R5.D.04 : Développement pour progiciels

Projet : Réalisation d'un module personnalisé

Cyberware

VINET LATRILLE Jules

TD D Alt

Semestre 5

11 / 01 / 2026

TABLE DES MATIERES

Table des illustrations :

Mise en place et Environnement

Pour lancer ce projet : Une fois le dossier du projet extrait (ce que vous avez fait étant donnez que vous lisez ce rapport), ouvrez un terminal dans le dossier contenant le fichier docker-compose.yml ainsi que le dossier addons. Lancez la commande **docker-compose up -d**, qui va permettre de démarrer les conteneurs Docker permettant l'utilisation de l'application.

Une fois cela fait, **attendez quelques secondes** le temps que les conteneurs démarrent correctement, et rendez vous sur le lien **http://localhost:8069**.

Une fois dessus, créez la base de données en remplissant les champs comme indiqué ci-dessous :

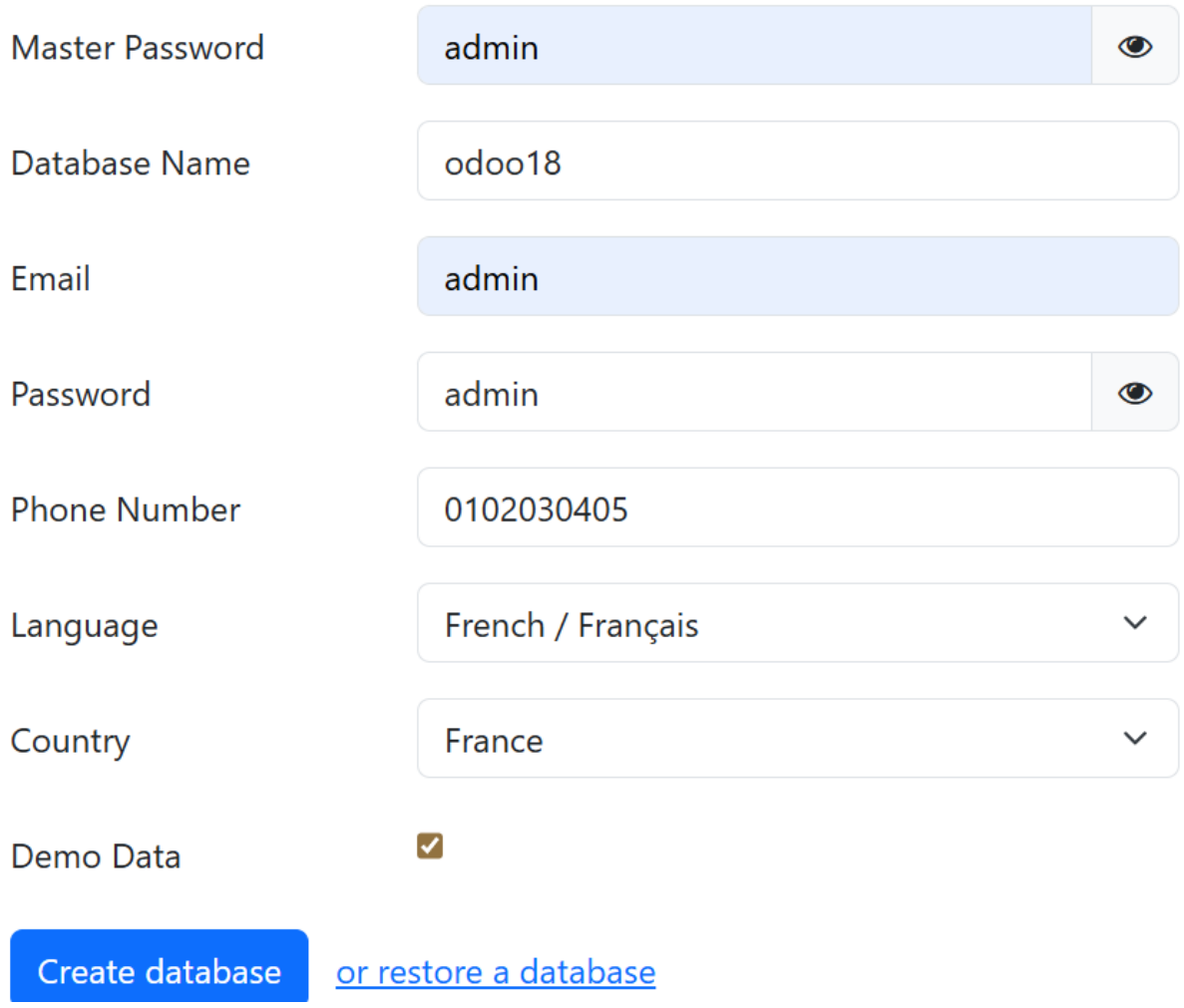

Figure 1 : Remplissage du formulaire de Base de Données

Une fois la Base de données créée, connectez-vous à votre compte avec comme utilisateur « admin » et mot de passe « admin ».

Enfin, activez les applications nécessaires au bon fonctionnement du module : Ventes, Inventaire, Site Web, Contacts, et évidement Cyberware.

Groupe de sécurité

Il y a aux moins deux groupes de sécurité

Mes deux groupes de sécurité sont définis dans mon fichier **security/cyberware_security.xml** :

- Un groupe **Client** nommé cyberware_group_client, qui corresponds aux utilisateurs standards ou aux patients qui consultent le catalogue. Il est configuré pour hériter des droits de l'utilisateur de base Odoo (base.group_user).
- Un groupe **Charcudoc** nommé cyberware_group_charcudoc, qui corresponds aux administrateurs ou médecins. Tout comme le client, il est configuré pour hériter des droits de l'utilisateur de base Odoo (base.group_user) et aussi hériter des droits root/admin d'Odoo (base.user_admin et base.user_root), lui donnant accès à l'interface de gestion backend.

Pour vérifier l'existence de ces groupes, vous pouvez aller dans Paramètres -> activer le mode développeur. Ensuite, toujours dans paramètres, cliquer sur « Utilisateurs & sociétés » -> Groupes -> Rechercher cyberware.

Des droits d'accès différents aux modèles ont été accordés aux groupes de sécurité

Les droits des différents groupes sont gérés via le fichier **security/ir.model.access.csv**.

Ce fichier permet de définir précisément ce que chaque groupe peut faire sur chaque modèle (Lecture, Écriture, Création, Suppression) :

- Le groupe **Charcudoc** dispose d'un contrôle total sur l'application. Il possède les droits de lecture, d'écriture, de création et de suppression (1,1,1,1) sur l'ensemble des modèles : les implants, les fabricants, les clients et les implantations. Cela lui permet de gérer intégralement le cabinet et le catalogue.
- Le groupe **Client** possède des droits beaucoup plus restreints. Il est limité à la simple lecture (1,0,0,0) sur les modèles implant, manufacturer et client, ce qui lui permet de consulter le catalogue et son profil sans pouvoir altérer les données. Une exception a été faite pour le modèle implantation (les interventions), sur lequel le client possède des droits de lecture et d'écriture (1,1,0,0) pour pouvoir consulter et mettre à jour son historique médical, mais sans pouvoir créer de nouvelles opérations.

Modèles

Il y a plusieurs modèles.

L'architecture des données du module a été construite autour de plusieurs entités distinctes, définies dans le dossier *models/*. Le cœur du système repose sur le modèle **cyberware.implant**, qui gère le catalogue des produits. Il interagit avec le modèle **cyberware.client**, représentant les patients, et le modèle **cyberware.manufacturer**, référençant les entreprises fabricantes. Un modèle intermédiaire, **cyberware.implantation**, a également été créé pour historiser les opérations.

Il y a dans au moins un modèle un champ textuel, un champ numérique, un champ date, un champ image, un champ select, un champ calculé et un champ related.

J'ai concentré l'ensemble des types de champs requis au sein du modèle principal cyberware.client (fichier *models/cyberware_client.py*) :

- Un **champ textuel** (notes_medicales) pour saisir l'historique médical complet.
- Un **champ numérique** (niveau_essence_max) définissant la capacité du patient.
- Un **champ date** (date_naissance) indispensable pour le suivi.
- Un **champ image** (image_client) pour l'avatar.
- Un **champ select** (groupe_sanguin) proposant une liste de choix pré-définis (A+, B-, etc.).
- Un **champ calculé** (age), qui se met à jour automatiquement selon la date de naissance.
- Un **champ related** (email_user), qui récupère automatiquement l'adresse email depuis le compte utilisateur lié, sans duplication de données.

Il y a au moins dans un modèle une méthode qui contrôle des données.

Pour garantir la cohérence et la fiabilité des informations enregistrées, une logique de validation a été intégrée directement dans le modèle principal **cyberware.implant** (dans le fichier *models/cyberware_implant.py*.)

Cette validation est assurée par la méthode **_check_valeurs_positives**, qui utilise le décorateur @api.constrains. Cette méthode est automatiquement exécutée à chaque création ou modification d'un implant. Elle vérifie spécifiquement que les champs prix_euro et cout_essence ne contiennent pas de valeurs négatives. Si cette règle est enfreinte, le système interrompt l'opération de sauvegarde et déclenche une

exception ValidationError, affichant un message d'erreur à l'utilisateur pour l'empêcher d'enregistrer des données invalides.

Vous pouvez tester le fonctionnement ce cette fonctionnalité depuis n'importe quel cyberware (donc pour y accéder, depuis la page d'accueil, sélectionnez un cyberware), en faisant une modification « interdite » puis en enregistrant.

Il existe au moins une relation un à plusieurs et une relation plusieurs à plusieurs.

J'ai intégré les deux types de relations requises dans plusieurs de mes modèles, mais celui que je vais prendre en exemple est le modèle cyberware.client (fichier cyberware_client.py).

Relation "Un à Plusieurs"

J'ai inclus une relation de type Un à Plusieurs en utilisant le champ **implantation_ids**. Ce champ permet de relier le client à tous les enregistrements de son historique d'opérations, ce qui est essentiel pour afficher l'historique complet de toutes ses implantations passées et futures.

Relation "Plusieurs à Plusieurs"

J'ai également défini une relation de type Plusieurs à Plusieurs en utilisant le champ **implant_ids**. Cette relation permet de lister tous les implants actuellement possédés par le client. Ce type de lien est nécessaire car un client peut posséder plusieurs implants, et chaque modèle d'implant peut être installé sur de nombreux clients différents.

Il existe des données de démo au format xml. Certains enregistrements sont liés par les relations d'autres non.

Le fichier principal est au format XML (**demo/cyberware.demo.xml**), qui est utilisé pour créer la majorité des enregistrements comme les implants et les clients. Ce format est essentiel, car il permet d'établir les liens relationnels entre les différents modèles. Par exemple, pour créer un implant, j'utilise l'attribut ref dans le XML (ex: ref="demo_manuf_militech") afin de le lier immédiatement à un fabricant spécifique.

En parallèle, certains enregistrements sont définis de manière autonome sans lien relationnel immédiat, comme les profils clients créés dans **demo/cyberware.client.xml**. Ces enregistrements sont définis avec leurs propres valeurs (nom, pseudo, date de naissance calculée) sans être dépendants d'autres objets du module lors de leur création initiale.

Un des modèles étend un modèle d'Odoo.

J'ai choisi d'intégrer certaines fonctionnalités du module directement sur les fiches de contact existantes dans Odoo plutôt que de recréer une entité de personne (c'est pour cela qu'il fallait activer le module Contacts au début). Pour cela, j'ai utilisé l'héritage dans le fichier **models/res_partner.py**.

Grâce à l'attribut _inherit = 'res.partner', je me suis basé sur le modèle natif de Contact d'Odoo pour y ajouter deux champs spécifiques. J'ai notamment ajouté une case à cocher booléenne, is_charcudoc, et un champ pour la spécialité. Cette approche permet de transformer un contact standard en "Charcudoc" sans dupliquer toutes les informations de base du contact.

Vues

Il existe une vue liste personnalisée.

J'ai défini une vue liste personnalisée dans le fichier **views/cyberware_implant_views.xml**. Cette vue, identifiée sous le nom technique view_cyberware_implant_list. Pour y accéder sur le site, il suffit de naviguer via l'URL http://localhost:8069/odoo/action-480?view_type=list.

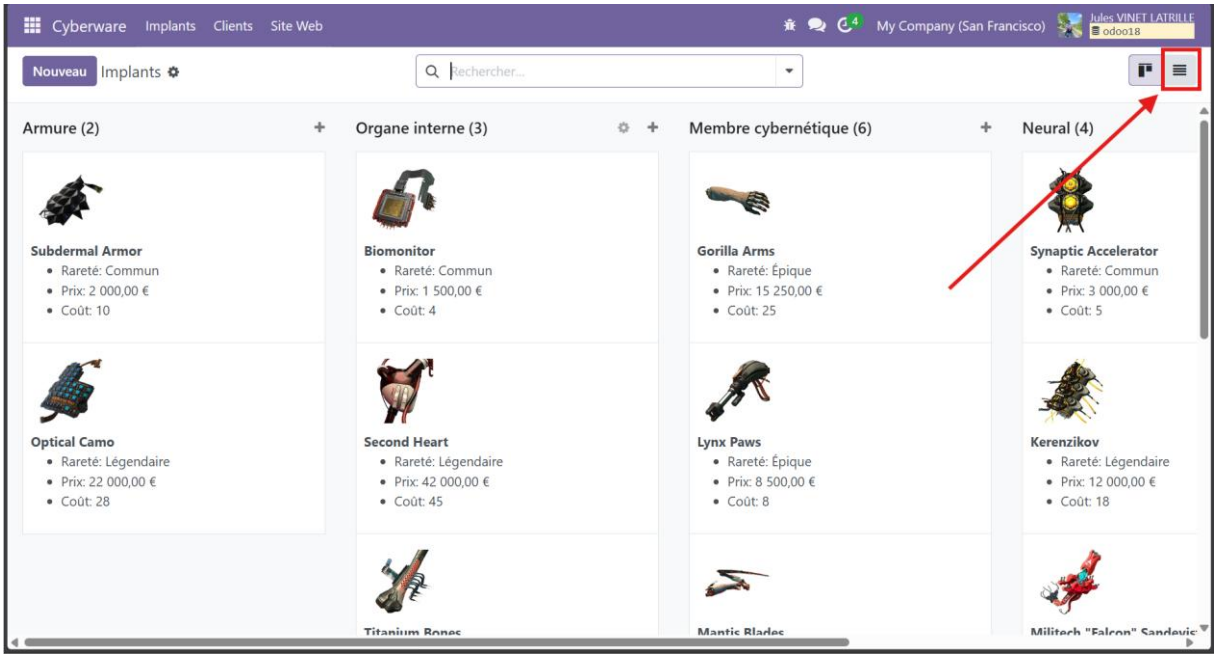

Figure 2 : Sélectionner la vue liste personnalisée

J'ai personnalisé cet affichage pour un aperçu rapide du catalogue. La vue liste n'affiche pas uniquement l'ID des enregistrements, mais présente directement les informations clés du catalogue : le nom de l'implant (nom_implant), son type, sa rareté

et son prix. Cela permet à l'utilisateur une lecture immédiate et fonctionnelle des produits disponibles.

Il existe une vue formulaire personnalisée.

J'ai créé une vue formulaire dédiée pour la consultation et la saisie détaillée des implants, définie dans le fichier **views/cyberware_implant_views.xml**. Cette vue, nommée view_cyberware_implant_form, est essentielle pour organiser les nombreuses informations d'un produit.

Accès à la vue sur le site, vous pouvez essayer de créer un nouvel implant grâce au bouton dédié de la page d'accueil.

Il existe un champ recherche personnalisé.

J'ai optimisé l'expérience utilisateur en définissant une vue de recherche dédiée dans le fichier **views/cyberware_implant_views.xml**. Cette vue, nommée view_cyberware_implant_search, est essentielle pour faciliter le tri et la localisation des implants dans le catalogue.

Accès à la vue sur le site : La vue est accessible dans l'interface Odoo via la barre de recherche et les menus Filtres et Regrouper par, qui apparaissent lorsque l'on navigue vers le menu Cyberware depuis la page Implants.

Figure 3 : Localisation des filtres de recherche personnalisé

J'ai configuré des filtres rapides pré-définis, comme le filtre "Légendaire" ou le filtre "Abordable (< 5000€)". De plus, l'option Grouper par permet à l'utilisateur de classer dynamiquement le catalogue par Type d'implant, Rareté ou Emplacement corporel, ce qui est particulièrement utile avec la vue Kanban.

Il existe une vue calendrier ou une vue kanban.

J'ai choisi d'intégrer des **vues Kanban** dans le module, car elles offrent un affichage plus visuel et intuitif que les listes classiques. J'en ai créé une pour les implants **view_cyberware_implant_kanban** et une pour les clients **view_cyberware_client_kanban**.

Accès à la vue sur le site : Ces vues visibles sur la page d'accueil, via la sélection Kanban.

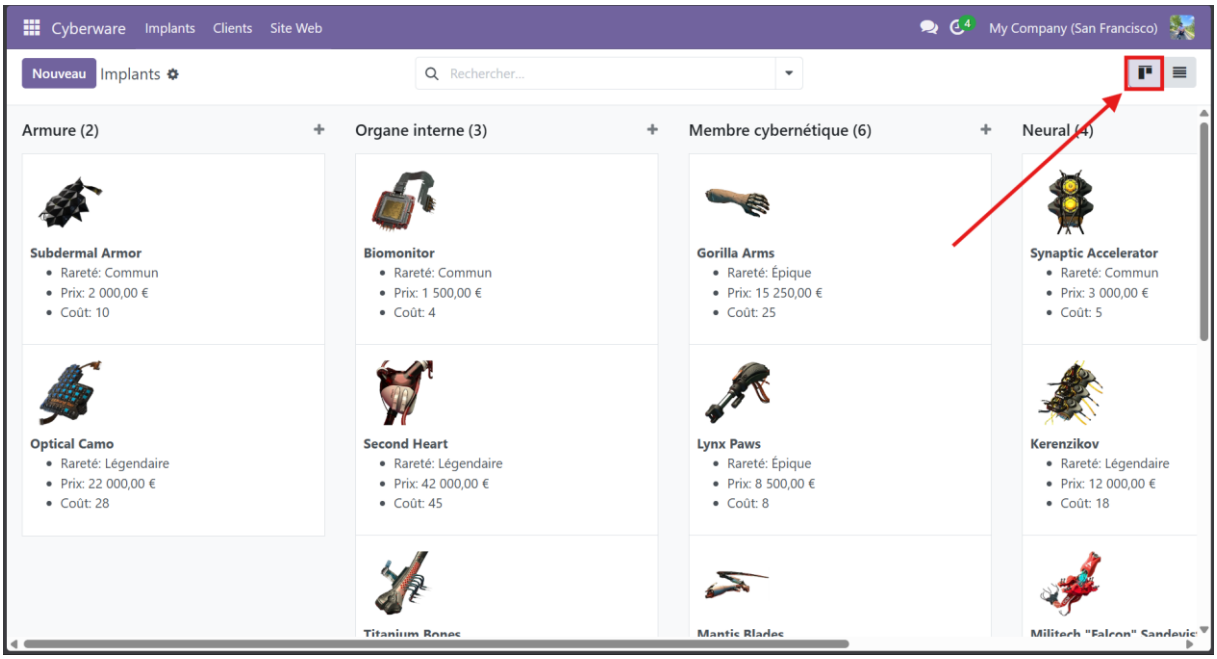

Figure 4 : Accéder a la vue Kanban

Une page web est disponible.

J'ai créé une page web qui sert de catalogue public pour présenter les implants disponibles aux visiteurs. Le contrôleur Python gère la route et récupère les données. Le fichier **controllers/cyberware_controller.py** définit spécifiquement la route publique à l'URL /cyberware/market. Le template QWeb (**views/cyberware_website_templates.xml**) génère ensuite le code HTML pour afficher le catalogue sous forme de grille.

Deux méthodes d'accès sont disponibles :

- L'interface publique est directement consultable via l'adresse http://localhost:8069/cyberware/market.
- J'ai configuré une action de type URL, qui est liée à un bouton dans le menu principal d'Odoo, à côté de l'application Cyberware. Ce lien ouvre le catalogue dans le navigateur, permettant un accès facile depuis le backend.

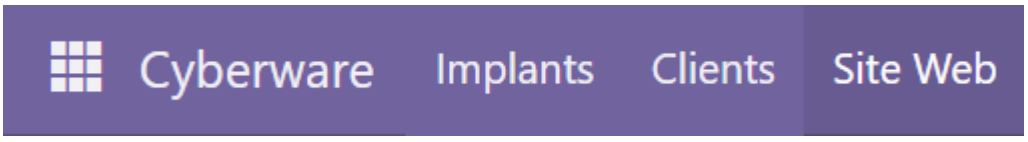

Figure 5 : Localisation bouton Site Web

Dans une des vues on trouve au moins un exemplaire de tous les champs exigés dans les consignes sur les modèles.

J'ai configuré la **vue formulaire du client** (view_cyberware_client_form) dans le fichier *views/cyberware_client_views.xml* pour qu'elle regroupe l'intégralité des types de champs exigés, offrant ainsi une démonstration technique complète sur un seul écran.

En consultant la fiche d'un client, on peut identifier :

- Un champ Image (image_client) pour l'avatar.
- Un champ Date (date_naissance).
- Un champ Calculé (age), dérivé de la date.
- Un champ Select (groupe_sanguin) avec liste déroulante.
- Un champ Related (email_user) affichant le login de l'utilisateur lié.
- Un champ Numérique (niveau_essence_max).
- Un champ Textuel (notes_medicales) pour les commentaires longs.
- Un champ Relationnel (implant_ids) pour la liste des implants.

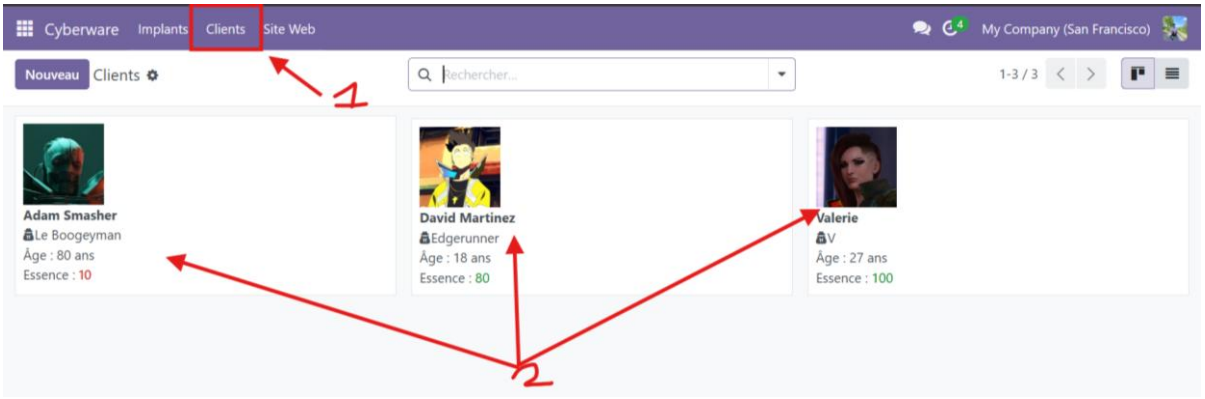

Figure 6 : Consulter la fiche d'un client

Un bouton permet d'exécuter la méthode qui contrôle les données

J'ai implémenté un mécanisme de **contrôle manuel** directement accessible depuis l'interface. J'ai ajouté un bouton **Vérifier le dossier** situé dans l'en-tête de la vue formulaire du client (fichier *views/cyberware_client_views.xml*, lien http://localhost:8069/odoo/action-486/3). Ce bouton déclenche la méthode Python action_controler_donnees définie dans le modèle cyberware.client.

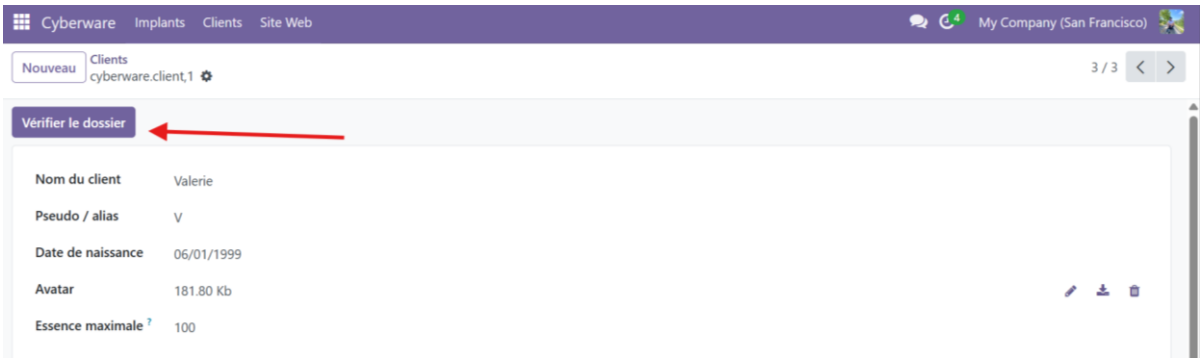

Figure 7 : Bouton pour vérifier le dossier d'un client

L'héritage du modèle étendu est utilisé dans une vue (affichage d'un champ parent ou modification d'une des vues du modèle parent)

Pour intégrer les nouvelles fonctionnalités sans altérer le code natif d'Odoo, j'ai mis en œuvre le mécanisme d'héritage de vue dans le fichier *views/res_partner_views.xml*.

Techniquement, la vue view_partner_form_cyberware étend la vue formulaire standard des contacts (base.view_partner_form).

L'implémentation inclut une logique d'affichage conditionnelle pour préserver l'ergonomie :

- Le champ booléen is_charcudoc permet d'identifier le rôle du contact.
- Le champ de spécialisation charcudoc_speciality possède l'attribut invisible="not is_charcudoc". Il reste donc masqué par défaut et n'apparaît à l'écran que lorsque l'utilisateur définit le contact comme étant un "Charcudoc".

Cette extension est visible pour n'importe quel contact.

Un ou plusieurs champs ne sont visibles que pour un groupe d'utilisateur ou en fonction de la valeur d'un autre champ.

J'ai implémenté la **visibilité conditionnelle** sur la vue étendue du modèle Contact (res.partner) pour ne pas surcharger l'interface. Cette fonctionnalité est mise en œuvre dans le fichier **views/res_partner_views.xml**.

Le champ charcudoc_speciality, qui définit la spécialité du praticien, est masqué par défaut. J'ai utilisé l'attribut invisible avec la condition invisible="not is_charcudoc". Ainsi, ce champ n'apparaît à l'utilisateur que si la case Est un Charcudoc est cochée. Cela garantit que les informations spécifiques aux médecins ne sont affichées que pour les contacts qui ont effectivement ce rôle, simplifiant l'interface pour les autres contacts.