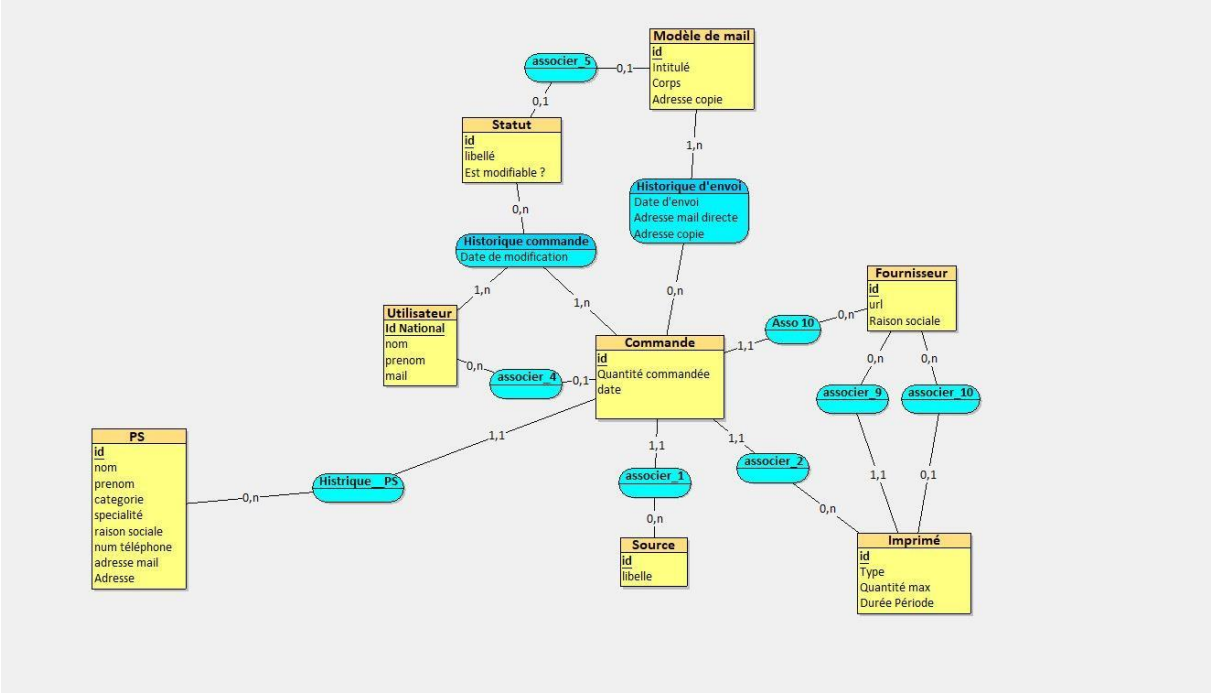
SCIPS

Présentation du Projet SCIPS

SCIPS est un applicatif de gestion de commande d’imprimés médicaux conçu pour les professionnels de santé, à destination des responsables de ces professionnels travaillant à la CPAM (Caisse Primaire d'Assurance Maladie). L'application a pour objectif principal de tracer, gérer et historiser les commandes d'imprimés médicaux, tout en facilitant les interactions avec le service logistique de la CPAM.

MCD :

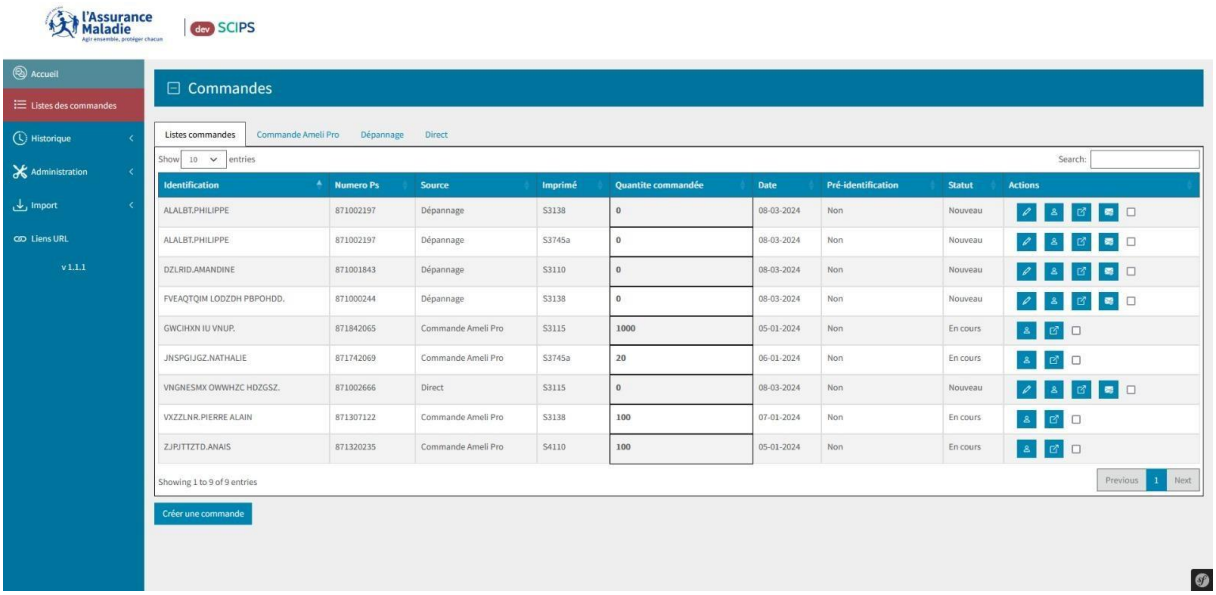


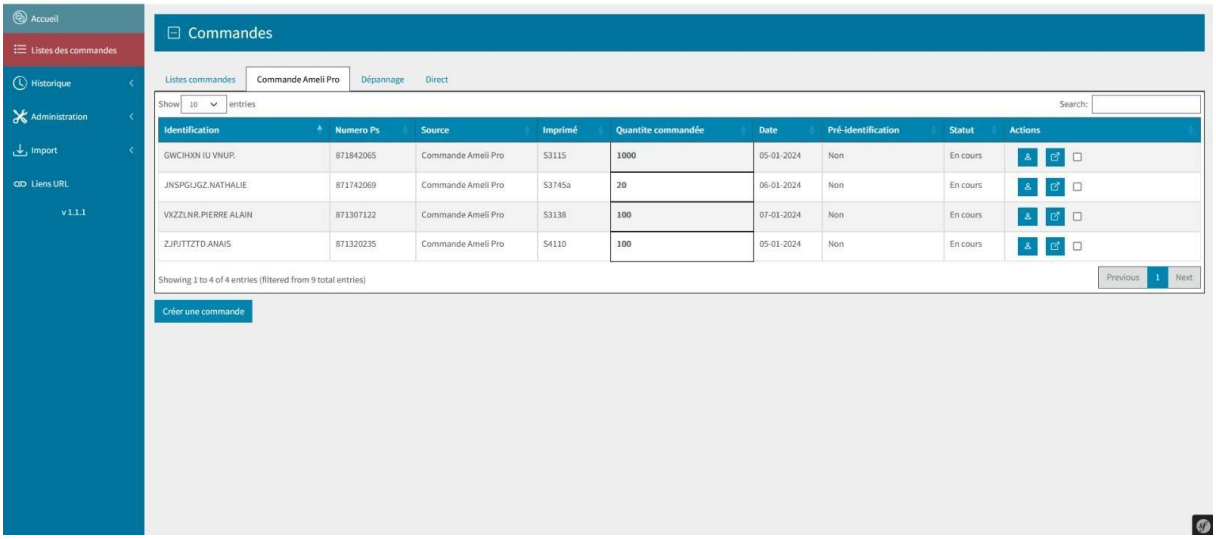
Page d’Accueil :

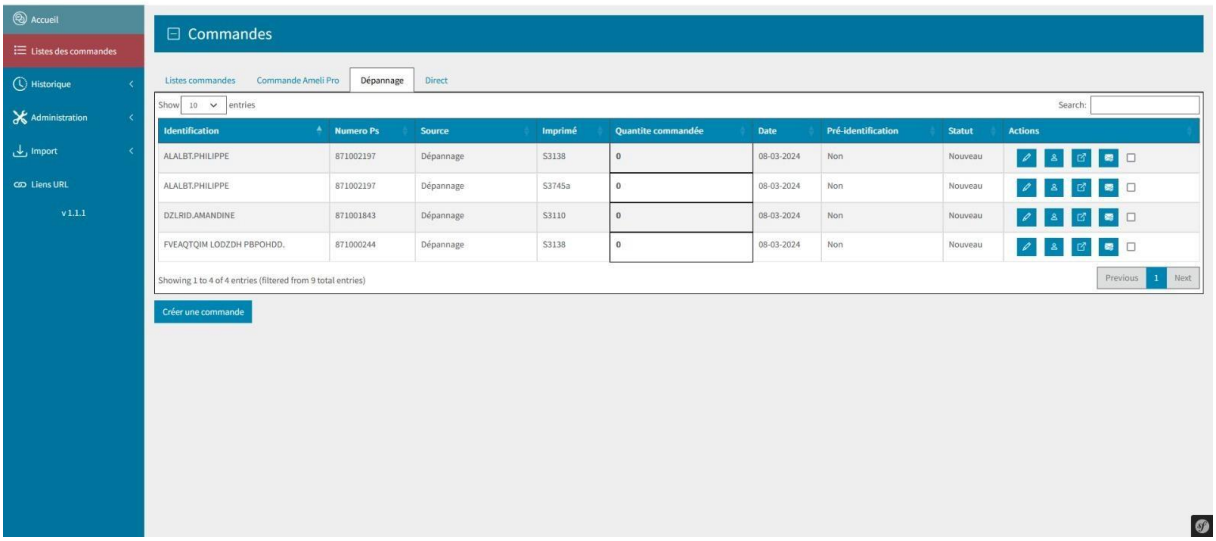
Liste Commandes :

Vue :

Cette section affiche toutes les commandes en cours, y compris leur statut et d'autres informations pertinentes.



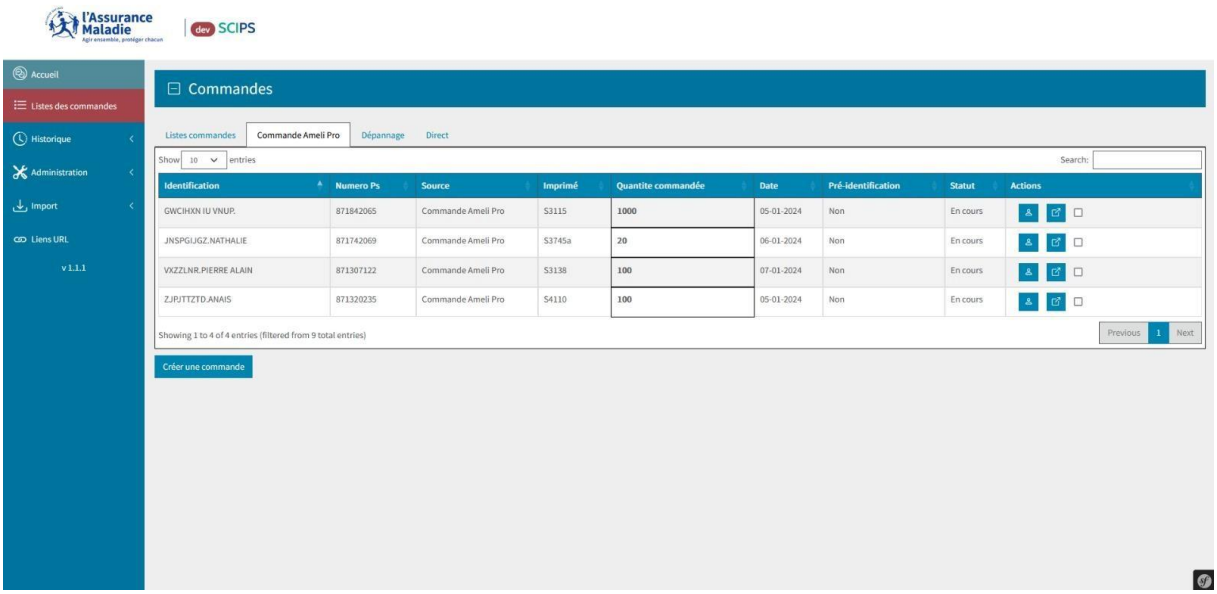

Accueil
Listes des commandes
Historique
Administration
Import
Liens URL
v 1.1.1
Commandes
Listes commandes
Commande Ameli Pro
Dépannage
Direct
Show 10 entries
Search:
Identification
Numero Ps
Source
Imprimé
Quantite commandée
Date
Pré-identification
Statut
Actions
GWCIHXN IU VNUP. 871842065 Commande Ameli Pro S3115 1000 05-01-2024 Non En cours
JNSPGIJGZ.NATHALIE 871742069 Commande Ameli Pro S3745a 20 06-01-2024 Non En cours
VXZZLNR.PIERRE ALAIN 871307122 Commande Ameli Pro S3138 100 07-01-2024 Non En cours
ZJPJTTZTD.ANAIS 871320235 Commande Ameli Pro S4110 100 05-01-2024 Non En cours
Showing 1 to 4 of 4 entries (filtered from 9 total entries)
Previous 1 Next
Créer une commande

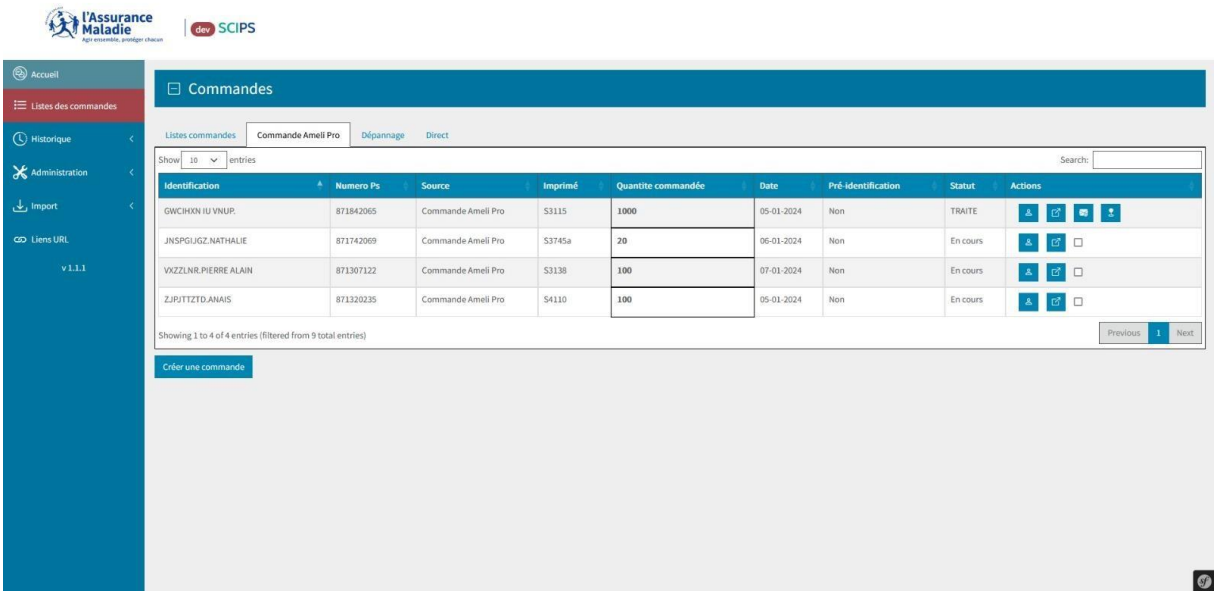

Accueil
Listes des commandes
Historique
Administration
Import
Liens URL
v 1.1.1
Commandes
Listes commandes
Commande Ameli Pro
Dépannage
Direct
Show 10 entries
Search:
Identification
Numero Ps
Source
Imprimé
Quantite commandée
Date
Pré-identification
Statut
Actions
ALALBT.PHILIPPE 871002197 Dépannage S3138 0 08-03-2024 Non Nouveau
ALALBT.PHILIPPE 871002197 Dépannage S3745a 0 08-03-2024 Non Nouveau
DZLRID.AMANDINE 871001843 Dépannage S3110 0 08-03-2024 Non Nouveau
FVEAQTQIM LODZDH PBPOHDD. 871000244 Dépannage S3138 0 08-03-2024 Non Nouveau
Showing 1 to 4 of 4 entries (filtered from 9 total entries)
Previous 1 Next
Créer une commande

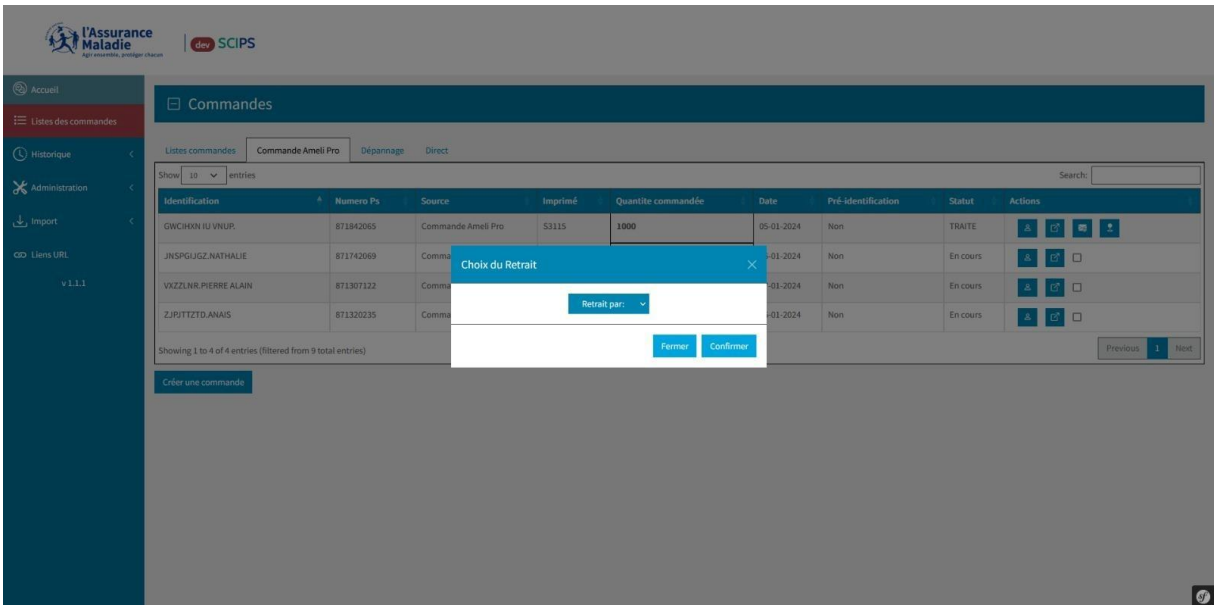
Fonctionnalité des buttons :

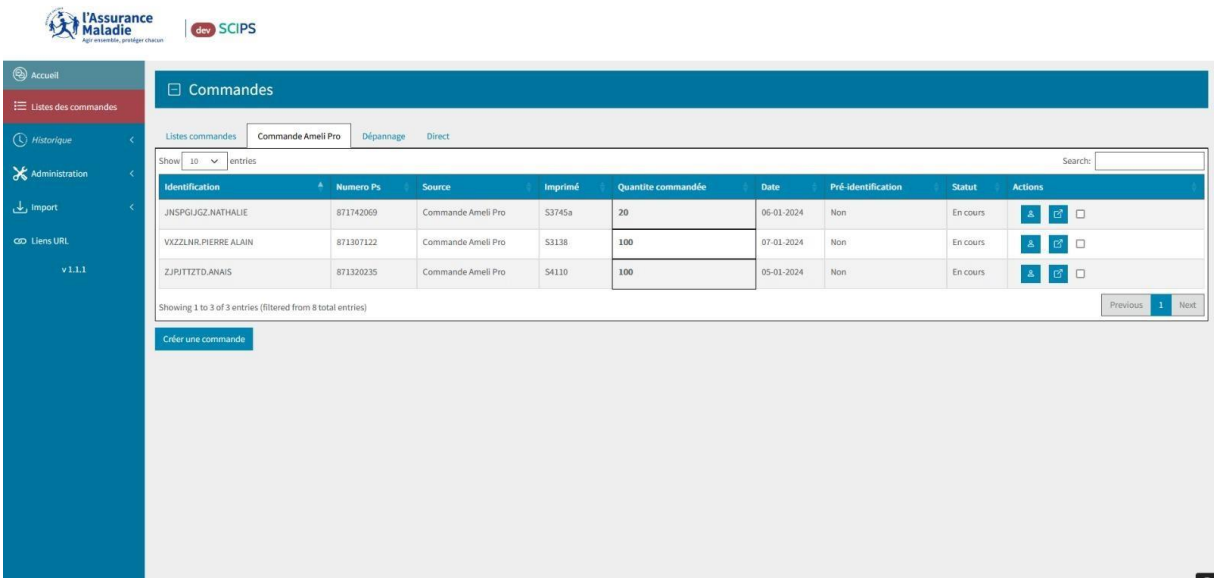
Checkbox pour les commandes Ameli Pro :

- Permet de gérer les statuts des commandes. Initialement, les commandes sont en cours. En cliquant sur la checkbox, elles passent au statut "Commande traitée".
- Le client doit cliquer sur le bouton "Retrait" et confirmer pour signaler que la commande est prête à être retirée. En choisissant un canal de retrait, la commande devient prête. Une fois prête, elle n'apparaît plus dans la liste des commandes, mais reste enregistrée dans la base de données pour la gestion de l'historique des commandes et des e-mails.



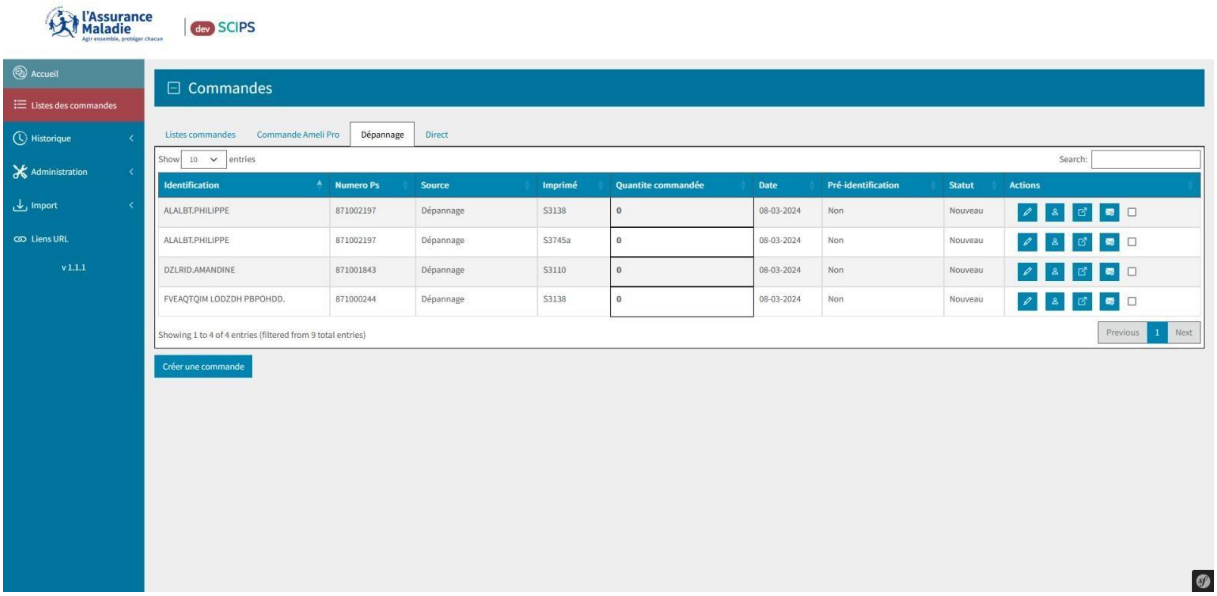


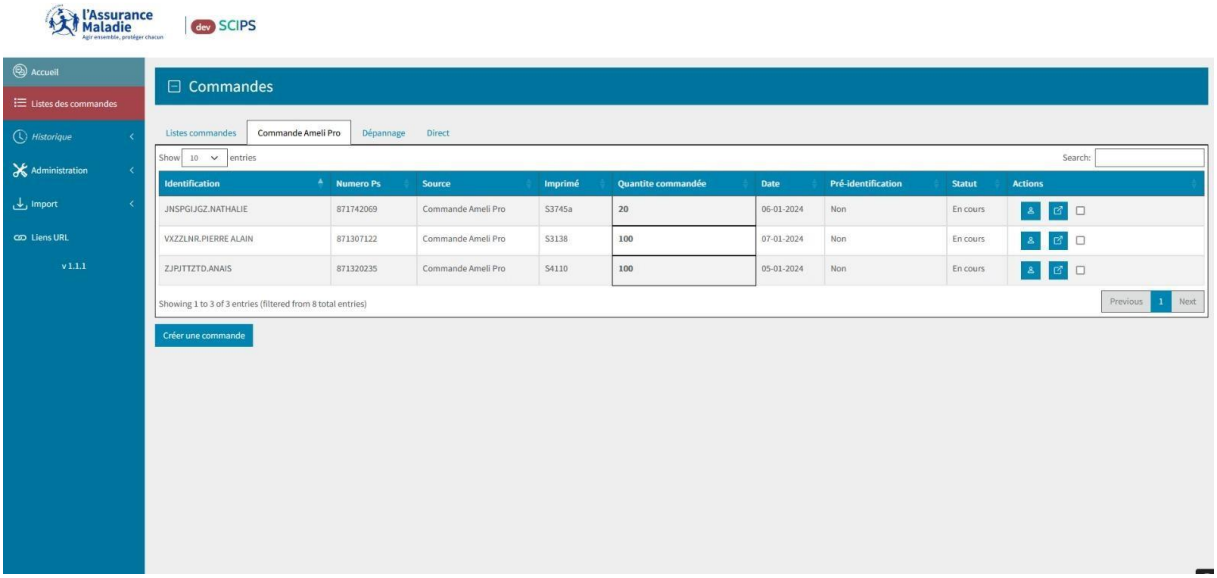




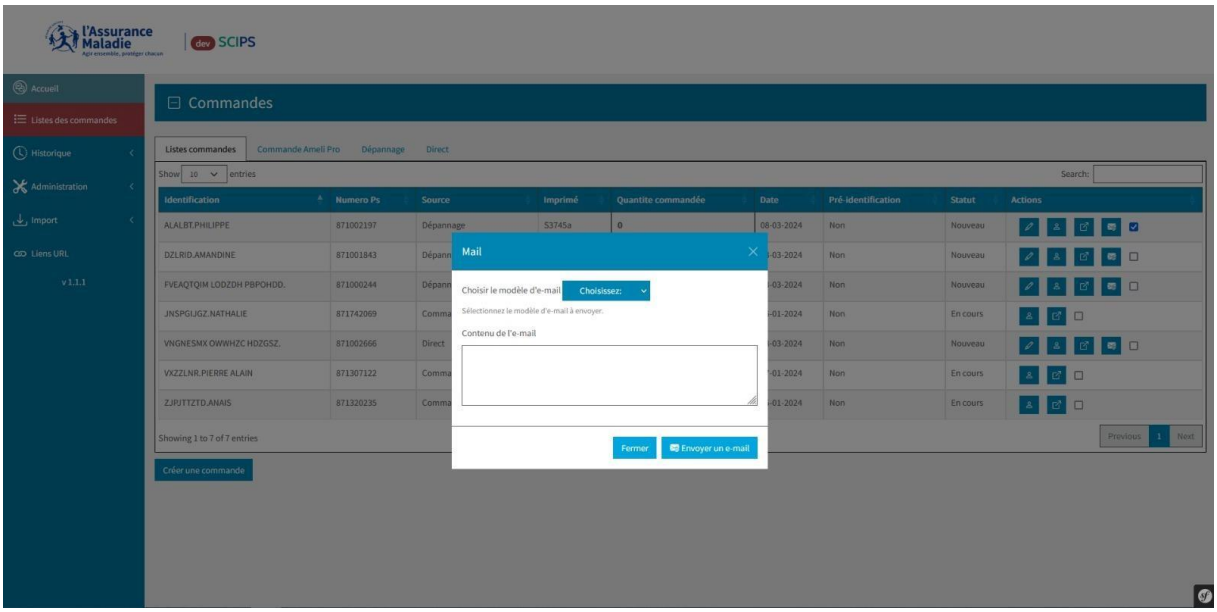
Checkbox pour les commandes directes et de dépannage :

- Les commandes de dépannage et directes sont créées par le représentant du service client. Elles ne sont pas importées.
- Lors de leur création, leur statut est par défaut "Nouvelle". En cliquant sur la checkbox, le statut passe à "Traitée". Une autre checkbox "Prête" est disponible pour signaler que la commande est prête à être traitée.

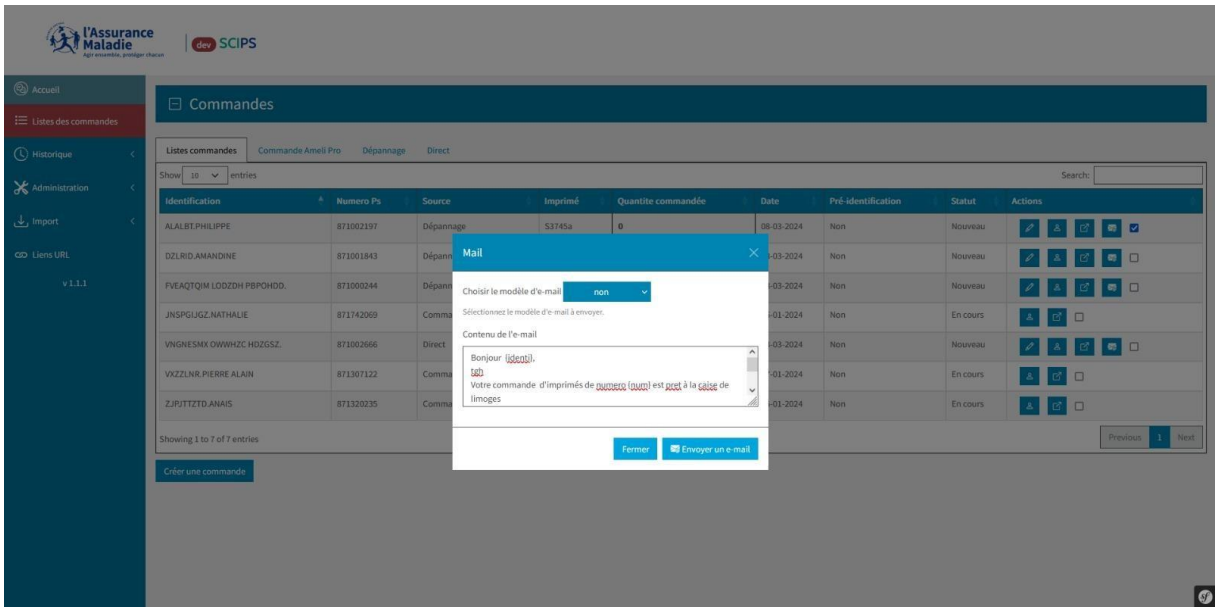

Accueil
Listes des commandes
Historique
Administration
Import
Liens URL
v 1.1.1
Commandes
Listes commandes
Commande Ameli Pro
Dépannage
Direct
Show 10 entries
Search:
Identification
Numero Ps
Source
Imprimé
Quantite commandée
Date
Pré-identification
Statut
Actions
ALALBT.PHILIPPE 871002197 Dépannage S3138 0 08-03-2024 Non Nouveau
ALALBT.PHILIPPE 871002197 Dépannage S3745a 0 08-03-2024 Non Nouveau
DZLRID.AMANDINE 871001843 Dépannage S3110 0 08-03-2024 Non Nouveau
FVEAQTQIM LODZDH PBPOHDD. 871000244 Dépannage S3138 0 08-03-2024 Non Nouveau
Showing 1 to 4 of 4 entries (filtered from 9 total entries)
Previous 1 Next
Créer une commande


Accueil
Listes des commandes
Historique
Administration
Import
Liens URL
v 1.1.1
Commandes
Listes commandes
Commande Ameli Pro
Dépannage
Direct
Show 10 entries
Search:
Identification
Numero Ps
Source
Imprimé
Quantite commandée
Date
Pré-identification
Statut
Actions
JNSPGIJGZ.NATHALIE 871742069 Commande Ameli Pro S3745a 20 06-01-2024 Non En cours
VXZZLNR.PIERRE ALAIN 871307122 Commande Ameli Pro S3138 100 07-01-2024 Non En cours
ZJPJTTZTD.ANAIS 871320235 Commande Ameli Pro S4110 100 05-01-2024 Non En cours
Showing 1 to 3 of 3 entries (filtered from 8 total entries)
Previous 1 Next
Créer une commande

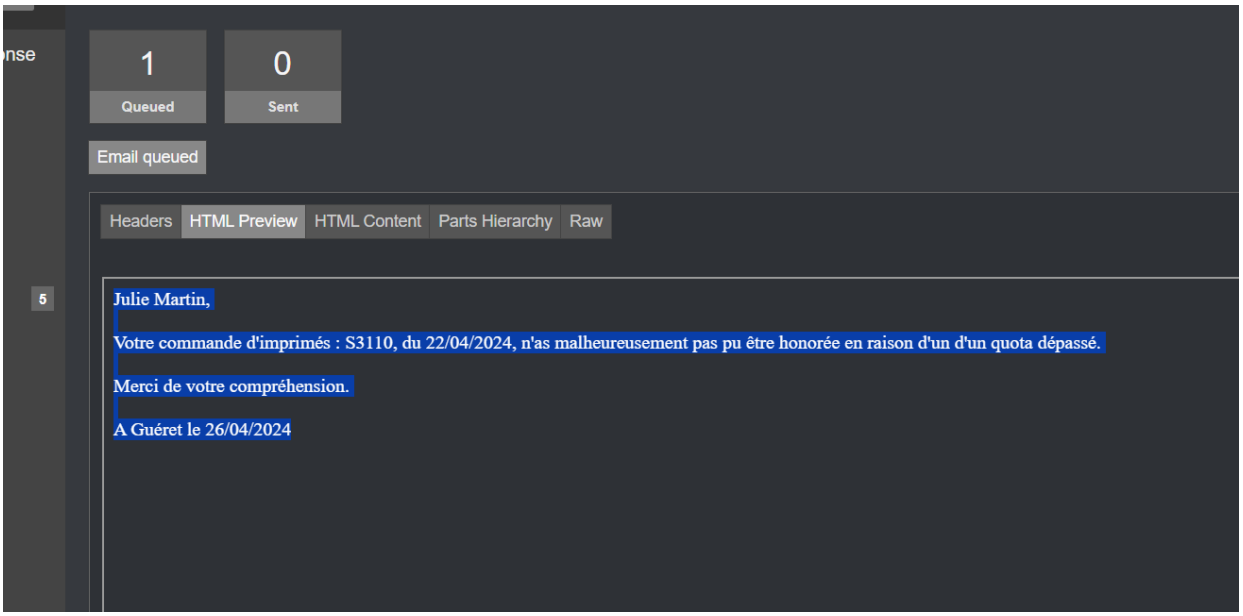
Bouton « Envoi de mail »



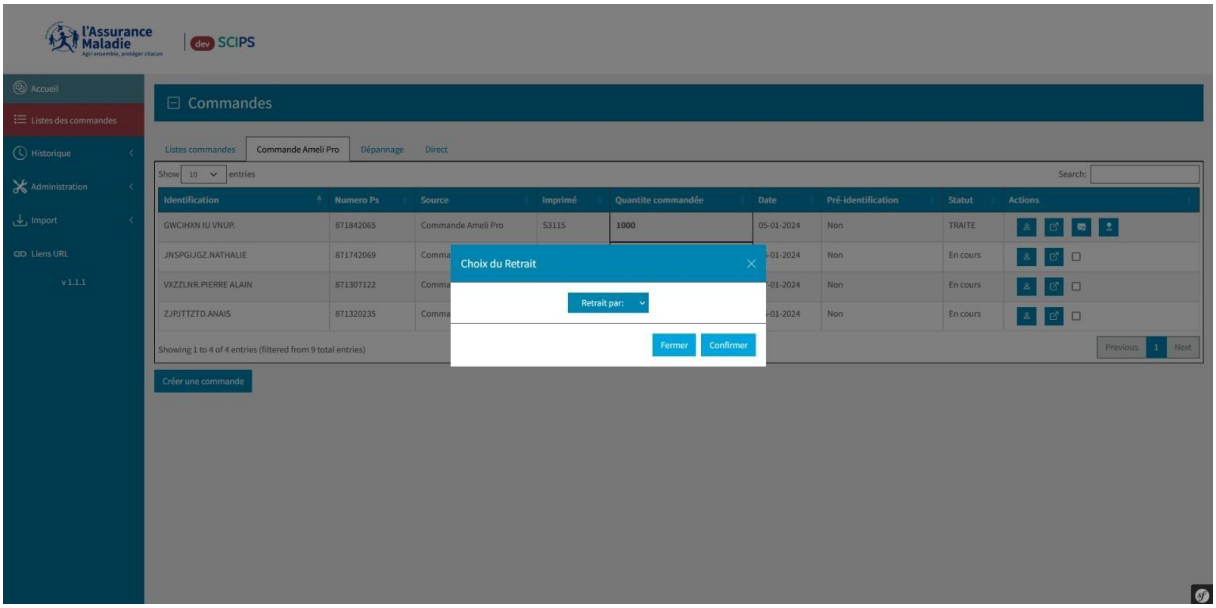
Ouvre une fenêtre modale Bootstrap où le l'utilisateur peut choisir un modèle de mail prédéfini, il peut également en modifier le contenu et l'envoyer en cliquant sur le bouton « Envoyer ».



Exemple d’un courrier lors de son envoi :



Bouton retrait :



Affiche une fenêtre modale où le client choisit le canal de retrait. Cette information est enregistrée dans la base de données avec la date du retrait.

Bouton lien :

Lorsqu'il est cliqué, ajoute un onglet où le site du fournisseur de l'imprimé associé à la commande est affiché.

Bouton "Éditer" :

- Permet d'éditer les commandes directes et de dépannage.

Colonne des quantités commandées :

- Permet de modifier les quantités directement dans la colonne.

Bouton "Historique" :

- Affiche l'historique des commandes sur une période d'un an.

Code :

Partie statut « checkbox »

Dans cette partie du projet, l'objectif est de permettre aux utilisateurs de modifier le statut d'une commande en utilisant des cases à cocher. Voici comment cela fonctionne :

Twig :

Dans le fichier Twig, deux blocs conditionnels sont utilisés pour afficher les cases à cocher en fonction du statut actuel de la commande :

Si le statut de la commande est 'TRAITE' et la source de la commande n'est pas 'Commande Ameli Pro', une case à cocher est affichée avec le libellé "Statut est prête".

Si le statut de la commande n'est pas 'TRAITE', une autre case à cocher est affichée avec le libellé "Statut est traité".

```
{% if commande.idStatut.libelle=='TRAITE' and commande.idSource.libelle != 'Commande
Ameli Pro'%}
```

```
  <div class="btn-text-container btn-text-container-statut" id="btnTextContainer">
        <input type="checkbox" class="btn change-status" data-commande-id="{{ commande.id }}" data-statut-id="{{ commande.idStatut.id }}" data-source-id="{{ commande.idSource.id }}">
        <span class="btn-text">Statut est prête</span>
   </div>
{% endif %}
{% if commande.idStatut.libelle!='TRAITE'  %}
   <div class="btn-text-container btn-text-container-statut" id="btnTextContainer">
        <input type="checkbox"  class="btn change-status" data-commande-id="{{ commande.id }}" data-statut-id="{{ commande.idStatut.id }}" data-source-id="{{ commande.idSource.id }}">
        <span class="btn-text">Statut est traité </span>
   </div> {% endif %}
```

JavaScript :

Lorsqu'une case à cocher change d'état, une fonction est déclenchée.

Cette fonction récupère l'ID du statut actuel de la commande et détermine le nouveau statut en fonction de l'état de la case et du statut actuel.

Ensuite, une requête AJAX est envoyée au serveur avec les informations nécessaires pour mettre à jour le statut de la commande.

En fonction de la réponse de la requête, la page est rechargée pour refléter les changements.

```
//Parite statut
      $(document).ready(function() {
          $('.change-status').change(function() {              var isChecked =
$(this).is(':checked');              var statutId = $(this).data('statut-id');
var newStatutId;
            if (isChecked) {                if (statutId === 1
) {
                  newStatutId = 4;

              } else if (statutId === 3 || statutId === 2) {
                  newStatutId = 1;



            }
```

```
            } else {
                // Définir une autre valeur par défaut si besoin                 newStatutId = 0;
            }                 var commandeId = $(this).data('commande-id');
var idSource= $(this).data('source-id');                 if (idSource ===
'Commande Ameli Pro') {
                $('.btn-envoi').show(); // Afficher le bouton d'envoi de mail
                $('.btn-confirm-retrait-par').show(); // Afficher le bouton de
retrait par
            }
            var actionUrl = "/updateStatut/" + commandeId + "/" + statutId;
            fetch(actionUrl, {
method: 'POST',                 headers: {
                    'Content-Type': 'application/json'
                },                 body: JSON.stringify({
id_statut: newStatutId
                })
            })
                .then(response => {                 if (response.ok) {
console.log('Statut mis à jour avec succès');                 window.location.reload();
                } else {                 console.error('Erreur lors de la mise à jour du statut');
                }
                })
                .catch(error => {
                    console.error('Erreur lors de la mise à jour du statut:',
error);
                });
        });
    });
```

Ce code gère la logique du changement de statut en fonction de l'état de la case à cocher. Il envoie également une requête POST à l'URL de l'action du contrôleur pour mettre à jour le statut de la commande.

Action du contrôleur : L'action du contrôleur associée à la modification du statut d'une commande est accessible via la route /commande/updateStatut/{id}/{id_statut}. Cette action attend les paramètres de l'ID de la commande et du nouvel ID du statut à mettre à jour.

Elle traite la requête JSON envoyée par le client pour extraire les informations nécessaires, telles que le nouvel ID du statut.

Ensuite, elle met à jour le statut de la commande en fonction des données fournies dans la requête.

```
/**
 * @Route("/commande/updateStatut/{id}/{id_statut}", name="update_statut", methods={"POST"})
 */
public function updateStatut($id, $id_statut, Request $request, CommandeRepository $commandeRepository, StatutRepository $statutRepository): Response
{
    $content = $request->getContent();
    $decoded = json_decode($content, true);

    // Vérifier si 'id_statut' existe dans les données JSON décodées
if (isset($decoded['id_statut'])) {
        $newStatutId = $decoded['id_statut'];
        $statut = $statutRepository->find($newStatutId);
        $commande = $commandeRepository->find($id);

        if ($statut && $commande) {
            $commande->setIdStatut($statut);
            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->persist($commande);
            $entityManager->flush();
            return new JsonResponse(['success' => 'Statut mis à jour avec succès'], Response::HTTP_OK);
        } else {
            return new JsonResponse(['error' => 'Statut ou commande introuvable'], Response::HTTP_NOT_FOUND);
        }
    } else {
```

```
        return new JsonResponse(['error' => 'Paramètre id_statut manquant'],
Response::HTTP_BAD_REQUEST);
    }
}
```

Conclusion

Cette fonctionnalité permet une gestion interactive et en temps réel des statuts des commandes, offrant aux utilisateurs une manière efficace de mettre à jour les statuts de manière intuitive. Elle repose sur une interaction fluide entre la vue Twig, le JavaScript et l'action du contrôleur pour garantir des mises à jour rapides et précises des données.

Partie mail

Twig : La vue Twig affiche un lien qui, lorsqu'il est cliqué, ouvre un formulaire modal permettant à l'utilisateur de choisir un modèle d'e-mail et de rédiger le contenu de l'e-mail.

Formulaire Modal

Le formulaire modal contient les champs suivants :

Un menu déroulant pour choisir un modèle d'e-mail prédéfini.

Un champ de texte pour saisir le contenu de l'e-mail personnalisé.

```
 {#  <div class="btn-text-container">
                    <a  href="{{ path('envoyer_email', {'statut': commande.idStatut.id, 'ps': commande.id
})}}" id="btn-envoyer-email" class="btn btn-primary btn-sm">
                       <i class="bi bi-envelope-at-fill btn-primary"></i>
                       <span class="btn-text">Envoyer un e-mail</span>
                    </a>
                 </div>#}
```

Quand un RPS clique envoyer on affiche ce formulaire :

```
 <form  method="post" id="emailForma" class="emailForma">
       <div class="modal fade" id="emailForm" tabindex="-1" aria-labelledby="exampleModalLabel"
aria-hidden="true">
          <div class="modal-dialog" style=" position: relative; top: 35%;">
```

```
<div class="modal-content" style="z-index: 1000;">
  <!-- Contenu de votre div modale -->
  <div class="modal-header">
    <h5 class="modal-title" id="exampleModalLabel">Mail</h5>
    <button type="button" class="btn-close" data-bs-dismiss="modal"
arialabel="Close"></button>
  </div>
  <div class="modal-body">
    <!-- Formulaire pour choisir et modifier le modèle d'e-mail -->
    <div class="mb-3">
      <label for="select-model" class="form-label">Choisir le modèle d'e-mail</label>
      <select class="  btn btn-primary" id="select-model" name="intitule_email">
        <option value="">Choisissez:</option>
        {% for model in models %}
          <option value="{{ model.id }}" data-model-content="{{ model.corps }}">{{
model.intitule }}</option>
        {% endfor %}
      </select>
      <div id="modeleEmailHelp" class="form-text">Sélectionnez le modèle d'e-mail à
envoyer.</div>
    </div>
    <div class=" mb-3">
      <label for="exampleFormControlTextarea1" class="form-label">Contenu de l'e-mail
</label>
      <textarea name="contenu_email" class="form-control"
id="exampleFormControlTextarea1" rows="3" aria-describedby="contenuEmailHelp"></textarea>
    </div>
  </div>
  <div class="modal-footer">
    <button type="button" class="btn btn-secondary" data-
bsdismiss="modal">Fermer</button>
    <button type="submit" class="btn btn-secondary btn-send-email"
id="submitEmailForm" data-bs-dismiss="modal">
      <i class="bi bi-envelope-at-fill btn-primary"></i>
```

```
                    <span class="btn-text">Envoyer un e-mail</span>
                </button>
            </div>
        </div>
    </div>
</form>
```

JavaScript : Le JavaScript associé au formulaire modal traite l'envoi des données du formulaire au serveur :

Il écoute le clic sur le bouton d'envoi de l'e-mail dans le formulaire modal.

Il récupère les données du formulaire, notamment le contenu de l'e-mail et l'intitulé du modèle d'e-mail.

Il envoie ces données au serveur via une requête AJAX POST.

Il traite la réponse du serveur, affichant les éventuelles erreurs ou confirmations de succès dans la console du navigateur.

```
document.addEventListener("DOMContentLoaded", function()
{$('#submitEmailForm').click(function() { event.preventDefault(); // Empêche le
comportement par défaut du formulaire
// Sélection du formulaire let forma =document.querySelector(".emailForma");
let fd = new FormData(); var contenuEmail =
document.querySelector('[name="contenu_email"]').value; var intituleEmail =
document.querySelector('[name="intitule_email"]').value;
console.log(intituleEmail); var commandeId = $('.btn-envoi').data('commande-
id'); var commandeIdPs = $('.btn-envoi').data('ps-id');
var actionUrl = "{{ path('envoyer_email', {'id': 'REPLACE_WITH_CM_ID','ps': 'REPLACE_WITH_PS_ID'})
}}";
        actionUrl = actionUrl.replace('REPLACE_WITH_PS_ID', commandeIdPs);
actionUrl = actionUrl.replace('REPLACE_WITH_CM_ID', commandeId);
fd.append("contenu_email",contenuEmail);
```

```
fd.append("intituleEmail",intituleEmail);              // Envoi des données au
serveur
            fetch(actionUrl, {
method: 'POST',
headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
contenu_email: contenuEmail,
intitule_email: intituleEmail,
ps:commandeIdPs,                    id:commandeId
            })
        })
            .then(response => {
                // Gérer la réponse du serveur
console.log('Réponse du serveur:', response);
                // Vous pouvez ajouter ici la logique pour gérer la réponse du serveur
            })
            .catch(error => {
                console.error('Erreur lors de l\'envoi des données au serveur:', error);
            });
    });
});
```

Action du Controller :

Elle récupère les données envoyées par le client, telles que le contenu de l'e-mail et l'ID du modèle d'e-mail.

Elle remplace les variables {num} et {identi} dans le contenu de l'e-mail par l'ID de la commande et l'identifiant du client, respectivement.

Elle récupère l'adresse e-mail de destination et le sujet de l'e-mail.

Elle envoie l'e-mail en utilisant le service de messagerie fourni par Symfony.

```
/**
    * @Route("/envoyer-email/{id}/{ps}", name="envoyer_email", methods={"POST"}))
    */
   public function sendEmailAction($ps,$id, MailerInterface $mailer, Request
$request,CommandeRepository $commandeRepository,ModeleMailRepository
$ModeleMailRepository): Response
   {
      $content = trim(file_get_contents("php://input"));
      $content = $request->getContent();
      $user=$commandeRepository->findOneBy(['id' =>$id ]);
      $id=$user->getId();
      $nom=$user->getIdPs()->getNom();
      $prenom=$user->getIdPs()->getPrenom();
      $ps=$nom.$prenom;
      $decoded = json_decode($content, true);
var_dump($decoded);
      $contenuEmail = $request->request->get('contenu_email');
      $intituleEmail = $request->request->get('intitule_email');
$modelmail=$ModeleMailRepository->findOneBy(['id'=> $decoded['intitule_email']]);


          var_dump( $decoded['contenu_email']);


         var_dump($decoded['intitule_email']);
         $contenuEmail = $decoded['contenu_email'];
         $contenuEmail = str_replace('{num}', $id, $contenuEmail);
         $contenuEmail = str_replace('{identi}', $ps, $contenuEmail);
         $intituleEmail =$modelmail->getIntitule();
         $email = "saad.idrharnane-ext@assurance-maladie.fr";



         $message = (new Email())
```

```
            ->from(new Address('ne-pas-repondre@assurance-maladie.fr', 'SCIPS'))
            ->to($email)
            ->subject($intituleEmail)
            ->html(nl2br($contenuEmail));

        $mailer->send($message);

    }
}
```

Conclusion

Cette fonctionnalité offre aux utilisateurs la possibilité d'envoyer des e-mails associés à des commandes de manière efficace et personnalisée. Elle s'appuie sur une interface utilisateur conviviale, un traitement JavaScript fluide et une logique de contrôleur robuste pour garantir la livraison des e-mails aux destinataires appropriés.

Partie Button retrait :

La partie "Button Retrait" permet à l'utilisateur de confirmer le retrait d'une commande en choisissant le canal de retrait approprié. Lorsque l'utilisateur clique sur le bouton de confirmation, un formulaire modal s'affiche, lui permettant de sélectionner le canal de retrait.

Twig : La vue Twig affiche le bouton de retrait seulement si la commande provient de "Commande Ameli Pro" et si son statut n'est pas "En cours". Le bouton est associé à un formulaire modal permettant de sélectionner le canal de retrait.

Formulaire Modal

Le formulaire modal contient un menu déroulant pour sélectionner le canal de retrait. L'utilisateur peut choisir parmi une liste de canaux de retrait disponibles.

```
        {% if commande.idSource.libelle == 'Commande Ameli Pro' and not (commande.idSource.libelle == 'Commande Ameli Pro' and commande.idStatut.libelle == 'En cours') %}
                        <div class="btn-text-container">
                            <a data-bs-toggle="modal" href="#retraitParModal" class="btn btn-primary btnsm btn-se btn-confirm-retrait-par" data-commande-id="{{ commande.id }}">
                                <i class="bi bi-geo-fill"></i>
                                <span class="btn-text">Retrait par</span>
                            </a>
                        </div>
```

```
                {% endif %}

  <form id="retraitParForm{{ commande.id }}" action="{{ path('app_commande_confirm_retrait', {'id': commande.id })}}" method="post">

            <div class="modal fade" id="retraitParModal" tabindex="-1" arialabelledby="exampleModalLabel" aria-hidden="true">

              <div class="modal-dialog modal-dialog-centered">

                <div class="modal-content">

                  <!-- Contenu du modal -->

                  <div class="modal-header">

                    <h5 class="modal-title" id="exampleModalLabel">Choix du Retrait</h5>

                    <button type="button" class="btn-close" data-bs-dismiss="modal" arialabel="Close"></button>

                  </div>

                  <div class="modal-body d-flex justify-content-center  ">

                    <select class="btn btn-primary select-canal" id="select-canal">

                      <option value="">Retrait par:</option>

                      {%  for canal in canals %}


                        <option value="{{ canal.id }}">{{ canal.libelle }}</option>


                      {% endfor %}


                    </select>

                  </div>

                  <div class="modal-footer">

                    <button type="button" class="btn btn-secondary" data-bsdismiss="modal">Fermer</button>

                    <button type="button" class="btn btn-secondary   btn-confirm-retrait-par" data-commande-id="{{ commande.id }}"  data-statut-id="{{ commande.idStatut.id }} data-sourceid="{{ commande.idSource.id }}">

                      Confirmer

                    </button>


                  </div>
```

```
                    </div>
                </div>
            </div>
</form>
```

JavaScript : Il écoute le clic sur le bouton de confirmation de retrait.

Il récupère l'identifiant de la commande associée au bouton de confirmation.

Il récupère l'identifiant du canal de retrait sélectionné par l'utilisateur.

Il envoie ces données au serveur via une requête AJAX POST.

Il cache le formulaire modal une fois que la requête est réussie.

```
$(document).ready(function() {

        // Gestion du clic sur le bouton de confirmation de retrait par
        $('.btn-confirm-retrait-par').click(function() {
            // Récupérer l'identifiant de la commande associée à ce bouton de confirmation
var commandeId = $(this).data('commande-id');
            var statutId = $(this).data('statut-id');
var sourceId = $(this).data('source-id');

            // Récupérer la valeur sélectionnée du canal de retrait spécifique à cette commande
var selectedCanalId = $('#select-canal').val();              console.log(selectedCanalId);

            // Effectuer une requête AJAX pour envoyer la valeur au serveur
            $.ajax({
type: 'POST',
                url: '{{ path("app_commande_confirm_retrait", {'id': 0}) }}'.replace('0', commandeId),
data: { canal_id: selectedCanalId,                commandeId:commandeId},                success:
function(response) {
```

```
                // Cacher le modal une fois que la requête est réussie
                $('#retraitParModal-' + commandeId).modal('hide');
            },
            error: function(xhr, status, error) {
console.error(error);
            }
        });
        $('#retraitParModal').modal('hide');
    });

    // Gestion du clic sur le bouton de confirmation global
    $('#btn-confirm-retrait-par').click(function() {
        // Effectuez les actions appropriées lorsque le mode de retrait par est confirmé
window.location.reload();
        // Fermez le modal une fois que l'action est terminée
        $('#retraitParModal').modal('hide');
    });


});
```

Action contrôleur :

Elle récupère les données envoyées par le client, notamment l'ID du canal de retrait et l'ID de la commande.

Elle met à jour les informations de la commande dans la base de données, en ajoutant le canal de retrait sélectionné et la date du retrait.

Elle met à jour le statut de la commande si la statut de command est traité et de source Ameli pro.

```
/**
 * @Route("/commande/{id}/confirm-retrait", name="app_commande_confirm_retrait", methods={"POST"})
 */
public function confirmRetrait(Request $request, Commande $commande, CanalRepository $canalRepository , StatutRepository $statutRepository,SourceRepository $sourceRepository, CommandeRepository $commandeRepository): JsonResponse
{
```

```
        $dateActuelle = new DateTime();
        $canalId = $request->request->get('canal_id'); // Assurez-vous que la clé correspond à celle envoyée depuis la vue
$commandeId=$request->request->get('commandeId');
        // Mettre à jour la colonne lieuxRet dans la base de données pour la commande actuelle
        $canalRet = $canalRepository->findOneBy(['id' => $canalId]);
        $commande=$commandeRepository->findOneBy(['id'=>$commande]);
        // Ensure that Commande object exists
if ($commande !== null) {
            // Get the ID of the Statut and Source
            $desiredStatutId = 1; // Update this with the desired Statut ID
            $desiredSourceId = 1; // Update this with the desired Source ID

// Retrieve the ID of the Statut entity associated with the Commande
            $commandeStatutId = $commande->getIdStatut()->getId(); // Assuming getId() returns the ID

// Retrieve the ID of the Source entity associated with the Commande
            $commandeSourceId = $commande->getIdSource()->getId(); // Assuming getId() returns the ID

// Compare the IDs of Statut and Source entities            if ($commandeStatutId == $desiredStatutId
&& $commandeSourceId == $desiredSourceId) {                // Assuming the ID of the desired Statut
entity you want to set                $desiredStatutId = 4;

                // Retrieve the Statut object based on its ID
                $statut = $statutRepository->find($desiredStatutId);

                // If Statut object is found, set it to the Commande object
if ($statut !== null) {
                    $commande->setIdStatut($statut);
                } else {
                    // Handle the case where Statut object with the given ID is not found
throw new \Exception('Statut with ID ' . $desiredStatutId . ' not found.');
```

```
            }
        }

        // Set CanalRet and DateRet for the Commande object
        $commande->setCanalRet($canalRet);
        $commande->setDateRet($dateActuelle);

        // Flush changes to the database
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->flush();

        return new JsonResponse(['success' => true]);
    } else {
        // Handle the case where Commande object with the given ID is not found
throw new \Exception('Commande with ID ' . $commandeId . ' not found.');
    }
  }
```

Partie modification de quantité commandé :

La fonctionnalité de modification de la quantité commandée permet aux utilisateurs de mettre à jour la quantité commandée pour une commande spécifique. Les utilisateurs peuvent double-cliquer sur la quantité affichée, modifier la valeur et appuyer sur la touche "Entrée" pour enregistrer les modifications.

Twig : La quantité commandée est affichée dans une cellule de tableau avec l'attribut **`contenteditable="true"`**, ce qui permet à l'utilisateur de modifier directement la valeur.

```
<td class="editable  editable-cell  " contenteditable="true" data-id="{{ commande.id }}">
{{ commande.quantiteCommandee }}</td>
```

JavaScript

L'utilisateur double-clique sur la quantité commandée à modifier.

Il modifie la valeur directement dans la cellule du tableau.

L'utilisateur appuie sur la touche "Entrée" pour valider les modifications.

Une boîte de dialogue de confirmation s'affiche pour confirmer l'enregistrement des modifications.

Les modifications sont envoyées au serveur via une requête AJAX.

Le contrôleur Symfony traite la requête, met à jour la quantité commandée dans la base de données et renvoie une réponse.

```
$(document).ready(function(){
        $('.editable')
            .dblclick(function(){
                $(this).focus();
            })
            .keydown(function(event){                if (event.keyCode == 13) {
// 13 correspond à la touche "Entrer"
event.preventDefault(); // Empêcher le saut de ligne                    var
currentCell = $(this);                    var newValue = currentCell.text().trim();

                    // Afficher la boîte de dialogue de confirmation                    if
(confirm("Êtes-vous sûr de vouloir enregistrer les modifications ?")) {
currentCell.blur(); // Perdre le focus
                    } else {
                        // Restaurer la valeur précédente si l'utilisateur annule
currentCell.text(currentCell.data('originalValue'));
                    }
                }
            })
            .blur(function(){
                var quantiteCommandee = $(this).text().trim();
var idCommande = $(this).data('id');

                // Envoi des données au serveur via AJAX
                $.ajax({
url: '/enregi' +
```

```
'strer-quantite',
method: 'POST',
                data: {
id: idCommande,
                    quantiteCommandee: quantiteCommandee
                },
                success: function(response) {
                    console.log(response);
                },
                error: function(xhr, status, error) {
console.error(error);
                }
            });
        });
    });
```

Action contrôleur

```
 /**
    * @Route("/enregistrer-quantite", name="enregistrer_quantite", methods={"POST"})
    */
   public function enregistrerQuantite(Request $request, CommandeRepository
$commandeRepository): Response
   {
      $idCommande = $request->request->get('id');
      $quantiteCommandee = $request->request->get('quantiteCommandee');

      $entityManager = $this->getDoctrine()->getManager();
      $commande = $commandeRepository->findOneBy(['id' => $idCommande]);

      if (!$commande) {          throw $this-
>createNotFoundException(
          'Aucune commande trouvée pour l\'identifiant '.$idCommande
```

```
        );
    }

    // Mettre à jour la quantité commandée
    $commande->setQuantiteCommandee($quantiteCommandee);
    $entityManager->persist($commande);
    $entityManager->flush();

    return new Response('Quantité mise à jour avec succès');
}
```

Partie Button fournisseur :

Twig : La partie "Button Fournisseur" permet à l'utilisateur d'accéder au site web du fournisseur associé à une commande spécifique. Lorsque l'utilisateur clique sur le bouton "Fournisseur", il est redirigé vers la page web du fournisseur dans un nouvel onglet ou une nouvelle fenêtre, en fonction de l'attribut **target="_blank"**

```
<div class="btn-text-container">
                    <a  target="_blank" href="{{ commande.idImprime.fournisseurPrimaire.url }}" class="btn btn-primary btn-sm">
                        <i class="bi bi-box-arrow-up-right"></i>
                        <span class="btn-text">Fournisseur</span>
                    </a>
                </div>
```

Partie création d'une commande

La fonctionnalité de création d'une commande permet aux utilisateurs d'initier une nouvelle commande en fournissant les informations requises via un formulaire. Toutes les commandes créées ont par défaut le statut "Nouvelle". Twig :

Un lien "Créer une commande" redirige les utilisateurs vers la route **app_commande_new**, où le formulaire de création de commande est affiché.

```
<a href="{{ path('app_commande_new') }}"  class="btn btn-primary">Créer une commande</a>
```

Action contrôleur

```
 /**
    * @Route("/new", name="app_commande_new", methods={"GET", "POST"})
    */
   public function new(Request $request, CommandeRepository $commandeRepository,
StatutRepository $statutRepository): Response
   {
      $commande = new Commande();

      $commande->setDate( new \DateTime());

      $statut = $statutRepository->find(3);
      $commande->setIdStatut($statut);
      $commande->setQuantiteCommandee(0);
      $form = $this->createForm(CommandeType::class, $commande);        $form-
>handleRequest($request);

      if ($form->isSubmitted() && $form->isValid()) {

         $idPs = $commande->getIdPs()->getId();
         $idCo = $idPs.time();
         $commande->setId($idCo);
         $commandeRepository->add($commande, true);

         return $this->redirectToRoute('app_commande_index', [], Response::HTTP_SEE_OTHER);
      }

      return $this->renderForm('commande/new.html.twig', [
         'commande' => $commande,
         'form' => $form,
      ]);
   }
```

Pour la partie design du tableau datatable()) :

- La conception utilise DataTables pour rendre le tableau interactif et permettre le filtrage des données.
- L'initialisation de DataTables est effectuée dans le script JavaScript, avec des options de personnalisation et un fichier de langue français.
- Les boutons d'onglets permettent de filtrer les données en fonction des catégories spécifiques.
- La fonction **filterCommands()** est responsable de filtrer les données en fonction de l'onglet sélectionné.
- Cette implémentation améliore l'expérience utilisateur en permettant une navigation aisée dans les données de manière interactive.

Twig

```
<table class="table table-bordered display table-hover table-striped" id="tableCommandes">
    <thead >
    <tr >
      <th class="text-light">Identification</th>
      <th class="text-light">Numero Ps</th>
      <th class="text-light">Source</th>
      <th class="text-light">Imprimé</th>
      <th class="text-light">Quantite commandée</th>

      <th class="text-light">Date </th>
      <th class="text-light">Pré-identification</th>
      <th class="text-light">Statut</th>

      <th class="text-light">Actions</th>

    </tr>
    </thead>
    <tbody>
```

```
<ul class="nav nav-tabs" id="myTab" role="tablist">
  <li class="nav-item" role="presentation">
    <button class="nav-link active" id="home-tab" data-bs-toggle="tab" data-bs-target="#home" type="button" role="tab" aria-controls="home" aria-selected="true">Listes commandes</button>
  </li>
  <li class="nav-item" role="presentation">
    <button class="nav-link" id="profile-tab" data-bs-toggle="tab" data-bs-target="#profile" type="button" role="tab" aria-controls="profile" aria-selected="false">Commande Ameli Pro</button>
  </li>
  <li class="nav-item" role="presentation">
    <button class="nav-link" id="contact-tab" data-bs-toggle="tab" data-bs-target="#contact" type="button" role="tab" aria-controls="contact" aria-selected="false">Dépannage</button>
  </li>
  <li class="nav-item" role="presentation">
    <button class="nav-link" id="direct-tab" data-bs-toggle="tab" data-bs-target="#direct" type="button" role="tab" aria-controls="direct" aria-selected="false">Direct</button>
  </li>
</ul>
<div class="tab-content" id="myTabContent">
  <div class="tab-pane fade show active" id="home" role="tabpanel" aria-labelledby="hometab"></div>
  <div class="tab-pane fade" id="profile" role="tabpanel" aria-labelledby="profile-tab"></div>
  <div class="tab-pane fade" id="contact" role="tabpanel" aria-labelledby="contact-tab"></div>
  <div class="tab-pane fade" id="direct" role="tabpanel" aria-labelledby="direct-tab"></div>
</div>

{% for commande in commandes %}
  {% if commande.idPs and commande.idImprime %}
```

JavaScript

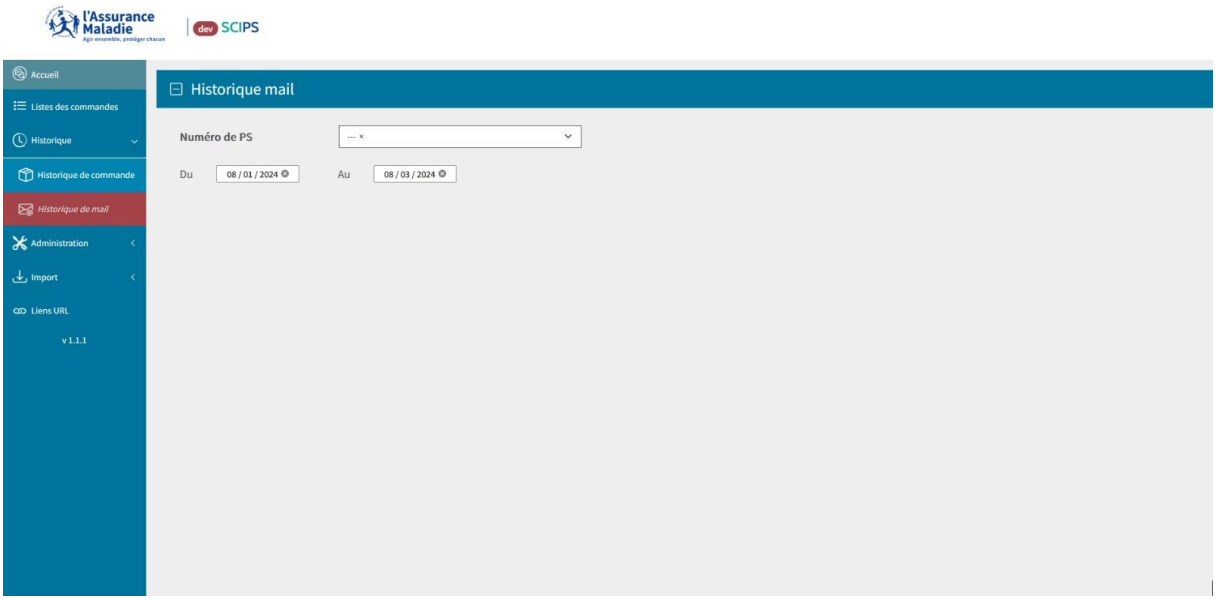
```
document.addEventListener('DOMContentLoaded', function() {
```

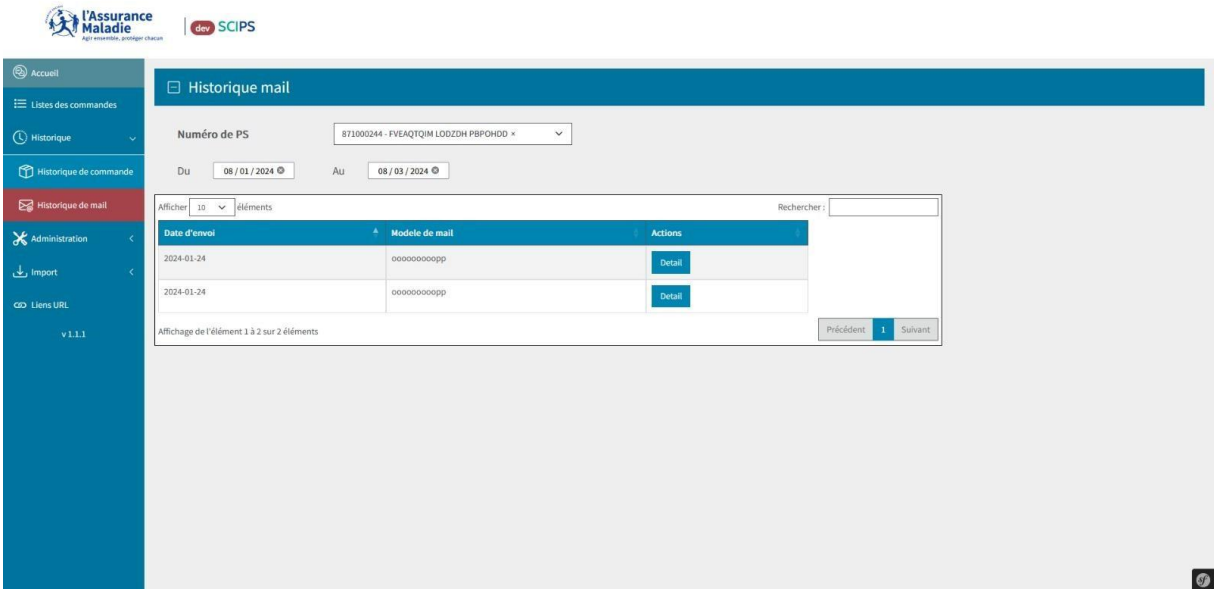
```
   // Initialisation de DataTables avec les options et le fichier de langue français
let table = $('#tableCommandes').DataTable({
      "language": {            url:
'dataTables.french.json',        }
   });

   // Sélection des boutons d'onglets     const tabButtons =
document.querySelectorAll('.nav-link');

   // Ajout d'écouteurs d'événements aux boutons d'onglets
tabButtons.forEach(button => {        button.addEventListener('click',
function() {           // Récupération de l'ID de l'onglet cible
const targetTabId = button.getAttribute('aria-controls');           //
Filtrage des commandes en fonction de l'onglet sélectionné
filterCommands(targetTabId, table);
      });
   });

   // Fonction pour filtrer les commandes en fonction de l'onglet sélectionné
function filterCommands(tabId, table) {
      switch (tabId) {
case 'home':
            // Filtrer les données de la colonne 2 avec une chaîne vide et redessiner la table
table.column(2).search("").draw();
            break;
case 'profile':
            // Filtrer les données de la colonne 2 avec "Commande Ameli Pro" et redessiner la table
table.column(2).search("Commande Ameli Pro").draw();
            break;
case 'contact':
```

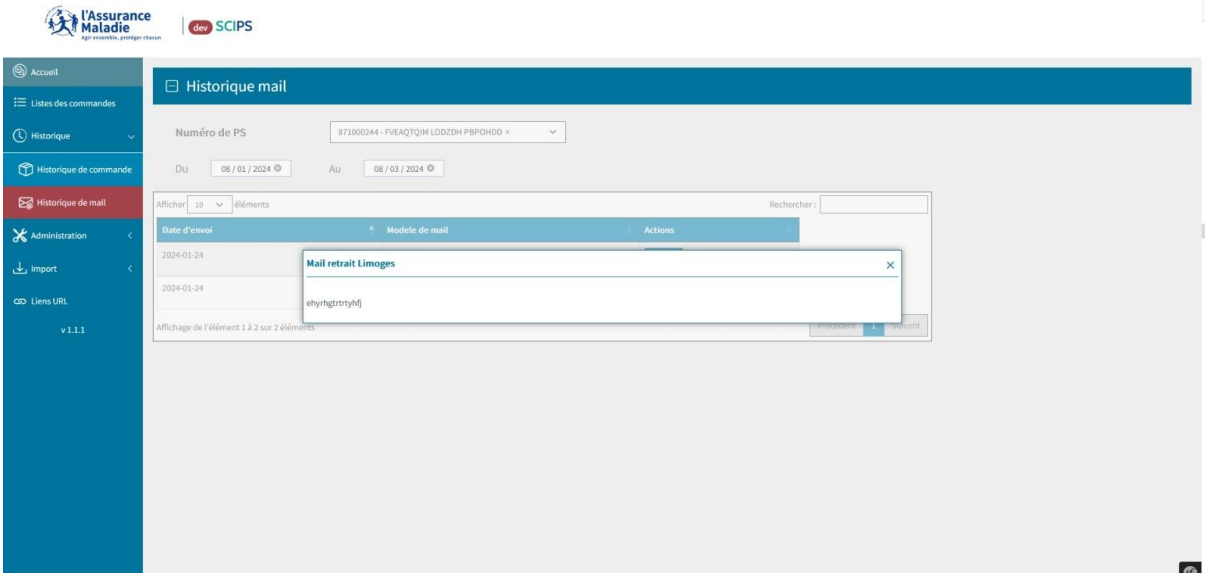
```
            // Filtrer les données de la colonne 2 avec "Dépannage" et redessiner la table
table.column(2).search("Dépannage").draw();
            break;
        case 'direct':
            // Filtrer les données de la colonne 2 avec "Direct" et redessiner la table
table.column(2).search("Direct").draw();
            break;
    }
  }
});
```

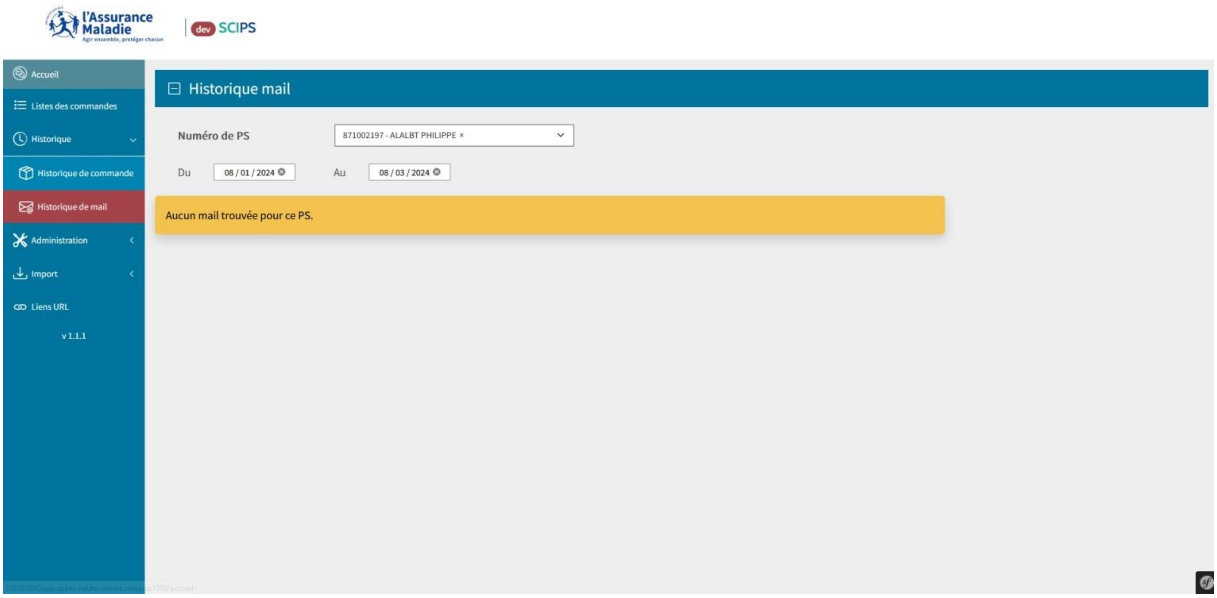
Historique :

Historique mail :




l'Assurance Maladie
dev SCIPS
Accueil
Listes des commandes
Historique
Historique de commande
Historique de mail
Administration
Import
Liens URL
v 1.1.1
Historique mail
Numéro de PS
871000244 - FVEAQTQIM LODZDH PBPOHDD ×
Du
08 / 01 / 2024
Au
08 / 03 / 2024
Afficher 10 éléments
Rechercher :
Date d'envoi
Modele de mail
Actions
2024-01-24
oooooooopp
Detail
2024-01-24
oooooooopp
Detail
Affichage de l'élément 1 à 2 sur 2 éléments
Précédent
1
Suivant


l'Assurance Maladie
dev SCIPS
Accueil
Listes des commandes
Historique
Historique de commande
Historique de mail
Administration
Import
Liens URL
v 1.1.1
Historique mail
Numéro de PS
871000244 - FVEAQTQIM LODZDH PBPOHDD ×
Du
08 / 01 / 2024
Au
08 / 03 / 2024
Afficher 10 éléments
Rechercher :
Date d'envoi
Modele de mail
Actions
2024-01-24
2024-01-24
Mail retrait Limoges
ehyrhgtrtrtyhfj
Affichage de l'élément 1 à 2 sur 2 éléments


l'Assurance Maladie
dev SCIPS
Accueil
Listes des commandes
Historique
Historique de commande
Historique de mail
Administration
Import
Liens URL
v 1.1.1
Historique mail
Numéro de PS
871002197 - ALALBT PHILIPPE ×
Du
08 / 01 / 2024
Au
08 / 03 / 2024
Aucun mail trouvée pour ce PS.

Dans la page d'historique des e-mails, nous présentons une liste des numéros PS avec une sélection comprenant le numéro PS, le nom et le prénom. Par défaut, les dates sont affichées pour les deux derniers mois, mais le représentant du service client peut les modifier selon ses besoins.

Lorsque l'utilisateur choisit un numéro PS, un tableau s'affiche avec la date d'envoi du mail, le modèle de mail, et une action associée à un bouton "Détails". En cliquant sur ce bouton, le contenu du mail envoyé est affiché.

Si aucun mail n'est trouvé pour le PS sélectionné, un message est affiché pour informer l'utilisateur de cette situation. Cela garantit une transparence et une clarté quant à l'absence de correspondance.

Code :

Twig :

1. **Formulaire de Sélection**:
    - La vue contient un formulaire avec un sélecteur pour choisir un numéro de PS.
    - Deux champs de date permettent de spécifier la plage de dates pour la recherche.
    - Un conteneur est prévu pour afficher les résultats de la recherche.
2. **Conteneur de Détails**:
    - Un conteneur est prévu pour afficher les détails du mail sélectionné.
    - Il est caché par défaut et s'affiche lorsqu'un bouton "Détails" est cliqué. {% block content %}

```
<div class="col-lg-9">
  <div id="overlay">
    <form id="histCommande" method="post" class="forma">
      <div class="row ms-md-4 align-items-md-center">
        <label for="saad" class="col-md-4 col-form-label fs-5 fw-semibold">Numéro de PS</label>
          <select class="tom-select saad col-md-6 ms-md-2" onchange="executePromise()" required>
            <option  selected="selected" class="fw-bold">---</option>
```

```
                        {% for p in ps %}
                            <option value="{{ p.id }}">{{ p.id }} - {{ p.nom }} {{ p.prenom }}</option>
                        {% endfor %}
                    </select>
                </div>

                <div class="row my-md-4 ms-4 align-items-md-center">
                    <label class="col-md-1 w-auto mb-0 fs-5 ">Du</label>
                    <input type="date" id="start-date" name="start-date" class="col-md-2 w-auto ms-4">
                    <label class="col-md-1 w-auto ms-lg-5 mb-0 fs-5 d-md-inline-flex ">Au</label>
                    <input type="date" id="end-date" name="end-date" class="col-md-3 w-auto ms-4">
                </div>

            </form>
            <div id="commandes-container" class="mt-md-4 rounded"></div>
        </div>
    </div>

<div id="details-container" class="details-popup">
    <div class="flexissue">
        <h2 style="font-weight: bold; font-size: medium">Mail retrait Limoges</h2>
        <button class="close-popup" onclick="closePopup()"><i class="bi bi-x"></i></button>
    </div>
</div>


</table>
{% endblock %}
```

JavaScript

1. **Fonction executePromise()**:
    - Cette fonction est déclenchée lorsqu'un numéro de PS est sélectionné.
    - Elle envoie une requête asynchrone au contrôleur Symfony pour obtenir les détails des mails associés au numéro PS sélectionné et à la plage de dates spécifiée.
    - Les résultats sont affichés dans le conteneur prévu à cet effet.
2. **Fonction attachEventToDetailButtons()**:
    - Cette fonction attache un gestionnaire d'événements à chaque bouton "Détails" dans le tableau des résultats.
    - Lorsqu'un bouton est cliqué, les détails du mail associé sont affichés dans un conteneur dédié.
3. **Fonction closePopup()**:
    - Cette fonction permet de fermer le conteneur affichant les détails du mail lorsqu'un utilisateur clique sur le bouton de fermeture.
4. **Initialisation des Dates**:
    - Les dates de début et de fin sont initialisées pour afficher les mails des deux derniers mois par défaut.

```
{% block javascripts %}
  {{ parent() }}
  <script>

    async function executePromise() {
      try {
        let selectPS = document.querySelector('.saad');          let
selectedPSId = selectPS.value;          let startDate =
document.getElementById('start-date').value;          let endDate =
document.getElementById('end-date').value;

        let response = await
fetch(/Historique_mailController/details/${selectedPSId}?start_date=${startDate}&end_date=${endD
ate});          let jsonData = await response.json();          let commandesContainer =
document.getElementById('commandes-container');          commandesContainer.innerHTML = '';
```

```
            if (jsonData.length === 0) {
                commandesContainer.innerHTML = '<p class="fs-5 alert alert-warning rounded shadow">Aucune mail trouvée pour ce PS.</p>';
            } else  {
                let tableHtml = '<table class="table tom-select table-bordered table-hover w-83 tablestriped  " id="tableCommandes")>';
                tableHtml += '<thead><tr><th class="text-light">Date d\'envoi</th><th class="textlight">Modele de mail</th><th class="text-light">Actions</th></tr></thead>';
                tableHtml += '<tbody>';

                jsonData.forEach(histMail => {
                    if(histMail.date_modif >= startDate.split('/').reverse().join('/') &&histMail.date_modif <= endDate.split('/').reverse().join('/')){
                        tableHtml += `<tr><td>${histMail.date_modif}</td><td>${histMail.intitule_modele_mail} </td>
<td><button class="btn btn-primary detail-button" data-detail="${histMail.detail}"  >Detail </button></td></tr>`;
                        const detail = histMail.detail;
console.log(detail);
attachEventToDetailButtons(detail);


                    }});
// Ajout d'un gestionnaire d'événements pour chaque bouton "Detail"

                tableHtml += '</tbody></table>';
commandesContainer.innerHTML = tableHtml;
attachEventToDetailButtons();                 let table =
$('#tableCommandes').DataTable();
            }


        } catch (error) {
```

```
            console.error('Erreur lors de l\'exécution de la promesse :', error);
        }

        function attachEventToDetailButtons(detail) {
console.log(detail);              var dd=detail;
console.log(detail);
            document.querySelectorAll('.detail-button').forEach(button => {

                button.addEventListener('click', () => {

                    const details = button.getAttribute('data-details');                    const
detailsContainer = document.getElementById('details-container');

                    const overlay = document.getElementById('overlay');
overlay.style.pointerEvents='none';                    const detail =
button.getAttribute('data-detail');
console.log(detail);                    overlay.style.opacity='50%';

                    detailsContainer.style.display = 'block';


                    detailsContainer.innerHTML = `
            <div class="flexissue ">
                <h2 style="font-weight: bold;font-size: medium;color:#00749c">Mail retrait Limoges</h2>
                <button class="close-popup" onclick="closePopup()"><i class="bi bi-x"></i></button>
            </div>
            <div style="padding-top: 30px">
            <p>${detail}</p>
            </div>

`;
```

```
            });
        });
    }
}

function closePopup() {        const detailsContainer =
document.getElementById('details-container');
detailsContainer.style.display = 'none';        overlay.style.pointerEvents='all';

    overlay.style.opacity='100%';
    // Cache la fenêtre contextuelle
}
document.addEventListener("DOMContentLoaded", function() {
var date_debut = document.getElementById('start-date');        var
date_fin = document.getElementById('end-date');

    // Date de fin à aujourd'hui
    var auj = new Date();        var jj =
String(auj.getDate()).padStart(2, '0');        var mm =
String(auj.getMonth() + 1).padStart(2, '0');        var yyyy
= auj.getFullYear();

    auj =  yyyy+ '-' + mm + '-' +jj ;
date_fin.value = auj;

    // Date de début il y a deux mois        var deuxmois =
new Date();        deuxmois.setMonth(deuxmois.getMonth() -
2);        var jJ= String(deuxmois.getDate()).padStart(2, '0');
var mM = String(deuxmois.getMonth() + 1).padStart(2, '0');
var aa_a = deuxmois.getFullYear();
```

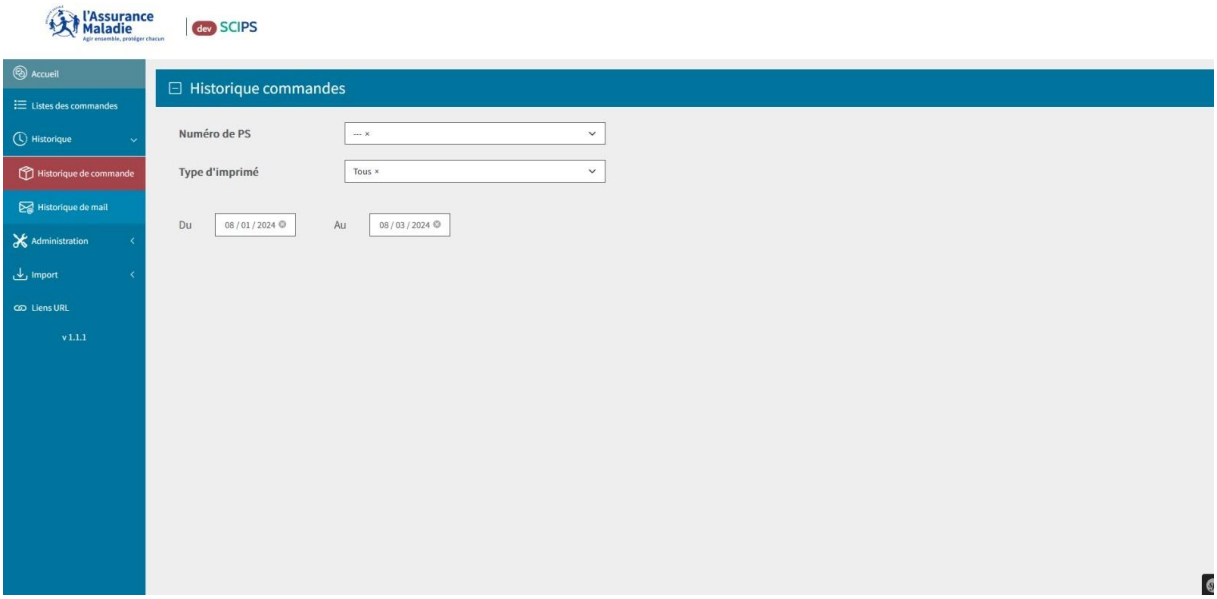
```
        deuxmois = aa_a + '-' + mM + '-' + jJ;
date_debut.value = deuxmois;
    });
  </script>
{% endblock %}
```

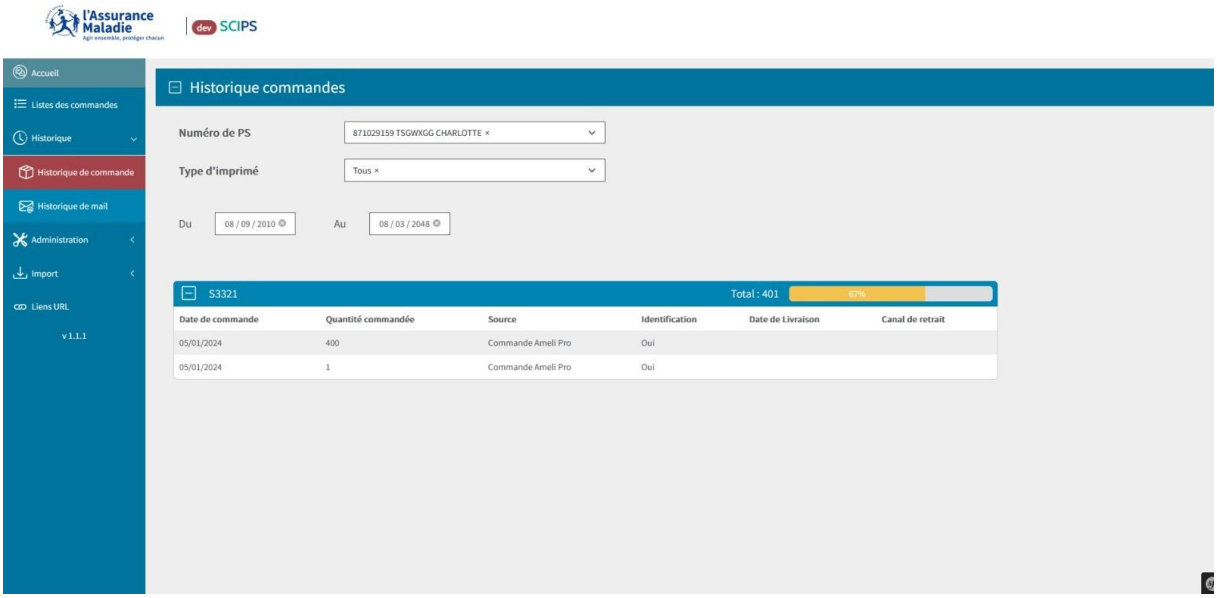
Action contrôleur

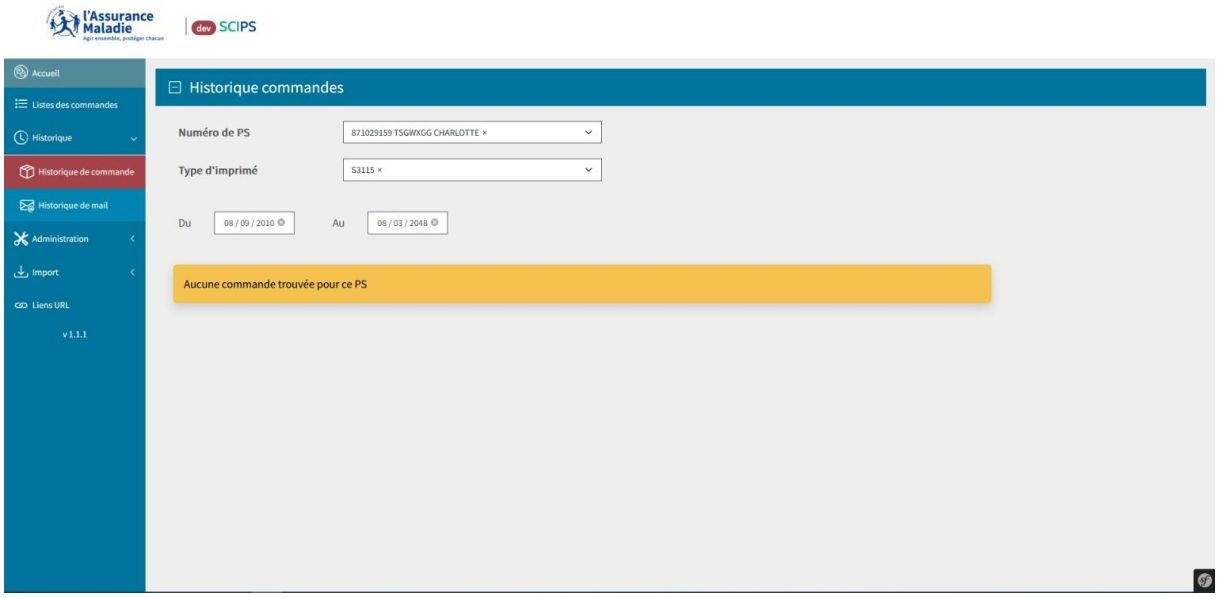
```
  /**
   * @Route("/details/{id}", name="historique_mail_controller_id")
   */
  public function getHistoriqueMailDetails($id, PSRepository $psRepository): JsonResponse
  {
    // Récupérer les détails du courrier historique en fonction de l'identifiant fourni
$ps = $psRepository->find($id);
     if (!$ps)
{          //
Retourner
une réponse
JSON avec un
message
d'erreur si
aucun
courrier
historique
n'est trouvé

        return new JsonResponse(['error' => 'Aucun courrier historique trouvé pour cet identifiant.'],
JsonResponse::HTTP_NOT_FOUND);
    }


    $historiqueMails = $ps->getHistoriqueMails();
```

```
        $historiqueMailDetails = [];



        // Générer les données à renvoyer sous forme de réponse JSON
foreach ($historiqueMails as $historiqueMail) {
            $modeleMail = $historiqueMail->getModeleMail();
            $intitule = $modeleMail ? $modeleMail->getIntitule() : null;
            $detail = $historiqueMail->getModeleMail()->getCorps();
            $historiqueMailDetails[] = [
                'id' => $historiqueMail->getModeleMail(),
                'intitule_modele_mail' => $intitule,
                'date_modif' => $historiqueMail->getDateModif() ? $historiqueMail-
>getDateModif()>format('Y-m-d') : null,
            'detail'=>  nl2br($detail),
                ];
        }



        // Créer une réponse JSON avec les données du courrier historique
$response = new JsonResponse($historiqueMailDetails);

        // Définir l'en-tête Content-Type sur application/json
        $response->headers->set('Content-Type', 'application/json');

        // Retourner la réponse JSON
        return $response;
    }
```

Historique commandes :







Dans la section de l'historique des commandes, nous offrons une sélection des types d'imprimés disponibles. Lorsque le client choisit un numéro PS et un type d'imprimé, ou tous les types d'imprimés, un tableau détaillé est généré. Ce tableau inclut la date de commande, la quantité commandée, la source, l'état de la préidentification de la commande, la date de livraison prévue, et le canal de retrait.

Pour chaque type d'imprimé, nous calculons le total des quantités commandées et le comparons avec le maximum autorisé. Ensuite, nous affichons un pourcentage pour indiquer le niveau de remplissage. Par exemple, si un PS a effectué deux commandes pour un total de 800 unités, et que le maximum autorisé est de 1000 unités pour ce type d'imprimé, nous affichons un pourcentage de 80%, indiquant qu'il reste 20% de la capacité disponible.

Si aucune commande n'est trouvée pour le PS sélectionné, un message est affiché pour informer l'utilisateur de cette

situation. Il s'accompagne d'une vérification pour garantir la clarté des informations fournies.

Administration :

L'administration comprend cinq pages dédiées à la gestion des impressions, des fournisseurs, des sources, des e-mails et des liens. Chacune de ces pages est configurée de manière similaire. Je vais vous montrer deux exemples : la gestion des impressions et une autre partie de l'administration.

Dans la gestion des impressions, nous avons deux actions principales :

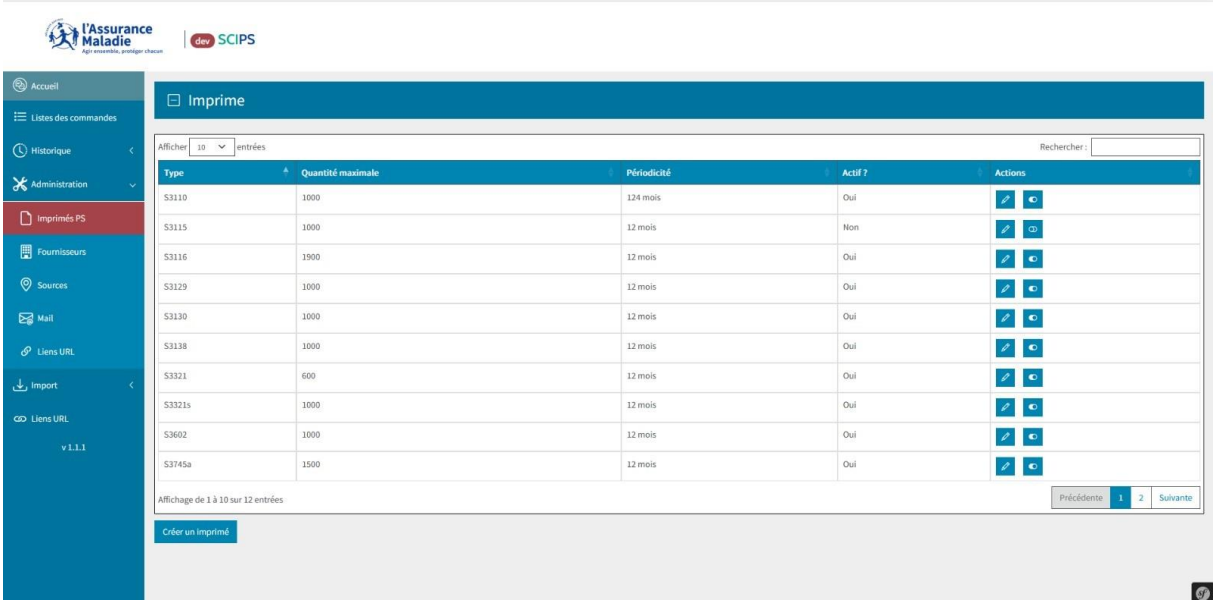
Le bouton "Éditer" qui est présent dans toutes les parties de l'administration. Il permet de modifier les informations liées à chaque élément.

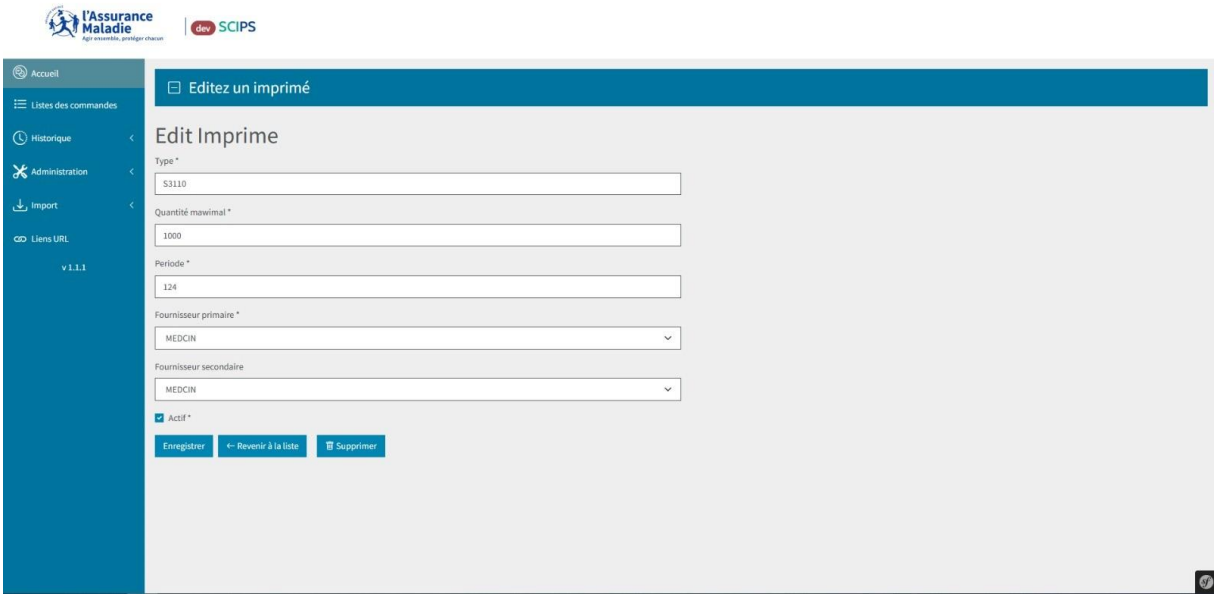
Le deuxième est un bouton bascule (toggle button) qui permet d'activer ou de désactiver un type d'imprimé. Le représentant du service client peut éditer les informations, mais il ne peut pas supprimer un type d'imprimé déjà utilisé dans une commande. Dans ce cas, un message informe le représentant du service client : "Vous ne pouvez pas supprimer cet imprimé car il est déjà utilisé dans une commande".

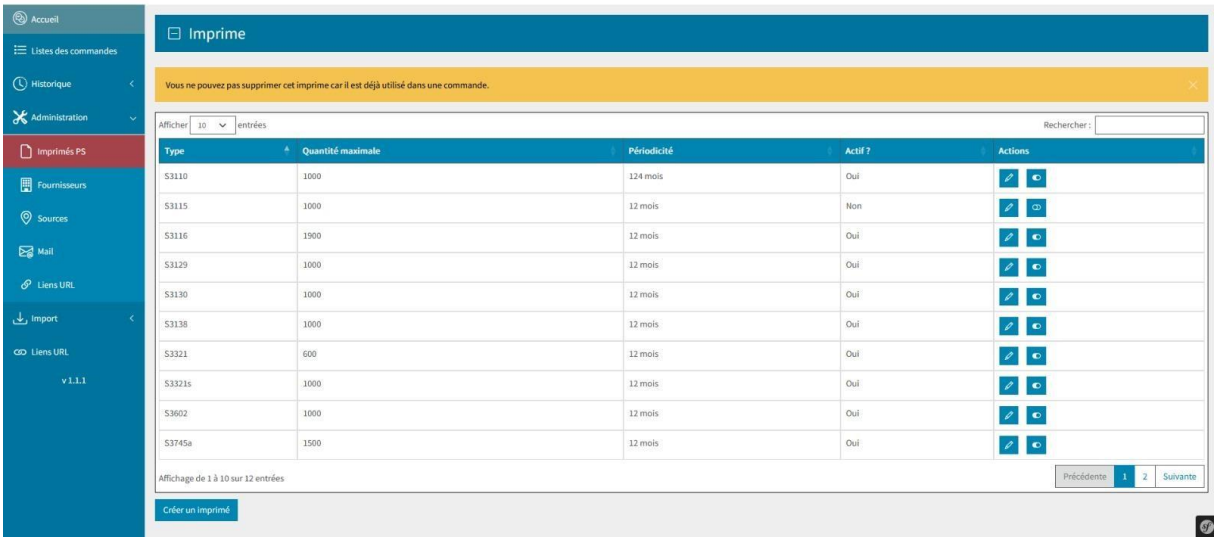
Dans d'autres parties de l'administration, comme la gestion des modèles d'e-mails, le même principe s'applique. Si un modèle d'e-mail est utilisé dans l'historique, le représentant du service client est informé : "Vous ne pouvez pas supprimer

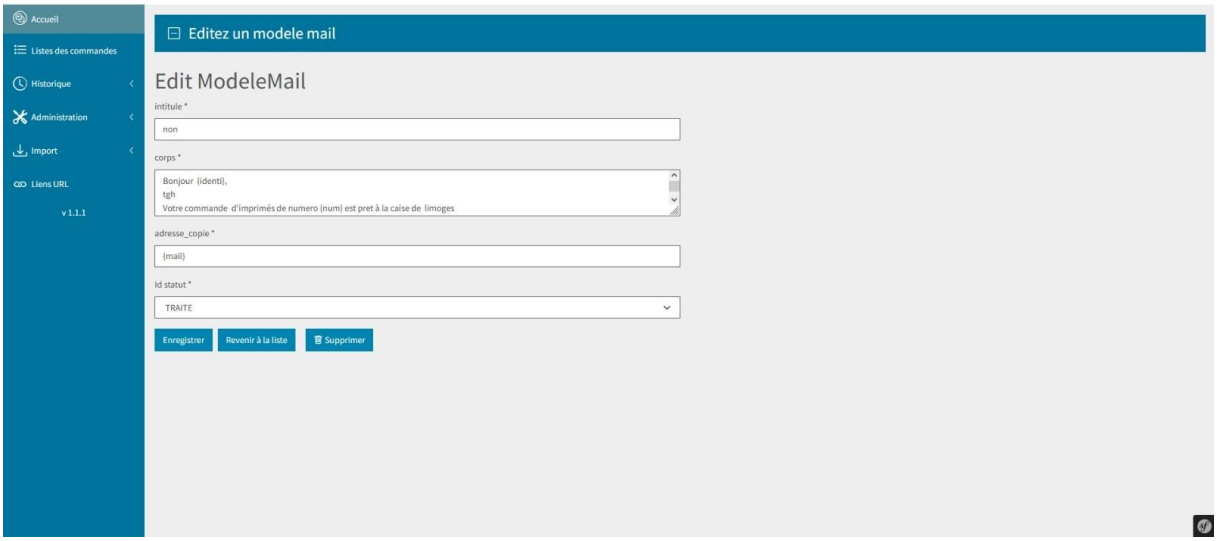
ce mail car il est déjà utilisé dans l'historique, mais vous pouvez le modifier".

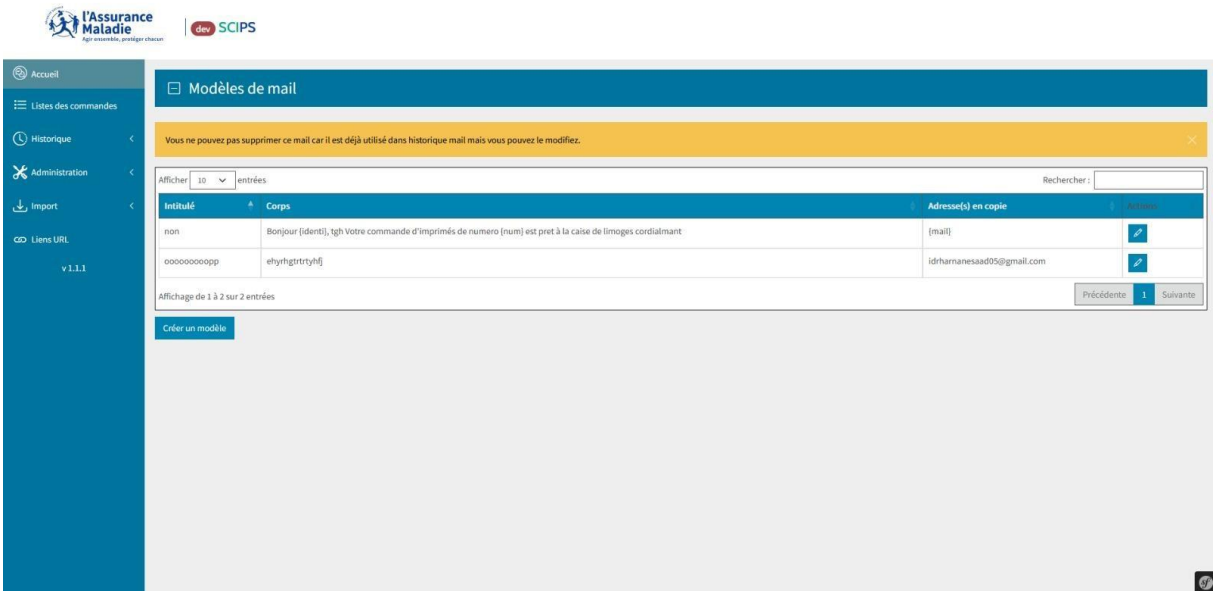
Cette approche vise à assurer que les éléments essentiels ne sont pas supprimés accidentellement tout en permettant des modifications nécessaires.




Accueil
Listes des commandes
Historique
Administration
Import
Liens URL
v 1.1.1
Editez un imprimé
Edit Imprime
Type *
S3110
Quantité mawimal *
1000
Periode *
124
Fournisseur primaire *
MEDCIN
Fournisseur secondaire
MEDCIN
Actif *
Enregistrer
← Revenir à la liste
Supprimer


Accueil
Listes des commandes
Historique
Administration
Imprimés PS
Fournisseurs
Sources
Mail
Liens URL
Import
Liens URL
v 1.1.1
Imprime
Vous ne pouvez pas supprimer cet imprime car il est déjà utilisé dans une commande.
Afficher 10 entrées
Rechercher :
Type
Quantité maximale
Périodicité
Actif ?
Actions
S3110 1000 124 mois Oui
S3115 1000 12 mois Non
S3116 1900 12 mois Oui
S3129 1000 12 mois Oui
S3130 1000 12 mois Oui
S3138 1000 12 mois Oui
S3321 600 12 mois Oui
S3321s 1000 12 mois Oui
S3602 1000 12 mois Oui
S3745a 1500 12 mois Oui
Affichage de 1 à 10 sur 12 entrées
Précédente 1 2 Suivante
Créer un imprimé


Accueil
Listes des commandes
Historique
Administration
Import
Liens URL
v 1.1.1
Editez un modele mail
Edit ModeleMail
intitule *
non
corps *
Bonjour {identi},
tgh
Votre commande d'imprimés de numero {num} est pret à la caise de limoges
adresse_copie *
{mail}
Id statut *
TRAITE
Enregistrer
Revenir à la liste
Supprimer



Import :

L'importation des données se déroule sur deux pages distinctes : une pour l'importation des PS (Personnel de Santé) et une autre pour l'importation des commandes. Sur ces deux pages, plusieurs vérifications sont effectuées pour garantir l'intégrité des données importées.

Importation des PS :

Si l'identifiant du numéro de commande ou l'identifiant du PS existe déjà dans la base de données, ces données ne sont pas importées à nouveau.

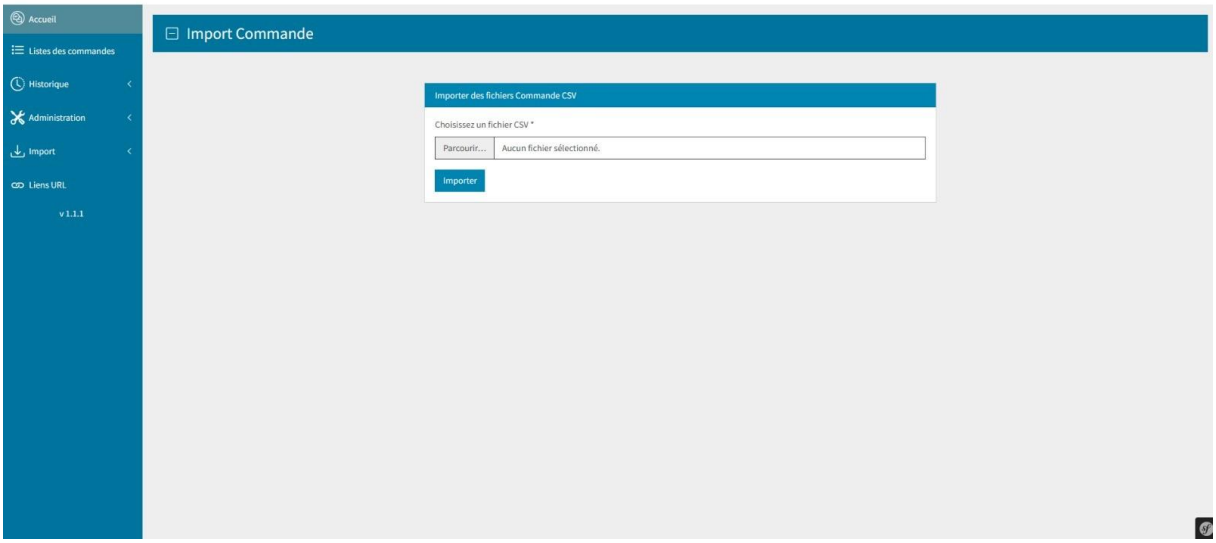
Pour les PS, si les données sont modifiées pour un même PS, ces modifications sont prises en compte lors de l'importation.

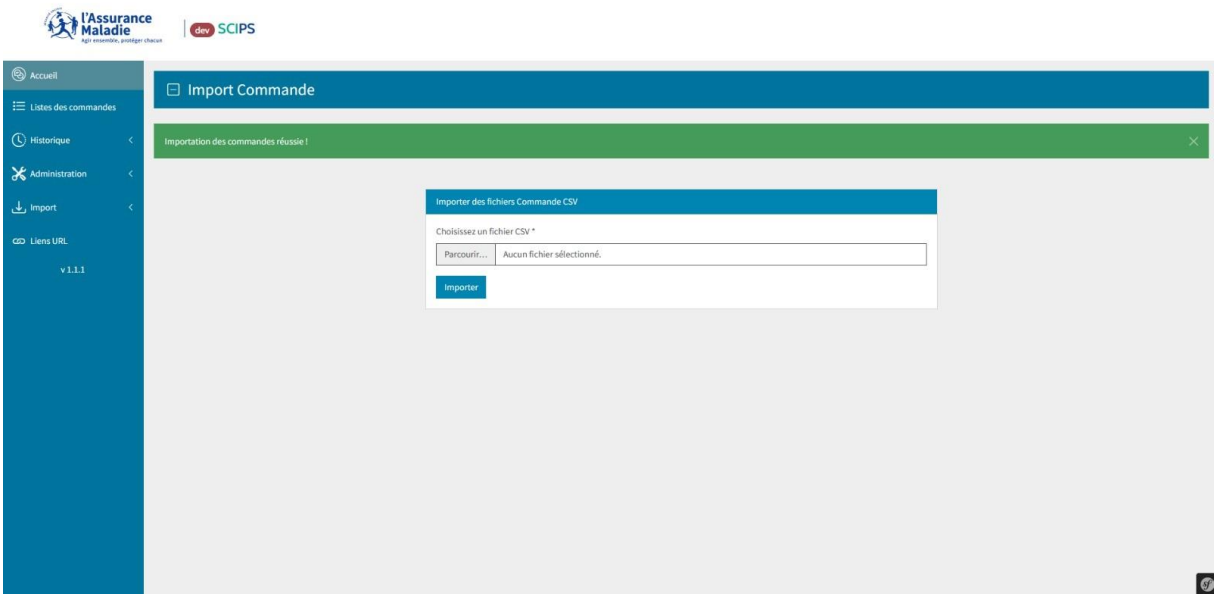
Importation des Commandes :

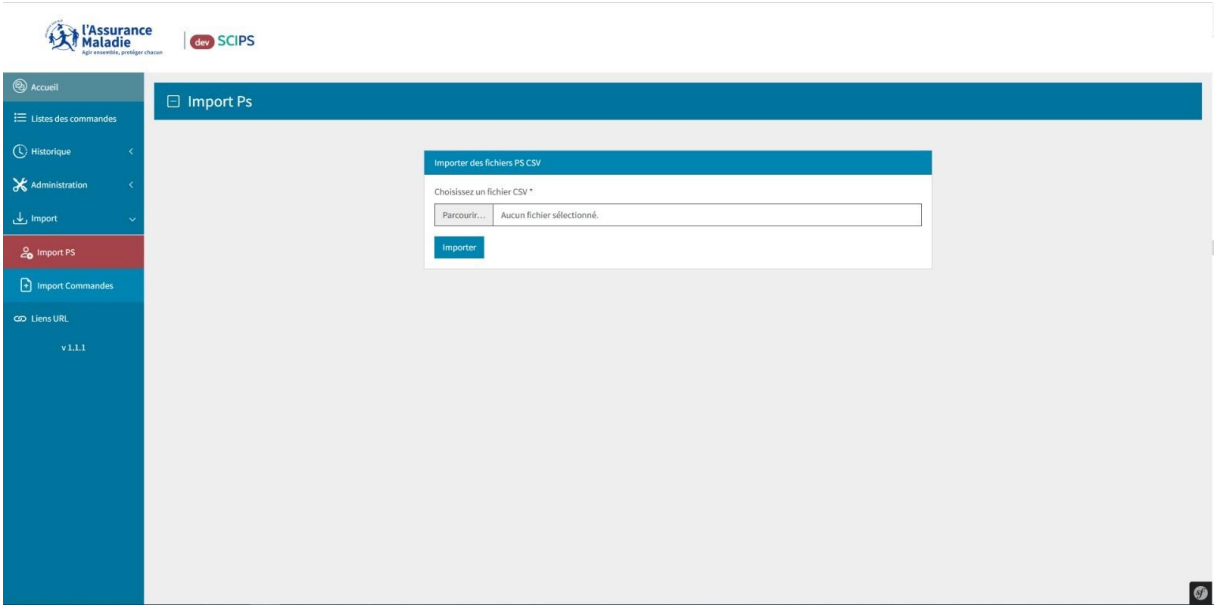
Les mêmes vérifications sont effectuées pour éviter les doublons dans les commandes et les PS.

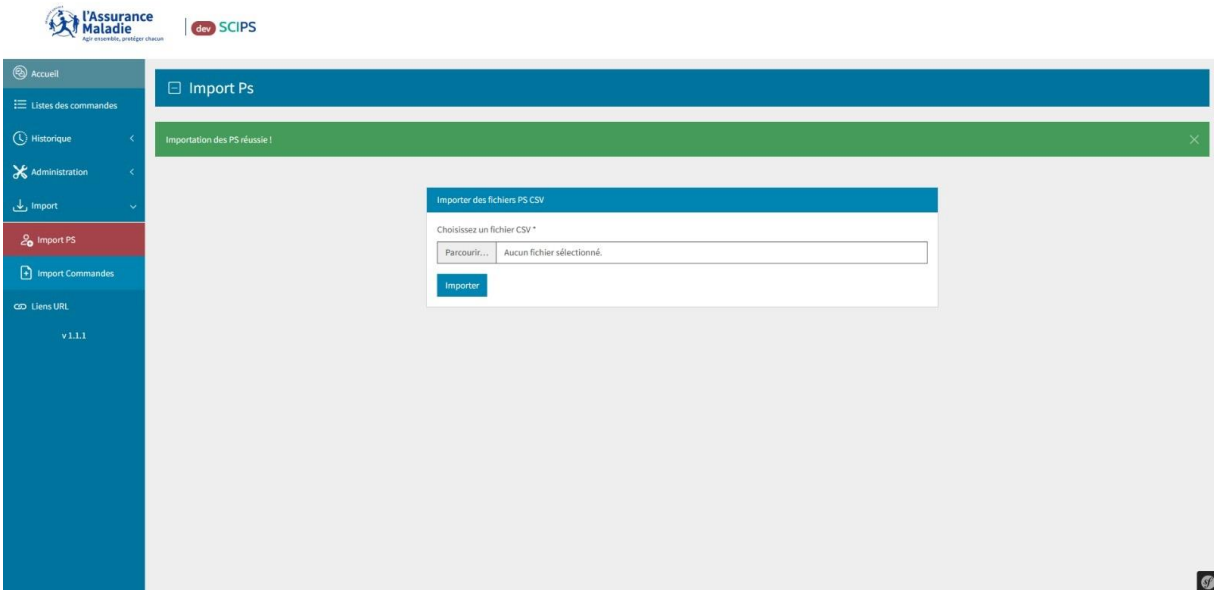
Pour les commandes non identifie identifiées et ayant le statut "traité", leur statut est automatiquement changé en "en cours" lors de l'importation et pour les commande identifie et « traité » leur statut est automatiquement changé en « Prête » .

Les modifications apportées aux PS lors de l'importation sont également prises en compte.


Accueil
Listes des commandes
Historique
Administration
Import
Liens URL
v 1.1.1
Import Commande
Importer des fichiers Commande CSV
Choisissez un fichier CSV *
Parcourir...
Aucun fichier sélectionné.
Importer


l'Assurance Maladie
dev SCIPS
Accueil
Listes des commandes
Historique
Administration
Import
Liens URL
v 1.1.1
Import Commande
Importation des commandes réussie !
Importer des fichiers Commande CSV
Choisissez un fichier CSV *
Parcourir...
Aucun fichier sélectionné.
Importer


l'Assurance Maladie
dev SCIPS
Accueil
Listes des commandes
Historique
Administration
Import
Import PS
Import Commandes
Liens URL
v 1.1.1
Import Ps
Importer des fichiers PS CSV
Choisissez un fichier CSV *
Parcourir...
Aucun fichier sélectionné.
Importer


l'Assurance Maladie
dev SCIPS
Accueil
Listes des commandes
Historique
Administration
Import
Import PS
Import Commandes
Liens URL
v 1.1.1
Import Ps
Importation des PS réussie !
Importer des fichiers PS CSV
Choisissez un fichier CSV *
Parcourir...
Aucun fichier sélectionné.
Importer

Partie d’import des commandes :

Twig :

```
{% block content %}
   {#  <div class="container" style="margin-top: 10rem;">
      {{ form_start(form, {'attr': {'class': 'mx-auto w-50', 'action': path('import_commande')}}) }}
<table class="form-group">
         <td>
            {{ form_row(form.file, {'attr': {'name': 'commande_file'}}) }}
         </td>
         <td>
            <button type="submit" class="btn btn-primary" name="import">Importer</button>
         </td>
      </table>
      {{ form_end(form) }}
   </div>#}
   <div class="container-fluid my-5">
      <div class="row justify-content-center">
         <div class="col-md-6">
            <div class="card">
               <div class="card-header bg-primary text-white">
                  Importer des fichiers Commande CSV
               </div>
               <div class="card-body">
                  {{ form_start(form, {'attr': {'class': 'w-100', 'action':
path('app_importer_commande_csv')}}) }}
                  <div class="form-group">
                     {{ form_row(form.file, {'attr': {'class': 'form-control-file'}}) }}
</div>
                  <button type="submit" class="btn btn-primary">Importer</button>
                  {{ form_end(form) }}
```

```
            </div>
          </div>
        </div>
      </div>
    </div>

{% endblock
```

Action contrôleur :

Cette méthode est responsable de deux choses principales :

- Rendre la page du formulaire d'importation.
- Traiter la soumission du formulaire et l'importation des données.

Rendre la page du formulaire :

- Le formulaire est créé en utilisant la classe ImporterFormType, qui définit les champs du formulaire.
- Le formulaire est rendu dans le template **importer_commande_csv/index.html.twig**.
- L'utilisateur peut choisir un fichier CSV à importer.

Traitement de la soumission du formulaire :

- Lorsque le formulaire est soumis, la méthode vérifie s'il est valide.
- Ensuite, elle récupère le fichier CSV à partir des données soumises dans le formulaire.
- Elle appelle ensuite la méthode **importFile()** pour traiter le fichier CSV.
- Après l'importation réussie, un message de succès est ajouté et l'utilisateur est redirigé vers la même page.

- Elle utilise la bibliothèque **league/csv** pour lire et parcourir le contenu du fichier CSV.

- Pour chaque enregistrement du fichier CSV, elle vérifie d'abord s'il existe déjà une commande avec le même numéro de commande dans la base de données.
- Si la commande n'existe pas, elle crée une nouvelle instance de l'entité **Commande** et lui attribue les valeurs du fichier CSV.
- Les relations avec d'autres entités telles que **PS**, **Statut**, **Imprime**, **Source**, etc., sont également gérées en fonction des valeurs du fichier CSV.
- Enfin, les instances de **Commande** sont persistées dans le contexte de persistance de Doctrine, et les changements sont sauvegardés dans la base de données.

En résumé, ce contrôleur offre une fonctionnalité d'importation de données robuste à partir de fichiers CSV, en gérant les doublons, les relations entre entités et en utilisant Doctrine pour interagir avec la base de données

```
class ImporterCommandeCsvController extends AbstractController
{
private $entityManager;     public function
__construct(EntityManagerInterface $entityManager)
   {
      $this->entityManager = $entityManager;
   }
   /**
    * @Route("/import_commande", name="import_commande")
    */
   public function importCommandeCsv(Request $request, ImporterController $importerController): Response
   {
      $form = $this->createForm(ImporterFormType::class);
      $form->handleRequest($request);
```

```
        if ($form->isSubmitted() && $form->isValid()) {
$csvFile = $form['file']->getData();
            $this->importFile($csvFile->getRealPath());

            $this->addFlash('success', 'Importation des commandes réussie !');
return $this->redirectToRoute('app_importer_commande_csv');
        }

        return $this->render('importer_commande_csv/index.html.twig', [
            'controller_name' => 'ImporterCommandeCsvController',
            'form' => $form->createView(),
        ]);
    }
    private function importFile(string $csvFile): void
    {
        $csv = Reader::createFromPath($csvFile);
        $csv->setDelimiter(';');
        $csv->setHeaderOffset(0);

        $records = $csv->getRecords();
 // Verification        foreach
($records as $record) {
// Vérifie si la pré-identification est égale à 'Oui'
            //if ($record['Pré-identification'] !== 'Non') {
              //  continue; // Passe à l'itération suivante si la pré-identification n'est pas 'Oui'
            //}
            $existingCommande = $this->getDoctrine()
                ->getRepository(Commande::class)
                ->findOneBy([
                    'id' => $record['Numéro de commande']
```

```
            ]);


        // Si la commande existe déjà, passer à l'itération suivante
if ($existingCommande !== null) {
            continue;
        }




        $commande = new Commande();
        $commande->setId($record['Numéro de commande']);


        // Si la commande n'existe pas, créez une nouvelle commande ***** import donc si la quantite,numero de commande et la preiden n'ont pas changer une creer une nouvelle table sinon, s'ils sont mmodifier on ecraseee
         try
{
            $ps = $this->getDoctrine()
                ->getRepository(PS::class)
                ->find((int)$record['Numero AM']);


            if ($ps !== null) {
                $commande->setIdPs($ps);
            }
        } catch (\Exception $e) {


        }
        $dateString = $record['Date commande'];
        $date = \DateTime::createFromFormat('d/m/Y', $dateString);
```

```
        if ($date instanceof \DateTime) {
            $commande->setDate($date);
        }
        $commandeId = (int)$record['Numéro de commande'];
        $commande->setId($commandeId);




        $statut = $this->entityManager
            ->getRepository(Statut::class)
            ->findOneBy(['libelle' => $record['Statut']]);
if ($record['Pré-identification'] === 'Non') {
            $id_statut=2;
            $statutEnCour = $this->entityManager
                ->getRepository(Statut::class)
                ->find($id_statut);
            $commande->setIdStatut($statutEnCour);
            }else{
            $id_statut=4;
            $statutEnCour = $this->entityManager
                ->getRepository(Statut::class)
                ->find($id_statut);
            $commande->setIdStatut($statutEnCour);
        }


        $imprime = $this->entityManager
            ->getRepository(Imprime::class)
            ->findOneBy(['type' => $record['Référence imprimé']]);
        $commande->setIdImprime($imprime); $preIdentificationValue = ($record['Pré-identification']
=== 'Oui') ? true : false;           $commande->setPreIdentification(  $preIdentificationValue);
```

```
            $commande->setQuantiteCommandee((int)$record['quantité']);

            $source = $this->entityManager
                ->getRepository(Source::class)
                ->findOneBy(['id' => 1]);
            $commande->setIdSource($source);

            $this->getDoctrine()->getManager()->persist($commande);
        }

        $this->getDoctrine()->getManager()->flush();
    }
}
```

Partie d'import des PS:

Twig :

```
{% block content %}
    {#  <div class="container" style="margin-top: 10rem;">
        {{ form_start(form, {'attr': {'class': 'mx-auto w-50', 'action': path('import_ps')}}) }}
<table class="form-group">
            <td>
                {{ form_row(form.file, {'attr': {'name': 'ps_file'}}) }}
            </td>
            <td>
                <button type="submit" class="btn btn-primary" name="import">Importer</button> </td>
        </table>
        {{ form_end(form) }}
    </div>#}
    <div class="container-fluid my-5">
        <div class="row justify-content-center">
```

```
        <div class="col-md-6">
            <div class="card">
                <div class="card-header bg-primary text-white">
                    Importer des fichiers Commande CSV
                </div>
                <div class="card-body">
                    {{ form_start(form, {'attr': {'class': 'w-100', 'action': path('app_importer_ps_csv')}}) }}
                    <div class="form-group">
                        {{ form_row(form.file, {'attr': {'class': 'form-control-file'}}) }}
</div>
                    <button type="submit" class="btn btn-primary">Importer</button>
                    {{ form_end(form) }}
                </div>
            </div>
        </div>
    </div>

{% endblock
```

Action contrôleur :

```
class ImporterPsCsvController extends AbstractController
{
    private $entityManager;

    public function __construct(EntityManagerInterface $entityManager)
    {
        $this->entityManager = $entityManager;
    }

    /**
     * @Route("/importer_ps_csv", name="app_importer_ps_csv")
```

```
     */
    public function index(Request $request): Response
    {
        $form = $this->createForm(ImporterFormType::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
$csvFile = $form['file']->getData();
            $this->importFile($csvFile->getRealPath());

            $this->addFlash('success', 'Importation des PS réussie !');
return $this->redirectToRoute('app_importer_ps_csv');
        }

        return $this->render('importer_ps_csv/index.html.twig', [
            'controller_name' => 'ImporterPsCsvController',
            'form' => $form->createView(),
        ]);
    }

    private function importFile(string $csvFile): void
    {
        $csv = \League\Csv\Reader::createFromPath($csvFile);
        $csv->setDelimiter(';');
        $csv->setHeaderOffset(0);

        $records = $csv->getRecords();

        foreach ($records as $record) {
```

```
        $psId = utf8_encode($record['No de l\'executant'] . $record['Clé du numéro du PS']);
// Vérifie si le PS existe déjà
        $existingPs = $this->entityManager
            ->getRepository(PS::class)
            ->findOneBy(['id' => $psId]);

        // Si le PS existe déjà, passer à l'itération suivante
if ($existingPs !== null) {             continue;
        }
        $ps = new PS();
        $ps->setId(utf8_encode($record['No de l\'executant'] . $record['Clé du numéro du PS']));
        $ps->setNom(utf8_encode($record['Nom publipostage executant']));
        $ps->setPrenom(utf8_encode($record['Prenom de l\'executant']));
        $ps->setCategorie(utf8_encode($record['Categorie de l\'executant']));
        $ps->setSpecialite(utf8_encode($record['Specialite de l\'executant']));
        $ps->setNumTelephone(utf8_encode($record['Numéro de téléphone']));
        $ps->setMail(utf8_encode($record['Adresse "e-mail"']));
        $ps->setAdresse(utf8_encode($record['No voie residence de l\'executant'] . $record['Nature voie residence executant'] . $record['Nom voie residence executant'] . $record['Compl. adresse de l\'executant'] . $record['Code postal de l\'executant'] . $record['Nom localite residence Executant']));
$this->entityManager->persist($ps);
    }

    $this->entityManager->flush();
  }
}
```