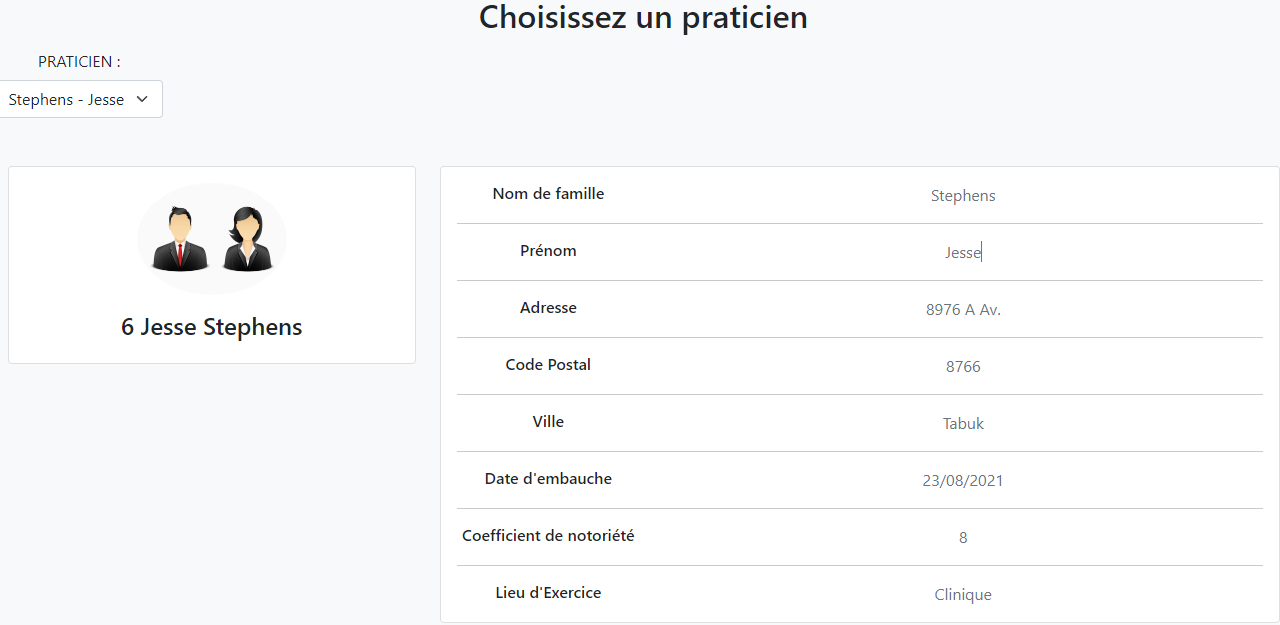


1. Choisir un praticien

Une image contenant texte

Description générée automatiquement

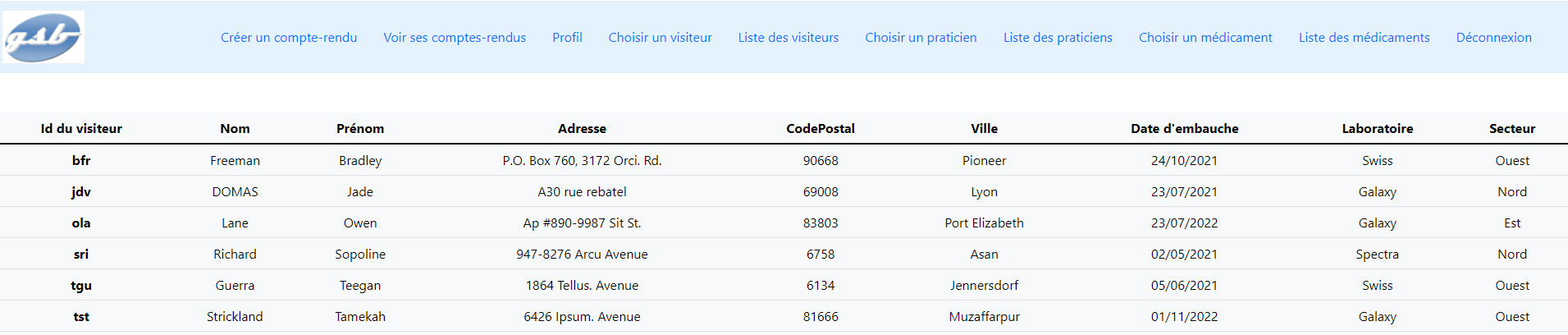
Avant



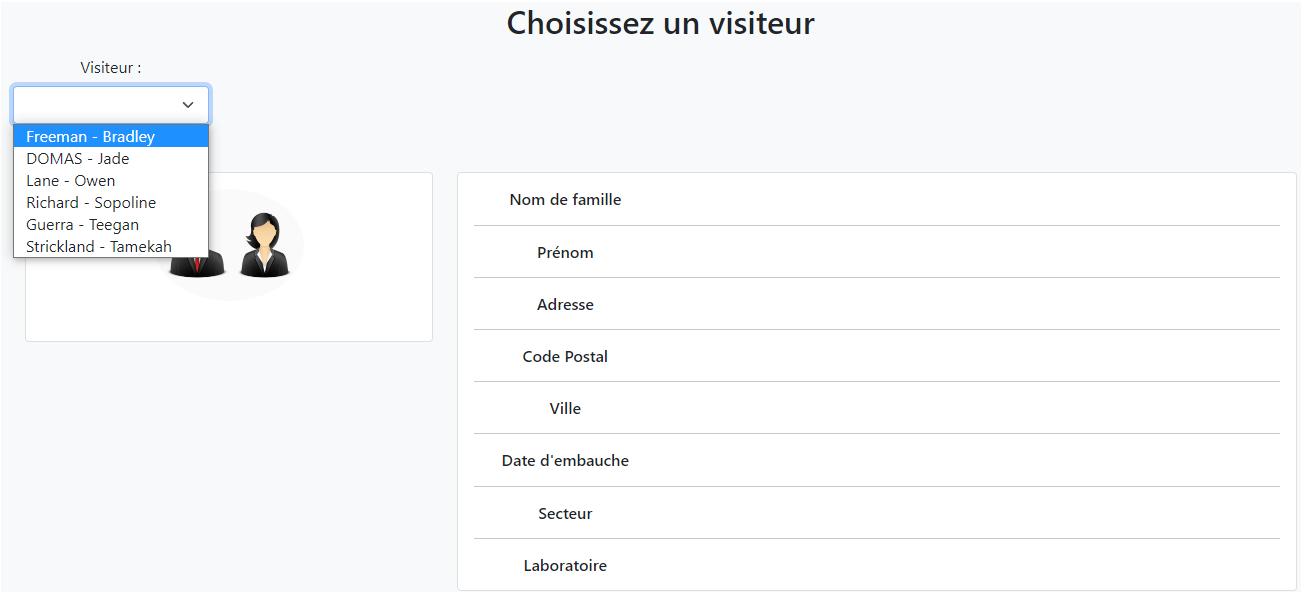
Après

**DEVELOPPEMENT VISITEURS**

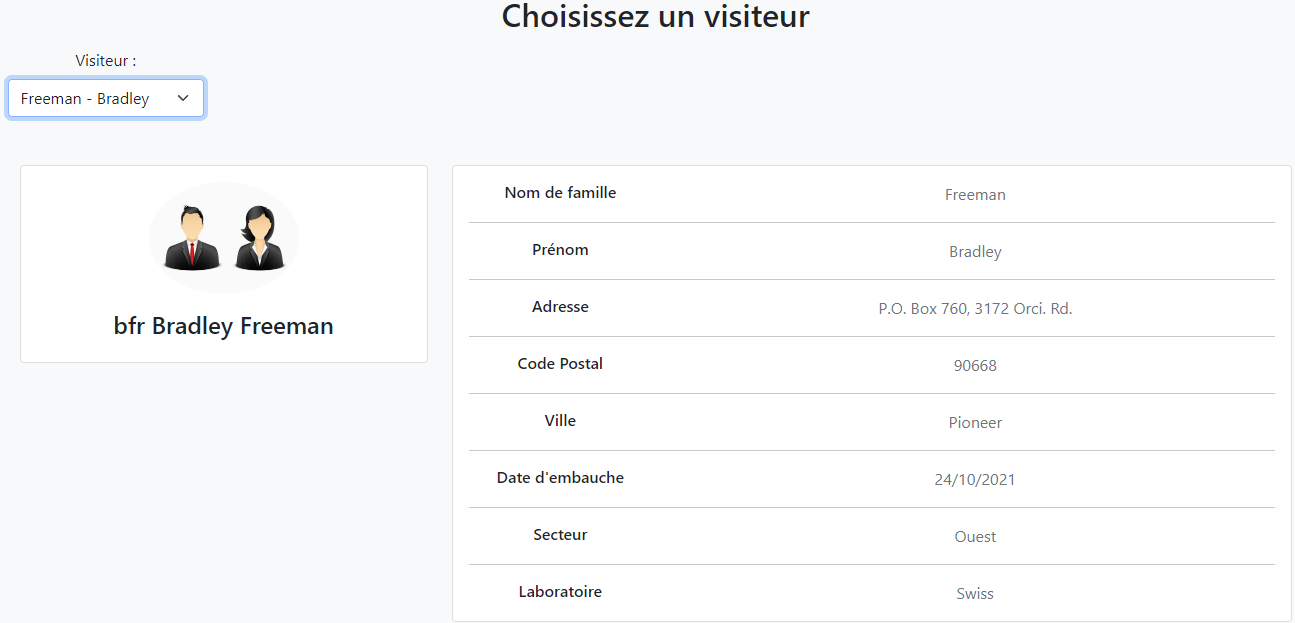
1. Liste des visiteurs



1. Choisir un visiteur



Avant



Après

**DEVELOPPEMENT INTERFACE UTILISATEUR CONNECTE**

1. Espace Profil

J’ai créé un composant Profil.vue que j’ai ajouté dans mon app.js

Mon fichier twig appelle ce composant ainsi que le sommaire en haut.

Une image contenant texte

Description générée automatiquement

On arrive sur cette page lorsqu’on clique sur « Profil » dans le sommaire.

Cela dirige vers la route /visiteur/

Dans notre vue, on appelle la requête axios /visiteur/session de l’API que j’ai créée, elle retourne un JSON avec les informations du visiteur dans la session. Puis, on affiche son nom et prénom.

On peut voir les informations si on clique sur « voir les infos ».

J’ai utilisé la même mise en page que pour « choisir le médicament ».

Lorsque l’utilisateur clique sur le bouton, cela lance une requête axios vers l’API.

Puis, j’affiche les informations si le bouton a été cliqué.

Une image contenant texte

Description générée automatiquement

1. Espace «Voir ses comptes-rendus »

J’ai créé un composant ShowCr.vue que j’ai ajouté dans mon app.js

Mon fichier twig appelle ce composant ainsi que le sommaire en haut.

Je fais un appel API grâce à Axios de l’API /rapportvisite/show/list que j’ai créé. Je recherche les rapports de visites du visiteur enregistré dans la session.

Je return un tableau (array) au format JSON comprenant (id, dateVIsite, estRemplaçant, bilan, idmotif, motifText, le nom et prénom du praticen et celui du remplaçant s’il y en a un).

Une image contenant texte, ordinateur, intérieur, capture d’écran

Description générée automatiquement

On peut cliquer sur le bouton « voir Infos » pour voir les détails Produits du rapport sélectionné.

On fait un appel API vers la route /rapportvisite/voirProduits/{id}

Dans notre back, cette route retourne un JSON. On cherche tout d’abord les RapportMedicament qui ont l’idRapport passé en paramètre.

Et pour chaque rapportMedicament, on a créé un tableau avec id, quantité, estEchantillon, estPresente, nomMedicament, idRapport.

On retourne un tableau comprenant tous les tableaux. Une image contenant texte

Description générée automatiquement

**CREATION DE COMPTE-RENDU**

J’ai créé un composant Cr.vue que j’ai ajouté dans mon app.js

Mon fichier twig appelle ce composant ainsi que le sommaire en haut.

J’ai créé du côté front-end le formulaire qui a comme action la route /rapportvisite/new

Grâce à axios j’ai fait appel à des API de API Platform afin de récupérer la liste des médicaments, des motifs et des praticiens



Je peux ensuite les afficher dans des <select> et <option> grâce à un v-for.

Je récupère la valeur grâce à v-model sur le select en passant une variable

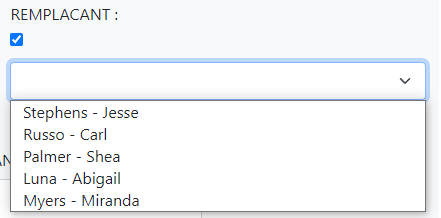
Une image contenant texte, écran, capture d’écran

Description générée automatiquement

J’ai géré la gestion de l’affichage et du comportement de mes inputs.

- Voir si remplaçant a été coché afin de disabled ou non le select du remplaçant.





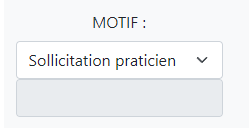
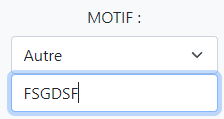
- Voir si un praticien a été choisi afin d'afficher son coefficient de notoriété en dessous

Une image contenant texte

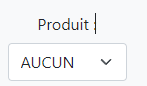
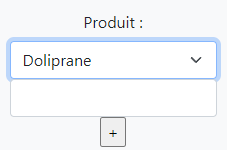
Description générée automatiquement Une image contenant texte

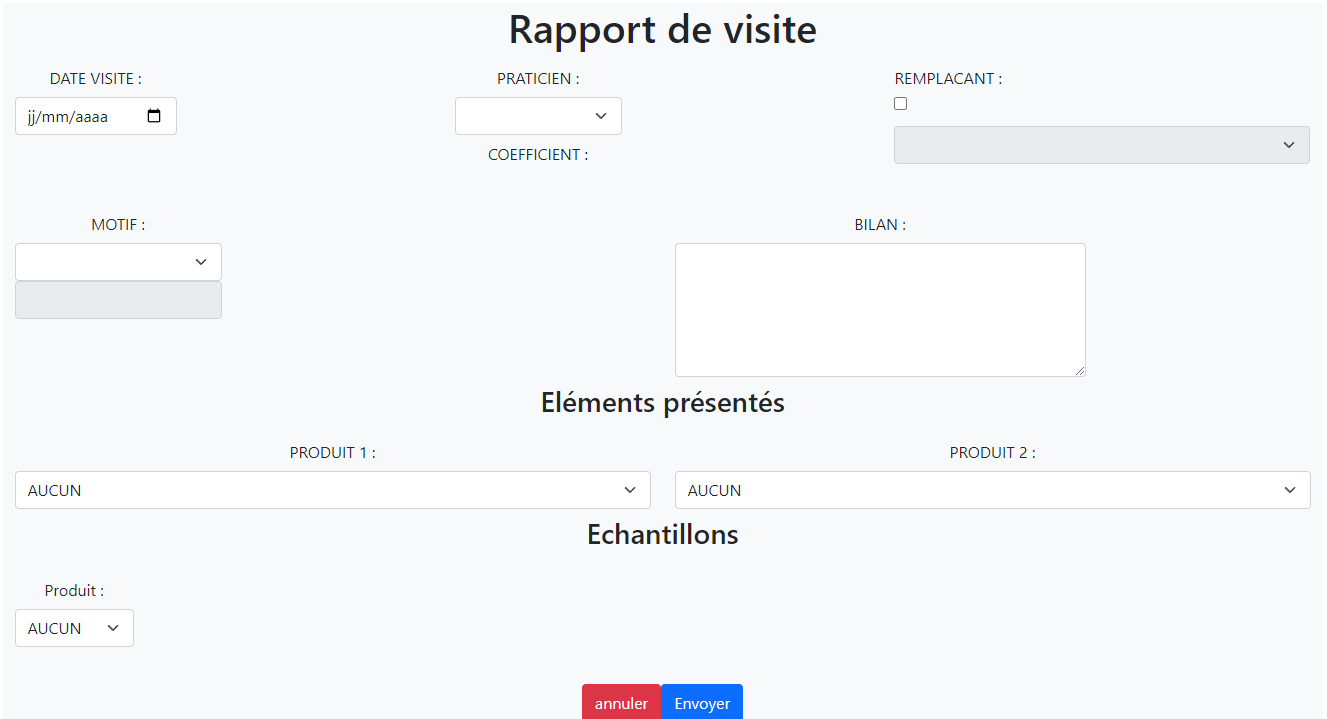
Description générée automatiquement

- Voir si le motifs choisi est == à "Autre" pour disabled ou non l'input text

- Voir si le produit échantillon = AUCUN pour ne pas afficher (grâce à v-if) la quantité et le bouton +



Du côté Back-end, la route/rapportvisite/new

Une image contenant texte, capture d’écran, portable, écran

Description générée automatiquement

On récupère l’id du visiteur connecté grâce à SessionInterface et on le set dans un nouveau rapport de Visite qu’on a créé.

Avec les valeurs POST récupérées on les sets dans le rapportVisite.

Une image contenant texte

Description générée automatiquement

Exemple pour dateVisite (on l’on a transformé en dateTime) et bilan.

Un rapport de visite contient des clefs étrangères.

Pour l’id Motif par exemple, je recherche le motif correspondant à la valeur

POST (idMotif sélectionné)

Une image contenant texte

Description générée automatiquement

On fait la même pour le praticien et s’il y a un remplaçant.

Une image contenant texte

Description générée automatiquement

On utilise EntityManagerInterface pour persiste puis flush le rapport de visite.

Ensuite, à chaque rapport de visite, il faut ajouter les produits référencés dans la table rapportMedicament.

On regarde si l’utilisateur à entrer un produit (c’est-à-dire qu’on détecte la variable $\_POST) et si la valeur n’est pas égale à NULL (ce qui signifie que l’utilisateur ait entré AUCUN).

On doit créer un nouveau RapportMedicament

Exemple pour Produit 1 :

Une image contenant texte

Description générée automatiquement

On utilise EntityManagerInterface pour persiste puis flush.

**CONCLUSION**

J’ai évité de mettre trop de captures d’écran du code. Vous pouvez trouver le code sur GitHub et il est commenté.

Le fichier README.md récapitule l’ensemble des commandes utilisées.

Merci de votre lecture.

1. En référence aux *conditions de réalisation et ressources nécessaires* du bloc « Conception et développement d'applications » prévues dans le référentiel de certification du BTS SIO. [↑](#footnote-ref-1)
2. Les réalisations professionnelles sont élaborées dans un environnement technologique conforme à l’annexe II.E du référentiel du BTS SIO. [↑](#footnote-ref-2)
3. Conformément au référentiel du BTS SIO « *Dans tous les cas, les candidats doivent se munir des outils et ressources techniques nécessaires au déroulement de l’épreuve. Ils sont seuls responsables de la disponibilité et de la mise en œuvre de ces outils et ressources. La circulaire nationale d’organisation précise les conditions matérielles de déroulement des interrogations et les pénalités à appliquer aux candidats qui ne se seraient pas munis des éléments nécessaires au déroulement de l’épreuve.* ». Les éléments peuvent être un identifiant, un mot de passe, une adresse réticulaire (URL) d’un espace de stockage et de la présentation de l’organisation du stockage. [↑](#footnote-ref-3)
4. Lien vers la documentation complète, précisant et décrivant, si cela n’a été fait au verso de la fiche, la réalisation professionnelle, par exemple service fourni par la réalisation, interfaces utilisateurs, description des classes ou de la base de données. [↑](#footnote-ref-4)