Nathan Halioua BTS SIO A

Note de synthèse de stage

Plan :

I. Contexte

A. Présentation de l'entreprise

QuietRent est une Insurtech doublée d'une boutique de conseil immatriculée à l'ORIAS et spécialisée dans l'analyse et le scoring des risques immobiliers. Son siège social se trouve dans le IIème arrondissement de Paris, à proximité de la place de la Bourse.
QuietRent élabore des produits d'assurances innovants avec comme fil conducteur l'optimisation de trésorerie. Il s'agit d'une startup pionnière en matière de nouvelles technologies appliquées à l'assurance qui dispose d'une expertise reconnue par la presse professionnelle, assureurs, courtiers et cabinets experts en analyse de risques.

B. Présentation du projet

Le projet général est la création d'une application web de gestion des demandes de garantie immobilière des utilisateurs. Ce projet se décompose en 3 sous projets :

- un espace client (module admin-client : administration client)
- un espace super-admin (module super-admin)
- un espace courtier (module admin)

Mon travail consiste à développer le module client, c'est-à-dire un espace membre où le client aura accès à plusieurs fonctionnalités qui seront détaillées lors de cette note de synthèse.

C. Présentation de l'équipe

L'équipe informatique de l'entreprise est composée de trois personnes :

- Alexandre, stagiaire en 4ème année d'école d'ingénieur, qui a pour rôle la mise en place de l'espace courtier et l'inscription.
- Lilou, stagiaire en 4ème année d'école d'ingénieur, qui a pour rôle la création du module super-admin.
- Moi-même, Nathan, stagiaire en 1ère année de BTS SIO, qui a pour rôle la création de l'espace client.

II. Expression des besoins

A. Liste des besoins

L'application web doit fournir :

- Système de connexion
- Une page profil où l'utilisateur peut modifier ses informations personnelles ainsi que celles de ses entreprises.
- Une rubrique "Mes demandes" où l'utilisateur pourra consulter les informations relatives à ses demandes ainsi que leur statut de validation. (Détail des statuts de validation plus bas dans la note de synthèse)
- Une fonctionnalité d'upload de document de la part du client.
- Un gestionnaire des documents uploadés : consultation, suppression, validation. Si un document est validé, il n'est plus supprimable.
- Une rubrique "Mes tickets" où l'utilisateur pourra exprimer ses réclamations sur l'une de ses demandes. L'utilisateur pourra créer un nouveau ticket ou alors consulter les informations relatives à chacun des anciens tickets et être au courant de leur statut (en cours, clos ou résolu)
- Un système de notifications permettant d'informer le client des démarches à suivre dans un cas donné.
- Une rubrique "Nouvelle demande" où les données qui ne sont pas sensées changer d'une demande à l'autre sont déjà préremplies (avec tout de même la possibilité de les modifier). Cette rubrique, comme son nom l'indique, permettra à l'utilisateur de créer une nouvelle demande. Si la demande n'est pas éligible, elle n'est pas sauvegardée. L'utilisateur pourra commencer la création d'une demande et la reprendre plus tard sans la recommencer depuis le début.

B. Contraintes à respecter

- Ressource : Linux Ubuntu 20.04 recommandé.
- Ergonomique : L'application web devra être intuitive et facile d'utilisation.
- Technique : L'application devra être développée en Python avec la Framework Django. Il faudra utiliser des templates fournies par l'entreprise pour l'esthétique du site et les modifier. Ces templates utilisent les langages HTML, CSS et JavaScript, ainsi que la Framework Bootstrap.
- Fonctionnel : L'utilisateur ne peut accéder aux fonctionnalités de l'espace client si et seulement si il est connecté.

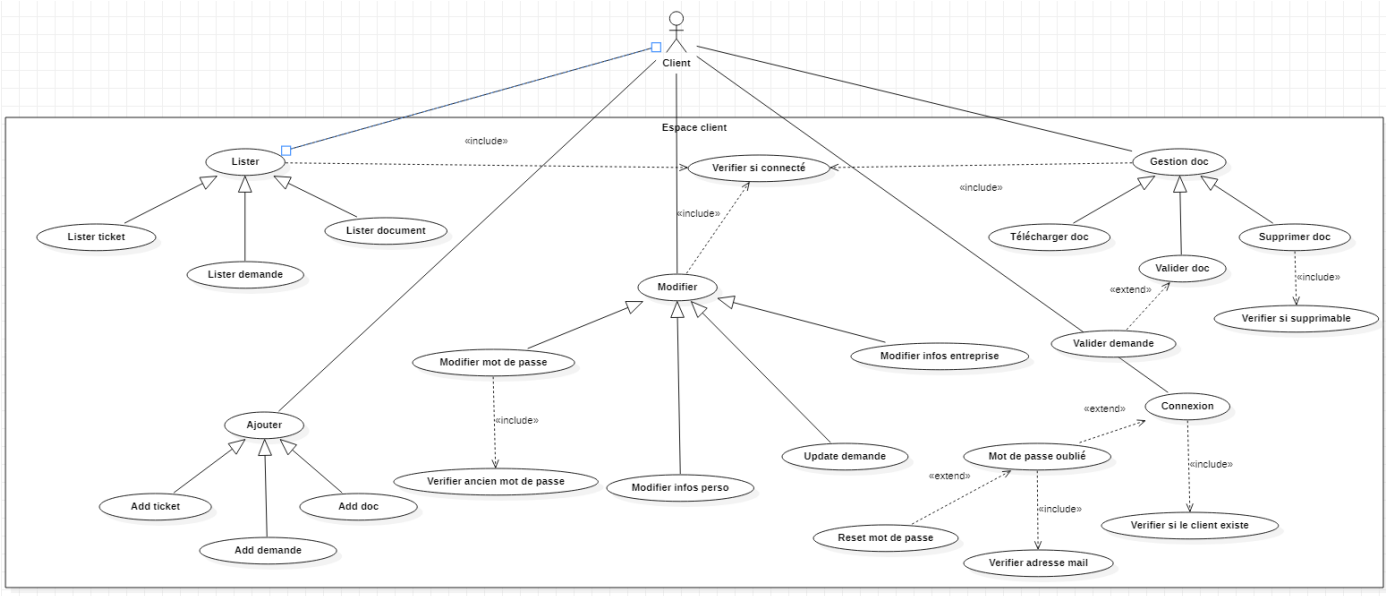
C. Définition des ressources matérielles et logicielles

- J'utilise mon ordinateur portable personnel : le Acer Nitro-5 sous Ubuntu 20.04.
- Je crée la base de donnée de zéro en utilisant le système de Models de Django
- J'utilise PyCharm de la suite JetBrains en tant qu'éditeur de code.
- J'utilise DB Browser For SQLite en tant que gestionnaire de base de données.

III. Analyse

A. Schéma des cas d'utilisations

Ce projet comporte deux types d'utilisateurs : client et admin.
Ma tâche étant de m'occuper de l'espace client, ce sont les cas d'usages du client que j'ai détaillé dans le schéma ci-dessous.



a. Connexion

Cette fonctionnalité sert à authentifier le client. Ce n'est qu'une fois connecté que le client peut accéder à son espace et ses fonctionnalités. Pour se connecter, le client doit :

- se rendre sur la page de connexion, son mail et son mot de passe lui sont demandés. Le client clique sur le bouton de connexion et est redirigé vers son profil.
  Exception : si le mail ou le mot de passe est incorrect, un message d'erreur s'affiche.
- la page dispose également d'un lien "Mot de passe oublié" qui permet au client de réinitialiser son mot de passe.
  Exception : si le mail renseigné pour réinitialiser le mot de passe ne correspond à aucun compte client, un message d'erreur s'affiche.

b. Lister

Cette fonctionnalité sert à lister les différents éléments de la base de données avec le détail des informations qui leurs sont relatives. Les 3 éléments principalement listés sont :

- les demandes du client qui sont listées sous forme d'un tableau qui indique l'id de la demande, l'adresse de la location, l'entreprise, la date d'émission de la demande ainsi que son état (les différents états de demande seront détaillés dans le prochain paragraphe). En fonction de son état, une demande peut donner accès à différents

boutons comme un bouton de téléchargement de document ou encore un bouton pour upload des documents. Une demande aura cependant toujours un bouton permettant de consulter en détail les informations de la demande.
Exception : si le client ne possède qu'une seule entreprise, celle-ci n'est pas affichée dans le tableau.

- les tickets du client qui sont listés sous forme d'un tableau qui indique le service sollicité, la demande concernée, la date d'émission du ticket, son objet ainsi que son statut (en cours, résolu ou clos). Le tableau dispose également d'un bouton pour chaque ticket qui permet de consulter le contenu de ce-dernier. Enfin, un bouton en haut à droite du tableau permet d'ajouter un nouveau ticket.

- les documents que le client upload afin de valider sa demande. Ces documents sont triés par demande et par type de document (pièce d'identité, bilan de société, bail etc…).

  Chacune de ces listes est accessible en un clic de la part de l'utilisateur qui choisit depuis le menu du site à quelle rubrique il souhaite accéder.

c. Ajouter

Cette fonctionnalité permet au client d'ajouter depuis le site de nouveaux éléments à la base de données. Les trois principaux éléments pouvant être ajoutés sont :

- les demandes : le client clique sur la rubrique du menu appelée "Nouvelle demande". C'est un lien qui le renvoie vers un formulaire lui permettant de créer sa nouvelle demande. Ce formulaire se divise en 3 étapes (qui seront détaillées plus bas dans la note de synthèse) puis d'une étape complémentaire qui est le formulaire d'éligibilité.
  Exception : si la demande n'est pas éligible, elle n'est pas enregistrée et le client reçoit un message l'en informant.
  Le client peut quitter l'un des questionnaires avant d'avoir terminé sa demande pour la reprendre plus tard. Pour ce faire, il doit aller dans la rubrique "Mes demandes" où il trouvera sa demande en cours avec le choix de la continuer ou alors de l'annuler.

- les tickets : le client clique sur le bouton "Nouveau ticket" disponible dans la rubrique "Mes tickets". Une pop-up lui propose alors de remplir un formulaire pour créer son ticket. Le client doit alors renseigner le service ainsi que la demande concernée, le sujet ainsi que l'objet de sa réclamation. Enfin, il peut rédiger son ticket et l'envoyer.

- les documents : pour ajouter un document, le client doit aller dans la rubrique "Mes demandes". Si une de ses demandes est à l'état "documents manquants", un bouton pour accéder à la page d'upload de documents sera disponible. Il peut également le faire depuis la rubrique "Mes docs". Le client n'a plus qu'à cliquer dessus pour y accéder. Une fois sur la page d'upload, un formulaire d'upload de fichier s'affiche, le client choisit les documents qu'il souhaite upload puis les soumets.
  Exception : si le document n'est pas au format image ou pdf ou alors qu'il dépasse la taille maximale indiquée, l'upload est refusé et un message d'erreur s'affiche.

d. Modifier

Cette fonctionnalité permet au client de modifier certains éléments de la base de données. Le client clique sur la rubrique “Profil”, ainsi il accède au profil client qui comporte une série de formulaire :

- un formulaire pour modifier ses informations personnelles (N°Tél, E-mail, nom, prénom etc…)
- un ou plusieurs formulaires pour modifier les informations de son ou ses entreprises
- un formulaire pour changer son mot de passe. L’utilisateur doit pour cela renseigner au préalable son ancien mot de passe.
  Exception : si le mot de passe renseigné comme étant l’ancien est incorrect, un message d’erreur est affiché.

Le client peut également modifier une demande qui est en cours de complétion, c’est-à-dire une demande qu’il n’a pas encore terminé. Pour ce faire, il lui suffit de cliquer sur la rubrique “Mes demandes” puis de cliquer sur le bouton “Continuer ma demande” afin de poursuivre ses formulaires.

e. Gestion de documents

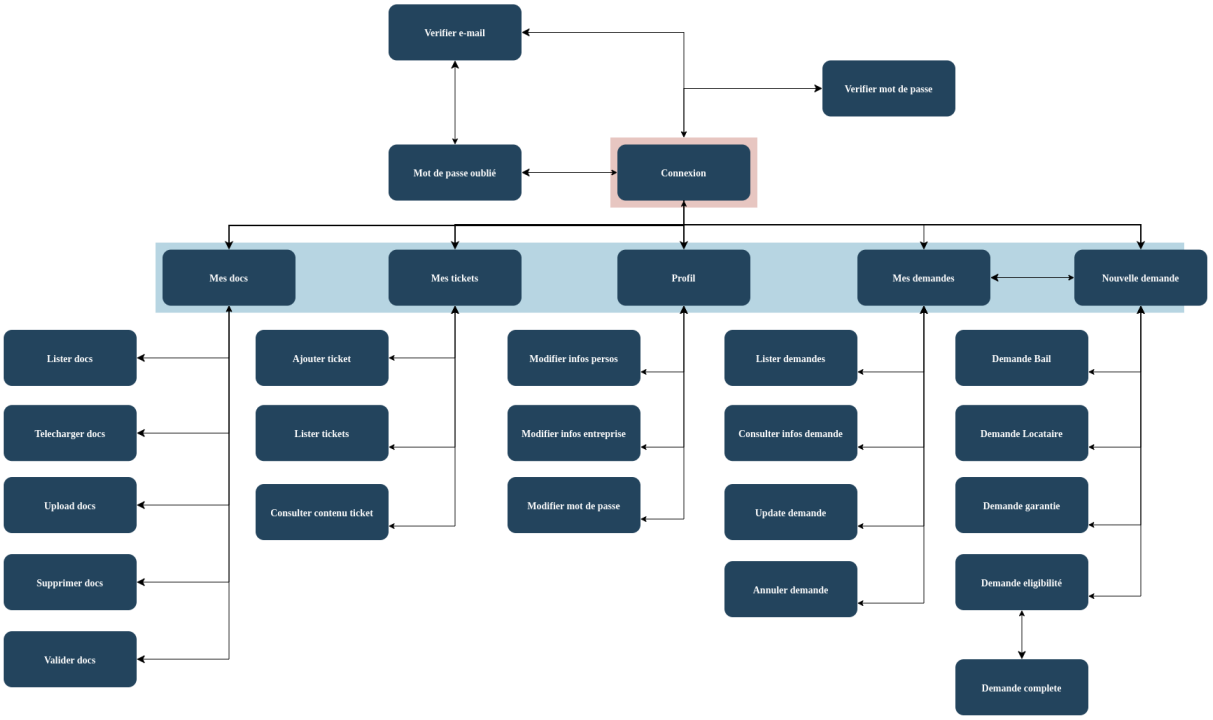
Cette fonctionnalité permet au client d’ajouter, télécharger, supprimer ou encore valider des documents. Comme dit plus haut, les documents sont triés en fonction de la demande qui leur est associée dans la rubrique “Mes docs” :

- pour chaque document, un bouton “Télécharger” est disponible.
- si le document n’est pas encore validé, un bouton "Supprimer" est disponible.
- il y a également un bouton “Upload” pour ajouter un document à la demande sélectionnée
- ainsi qu’un bouton “Valider mes documents” afin de valider l’envoi des documents. Une fois validés, les documents ne sont plus supprimables.
- Une fois tous les documents nécessaires validés, la demande est validée.

B. Spécification des IHM

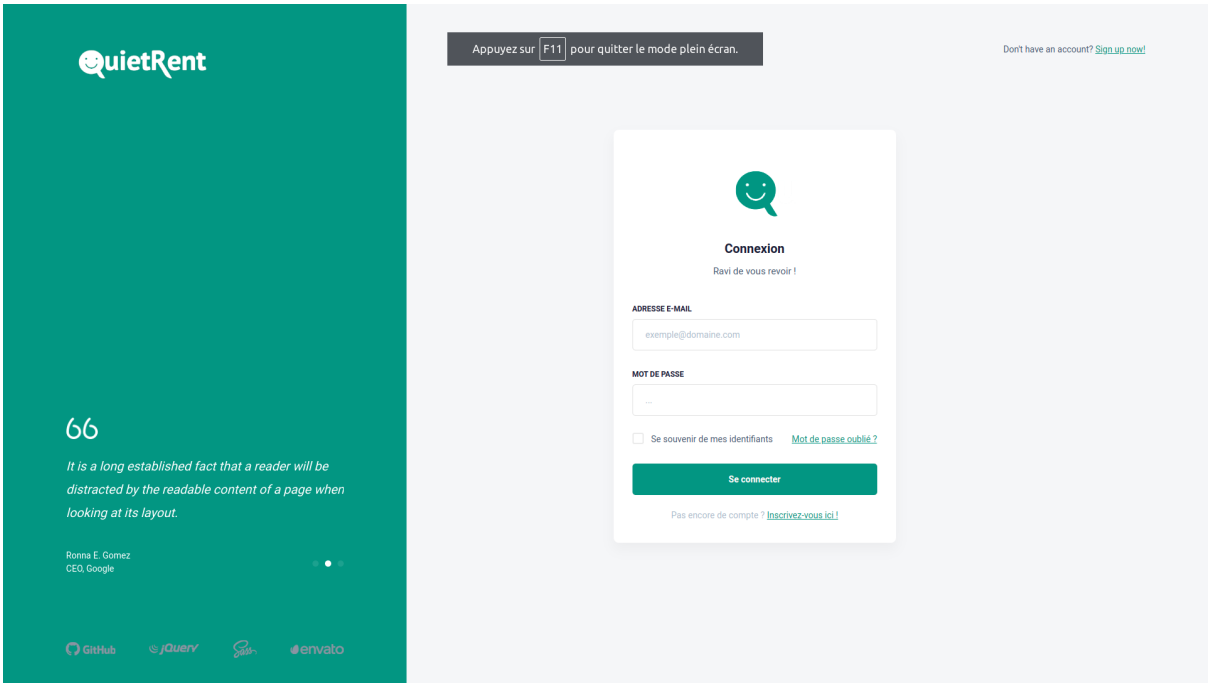
a. Arborescence des pages du site web

Ayant utilisé le Framework Django lors du développement de mon projet (Framework dont j'expliquerai le fonctionnement plus bas dans la note de synthèse) et dont le fonctionnement diffère d'un Modèle Vue Contrôleur classique, j'ai décidé de créer une arborescence des "Views" principales qui font en fait office de contrôleurs. En effet chaque View correspond à une page du site accessible depuis l'URL associée à cette View :

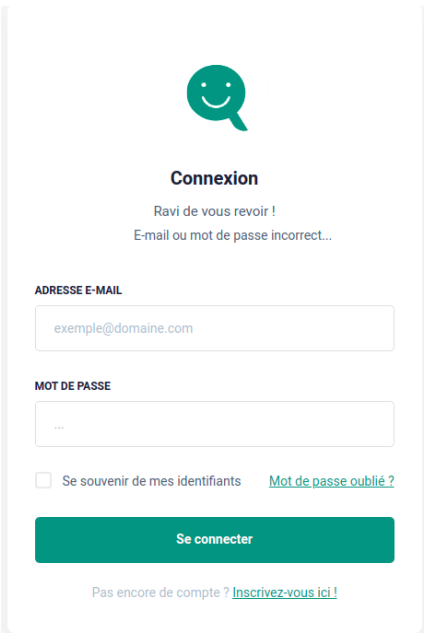


b. Exemples de pages du site web

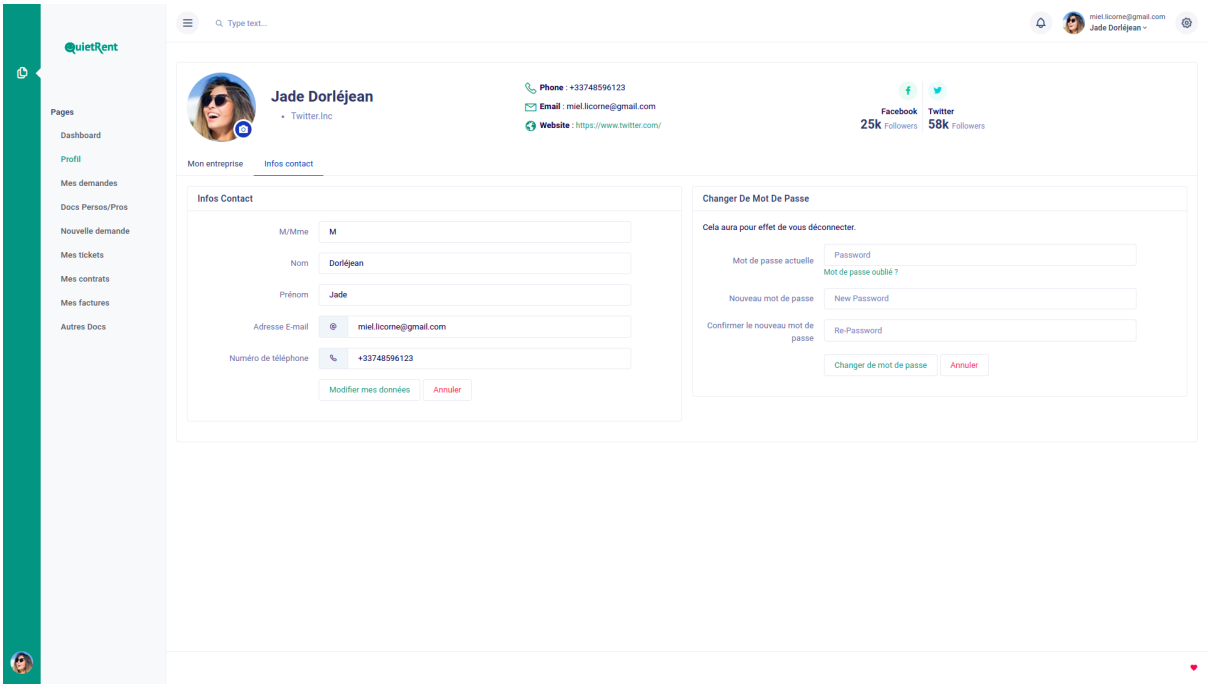
Page “Connexion”

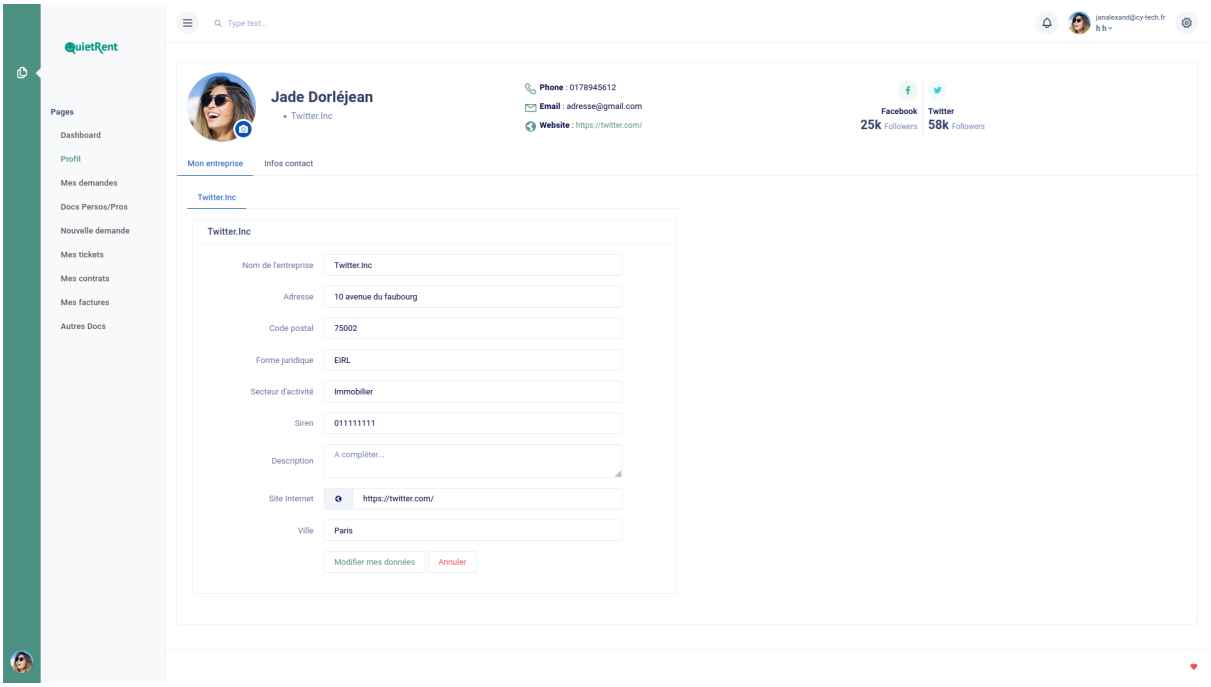


Cette page permet au client de se connecter à son espace en renseignant son adresse mail ainsi que son mot de passe.
Si l’une de ces deux informations est incorrecte, un message d’erreur s’affiche :



Page “Profil”



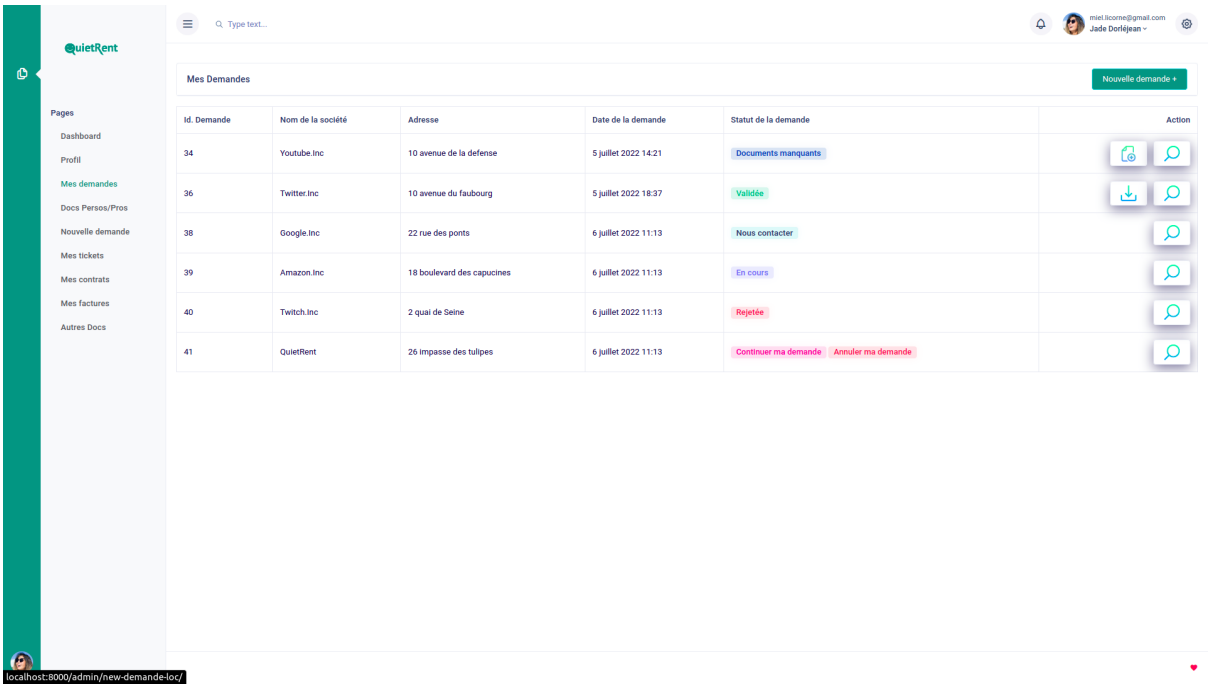


Sur cette page, le client a accès à trois différents formulaires :

- le premier lui permet de changer ses informations personnelles
- le second lui permet de changer son mot de passe
- le dernier lui permet de changer les informations de son entreprise

Il dispose également d’un affichage de ses informations.

Page “Mes demandes”

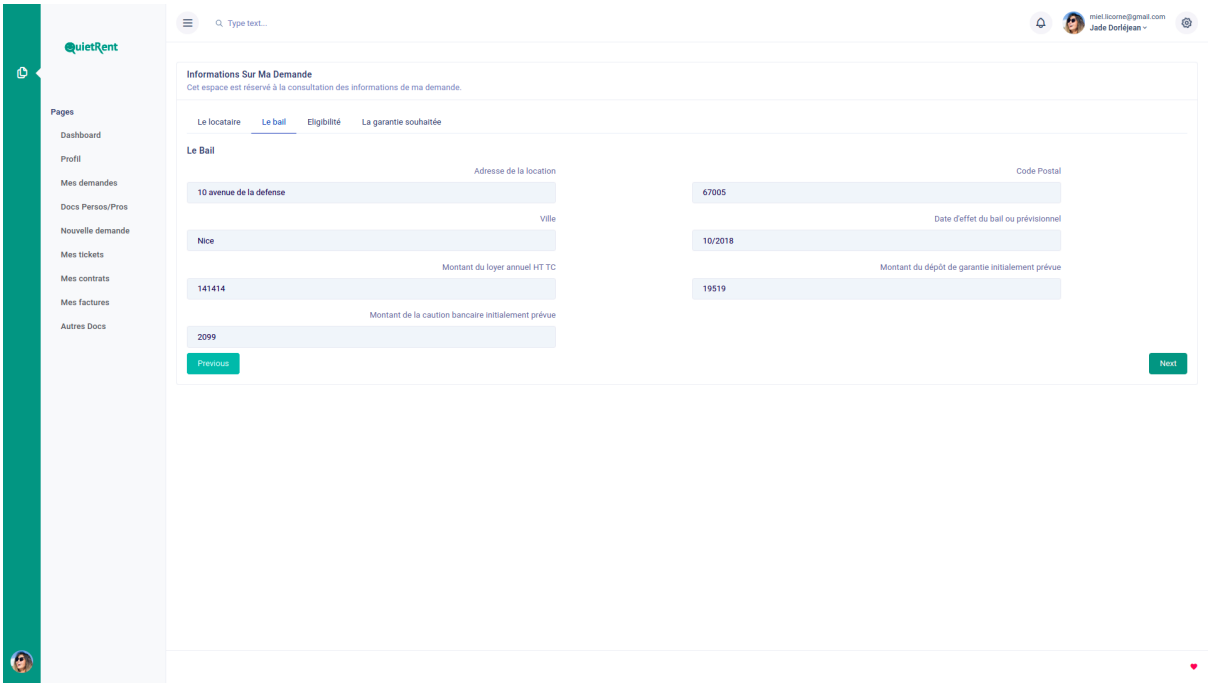


Cette page permet au client d’examiner les informations et l’état de ses demandes. Les actions possibles sur une demande changent en fonction de l’état de la demande. Ces états sont :

- Rejetée (lorsque la demande n’est pas validée par l’admin)
- Validée (lorsqu’elle est validée par l’admin). Le client peut alors télécharger son contrat depuis le bouton créé à cet effet à droite de la demande.
- Documents manquants indique au client qu’il ne lui reste plus qu’à upload des documents afin de valider sa demande. Il peut accéder à la page d’upload de documents depuis le bouton prévu à cet effet à droite de sa demande.
- En cours indique au client que sa demande est prête à être validée et qu’elle attend la validation finale de la part de l’admin.
- Dans le cas où le client avait mis en pause la complétion d’une de ses demandes, un choix s’affiche : Continuer ou Annuler ma demande. Continuer le renverra au formulaire d’éligibilité avec ses données enregistrées auparavant déjà préremplies, tandis que Annuler supprimera définitivement la demande.
- Nous contacter (dans le cas où la demande est éligible sous conditions). Le bouton “Nous contacter redirige alors vers un formulaire de contact)

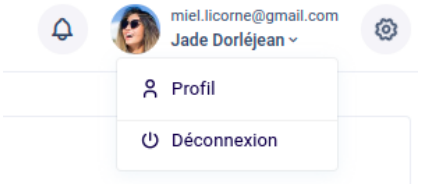
La page dispose également d’un bouton “Nouvelle demande +” qui est un lien vers le formulaire de nouvelle demande afin d’émettre une nouvelle demande.

Le bouton “Loupe” permet d’examiner en détail la totalité des informations d’une demande. Exemple :

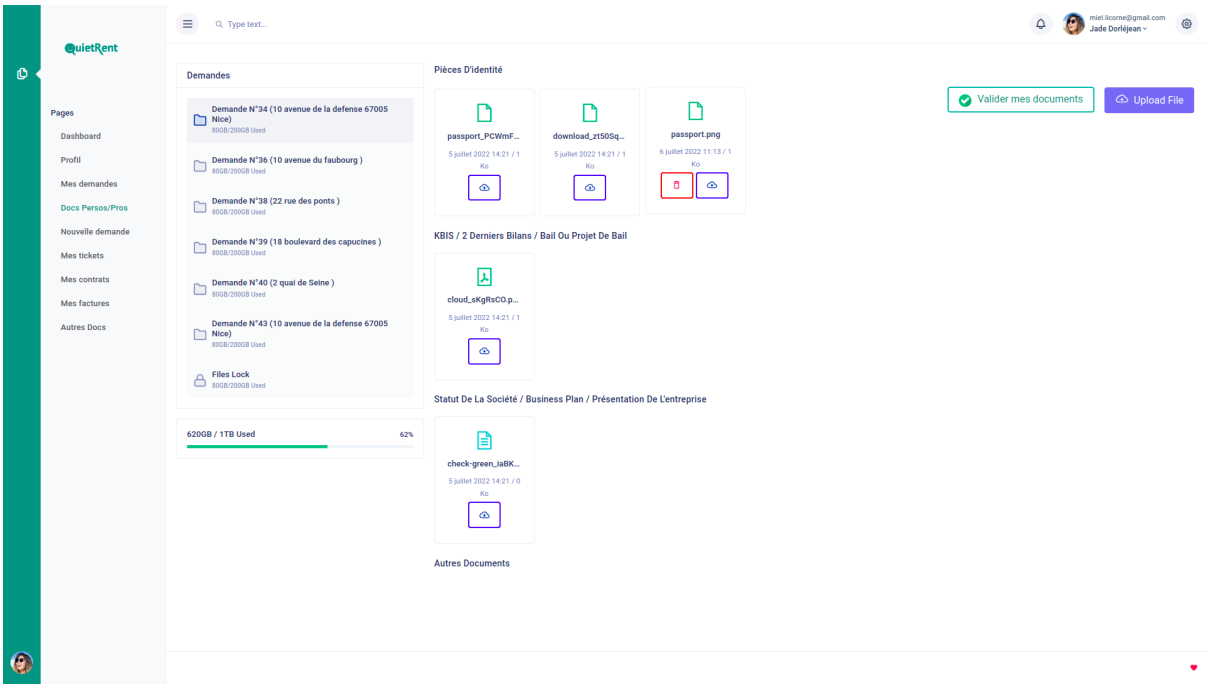


Cette consultation s'effectue sous forme de slides pour chaque type d'informations enregistrées lors de la demande.

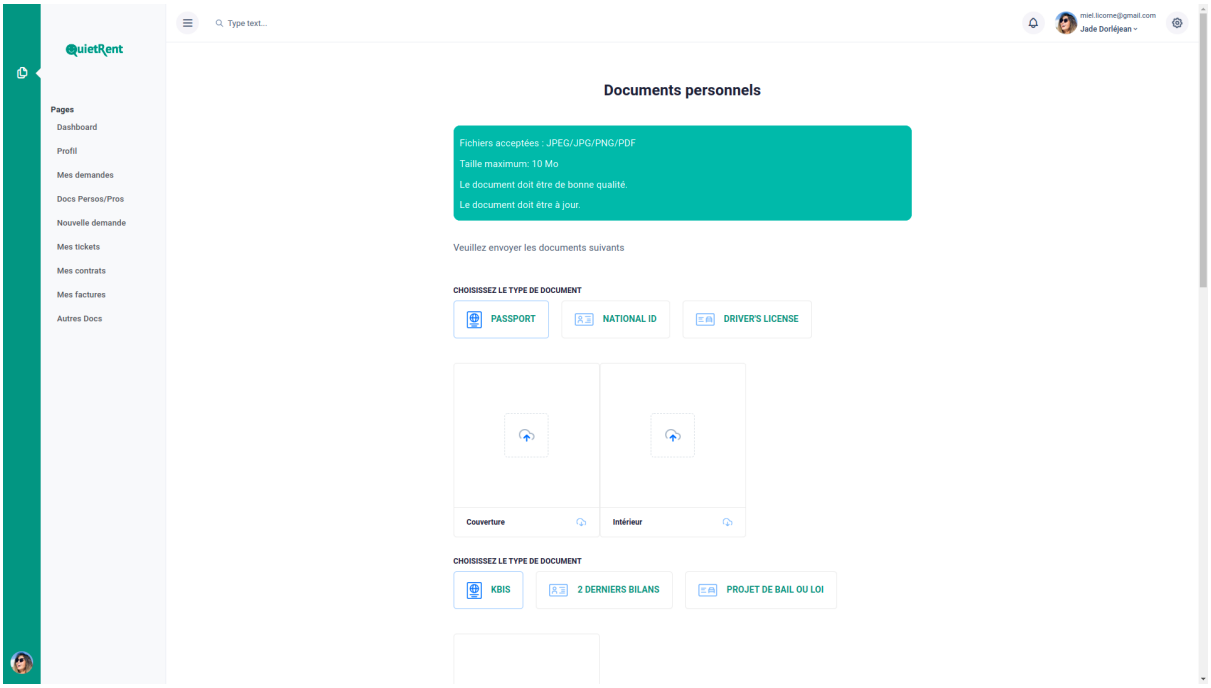
Le client peut également se déconnecter depuis le haut de page :

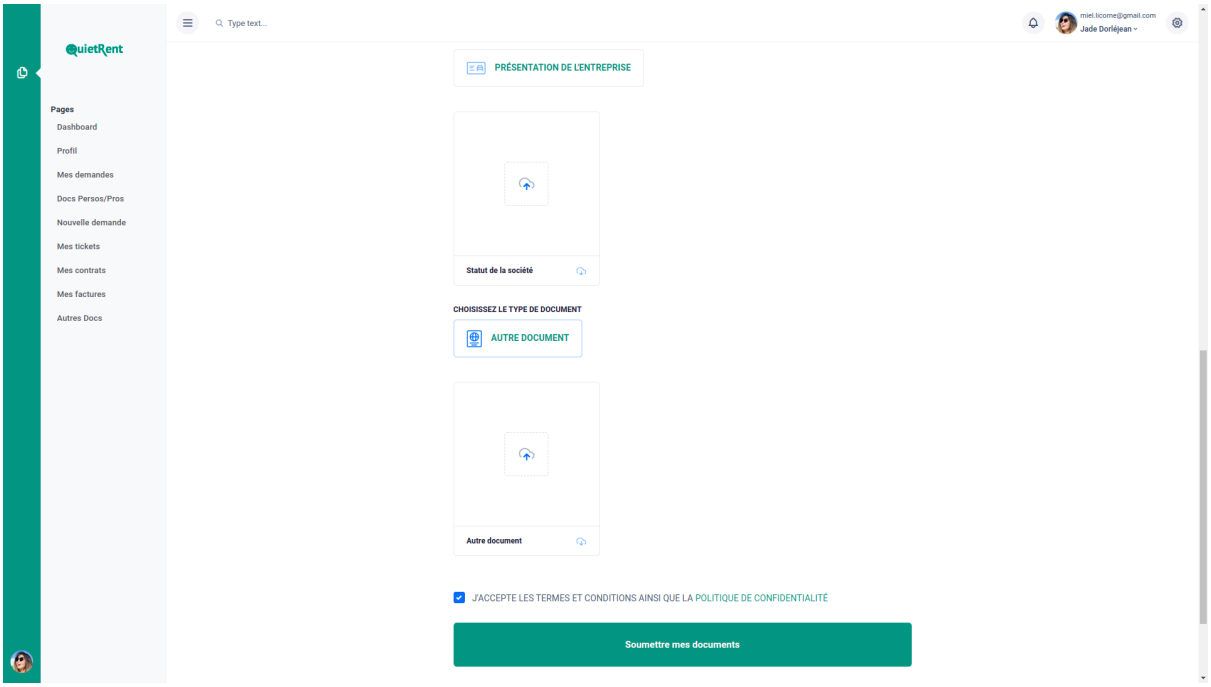


Page “Mes Docs”



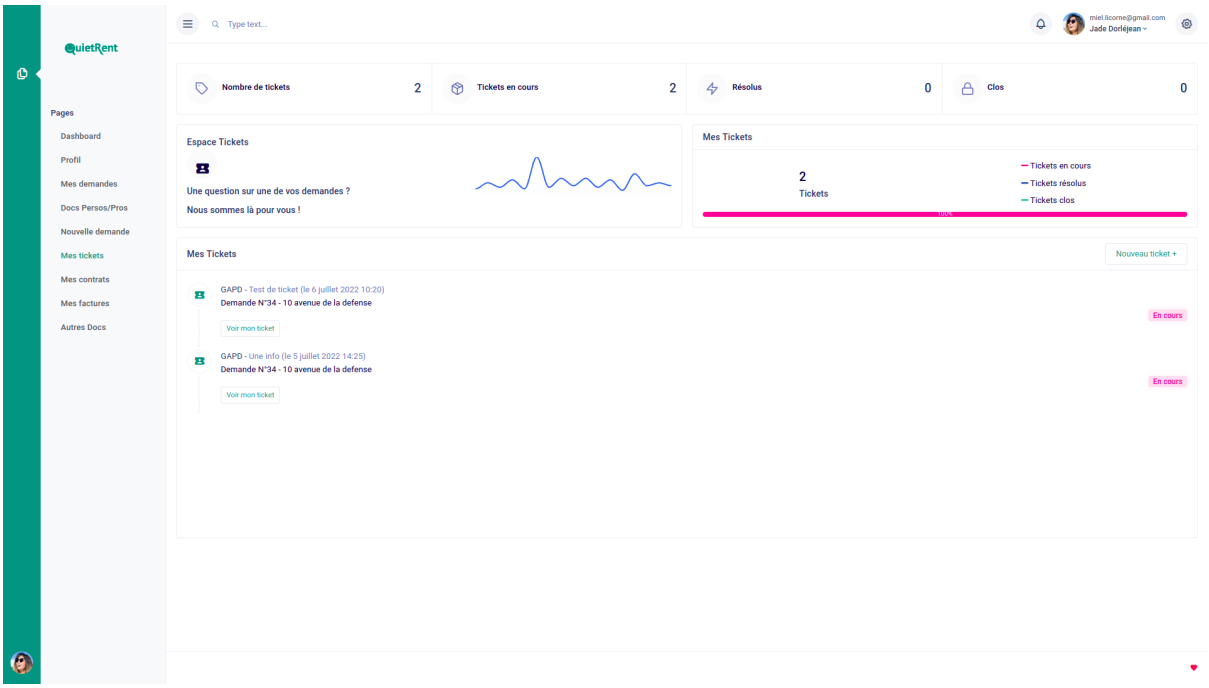
Cette page permet la consultation, la suppression et la validation des documents préalablement uploadés. Chaque document est consultable en cliquant dessus, téléchargeable en cliquant sur le bouton bleu ou supprimable en cliquant sur le bouton rouge.
Le bouton vert en haut à droite sert à valider l’envoi d’un document. Une fois validé, un document n’est plus supprimable, d’où le fait que certains documents aient le boutons supprimer alors que d’autres ne l’ont plus.
Le bouton violet “Upload File” renvoie vers la page d’upload de documents affiliée à la demande sélectionnée. Cette page s’affiche ainsi :



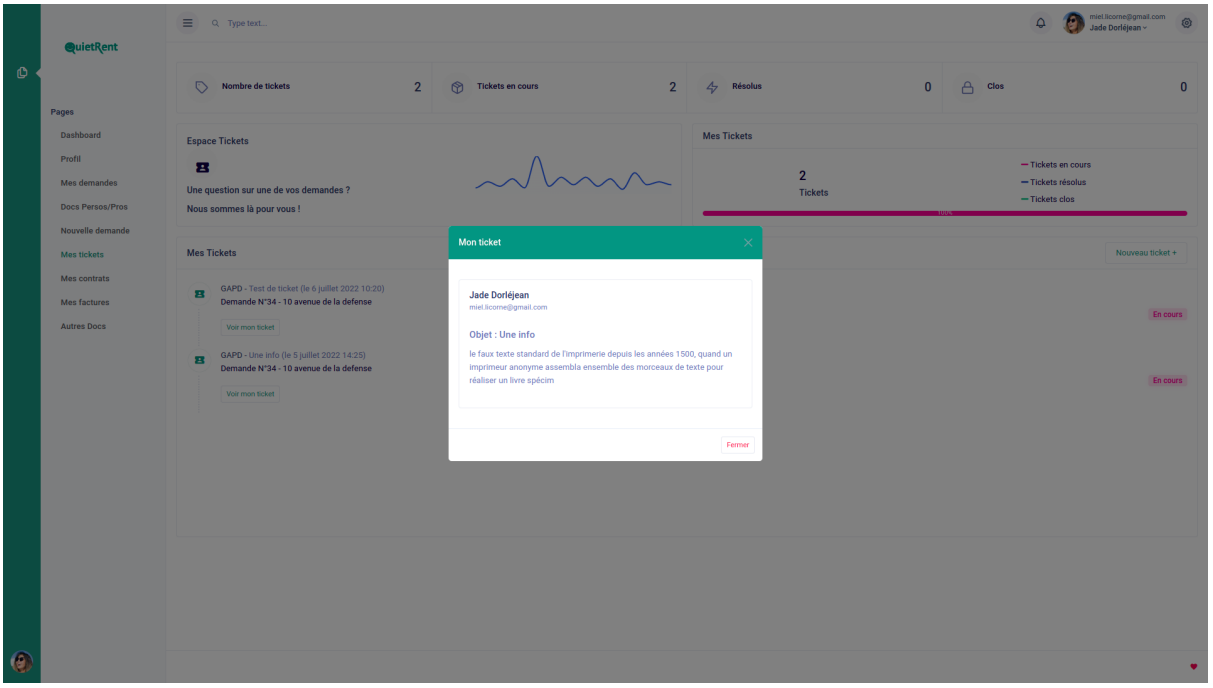


Le client doit y renseigner une liste de documents par type indiqué sur la page puis les soumettre.
Dans le cas où une des conditions n'est pas respectée, un message d'erreur s'affiche :

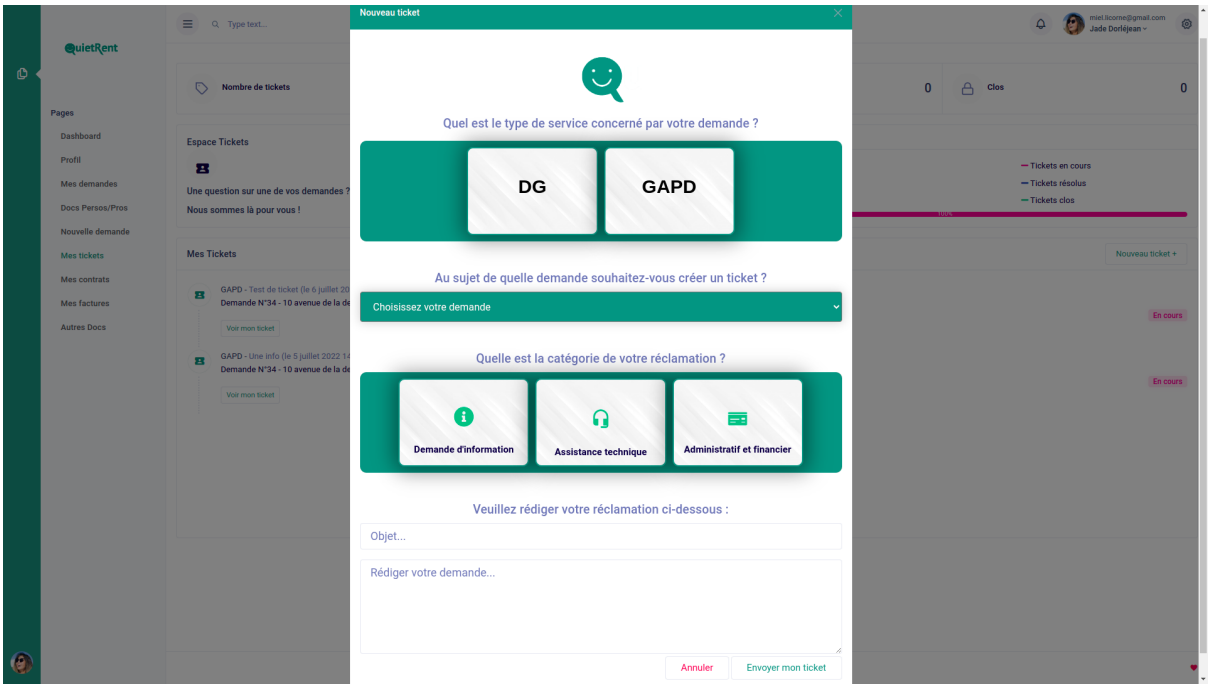
Page “Mes tickets”



Cette page permet de lister les différents tickets envoyés par le client avec les informations relatives à chaque ticket. Le nombre de tickets est afficher par état en haut de la page avec un affichage en pourcentage à droite.
Chaque ticket est accompagné d’un bouton “Voir mon ticket qui permet d’en afficher le contenu :

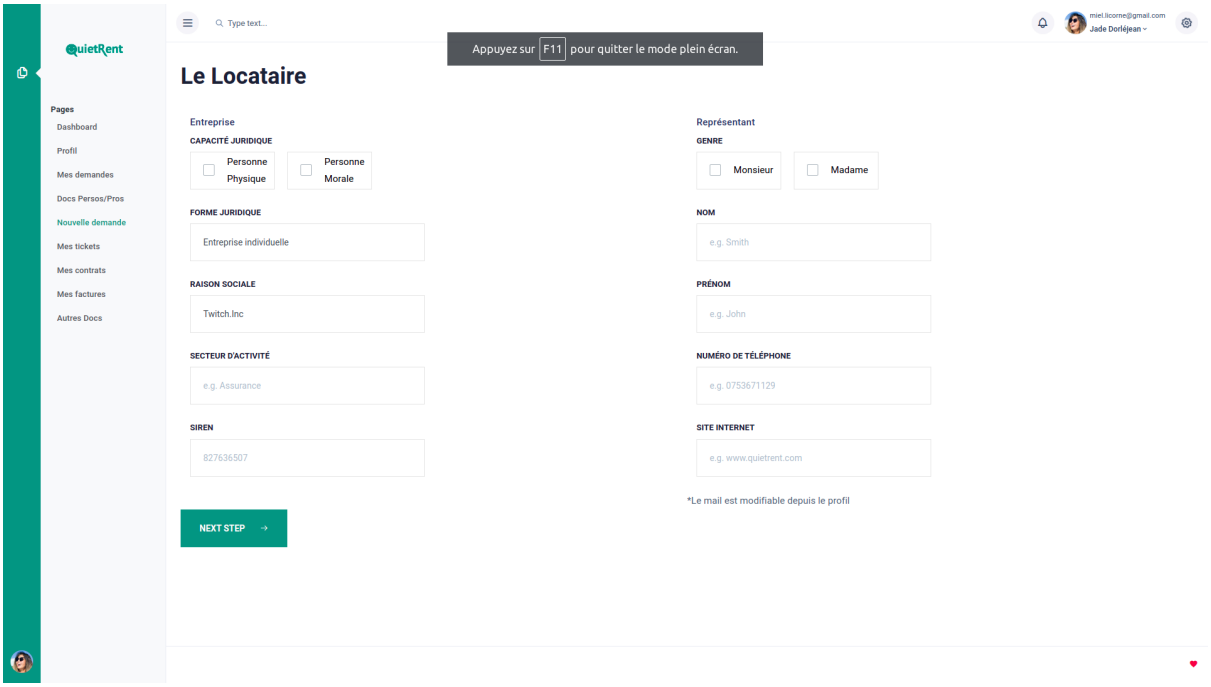


La page dispose également d’un bouton “Nouveau ticket +” permettant au client de créer un nouveau ticket. Le client devra alors renseigner quelle demande est concernée, le type de garantie évoqué, le type de service sollicité et rédiger son message accompagné de l’objet du message :

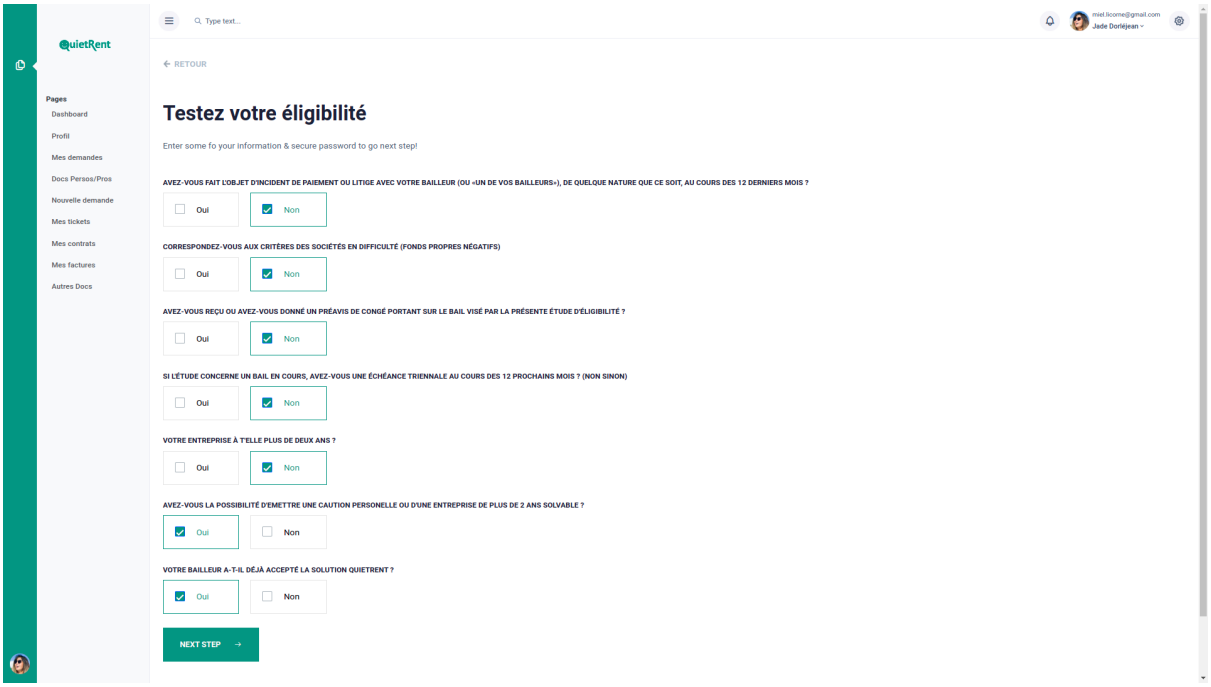


Page “Nouvelle demande”

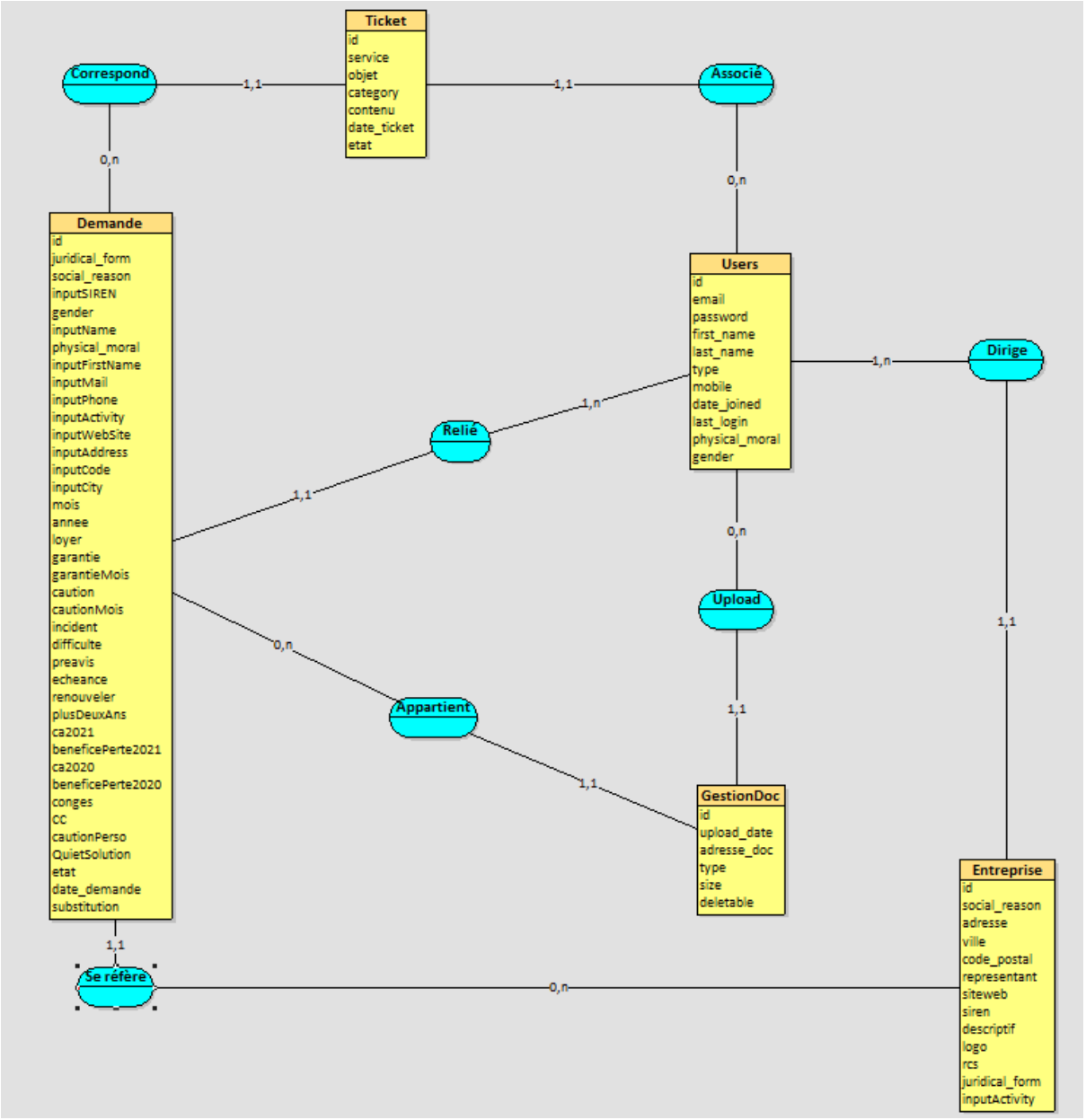
Cette page permet au client d’effectuer une nouvelle demande de garantie et s’illustre sous la forme d’une suite de questionnaire :
Exemple :



Le client doit ensuite répondre au questionnaire d’éligibilité pour déterminer si sa demande est éligible ou non :



C. Modèle conceptuel de données (MCD)



IV. Réalisation

A. Choix techniques

Langages utilisés : Python avec le Framework Django, HTML5, CSS3, JavaScript/Jquery
Editeur : PyCharm de la suite JetBrains
Templates utilisées : Metrica, Anfra (https://themeforest.net/)
Outils : DB Browser For SQLite, StarUML, Draw.io, FileZilla.

B. Spécifications techniques de l'application

a. Django et le MVT

Comme dit précédemment dans cette note de synthèse, l'application Web sur laquelle j'ai travaillé durant ce stage est codée en Python à l'aide du Framework Django. Ce Framework dispose d'une architecture d'application qui lui est spécifique : le Models Views Templates, plus communément appelé le MVT. Le MVT est assez similaire au MVC dans le sens où les tâches se divisent en trois :

- Les Models sont les tables qui composent la base de données de l'application. Ils sont rédigés en langage Python et associés à une base de données SQLite.
  Exemple du Model Ticket :

```python
class Ticket(models.Model):
    user = models.ForeignKey('User', on_delete=models.CASCADE)
    demande = models.ForeignKey('Demande', on_delete=models.CASCADE)
    service = models.CharField(max_length=255, blank=True)
    objet = models.CharField(max_length=255, blank=True)
    category = models.CharField(max_length=255, blank=True)
    contenu = models.TextField(blank=True)
    date = models.DateTimeField(default=datetime.now())
    etat = models.CharField(max_length=255)
```

  Ce sont en quelque sorte les objets qui seront traités par les fonctionnalités de l'application.

- Les Views sont les fonctions Python qui vont créer les fonctionnalités de l'application. Ce sont en quelque sorte des contrôleurs.
  Exemple de la View validate_docs :

```python
@login_required(login_url='../../register/login/')
def validate_docs(request, id_demande):
    """ Cette vue permet de valider l'ajout de documents depuis la page 'mes docs'. Une fois les documents validée, les docs ne sont plus supprimables.
    Si le nombre de docs attendus est atteint, la demande correspondante passe à l'état 'en cours'."""
    demande = mysite.models.Demande.objects.get(id=id_demande)
    if demande in mysite.models.Demande.objects.filter(id_user=request.session['id']):
        docs = mysite.models.GestionDoc.objects.filter(id_demande=id_demande)
        nb_piece_id = 0
        nb_kbis = 0
        nb_bilan = 0
        nb_bail = 0
        nb_statut_societe = 0
        request.session['id_demande_of_doc_deleted'] = id_demande
        for doc in docs:
            doc.deletable = False
            doc.save()
            if doc.type == "Piece_identite":
                nb_piece_id += 1
            elif doc.type == "Kbis":
                nb_kbis += 1
            elif doc.type == "Bilan":
                nb_bilan += 1
            elif doc.type == "Projet_de_bail_ou_LOI":
                nb_bail += 1
            elif doc.type == "Statut_societe":
                nb_statut_societe += 1

        if nb_statut_societe >= 1 and nb_bail >= 1 and nb_bilan >= 2 and nb_kbis >= 1 and nb_piece_id >= 2:
            demande.etat = 2
            demande.save()
    return redirect('../../documents/')
```

- Les Templates sont les pages HTML qui vont être affichées dans le navigateur web. On peut y envoyer des variables depuis la View associée afin de les utiliser dans le code HTML de la Template.
  Exemple d'une partie de la Template ticket-consulting :

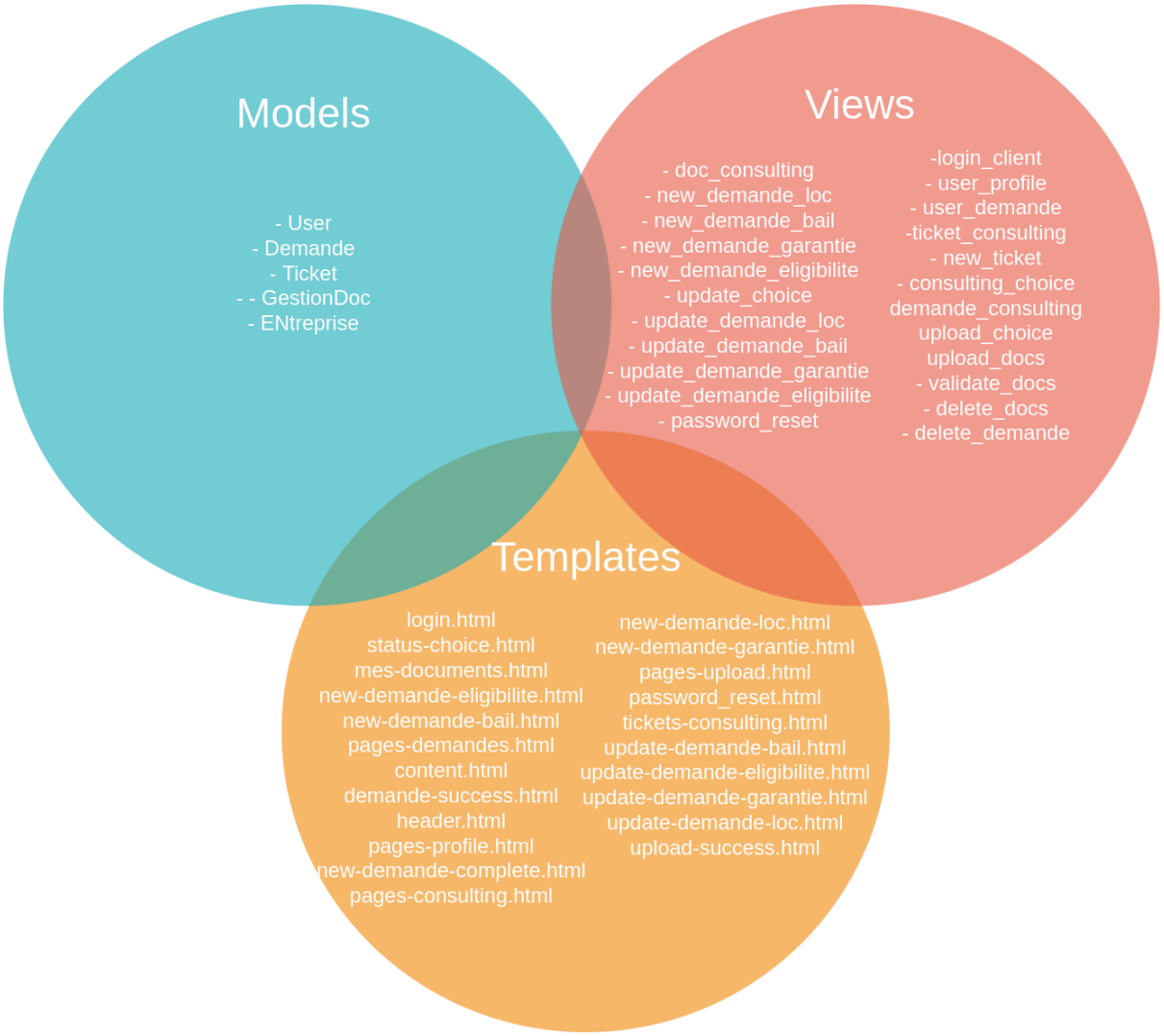
```
{% for ticket in tickets %}
<div class="modal fade" id="exampleModalCenter{{ ticket.id }}" tabindex="-1" role="dialog" aria-labelledby="exampleModalCenterTitle" aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered" role="document">
        <div class="modal-content">
            <div class="modal-header" style="background-color: #4C9082;">
                <h6 class="modal-title m-0" id="exampleModalCenterTitle">Mon ticket</h6>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div><!--end modal-header-->
            <div class="modal-body">
                <div class="card mt-3">
                    <div class="card-body">
                        <div class="media mb-4">
                            <div class="media-body align-self-center">
                                <h5 class="font-14 m-0">{{ request.user.first_name }} {{ request.user.last_name }}</h5>
                                <small class="text-muted">{{ request.user.email }}</small>
                            </div>
                        </div>

                        <h4 class="mt-0 font-15">Objet : {{ ticket.objet }}</h4>

                        <p>{{ ticket.contenu }}</p>
                    </div>
                </div>
            </div><!--end modal-body-->
            <div class="modal-footer">
                <button type="button" class="btn btn-de-danger btn-sm" data-bs-dismiss="modal">Fermer</button>
            </div><!--end modal-footer-->
        </div><!--end modal-content-->
    </div><!--end modal-dialog-->
</div><!--end modal-->
{% endfor %}
```

Une liste d'objets Tickets ont été "render" dans la Template. Cette liste est parcourue dans la Template à l'aide d'une boucle "for".

Ces trois piliers du MVT sont associés à un système d'URLs qui pointent vers les Views qui leur sont assignées.
Exemple :

```
urlpatterns = [
    path('dashboard/', views.user_dashboard, name="dashboard"),
    path('profil/', views.user_profile, name="profil"),
    path('delete_notif/<int:id_notif>/', views.delete_notif, name="delete_notif"),
    path('demande/', views.user_demande, name="demande"),
    path('contrat/', views.user_contrat, name="contrat"),
    path('facture/', views.user_facture, name="facture"),
    path('autre/', views.user_autre, name="autre"),
    path('consulting-choice/', views.consulting_choice, name="consulting-choice"),
    path('consulting/', views.demande_consulting, name="consulting"),
    path('upload-choice/', views.upload_choice, name="upload-choice"),
    path('upload/', views.upload_docs, name="upload"),
    path('validate-docs/<int:id_demande>/', views.validate_docs, name="validate-docs"),
    path('documents/', views.doc_consulting, name="documents"),
    path('documents/<int:id_doc>/', views.delete_docs, name="delete"),
    path('demande/<int:id_demande>/', views.delete_demande, name="delete-demande"),
    path('return-loc/<str:id_demande>/', views.return_loc, name="return-loc"),
    path('new-demande-loc/', views.new_demande_loc, name="new-demande-loc"),
    path('new-demande-bail/', views.new_demande_bail, name="new-demande-bail"),
    path('new-demande-garantie/', views.new_demande_garantie, name="new-demande-garantie"),
    path('new-demande-eligibilite/', views.new_demande_eligibilite, name="new-demande-eligibilite"),
    path('new-demande-complete/', views.new_demande_complete, name="new-demande-complete"),
    path('delete-demande/<int:id_demande>', views.delete_demande),
```

b. Liste de l'ensemble des fichiers composant le modèle MVT



c. Listes des Views et explication brève

```
def login_client(request):
    """ Cette vue permet de se connecter en tant que client"""
def logout_user(request):
    """ Cette vue permet de se déconnecter"""
def user_profile(request):
    """ Vue reliée à la page profil où l'utilisateur peut changer ses
infos perso et celles de ses entreprises.
    Il peut également changer de mdp."""
def user_demande(request):
    """ Vue reliée à la page qui liste les demandes de l'utilisateur,
avec leur état."""
def demande_consulting(request):
    """ Vue reliée à la page de consultation des infos d'une demande.
    Elle permet de consulter les infos de la demande sélectionnée en
fct de l'id de session instanciée
    lors du choix de la demande à consultée."""
def consulting_choice(request):
    """ Cette vue permet d'instancier l'id de session permettant la
consultation de la demande sélectionnée."""
def upload_choice(request):
    """ Cette vue permet d'instancier l'id de session permettant
l'upload de docs pour la demande sélectionnée."""
def upload_docs(request):
    """ Cette vue permet d'upload des docs en fct de la demande
sélectionnée.
    Elle contrôle le type de doc ainsi que sa taille."""
def validate_docs(request, id_demande):
    """ Cette vue permet de valider l'ajout de documents depuis la
page 'mes docs'. Une fois les documents validée, les docs ne sont
plus supprimables.
    Si le nombre de docs attendus est atteint, la demande
correspondante passe à l'état 'en cours'."""
def delete_docs(request, id_doc):
    """ Cette vue permet de supprimer des docs ajoutés. Ils sont
supprimés de la BDD ainsi que du storage."""
def delete_demande(request, id_demande):
    """ Cette vue permet de supprimer/annuler une demande en cours
d'enregistrement."""
def doc_consulting(request):
    """ Cette vue permet la consultation des documents upload."""
def new_demande_loc(request):
    """ Cette vue permet d'enregistrée les données saisies par
l'utilisateur lors de sa nouvelle demande.
    Les données de la demande précedente sont utilisées pour
préremplir celle de la nouvelle. Partie locataire."""
def new_demande_bail(request):
```

```
    """ Cette vue permet d'enregistrée les données saisies par
l'utilisateur lors de sa nouvelle demande.
    Partie bail."""
def new_demande_garantie(request):
    """ Cette vue permet d'enregistrée les données saisies par
l'utilisateur lors de sa nouvelle demande.
    Partie garantie."""
def new_demande_eligibilite(request):
    """ Cette vue permet d'enregistrée les données saisies par
l'utilisateur lors de sa nouvelle demande.
    Partie éligiblité. L'algo contrôle alors si la demande est
éligible et défini ainsi l'état de la demande.
    Si non éligible, la demande est supprimée."""
def new_demande_complete(request):
    """ Cette vue permet d'afficher un message à l'utilisateur en fct
de l'état de sa nouvelle demande."""
def update_choice(request):
    """ Cette vue permet d'instancier l'id de session en fct de la
demande à update sélectionnée."""
def update_demande_loc(request):
    """ Cette vue permet d'update la demande correspondant à l'id de
session actuel.
    Les données déjà saisies sont alors préremplies. Partie
locataire."""
def update_demande_bail(request):
    """ Cette vue permet d'update la demande correspondant à l'id de
session actuel.
    Les données déjà saisies sont alors préremplies. Partie bail."""
def update_demande_garantie(request):
    """ Cette vue permet d'update la demande correspondant à l'id de
session actuel.
    Les données déjà saisies sont alors préremplies. Partie
garantie."""
def update_demande_eligibilite(request):
    """ Cette vue permet d'update la demande correspondant à l'id de
session actuel.
    Les données déjà saisies sont alors préremplies. Partie
éligibilité."""
def tickets_consulting(request):
    """ Cette vue permet de calculer toutes les données nécessaires à
l'affichage de la page 'Mes tickets'."""
def new_ticket(request):
    """ Cette vue permet d'enregistrer les données saisies par
l'utilisateur lors de la création d'un nouveau ticket."""
```

C. Exemple de pages de code

La fonction de contrôle d'éligibilité :

```python
@login_required(login_url='../../register/login/')
def new_demande_eligibilite(request):
    """ Cette vue permet d'enregistrée les données saisies par l'utilisateur lors de sa nouvelle demande.
    Partie éligiblité. L'algo contrôle alors si la demande est éligible et défini ainsi l'état de la demande.
    Si non éligible, la demande est supprimée."""
    context = {}
    demande = mysite.models.Demande.objects.get(id=request.session['id_demande_new'])

    if request.method == 'POST':
        demande.incident = request.POST.get('incident')
        demande.difficulte = request.POST.get('difficulte')
        demande.preavis = request.POST.get('preavis')
        demande.echeance = request.POST.get('echeance')
        if request.POST.get('echeance') == "yes":
            demande.renouveler = request.POST['renouveler']
        demande.plusDeuxAns = request.POST.get('plusDeuxAns')
        if request.POST.get('plusDeuxAns') == "yes":
            demande.ca2021 = request.POST.get('ca2021')
            demande.beneficePerte2021 = request.POST.get('beneficePerte2021')
            demande.ca2020 = request.POST.get('ca2020')
            demande.beneficePerte2020 = request.POST.get('beneficePerte2020')
            demande.conges = request.POST.get('conges')
            if request.POST.get('conges') == "yes":
                demande.CC = request.POST.get('CC')
        else:
            demande.cautionPerso = request.POST.get('cautionPerso')
            demande.QuietSolution = request.POST.get('QuietSolution')
        demande.date_demande = datetime.now()
```

```python
        if demande.incident == "yes" or demande.difficulte == "yes" or demande.preavis == "yes" or (demande.echeance == "yes" and demande.renouveler == "no"):
            etat = 0

        elif demande.cautionPerso == "no" or demande.QuietSolution == "no":
            etat = 0

        elif demande.plusDeuxAns == "yes":
            benefice = demande.beneficePerte2020 + demande.beneficePerte2021
            if int(benefice) > 0 and int(demande.loyer) < 150000 and int(demande.loyer) >50000:
                etat = 3
            else:
                etat = 1

        else:
            etat = 1

        demande.etat = etat

        if demande.etat != 0:
            demande.save()
        else:
            demande.delete()
            del request.session['id_demande_new']
        return redirect('../new-demande-complete/')

    return render(request, './admin/new-demande-eligibilite.html')
```

La ligne `@login_required(login_url='../../register/login/')` permet le lancement de la View et donc de l'accès à la page associée seulement si le client est connecté. Dans le cas contraire, le client est redirigé vers l'url indiqué : la page de connexion.

```
if request.method == 'POST':
```
La vue récupère une requête HTTP puis si la requête est un POST, le traitement des données est effectué.

```
demande.beneficePerte2021 = request.POST.get('beneficePerte2021')
```
Chaque champ de formulaire récupéré est alors incrémenté à une instance de Model “Demande” afin de créer une nouvelle demande.

```
elif demande.plusDeuxAns == "yes":
    benefice = demande.beneficePerte2020 +
demande.beneficePerte2021
    if int(benefice) > 0 and int(demande.loyer) < 150000 and
int(demande.loyer) >50000:
        etat = 3
    else:
        etat = 1
```

```
else:
    etat = 1
```
Les données subissent un contrôle afin de déterminer si la demande est éligible. La demande récupère alors un “état”.

```
if demande.etat != 0:
    demande.save()
else:
    demande.delete()
    del request.session['id_demande_new']
return redirect('../new-demande-complete/')
```
Si la demande est éligible, elle est enregistrée, sauvegardée. Sinon, elle est automatiquement supprimée.

```
return render(request, './admin/new-demande-eligibilite.html')
```
Enfin, la Template, c’est-à-dire la page HTML est “render”, c’est-à-dire affichée, rendue.

La fonction ticket_consulting :

```
@login_required(login_url='../../register/login/')
def tickets_consulting(request):
    """ Cette vue permet de calculer toutes les données nécessaires à l'affichage de la page 'Mes tickets'."""
    context = {}
    context['tickets'] = mysite.models.Ticket.objects.filter(user=mysite.models.User.objects.get(id=request.session['id'])).order_by('-date')
    context['open_tickets'] = mysite.models.Ticket.objects.filter(user=mysite.models.User.objects.get(id=request.session['id']), etat='1')
    context['closed_tickets'] = mysite.models.Ticket.objects.filter(user=mysite.models.User.objects.get(id=request.session['id']), etat='0')
    context['done_tickets'] = mysite.models.Ticket.objects.filter(user=mysite.models.User.objects.get(id=request.session['id']), etat='2')

    if context['tickets'].count() > 0:
        context['done_percent'] = 100 * context['done_tickets'].count() // context['tickets'].count()
        context['closed_percent'] = 100 * context['closed_tickets'].count() // context['tickets'].count()
        context['open_percent'] = 100 * context['open_tickets'].count() // context['tickets'].count()
    else:
        context['done_percent'] = 0
        context['closed_percent'] = 0
        context['open_percent'] = 0

    context['demandes'] = mysite.models.Demande.objects.filter(id_user=request.session['id'])
    return render(request, './admin/tickets-consulting.html', context)
```

Une View permet également d'envoyer des variables vers la Template de la façon suivante :
`context = {}`
Un dictionnaire "context" est créé et on le remplit avec des variables qu'on nomme.
`context['tickets'] = mysite.models.Ticket.objects.filter(user=mysite.models.User.objects.get(id=request.session['id'])).order_by('-date')`
Ici la variables "tickets" contient la liste des tickets filtrée par client et ordonnée du ticket le plus récent au ticket le plus ancien. Ceci permet l'affichage des tickets dans la Template comme ceci :

```
{% for ticket in tickets %}
<div class="modal fade" id="exampleModalCenter{{ ticket.id }}" tabindex="-1" role="dialog" aria-labelledby="exampleModalCenterTitle" aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered" role="document">
        <div class="modal-content">
            <div class="modal-header" style="background-color: #4C9082;">
                <h6 class="modal-title m-0" id="exampleModalCenterTitle">Mon ticket</h6>
                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div><!--end modal-header-->
            <div class="modal-body">
                <div class="card mt-3">
                    <div class="card-body">
                        <div class="media mb-4">
                            <div class="media-body align-self-center">
                                <h5 class="font-14 m-0">{{ request.user.first_name }} {{ request.user.last_name }}</h5>
                                <small class="text-muted">{{ request.user.email }}</small>
                            </div>
                        </div>

                        <h4 class="mt-0 font-15">Objet : {{ ticket.objet }}</h4>

                        <p>{{ ticket.contenu }}</p>
                    </div>
                </div>
            </div><!--end modal-body-->
            <div class="modal-footer">
                <button type="button" class="btn btn-de-danger btn-sm" data-bs-dismiss="modal">Fermer</button>
            </div><!--end modal-footer-->
        </div><!--end modal-content-->
    </div><!--end modal-dialog-->
</div><!--end modal-->
{% endfor %}
```

V. Conclusion

A. Problèmes rencontrés et solutions trouvées.

Un des premiers problèmes rencontrés fut le fait que la fonction authenticate() de Django qui permet l'authentification d'un utilisateur vérifiait le "username" et le mot de passe. Or, le client désirait une authentification/connexion à l'aide d'une adresse mail. J'ai donc dû customiser le Model User par défaut afin de changer ce paramètre.
Un deuxième problème rencontré a été le fait que lorsque j'utilisais des URLs qui contenaient des données en GET du type :

```
path('documents/<int:id_doc>/', views.delete_docs, name="delete"),
```

les autres liens présents sur la page devenaient obsolètes à cause du fait que les retour "../" ne prenait pas en compte le "/param/" en plus.
Pour contourner ce problème, j'ai fait passer ces données par variables de sessions de la manière suivante :

```
@login_required(login_url='../../register/login/')
def upload_choice(request):
    """ Cette vue permet d'instancier l'id de session permettant l'upload de docs pour la demande sélectionnée."""
    if request.method == 'POST':
        request.session['id_demande_upload'] = request.POST['choice']
        request.session['id_demande_of_doc_deleted'] = int(request.POST['choice'])
        logging.error(request.session['id_demande_of_doc_deleted'])
        return redirect('../upload/')
    else:
        return redirect('../demande/')
```

Lorsque le client choisit à quelle demande il va ajouter un document, il est redirigé vers une View qui instancie en session l'id de la demande choisie. Il est ensuite redirigé vers la page d'upload de documents. Ceci permettra de relier l'upload du doc à la bonne demande.

B. Le produit a-t-il été mis en production ?

Oui, il est pour l'instant accessible à l'adresse suivante : http://149.202.84.203/index/
puis sûrement à terme sur https://www.quietrent.com/

C. Suites envisagées

Une des suites envisageables est de distinguer deux types de client différents : les partenaires et les clients. Pour l'instant, les deux sont confondus mais l'utilisateur a déjà le choix de se connecter en tant que partenaire ou en tant que client.

Pour conclure, cette première expérience dans le monde de l'entreprise m'a permis de prendre conscience de ce qu'est le métier de développeur, de mieux appréhender le travail d'équipe ainsi qu'organiser mon travail de façon réfléchie. J'ai également appris à être à l'écoute du client pour pouvoir satisfaire leurs attentes et faire au mieux. J'ai également eu la chance de m'initier au Framework Django qui est un outil très utile et optimisé.