Plateforme Team Spirit

Cahier des charges

Client
Team Spirit

Prestataire
Etienne BARBIER

Table des matières

1. Versions

Version	Motif	Auteur	Date
0.1	Création du document : spécifications fonctionnelles générales, début des spécifications techniques générales	Etienne BARBIER	25/05/2020
0.2	Spécifications techniques générales terminées User Stories génériques rédigées (US022 à US030)	Etienne BARBIER	30/07/2020
1.0	User Stories US001, US002, US007 et US008 rédigées	Etienne BARBIER	25/09/2020

2. Introduction

2.1. Objet du document

Le présent document constitue le cahier des charges fonctionnel et technique du projet « plateforme Team Spirit » (nommée dans la suite du document : « le projet »).
L'objectif de ce document est de décrire les spécifications fonctionnelles et techniques du projet.
Les éléments du présent document découlent d'échanges avec le bureau de l'association Team Spirit (nommée dans la suite du document : « le client »).

2.2. Références

Pour de plus amples informations, se référer également aux documents suivants :
- **P13_01_note_intention** : note d'intention
- **P13_02_note_cadrage** : note de cadrage

2.3. Besoin du client

2.3.1. Contexte

Le client se développe et souhaite se doter d'un outil numérique et collaboratif pour gérer et faire vivre l'association.

2.3.2. Enjeux et objectifs

Les enjeux et les objectifs du projet sont mentionnés dans la section 1 de la note de cadrage et rappelés ici :

Pour	les adhérents de l'association Team Spirit
qui	souhaitent participer à la vie de l'association,
la « plateforme Team Spirit » est	une application web
qui permet	de trouver les informations et les documents nécessaires,
à la difference de	plateformes spécialisées dans le stockage de données, l'agenda en ligne ou les réseaux sociaux,
la « plateforme Team Spirit »	centralise l'ensemble des informations dans une seule application.

3. Spécifications fonctionnelles générales

3.1. Description générale de la solution

Une solution CMS (Content Management System) de type Wordpress est exclue, car jugée pas suffisamment attractive et personnalisable en termes de design. Afin d'optimiser l'expérience utilisateur, d'assurer une maintenance efficace du code et des installations, de rendre le fonctionnement de l'application robuste avec la base de données et respecter les contraintes budgétaires du client, la solution proposée est une **application web** développée avec :

- Le langage Python (utilisé avec le framework Django) côté serveur (*back end*)
- Une base de données PostgreSQL
- Les langages HTML, CSS (utilisé avec le framework Bootstrap) et Javascript côté client (*front end*)

3.2. Les acteurs

De manière schématique et macroscopique, voici une représentation des interactions des acteurs avec le système, c'est-à-dire l'application à développer :

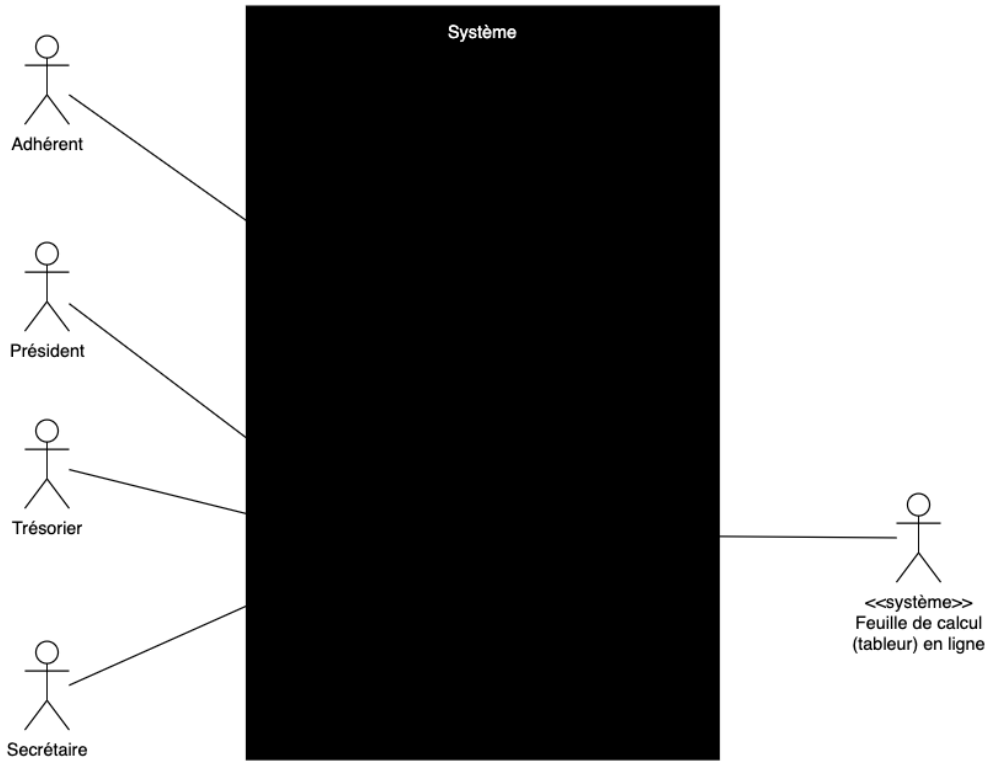

Figure 3.1 : diagramme de contexte

Les acteurs présentés dans ce diagramme de contexte sont de deux types :

- Des utilisateurs : un adhérent, le président, le trésorier, le secrétaire
- Un système externe : un système de feuille de calcul (tableur) en ligne

Il existe également deux autres types d'utilisateurs :

- L'administrateur, chargé de gérer les droits des utilisateurs
- Le développeur, chargé de concevoir et maintenir l'application

L'utilisation de l'application nécessite l'authentification de l'utilisateur.
Ainsi, un visiteur non authentifié n'a pas accès à l'application.

Voici les règles de gestion fonctionnelles, c'est-à-dire les droits attribués à ces différents profils :

Droits / Profils	Accès à la page de connexion	Fonctionnalités de base	Diffusion des informations	Gestion du budget	Administration générale de l'application
Visiteur	✓				
Adhérent	✓	✓			
Secrétaire	✓	✓	✓		
Trésorier	✓	✓		✓	
Président	✓	✓	✓	✓	
Administrateur	✓	✓			✓
Développeur	✓	✓	✓	✓	✓

Les fonctionnalités de base sont les suivantes :

- Accès aux différentes pages de l'application destinées aux adhérents :
  - Page d'accueil
  - Calendrier des événements
  - Compte-rendus de réunions
  - Organisation des entraînements
  - Catalogue des produits proposés
- Accès à son profil utilisateur

3.3. Le système

Le système peut être vu comme la combinaison de deux packages, regroupant des fonctionnalités différentes et ayant des objectifs distincts : la « vie associative » et la « gestion de l'association » :

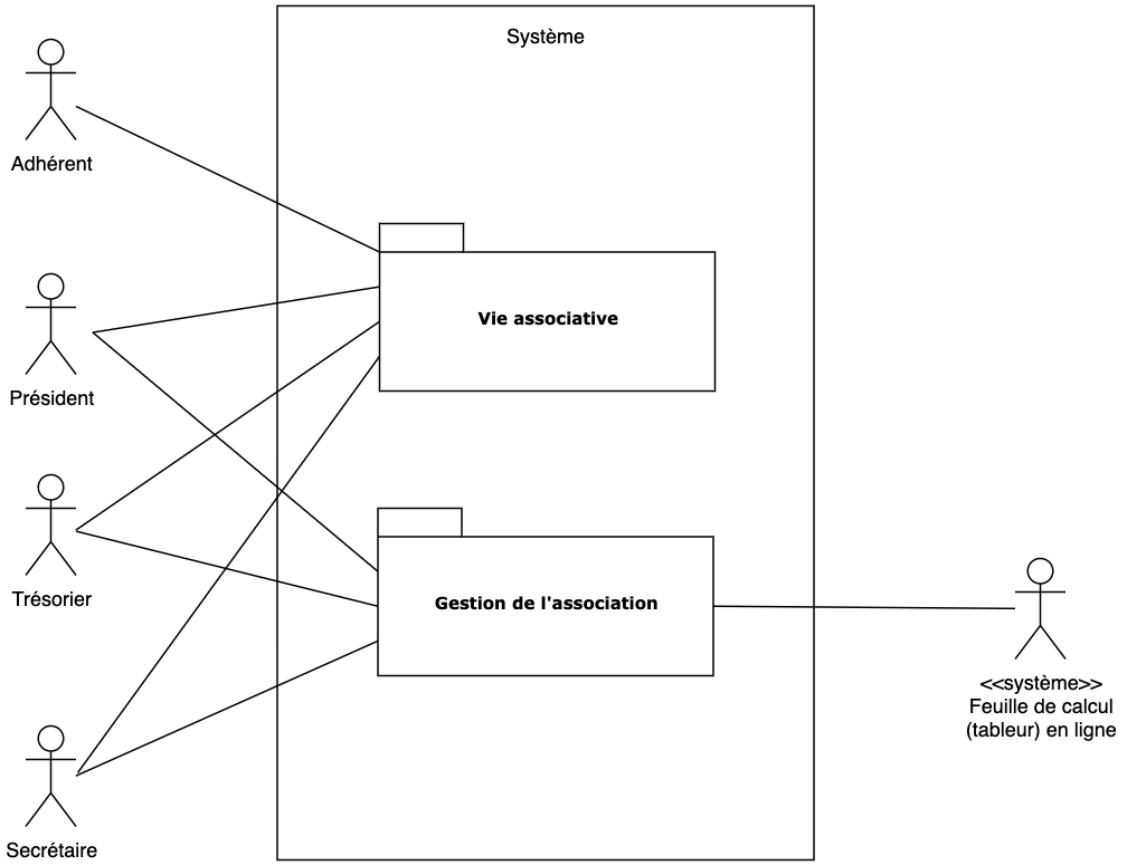

Figure 3.2 : diagramme de packages

3.4. Les fonctionnalités de l'application

3.4.1. Fonctionnalités à développer

Afin de détailler les fonctionnalités à développer dans le cadre de ce projet, deux *impact mappings* (cf. annexe B) ont été construits à partir de six *Personas* (cf. annexe A).

Voici la liste des fonctionnalités qui seront développées pour les adhérents (package « vie associative ») :

i. Renseigner ses coordonnées
ii. Poster ou supprimer un fichier sur la plateforme
iii. Renseigner des informations à destination des adhérents
iv. Consulter l'agenda
v. Télécharger ou ouvrir un document
vi. Accéder à certaines informations des adhérents
vii. Consulter la liste des articles disponibles
viii. Renseigner une pré-commande
ix. Connaître les adhérents intéressés par une pratique en groupe

Voici la liste des fonctionnalités qui seront développées pour les membres du bureau (package « gestion de l'association ») :

x. Répartir les rôles selon les tâches
xi. Accéder à toutes les tâches du bureau
xii. Mettre à jour la liste des adhérents
xiii. Contrôler le paiement des cotisations
xiv. Contrôler le statut du paiement des événements
xv. Contrôler le statut du paiement des vêtements
xvi. Contrôler le budget global
xvii. Mettre à jour l'agenda
xviii. Diffuser les compte-rendus de réunions
xix. Supprimer des documents
xx. Récupérer les documents des adhérents

A ces fonctionnalités spécifiques, s'ajoutent des fonctionnalités génériques :

xxi. Accéder à la page de connexion
xxii. Accéder à la page d'accueil
xxiii. Accéder à chaque page de l'application
xxiv. Accéder à la page de contact
xxv. Accéder à la page des mentions légales
xxvi. Modifier son mot de passe
xxvii. Réinitialiser son mot de passe
xxviii. Se connecter de l'application
xxix. Se déconnecter de l'application
xxx. Ajouter un utilisateur

3.4.2. User Stories

A partir des fonctionnalités listées dans la section précédente, des *User Stories* ont été écrites, afin de couvrir les comportements possibles des utilisateurs de l'application.

3.4.2.1. User Stories du package « vie associative »

US001 - En tant que Laurent, je veux indiquer mes coordonnées pour être joignable par d'autres adhérents.

US002 - En tant que Laurent, je veux poster ou supprimer un fichier sur la plateforme pour transmettre mon certificat médical.

US003 - En tant que Laurent, je veux renseigner des informations à destination des adhérents pour proposer des créneaux d'entraînement.

US004 - En tant que Claire, je veux consulter l'agenda pour connaître les événements passés et à venir.

US005 - En tant que Claire, je veux télécharger ou ouvrir un document pour lire un compte-rendu de réunion.

US006 - En tant que Pierre, je veux accéder à certaines informations des adhérents pour rencontrer d'autres coureurs.

US007 - En tant que Pierre, je veux consulter la liste des articles disponibles pour obtenir une tenue de sport.

US008 - En tant que Pierre, je veux renseigner une pré-commande pour obtenir une tenue de sport.

US009 - En tant que Pierre, je veux connaître les adhérents intéressés par une pratique en groupe pour s'entraîner ensemble.

3.4.2.2. User Stories du package « gestion de l'association »

US010 - En tant que Emmanuel, je veux répartir les rôles pour déléguer les tâches administratives.

US011 - En tant que Emmanuel, je veux accéder à toutes les tâches du bureau pour les superviser.

US012 - En tant que Maxime, je veux modifier la liste des adhérents pour tenir une liste à jour.

US013 - En tant que Maxime, je veux contrôler le paiement des cotisations pour suivre la situation.

US014 - En tant que Maxime, je veux contrôler le paiement des événements pour suivre la situation.

US015 - En tant que Maxime, je veux contrôler le paiement des vêtements pour suivre la situation.

US016 - En tant que Maxime, je veux contrôler le budget pour le gérer.

US017 - En tant que Axel, je veux mettre à jour l'agenda pour diffuser des informations.

US018 - En tant que Axel, je veux diffuser les compte-rendus de réunion pour informer les adhérents.

US019 - En tant que Axel, je veux supprimer des documents pour simplifier la communication.

US020 - En tant que Axel, je veux accéder aux documents des adhérents pour capter des informations.

US021 - En tant que Maxime, je veux ajouter un utilisateur.

3.4.2.3. User Stories génériques

US022 - En tant que Toto, je veux accéder à la page de connexion pour me connecter à l'application.

US023 - En tant que Toto, je veux me connecter à l'application.

US024 - En tant que Toto, je veux accéder à la page d'accueil pour naviguer sur le site.

US025 - En tant que Toto, je veux accéder à chaque page de l'application pour naviguer sur le site.

US026 - En tant que Toto, je veux accéder à la page de contact pour communiquer avec le bureau de l'association.

US027 - En tant que Toto, je veux accéder à la page des mentions légales pour le plaisir.

US028 - En tant que Toto, je veux modifier mon mot de passe.

US029 - En tant que Toto, je veux réinitialiser mon mot de passe.

US030 - En tant que Toto, je veux me déconnecter de l'application.

Lors du développement de l'application, les User Stories seront développées une à une, décrivant ainsi :

- Les parcours utilisateurs
- La documentation technique
- Les tests fonctionnels (également appelés tests d'acceptation) permettant de les valider

4. Spécifications techniques générales

4.1. Domaine fonctionnel

Voici le diagramme de classes du projet :

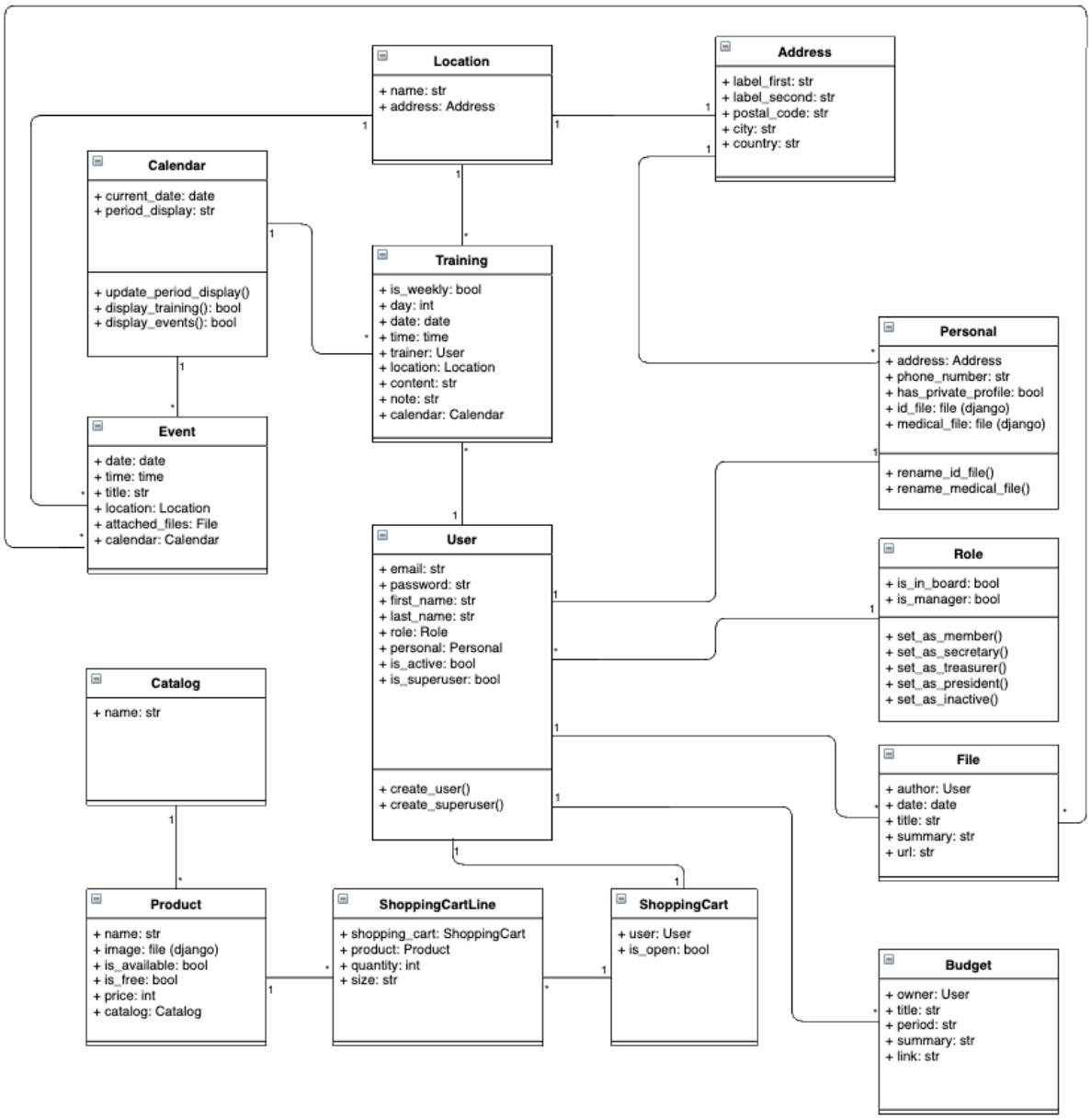

Figure 4.1 : diagramme de classes

4.2. Architecture technique

La pile logicielle du projet est la suivante :

- Application utilisant le langage de programmation *back end* est Python (version 3.8.3 ou ultérieure) avec le framework Django (version 3.0.7 ou ultérieure)
- Affichage géré par les langages de programmation *front end* sont HTML5, CSS3 (avec le framework Bootstrap 4) et Javascript
- Base de données PostgreSQL (version 12.2 ou ultérieure)

Voici le diagramme de composants du projet :

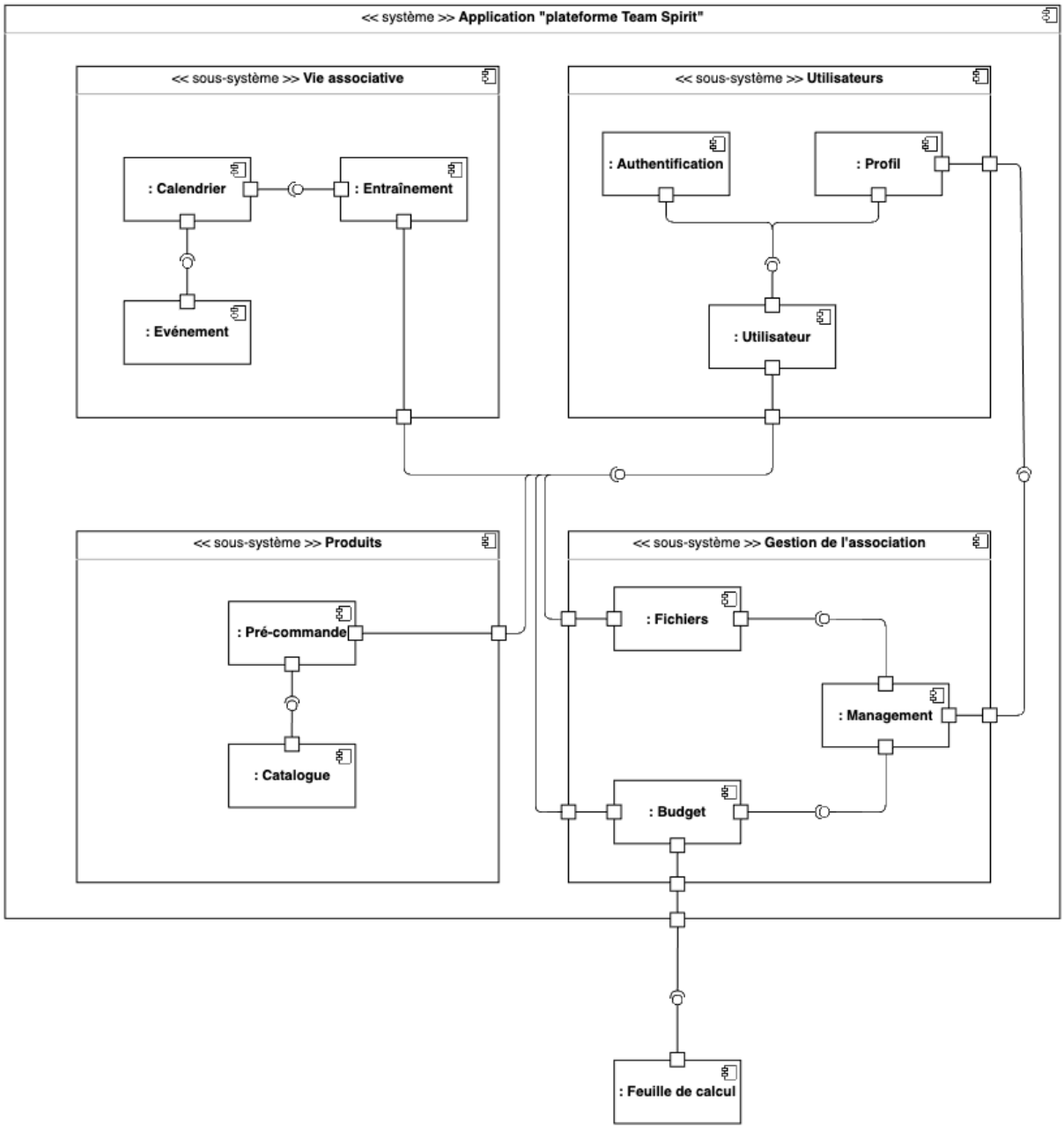

Figure 4.2 : diagramme de composants

Les composants sont présentés en détail dans les sections ci-après.

4.2.1. Sous-système « Vie associative »

4.2.1.1. Composant « Calendrier »

Le *Calendrier* affiche l'ensemble des événements proposés par l'association.
Il requiert des interfaces avec l'*Événement* et l'*Entraînement*.

4.2.1.2. Composant « Entraînement »

L'*Entraînement* est une proposition de lieu, de date et d'horaire d'un adhérent auprès des autres adhérents, pour une pratique collective de la discipline sportive proposée.
Il propose une interface au *Calendrier*.
Il requiert une interface avec l'*Utilisateur*.

4.2.1.3. Composant « Événement »

L'*Événement* est une manifestation proposée par l'association à ses adhérents.
Il propose une interface au *Calendrier*.

4.2.2. Sous-système « Utilisateurs »

4.2.2.1. Composant « Authentification »

L'*Authentification* permet de connecter un utilisateur (simple adhérent ou membre du bureau) à l'application.
Il requiert une interface avec l'*Utilisateur*.

4.2.2.2. Composant « Profil »

Le *Profil* permet de renseigner des informations à propos de l'adhérent, et de lui attribuer des droits sur l'application s'il est membre du bureau.
Il requiert des interfaces avec l'*Utilisateur* et le *Management*.

4.2.2.3. Composant « Utilisateur »

L'*Utilisateur* se sert de la plateforme, dispose d'un accès sécurisé à l'application et est à jour de ses droits d'inscription (cotisation annuelle) auprès de l'association.
Il propose des interfaces avec :

- Le *Profil* et *l'Authentification*
- L'*Entraînement* (du sous-système « Vie associative »)
- La *Pré-commande* (du sous-système « Produits »)
- Les *Fichiers* et le *Budget* (du sous-système « Gestion de l'association »)

4.2.3. Sous-système « Produits »

4.2.3.1. Composant « Pré-commande »

La *Pré-commande* permet à l'adhérent de sélectionner les produits qu'il souhaite commander auprès de l'association.
Elle requiert des interfaces avec :

- Le *Catalogue*
- L'*Utilisateur* (du sous-système « Utilisateurs »)

4.2.3.2. Composant « Catalogue »

Le *Catalogue* constitue l'ensemble des produits disponibles pour les adhérents qui souhaitent en commander auprès de l'association.
Il propose une interface à la *Pré-commande*.

4.2.4. Sous-système « Gestion de l'association »

4.2.4.1. Composant « Administration »

Le *Management* permet de créer les comptes utilisateurs des adhérents et de déléguer les tâches auprès des membres du bureau.
Il propose des interfaces avec :

- Les *Fichiers* et le *Budget*
- Le *Profil* (du sous-système « Utilisateurs »)

4.2.4.2. Composant « Fichiers »

Les *Fichiers* permettent aux utilisateurs de télécharger, poster et supprimer des documents sur la plateforme.
Ils requièrent des interfaces avec :

- Le *Management*
- L'*Utilisateur* (du sous-système « Utilisateurs »)

4.2.4.3. Composant « Budget »

Le *Budget* permet au trésorier de suivre l'activité financière de l'association.
Il requiert des interfaces avec :

- Le *Management*
- La *Feuille de calcul* (composant externe)

4.2.5. Composants externes

4.2.5.1. Composant « Feuille de calcul »

La *Feuille de calcul* permet de traiter des données dans un tableur.
Elle propose une interface avec le *Budget* (du sous-système « Gestion de l'association »).

4.3. Déploiement

Le projet est principalement déployé avec Heroku qui est utilisé en tant que plateforme en tant que service (*PaaS*) : cela concerne l'application Python/Django, la base de données, le serveur web et les fichiers statiques.
En revanche, les fichiers de type *media*, c'est-à-dire téléversés (*uploadés*) par les utilisateurs, sont stockés sur des serveurs S3 d'Amazon Web Services (AWS S3).

4.4. Charte graphique

4.4.1. Choix du thème pour le front-end

4.4.1.1. Propositions formulées au client

La trame graphique est mise en place avec l'outil Bootstrap. Voici des exemples de thèmes open source et disponibles gratuitement : https://startbootstrap.com/themes/

En particulier, le choix du client pouvait être parmi les propositions suivantes :
1. *Agency* : https://startbootstrap.com/previews/agency/
2. *Landing page* : https://startbootstrap.com/previews/landing-page/
3. *Business casual* : https://startbootstrap.com/previews/business-casual/
4. *New Age* : https://startbootstrap.com/previews/new-age/
5. *Stylish portfolio* : https://startbootstrap.com/previews/stylish-portfolio/
6. *One page wonder* : https://startbootstrap.com/previews/one-page-wonder/

La palette de couleurs utilisée peut être modifiée par rapport au rendu présenté dans les exemples précédents.

Afin de sélectionner le thème, il est conseillé au client de le visualiser sur ordinateur, et faire varier la largeur de la fenêtre pour voir comment l'affichage s'adapte, ou tout simplement basculer entre les vues « ordinateur » et « mobile ».

4.4.1.2. Décision du client

Le choix du client s'est porté sur le thème n°3 : *Business casual*.

4.4.2. Sélection de visuels pour l'application

Des photographies sont également transmises par le client, notamment pour la page d'accueil.

4.4.3. Police de caractères

Le client a choisi de conserver la police de caractères par défaut du thème Bootstrap choisi : *Lora*.

4.4.4. Palette de couleurs principales

Afin d'intégrer au mieux la couleur principale de son logo (code hexadécimal : #ff4502), le client a choisi la palette de couleur suivante : https://www.canva.com/colors/color-palettes/flaming-flower/.

Ainsi, la couleur principale est :
- Red Orange (code hexadécimal : #FF4500)

Les couleurs secondaires sont :
- Orange (code hexadécimal : #FFA303)
- Peach (code hexadécimal : #FFE0AE)
- Olive Green (code hexadécimal : #6C8255)

5. Développement des user stories

5.1. User Stories du package « vie associative »

5.1.1. US001 : renseigner ses coordonnées

5.1.1.1. Titre de la User Story

US001 - En tant que Laurent, je veux indiquer mes coordonnées pour être joignable par d'autres adhérents.

5.1.1.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Depuis la page d'accueil (cf. US024) ou depuis une autre page, dans la barre de navigation située en haut de page, cliquer sur « Mon espace ».
Ensuite, cliquer sur l'onglet « État civil » ou sur l'onglet « Coordonnées ». Dans les deux cas, pour mettre à jour les données, cliquer sur le bouton « Modifier », mettre à jour le formulaire et valider en cliquant sur le bouton « Mettre à jour ».

5.1.1.3. Spécifications techniques : documentation technique pour les développeurs

On peut distinguer trois sous-parties dans cette User Story :
- La visualisation des données
- La modification du nom et du prénom
- La modification des coordonnées (téléphone, adresse) et de leur confidentialité

5.1.1.3.1. Visualisation des données

Application django :
"teamspirit.profiles"

URL :
"/profile/"

Vue (view) :
La vue est fondée sur la classe `ProfileView`, qui hérite de la classe `TemplateView` du module `django.views.generic`.

```
class ProfileView(TemplateView):


    template_name = "profiles/profile.html"



profile_view = ProfileView.as_view()
profile_view = login_required(profile_view)
```

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
N/A

Modèle :
N/A

Gabarit (template) :
RAS

5.1.1.3.2. Modification du nom et du prénom

Application django :
"teamspirit.profiles"

URL :

"profile/update_personal_info/"

Vue (view) :

La vue est fondée sur la classe `PersonalInfoView`, qui hérite de la classe `FormView` du module `django.views.generic.edit`.

```
class PersonalInfoView(FormView):


    template_name = 'profiles/update_personal_info.html'
    form_class = PersonalInfoForm
    success_url = reverse_lazy('profiles:profile')


    def get_form_kwargs(self):
        kwargs = super(PersonalInfoView, self).get_form_kwargs()
        kwargs.update({'user': self.request.user})
        return kwargs



personal_info_view = PersonalInfoView.as_view()
personal_info_view = login_required(personal_info_view)
```

La vue s'appuie sur le formulaire `PersonalInfoForm` présenté ci-dessous. En cas de succès lors de la soumission de ce formulaire, l'utilisateur est redirigé vers sa page de profil et peut visualiser ses données (cf. sous-partie précédente).

Afin de permettre l'accès aux attributs du modèle `User` depuis le gabarit, il est nécessaire de réécrire la méthode `get_form_kwargs`.

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :

Le formulaire `PersonalInfoForm` hérite du formulaire `ModelForm` du module `django.forms`, s'appuie sur le modèle `User` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.

```
class PersonalInfoForm(ModelForm):


    class Meta:
        model = User
        fields = ['last_name', 'first_name']


    def __init__(self, *args, **kwargs):
        # paramétrage pour crispy_forms
        # [...]
        if self.is_valid():
            self.save()


    def save(self, commit=True):
        last_name = self.cleaned_data["last_name"]
```

```
        first_name = self.cleaned_data["first_name"]
        if commit:
            self.user.last_name = last_name.upper()
            self.user.first_name = first_name.capitalize()
            self.user.save()
        return self.user
```

Les champs `last_name` et `first_name` sont requis.

La méthode `save` est réécrite pour mettre à jour le nom et le prénom en base de données.

Modèle :
Le modèle `User` utilisé ici hérite notamment du modèle `AbstractBaseUser` du module `django.contrib.auth.base_user`. On ajoute notamment les attributs `last_name` et `first_name`.

Gabarit (template) :
Le gabarit contenant le formulaire de modification du nom et du prénom utilise le package *cripsy_forms*.

5.1.1.3.3. Modification des coordonnées

Applications django :
"teamspirit.profiles" et "teamspirit.core"

URL :
"/profile/update_phone_address/"

Vue (view) :
La vue `phone_address_view` est une fonction qui prend en paramètre `request` (de type `HttpRequest`, avec respectivement une méthode `GET` lors de l'affichage de la page, ou une méthode `POST` lors de la soumission des trois formulaires qu'elles contient) et retourne un objet de type `HttpResponse`.

```
@login_required
def phone_address_view(request):
    context = {}
    if request.method == 'POST':
        phone_form = PhoneForm(request.POST, user=request.user)
        address_form = AddressForm(request.POST, user=request.user)
        confidentiality_form = ConfidentialityForm(
            request.POST,
            user=request.user
        )
        if all([
            phone_form.is_valid(),
            address_form.is_valid(),
            confidentiality_form.is_valid()
        ]):
            # process forms
            phone_form.save()
            address_form.save()
            confidentiality_form.save()
            # redirect to the profile url
```

```
            return redirect(reverse_lazy('profiles:profile'))
        else:
            phone_form = PhoneForm(user=request.user)
            address_form = AddressForm(user=request.user)
            confidentiality_form = ConfidentialityForm(user=request.user)
        context['phone_form'] = phone_form
        context['address_form'] = address_form
        context['confidentiality_form'] = confidentiality_form
        return render(
            request,
            'profiles/update_phone_address.html',
            context,
        )
```

Formulaires :
Les trois formulaires `PhoneForm`, `AddressForm` et `ConfidentialityForm` héritent du formulaire `ModelForm` du module `django.forms`, s'appuient sur les modèles `Personal` et `Address`, et utilisent le package *cripsy_forms* pour le rendu HTML et CSS.
Les champs `phone_number`, `label_first`, `postal_code`, `city`, `country` et `has_private_profile` sont requis. Le champ `label_second` est optionnel.

De même que pour le formulaire `PersonalInfoForm` de la sous-partie précédente, la méthode `save` est réécrite pour mettre à jour les données en base de données.

Modèles :
Les modèles `Personal` et `Address` utilisés ici complètent le modèle `User`. Cela permet d'ajouter des données personnelles sans surcharger directement le modèle `User`. Le modèle `User` contient un attribut `personal` qui est une clé étrangère qui pointe vers le modèle `Personal`. De même, le modèle `Personal` contient un attribut `address` qui est une clé étrangère qui pointe vers le modèle `Address`.

Gabarit (template) :
Le gabarit contenant le formulaire d'authentification utilise le package *cripsy_forms*.

5.1.1.4. Tests d'acceptation

Les tests d'acceptation (ou tests fonctionnels) sont présents dans les modules `test_us001_to_009.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Les tests fonctionnels permettant de valider la User Story sont :
US001-AT01 : Modification du nom et du prénom de l'utilisateur
US001-AT02 : Modification des coordonnées de l'utilisateur

Ils nécessitent que l'utilisateur soit connecté au préalable, ainsi les tests sont des méthodes de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing.`

5.1.2. US002 : poster ou supprimer un fichier

5.1.2.1. Titre de la User Story

US001 - En tant que Laurent, je veux poster ou supprimer un fichier sur la plateforme pour transmettre mon certificat médical.

5.1.2.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Depuis la page d'accueil (cf. US024) ou depuis une autre page, dans la barre de navigation située en haut de page, cliquer sur « Mon espace ».
Ensuite, cliquer sur l'onglet « Certificat ou licence » ou sur l'onglet « Pièce d'identité ». Dans les deux cas, pour poster un fichier, cliquer sur le bouton « Ajouter », sélectionner un fichier en cliquant sur le bouton « Parcourir » et valider en cliquant sur le bouton « Soumettre ». De même, pour supprimer un fichier préalablement posté, cliquer sur le bouton « Supprimer » et valider en cliquant sur le bouton « Confirmer ».

5.1.2.3. Spécifications techniques : documentation technique pour les développeurs

On peut distinguer trois sous-parties dans cette User Story :
- La visualisation des données
- Le téléversement (upload) d'un fichier personnel par l'utilisateur
- La suppression d'un fichier personnel par l'utilisateur

5.1.2.3.1. Visualisation des données

Cette sous-partie est présentée dans ce paragraphe de la US001.

5.1.2.3.2. Téléversement d'un fichier

Application django :
"teamspirit.profiles"

URLs :
"profile/add_medical_file/" et "profile/add_id_file/"

Vues (views) :
Les vues sont fondées respectivement sur les classes `AddMedicalFileView` et `AddIdFileView`, qui héritent de la classe `FormView` du module `django.views.generic.edit`.

```
class AddMedicalFileView(FormView):


    template_name = 'profiles/add_medical_file.html'
    form_class = AddMedicalFileForm
    success_url = reverse_lazy('profiles:profile')


    def get_form_kwargs(self):
        kwargs = super(AddMedicalFileView, self).get_form_kwargs()
        kwargs.update({'user': self.request.user})
        return kwargs



add_medical_file_view = AddMedicalFileView.as_view()
add_medical_file_view = login_required(add_medical_file_view)
```

Dans les lignes suivantes, seul l'exemple de la vue `add_medical_file_view` et du formulaire `AddMedicalFileForm` est présenté, puisque l'exemple de la vue `add_id_file_view` et du formulaire `AddIdFileForm` est similaire.

La vue s'appuie sur le formulaire `AddMedicalFileForm` présenté ci-dessous. En cas de succès lors de la soumission de ce formulaire, l'utilisateur est redirigé vers sa page de profil et peut visualiser ses données (cf. ce paragraphe).

Afin de permettre l'accès aux attributs du modèle `User` depuis le gabarit, il est nécessaire de réécrire la méthode `get_form_kwargs`.

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :

Le formulaire `AddMedicalFileForm` hérite du formulaire `ModelForm` du module `django.forms`, s'appuie sur le modèle `Personal` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.

```
class AddMedicalFileForm(ModelForm):


    class Meta:
        model = Personal
        fields = ['medical_file']


    def __init__(self, *args, **kwargs):
        # paramétrage pour crispy_forms
        # [...]
        if self.is_valid():
            self.save()


    def save(self, commit=True):
        medical_file = self.cleaned_data["medical_file"]
        if commit and medical_file:
            self.user.personal.medical_file = medical_file
            self.user.personal.save()
        return self.user.personal
```

Le champ `medical_file` n'est pas requis et peut être laissé vide.

La méthode `save` est réécrite pour mettre à jour l'url du fichier en base de données.

Modèle :

Le modèle `Personal` utilisé ici complète le modèle `User`. On ajoute notamment les attributs `id_file` et `medical_file`.

```
class Personal(models.Model):
    """Contain personal information."""


    # autres attributs
    # [...]
    id_file = models.FileField(
        null=True,
        blank=True,
```

```
        verbose_name='Pièce d\'identité',
        upload_to=rename_id_file,
    )
    medical_file = models.FileField(
        null=True,
        blank=True,
        verbose_name='Certificat médical ou licence',
        upload_to=rename_medical_file,
    )
```

Gabarits (templates) :
Les gabarits contenant le formulaire de téléversement des fichiers de type « certificat médical ou licence » et « pièce d'identité » utilisent le package *cripsy_forms*.

5.1.2.3.3. Suppression d'un fichier

Application django :
"teamspirit.profiles"

URLs :
"profile/drop_medical_file/", "profile/drop_id_file/" et "profile/drop_file/"

Vues (views) :
Les deux premières vues sont fondées respectivement sur les classes `DropMedicalFileView` et `DropIdFileView`, qui héritent de la classe `FormView` du module `django.views.generic.edit`.

```
class DropMedicalFileView(FormView):


    template_name = 'profiles/drop_medical_file.html'
    form_class = DropMedicalFileForm
    success_url = reverse_lazy('profiles:drop_file')


    def get_form_kwargs(self):
        kwargs = super(DropMedicalFileView, self).get_form_kwargs()
        kwargs.update({'user': self.request.user})
        return kwargs



drop_medical_file_view = DropMedicalFileView.as_view()
drop_medical_file_view = login_required(drop_medical_file_view)
```

Dans les lignes suivantes, seul l'exemple de la vue `drop_medical_file_view` et du formulaire `DropMedicalFileForm` est présenté, puisque l'exemple de la vue `drop_id_file_view` et du formulaire `DropIdFileForm` est similaire.

La vue s'appuie sur le formulaire `DropMedicalFileForm` présenté ci-dessous. En cas de succès lors de la soumission de ce formulaire, l'utilisateur est redirigé vers une page de confirmation dont la vue est `drop_file_view` fondée sur la classe `DropFileView` qui hérite de la classe `TemplateView` du module `django.views.generic`.

Afin de permettre l'accès aux attributs du modèle `User` depuis le gabarit, il est nécessaire de réécrire la méthode `get_form_kwargs`.

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
Le formulaire `DropMedicalFileForm` hérite du formulaire `ModelForm` du module `django.forms`, s'appuie sur le modèle `Personal` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.

```
class DropMedicalFileForm(ModelForm):


    class Meta:
        model = Personal
        fields = []


    def __init__(self, *args, **kwargs):
        # paramétrage pour crispy_forms
        # [...]
        if self.is_valid():
            self.save()


    def save(self, commit=True):
        if commit:
            self.user.personal.medical_file.delete()
            self.user.personal.save()
        return self.user.personal
```

Aucun champ n'est disponible, puisque l'intérêt de ce formulaire consiste à supprimer le fichier déjà téléversé par l'utilisateur.

La méthode `save` est réécrite pour supprimer l'url du fichier en base de données.

Modèle :
Cf. modèle de la sous-partie précédente.

Gabarits (templates) :
Les gabarits contenant le formulaire de suppression des fichiers de type « certificat médical ou licence » et « pièce d'identité » utilisent le package *cripsy_forms*.

5.1.2.3.4. Stockage des fichiers

Les fichiers téléversés par les utilisateurs sont stockés sur un ou plusieurs serveurs de type S3 de la société Amazon Web Services (AWS). L'interface entre l'application Django et ces serveurs est paramétrée dans le fichier de configuration et utilise les libraires *django-storages* et *boto3* :

```
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
AWS_STORAGE_BUCKET_NAME = env('AWS_STORAGE_BUCKET_NAME')
AWS_ACCESS_KEY_ID = env('AWS_ACCESS_KEY_ID')
AWS_SECRET_ACCESS_KEY = env('AWS_SECRET_ACCESS_KEY')
AWS_S3_CUSTOM_DOMAIN = f'{AWS_STORAGE_BUCKET_NAME}.s3.amazonaws.com'
AWS_S3_OBJECT_PARAMETERS = {
    'CacheControl': 'max-age=86400',
}
```

Un utilisateur IAM a été créé sur la plateforme AWS, l'application Django utilise ses identifiants pour permettre l'ajout et la suppression de fichiers sur les serveurs AWS.

5.1.2.4. Tests d'acceptation

Les tests d'acceptation (ou tests fonctionnels) sont présents dans les modules `test_us001_to_009.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Les tests fonctionnels permettant de valider la User Story sont :
US002-AT01 : Ajout puis suppression d'un fichier de type « certificat médical ou licence »
US002-AT02 : Ajout puis suppression d'un fichier de type « pièce d'identité »

Ils nécessitent que l'utilisateur soit connecté au préalable, ainsi les tests sont des méthodes de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing`.

5.1.3. US003 : renseigner des informations à destination des autres adhérents

5.1.3.1. Titre de la User Story

US003 - En tant que Laurent, je veux renseigner des informations à destination des adhérents pour proposer des créneaux d'entraînement.

5.1.3.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.1.3.3. Spécifications techniques : documentation technique pour les développeurs

5.1.3.4. Tests d'acceptation

5.1.4. US004 : consulter l’agenda

5.1.4.1. Titre de la User Story

US004 - En tant que Claire, je veux consulter l'agenda pour connaître les événements passés et à venir.

5.1.4.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.1.4.3. Spécifications techniques : documentation technique pour les développeurs

5.1.4.4. Tests d’acceptation

5.1.5. US005 : télécharger ou ouvrir un document

5.1.5.1. Titre de la User Story

US005 - En tant que Claire, je veux télécharger ou ouvrir un document pour lire un compte-rendu de réunion.

5.1.5.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.1.5.3. Spécifications techniques : documentation technique pour les développeurs

5.1.5.4. Tests d'acceptation

5.1.6. US006 : accéder à certaines informations concernant d'autres adhérents

5.1.6.1. Titre de la User Story

US006 - En tant que Pierre, je veux accéder à certaines informations des adhérents pour rencontrer d'autres coureurs.

5.1.6.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.1.6.3. Spécifications techniques : documentation technique pour les développeurs

5.1.6.4. Tests d'acceptation

5.1.7. US007 : consulter les articles disponibles en catalogue

5.1.7.1. Titre de la User Story

US007 - En tant que Pierre, je veux consulter la liste des articles disponibles pour obtenir une tenue de sport.

5.1.7.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Depuis la page d'accueil (cf. US024) ou depuis une autre page, dans la barre de navigation située en haut de page, cliquer sur « Catalogue ». Les articles disponibles sont affichés sur cette page.

5.1.7.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.catalogs"

URL :
"catalog/"

Vue (view) :
La vue est fondée sur la classe `CatalogView`, qui hérite de la classe `ListView` du module `django.views.generic`.

```
class CatalogView(ListView):


    model = Product

    template_name = "catalogs/catalog.html"



catalog_view = CatalogView.as_view()

catalog_view = login_required(catalog_view)
```

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
N/A

Modèles :
Le modèle principal est le modèle `Product` qui définit plusieurs attributs requis (`name`, `is_available`, `is_free`, `price` et `catalog`) et un attribut optionnel (`image`). Le modèle `Catalog` complète le modèle `Product`.

Gabarit (template) :
RAS

5.1.7.4. Tests d'acceptation

Le test d'acceptation (ou test fonctionnel) est présent dans les modules `test_us001_to_009.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Le test fonctionnel permettant de valider la User Story est :
US007-AT01 : Consultation des articles répertoriés dans le catalogue

Il nécessite que l'utilisateur soit connecté au préalable, ainsi le test est une méthode de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing`.

5.1.8. US008 : renseigner une pré-commande

5.1.8.1. Titre de la User Story

US008 - En tant que Pierre, je veux renseigner une pré-commande pour obtenir une tenue de sport.

5.1.8.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Depuis le catalogue (cf. US007), choisir un article et cliquer sur le bouton « Ajouter au panier » correspondant. Renseigner la quantité et la taille souhaitées, puis valider en cliquant sur le bouton « Ajouter au panier ». Cette étape peut être renouvelée autant de fois que nécessaire.
Pour modifier son panier, cliquer sur le lien « Voir mon panier », puis cliquer le bouton « Supprimer » correspondant au produit que l'on souhaite retirer du panier.

5.1.8.3. Spécifications techniques : documentation technique pour les développeurs

On peut distinguer trois sous-parties dans cette User Story :
- La visualisation du panier
- L'ajout de produit au panier
- La suppression d'un produit du panier

5.1.8.3.1. Visualisation du panier

Application django :
"teamspirit.preorders"

URL :
"shopping_cart/"

Vue (view) :
La vue est fondée sur la classe `ShoppingCartView`, qui hérite de la classe `ListView` du module `django.views.generic`.

```
class ShoppingCartView(ListView):


    model = ShoppingCartLine
    template_name = "preorders/shopping_cart.html"


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['shopping_cart_amount'] = ShoppingCart.objects.get_or_create(
            user=self.request.user
        )[0].get_cart_amount()
        return context


    def get_queryset(self):
        queryset = super().get_queryset()
        shopping_cart = ShoppingCart.objects.get_or_create(
            user=self.request.user
        )[0]
        queryset = ShoppingCartLine.objects.filter(
            shopping_cart=shopping_cart
        )
        return queryset
```

```
shopping_cart_view = ShoppingCartView.as_view()
shopping_cart_view = login_required(shopping_cart_view)
```

Afin d'afficher le montant total du panier, on ajoute cette variable au contexte avec la méthode `get_context_data`. De plus, afin de limiter l'affichage aux produits sélectionnés par l'utilisateur (et ne pas afficher les produits des autres utilisateur), on redéfinit la méthode `get_queryset`.

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
N/A

Modèles :
Deux modèles sont utilisés ici : `ShoppingCart` (dont une instance correspond à un panier dans sa globalité, et est lié à un utilisateur avec l'attribut `user`) et `ShoppingCartLine` (dont une instance correspond à une ligne de panier) qui définit plusieurs attributs requis (`shopping_cart`, `product`, `quantity` et `size`).

Gabarit (template) :
RAS

5.1.8.3.2. Ajout de produit au panier

Application django :
"teamspirit.preorders"

URL :
"shopping_cart/add_product/<product_id>/"

Vue (view) :
La vue est fondée sur la classe `AddToCartView`, qui hérite de la classe `FormView` du module `django.views.generic.edit`.

```
class AddToCartView(FormView):

    template_name = "preorders/add_to_cart.html"
    form_class = AddToCartForm
    success_url = reverse_lazy('catalogs:catalog')

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['product'] = Product.objects.get(id=self.kwargs['product_id'])
        return context

    def get_initial(self):
        initial = super().get_initial()
        initial['shopping_cart'] = ShoppingCart.objects.get_or_create(
            user=self.request.user
        )[0]
        initial['product'] = Product.objects.get(id=self.kwargs['product_id'])
        return initial
```

```
add_to_cart_view = AddToCartView.as_view()
add_to_cart_view = login_required(add_to_cart_view)
```

Afin d'afficher certains attributs du produit (son nom, sa photo, son prix), on ajoute cette variable au contexte avec la méthode `get_context_data`. De plus, afin de paramétrer l'affichage du formulaire, il est utile de renseigner des valeurs par défaut pour les champs `shopping_cart` et `product` (qui sont invisibles pour l'utilisateur), on redéfinit la méthode `get_initial`.

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
Le formulaire `AddToCartForm` hérite du formulaire `ModelForm` du module `django.forms`, s'appuie sur le modèle `ShoppingCartLine` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.

```
class AddToCartForm(ModelForm):

    class Meta:
        model = ShoppingCartLine
        fields = '__all__'

    def __init__(self, *args, **kwargs):
        # paramétrage pour crispy_forms
        # [...]
        if self.is_valid():
            self.save()
```

Les champs `shopping_cart` et `product` sont masqués à l'affichage, afin de ne pas pouvoir être modifiés.

Modèles :
Cf. sous-partie précédente

Gabarit (template) :
Le gabarit contenant le formulaire d'ajout de produit au panier utilise le package *cripsy_forms*.

5.1.8.3.3. Suppression d'un produit du panier

Application django :
"teamspirit.preorders"

URL :
"shopping_cart/drop_product/<line_id>/"

Vue (view) :
La vue est fondée sur la classe `DropFromCartView`, qui hérite de la classe `FormView` du module `django.views.generic.edit`.

```
class DropFromCartView(FormView):

    template_name = "preorders/drop_from_cart.html"
    form_class = DropFromCartForm
    success_url = reverse_lazy('preorders:shopping_cart')
```

```
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['shopping_cart_line'] = ShoppingCartLine.objects.get(
            id=self.kwargs['line_id']
        )
        return context

    def get_form_kwargs(self):
        kwargs = super(DropFromCartView, self).get_form_kwargs()
        kwargs.update({'request_user': self.request.user})
        kwargs.update({'line_id': self.kwargs['line_id']})
        kwargs.update({
            'shopping_cart_line': ShoppingCartLine.objects.get(
                id=self.kwargs['line_id']
            )
        })
        return kwargs


drop_from_cart_view = DropFromCartView.as_view()
drop_from_cart_view = login_required(drop_from_cart_view)
```

Afin d'afficher certains attributs de la ligne de panier (son produit, sa quantité, sa taille), on ajoute cette variable au contexte avec la méthode `get_context_data`. De plus, afin de pouvoir supprimer un enregistrement en base de données, il est nécessaire de passer au formulaire certains paramètres : request l'affichage du formulaire, il est utile de renseigner des valeurs par défaut pour les champs `request_user`, `line_id` et `shopping_cart_line`, on redéfinit la méthode `get_form_kwargs`.

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :

Le formulaire `DropFromCartForm` hérite du formulaire `ModelForm` du module `django.forms`, s'appuie sur le modèle `Personal` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.

```
class DropFromCartForm(ModelForm):


    class Meta:
        model = ShoppingCartLine
        fields = []


    def __init__(self, *args, **kwargs):
        # paramétrage pour crispy_forms
        # [...]
        if self.is_valid():
            self.save()
```

```
def save(self, commit=True):
    if commit:
        # /!\ prevent this action for other users!
        if self.request_user == self.line_user:
            self.shopping_cart_line.delete()
    return self
```

Aucun champ n'est disponible, puisque l'intérêt de ce formulaire consiste à supprimer la ligne de panier existante.

La méthode `save` est réécrite pour supprimer l'instance en base de données.

Modèles :
Cf. sous-partie précédente.

Gabarit (template) :
Le gabarit contenant le formulaire de suppression d'un produit du panier utilise le package *cripsy_forms*.

5.1.8.4. Tests d'acceptation

Le test d'acceptation (ou test fonctionnel) est présent dans les modules `test_us001_to_009.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Le test fonctionnel permettant de valider la User Story est :
US008-AT01 : Ajout puis suppression d'un produit dans le panier

Il nécessite que l'utilisateur soit connecté au préalable, ainsi le test est une méthode de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing.`

5.1.9. US009 : connaître des adhérents intéressés par une pratique en groupe

5.1.9.1. Titre de la User Story

US009 - En tant que Pierre, je veux connaître les adhérents intéressés par une pratique en groupe pour s'entraîner ensemble.

5.1.9.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.1.9.3. Spécifications techniques : documentation technique pour les développeurs

5.1.9.4. Tests d'acceptation

5.2. User Stories du package « gestion de l'association »

5.2.1. US010 : délégation des tâches administratives

5.2.1.1. Titre de la User Story

US010 - En tant que Emmanuel, je veux répartir les rôles pour déléguer les tâches administratives.

5.2.1.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.1.3. Spécifications techniques : documentation technique pour les développeurs

5.2.1.4. Tests d'acceptation

5.2.2. US011 : supervision de tâches administratives

5.2.2.1. Titre de la User Story

US011 - En tant que Emmanuel, je veux accéder à toutes les tâches du bureau pour les superviser.

5.2.2.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.2.3. Spécifications techniques : documentation technique pour les développeurs

5.2.2.4. Tests d'acceptation

5.2.3. US012 : modifier la liste des adhérents

5.2.3.1. Titre de la User Story

US012 - En tant que Maxime, je veux modifier la liste des adhérents pour tenir une liste à jour.

5.2.3.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.3.3. Spécifications techniques : documentation technique pour les développeurs

5.2.3.4. Tests d'acceptation

5.2.4. US013 : contrôler le paiement des cotisations

5.2.4.1. Titre de la User Story

US013 - En tant que Maxime, je veux contrôler le paiement des cotisations pour suivre la situation.

5.2.4.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.4.3. Spécifications techniques : documentation technique pour les développeurs

5.2.4.4. Tests d'acceptation

5.2.5. US014 : contrôler le paiement des événements

5.2.5.1. Titre de la User Story

US014 - En tant que Maxime, je veux contrôler le paiement des événements pour suivre la situation.

5.2.5.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.5.3. Spécifications techniques : documentation technique pour les développeurs

5.2.5.4. Tests d'acceptation

5.2.6. US015 : contrôler le paiement des vêtements

5.2.6.1. Titre de la User Story

US015 - En tant que Maxime, je veux contrôler le paiement des vêtements pour suivre la situation.

5.2.6.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.6.3. Spécifications techniques : documentation technique pour les développeurs

5.2.6.4. Tests d'acceptation

5.2.7. US016 : contrôle du budget

5.2.7.1. Titre de la User Story

US016 - En tant que Maxime, je veux contrôler le budget pour le gérer.

5.2.7.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.7.3. Spécifications techniques : documentation technique pour les développeurs

5.2.7.4. Tests d'acceptation

5.2.8. US017 : mise à jour de l’agenda

5.2.8.1. Titre de la User Story

US017 - En tant que Axel, je veux mettre à jour l'agenda pour diffuser des informations.

5.2.8.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.8.3. Spécifications techniques : documentation technique pour les développeurs

5.2.8.4. Tests d’acceptation

5.2.9. US018 : diffusion de documents

5.2.9.1. Titre de la User Story

US018 - En tant que Axel, je veux diffuser les compte-rendus de réunion pour informer les adhérents.

5.2.9.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.9.3. Spécifications techniques : documentation technique pour les développeurs

5.2.9.4. Tests d'acceptation

5.2.10. US019 : suppression de documents

5.2.10.1. Titre de la User Story

US019 - En tant que Axel, je veux supprimer des documents pour simplifier la communication.

5.2.10.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.10.3. Spécifications techniques : documentation technique pour les développeurs

5.2.10.4. Tests d'acceptation

5.2.11. US020 : accéder à des documents postés par autrui

5.2.11.1. Titre de la User Story

US020 - En tant que Axel, je veux accéder aux documents des adhérents pour capter des informations.

5.2.11.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.11.3. Spécifications techniques : documentation technique pour les développeurs

5.2.11.4. Tests d'acceptation

5.2.12. US021 : ajout d’un utilisateur

5.2.12.1. Titre de la User Story

US021 - En tant que Maxime, je veux ajouter un utilisateur.

5.2.12.2. Spécifications fonctionnelles : documentation pour les utilisateurs

5.2.12.3. Spécifications techniques : documentation technique pour les développeurs

5.2.12.4. Tests d’acceptation

5.3. User Stories génériques

5.3.1. US022 : accès à la page de connexion

5.3.1.1. Titre de la User Story

US022 - En tant que Toto, je veux accéder à la page de connexion pour me connecter à l'application.

5.3.1.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Dans un navigateur web, aller à l'adresse https://team-spirit.herokuapp.com/users/login/. La suite des instructions figure dans la US023.

5.3.1.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.users"

URL :
"/users/login/"

Pour connaître davantage de détails techniques : cf. US023.

5.3.1.4. Tests d'acceptation

Le test d'acceptation (ou test fonctionnel) est présent dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Le test fonctionnel permettant de valider la User Story est :
US022-AT01 : Accès réussi à la page de connexion

Il nécessite que l'utilisateur ne soit pas connecté au préalable, ainsi les tests sont des méthodes de la classe `GeneralUserStories`**`Anonymous`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing`.

5.3.2. US023 : connexion à l'application

5.3.2.1. Titre de la User Story

US023 - En tant que Toto, je veux me connecter à l'application.

5.3.2.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Depuis la page de connexion (cf. US022), saisir son courriel et son mot de passe, puis valider avec la touche « Entrée » ou en cliquant sur le bouton « Se connecter ».
En cas d'erreur(s), suivre les indications.
En cas de succès, le site web redirige vers la page d'accueil et l'utilisateur est connecté.

5.3.2.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.users"

URL :
"/users/login/"

Vue (view) :
La vue est fondée sur la classe `CustomLoginView`, qui hérite de la classe `LoginView` du module `django.contrib.auth.views`.

```
class CustomLoginView(LoginView):


    template_name = 'users/login.html'
    form_class = CustomAuthenticationForm



custom_login_view = CustomLoginView.as_view()
```

Formulaire :
Le formulaire `CustomAuthenticationForm` hérite du formulaire `AuthenticationForm` du module `django.contrib.auth.forms` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.
Les champs `email` et `password` sont requis.

Modèle :
Le modèle `User` utilisé ici hérite notamment du modèle `AbstractBaseUser` du module `django.contrib.auth.base_user`. On ajoute notamment l'attribut `email` qui remplace l'attribut `username`.

Gabarit (template) :
Le gabarit contenant le formulaire d'authentification utilise le package *cripsy_forms*.

5.3.2.4. Tests d'acceptation

Les tests d'acceptation (ou tests fonctionnels) sont présents dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Les tests fonctionnels permettant de valider la User Story sont :
US023-AT01 : Connexion réussie
US023-AT02 : Échec de connexion avec un utilisateur inconnu
US023-AT03 : Échec de connexion avec un mot de passe erroné
US023-AT04 : Échec de connexion avec un utilisateur inactif

Ils nécessitent que l'utilisateur ne soit pas connecté au préalable, ainsi les tests sont des méthodes de la classe `GeneralUserStories`**`Anonymous`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing`.

5.3.3. US024 : accès à la page d'accueil

5.3.3.1. Titre de la User Story

US024 - En tant que Toto, je veux accéder à la page d'accueil pour naviguer sur le site.

5.3.3.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Dans un navigateur web, aller à l'adresse https://team-spirit.herokuapp.com/.

5.3.3.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.core"

URL :
"/"

Vue (view) :
La vue est fondée sur la classe `HomeView`, qui hérite de la classe `TemplateView` du module `django.views.generic`.

```
class HomeView(TemplateView):


    template_name = "pages/home.html"



home_view = HomeView.as_view()
home_view = login_required(home_view)
```

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
N/A

Modèle :
N/A

Gabarit (template) :
RAS

5.3.3.4. Tests d'acceptation

Le test d'acceptation (ou test fonctionnel) est présent dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Le test fonctionnel permettant de valider la User Story est :
US024-AT01 : Accès réussi à la page d'accueil

Il nécessite que l'utilisateur soit connecté au préalable, ainsi le test est une méthode de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing.`

5.3.4. US025 : accès à chaque page de l'application

5.3.4.1. Titre de la User Story

US025 - En tant que Toto, je veux accéder à chaque page de l'application pour naviguer sur le site.

5.3.4.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Dans un navigateur web, aller à l'une des adresses suivantes :
- Pour accéder aux événements : https://team-spirit.herokuapp.com/events/
- Pour accéder aux entraînements : https://team-spirit.herokuapp.com/trainings/
- Pour accéder au catalogue : https://team-spirit.herokuapp.com/catalog/
- Pour accéder à l'espace privé : https://team-spirit.herokuapp.com/profile/

Ou, à partir de la page d'accueil (cf. US024), cliquer sur le lien correspondant dans la barre de navigation en haut de page.

5.3.4.3. Spécifications techniques : documentation technique pour les développeurs

Applications django :
"teamspirit.events", "teamspirit.trainings", "teamspirit.catalogs", "teamspirit.profiles"

URLs :
"/events/", "/trainings/", "/catalog/", "/profile/"

Vues (views) :
Les vues sont respectivement fondées sur les classes `EventsView`, `TrainingsView`, `CatalogView` et `ProfileView` qui héritent des classes `ListView` et `TemplateView` du module `django.views.generic`.

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
N/A

Modèle :
N/A

Gabarit (template) :
RAS

5.3.4.4. Tests d'acceptation

Les tests d'acceptation (ou tests fonctionnels) sont présents dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Les tests fonctionnels permettant de valider la User Story sont :
US025-AT01 : Accès réussi à la page des événements
US025-AT02 : Accès réussi à la page des entraînements
US025-AT03 : Accès réussi à la page du catalogue
US025-AT04 : Accès réussi à la page de l'espace privé

Ils nécessitent que l'utilisateur soit connecté au préalable, ainsi les tests sont des méthodes de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing`.

5.3.5. US026 : accès à la page de contact

5.3.5.1. Titre de la User Story

US026 - En tant que Toto, je veux accéder à la page de contact pour communiquer avec le bureau de l'association.

5.3.5.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Dans un navigateur web, aller à l'adresse https://team-spirit.herokuapp.com/contact/ ou, à partir de la page d'accueil (cf. US024), cliquer sur le lien « Nous contacter » en bas de page.

5.3.5.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.core"

URL :
"/contact/"

Vue (view) :
La vue est fondée sur la classe `ContactView`, qui hérite de la classe `TemplateView` du module `django.views.generic`.

```
class ContactView(TemplateView):


    template_name = "pages/contact.html"



contact_view = ContactView.as_view()
contact_view = login_required(contact_view)
```

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
N/A

Modèle :
N/A

Gabarit (template) :
RAS

5.3.5.4. Tests d'acceptation

Le test d'acceptation (ou test fonctionnel) est présent dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Le test fonctionnel permettant de valider la User Story est :
US026-AT01 : Accès réussi à la page de contact

Il nécessite que l'utilisateur soit connecté au préalable, ainsi le test est une méthode de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing.`

5.3.6. US027 : accès aux mentions légales

5.3.6.1. Titre de la User Story

US027 - En tant que Toto, je veux accéder à la page des mentions légales pour le plaisir.

5.3.6.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Dans un navigateur web, aller à l'adresse https://team-spirit.herokuapp.com/legal/ ou, à partir de la page d'accueil (cf. US024), cliquer sur le lien « Mentions légales » en bas de page.

5.3.6.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.core"

URL :
"/legal/"

Vue (view) :
La vue est fondée sur la classe `LegalView`, qui hérite de la classe `TemplateView` du module `django.views.generic`.

```
class LegalView(TemplateView):


    template_name = "pages/legal.html"



legal_view = LegalView.as_view()
legal_view = login_required(legal_view)
```

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
N/A

Modèle :
N/A

Gabarit (template) :
RAS

5.3.6.4. Tests d'acceptation

Le test d'acceptation (ou test fonctionnel) est présent dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Le test fonctionnel permettant de valider la User Story est :
US027-AT01 : Accès réussi à la page des mentions légales

Il nécessite que l'utilisateur soit connecté au préalable, ainsi le test est une méthode de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing.`

5.3.7. US028 : modifier son mot de passe

5.3.7.1. Titre de la User Story

US028 - En tant que Toto, je veux modifier mon mot de passe.

5.3.7.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Dans un navigateur web, aller à l'adresse https://team-spirit.herokuapp.com/profile/change_password/ ou, à partir de la page d'accueil (cf. US024) :

- Cliquer sur le lien « Mon Espace » dans la barre de navigation en haut de page
- Cliquer sur le bouton « changer de mot de passe »
- Saisir son mot de passe actuel (« ancien mot de passe »), puis deux fois le nouveau mot de passe choisi
- Valider en appuyant sur la touche « Entrée » ou en cliquant sur le bouton « modifier le mot de passe »

5.3.7.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.profiles"

URL :
"/change_password/" et "/change_password/done/"

Vues (views) :
La **première vue** est fondée sur la classe `CustomPasswordChangeView`, qui hérite de la classe `PasswordChangeView` du module `django.views.auth.views`.

```
class CustomPasswordChangeView(PasswordChangeView):


    template_name = 'profiles/change_password.html'
    success_url = 'done'
    form_class = CustomPasswordChangeForm



custom_password_change_view = CustomPasswordChangeView.as_view()
custom_password_change_view = login_required(custom_password_change_view)
```

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

La **seconde vue** est fondée sur la classe `PasswordChangedView`, qui hérite de la classe `TemplateView` du module `django.views.generic`.

```
class PasswordChangedView(TemplateView):


    template_name = 'profiles/password_changed.html'



password_changed_view = PasswordChangedView.as_view()
password_changed_view = login_required(password_changed_view)
```

La page n'est accessible que si l'utilisateur est authentifié, ce qui explique l'utilisation du décorateur `login_required` du module `django.contrib.auth.decorators`.

Formulaire :
Le formulaire `CustomPasswordChangeForm` hérite du formulaire `PasswordChangeForm` du module `django.contrib.auth.forms` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.
Les champs `old_password`, `new_password1` et `new_password2` sont requis.

Modèle :
cf. US023

Gabarit (template) :
Le gabarit contenant le formulaire d'authentification utilise le package *cripsy_forms*.

5.3.7.4. Tests d'acceptation

Les tests d'acceptation (ou tests fonctionnels) sont présents dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Les tests fonctionnels permettant de valider la User Story sont :
US028-AT01 : Changement de mot de passe réussi
US028-AT02 : Échec de changement de mot de passe avec un ancien mot de passe erroné
US028-AT03 : Échec de changement de mot de passe avec deux nouveaux mots de passe différents

Il nécessite que l'utilisateur soit connecté au préalable, ainsi le test est une méthode de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing`.

5.3.8. US029 : réinitialiser son mot de passe

5.3.8.1. Titre de la User Story

US029 - En tant que Toto, je veux réinitialiser mon mot de passe.

5.3.8.2. Spécifications fonctionnelles : documentation pour les utilisateurs

Depuis la page de connexion (cf. US022), cliquer sur le lien « Mot de passe oublié ? ».
Saisir son email dans le champ « Courriel », et cliquer sur le bouton « Réinitialiser mon mot de passe ».
Une nouvelle page s'affiche avec une indication : aller consulter le mail reçu, et cliquer sur le lien fourni pour réinitialiser son mot de passe.
Le lien dirige vers une nouvelle page : saisir son nouveau mot de passe dans les champs « Nouveau mot de passe » et « Confirmation du nouveau mot de passe », et cliquer sur le bouton « Définir un mot de passe ».
Une nouvelle page s'affiche, suggérant de se connecter en cliquant sur le bouton « Se connecter ». Sur la page de connexion, suivre les indications de la US023.

5.3.8.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.profiles"

URL :
"/reset_password/", "/reset_password/done/", "/reset_password_confirm/<uidb64>/<token>/" et "/reset_password_complete/"

Vues (views) :
La **première vue** est fondée sur la classe `CustomPasswordResetView`, qui hérite de la classe `PasswordResetView` du module `django.views.auth.views`.

```
class CustomPasswordResetView(PasswordResetView):


    template_name = 'profiles/reset_password/password_reset.html'
    form_class = CustomPasswordResetForm
    subject_template_name = 'profiles/reset_password/' \
        'password_reset_subject.txt'
    email_template_name = 'profiles/reset_password/password_reset_email.html'
    success_url = 'done'



custom_password_reset_view = CustomPasswordResetView.as_view()
```

La **deuxième vue** est fondée sur la classe `CustomPasswordResetDoneView`, qui hérite de la classe `PasswordResetDoneView` du module `django.views.auth.views`.

```
class CustomPasswordResetDoneView(PasswordResetDoneView):


    template_name = 'profiles/reset_password/password_reset_done.html'



custom_password_reset_done_view = CustomPasswordResetDoneView.as_view()
```

La **troisième vue** est fondée sur la classe `CustomPasswordResetConfirmView`, qui hérite de la classe `PasswordResetConfirmView` du module `django.views.auth.views`.

```
class CustomPasswordResetConfirmView(PasswordResetConfirmView):
```

```
    template_name = 'profiles/reset_password/password_reset_confirm.html'
    form_class = CustomSetPasswordForm
    success_url = reverse_lazy('profiles:reset_password_complete')


custom_password_reset_confirm_view = CustomPasswordResetConfirmView.as_view()
```

La **quatrième vue** est fondée sur la classe `CustomPasswordResetCompleteView`, qui hérite de la classe `PasswordResetCompleteView` du module `django.views.auth.views`.

```
class CustomPasswordResetCompleteView(PasswordResetCompleteView):


    template_name = 'profiles/reset_password/password_reset_complete.html'


custom_password_reset_complete_view = CustomPasswordResetCompleteView.as_view()
```

Formulaires :
Le formulaire `CustomPasswordResetForm` hérite du formulaire `PasswordResetForm` du module `django.contrib.auth.forms` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.
Le champ `email` est requis.

Le formulaire `CustomSetPasswordForm` hérite du formulaire `SetPasswordForm` du module `django.contrib.auth.forms` et utilise le package *cripsy_forms* pour le rendu HTML et CSS.
Les champs `new_password1` et `new_password2` sont requis.

Modèle :
cf. US023

Gabarit (template) :
Les gabarits contenant les formulaires utilisent le package *cripsy_forms*.

5.3.8.4. Tests d'acceptation

Les tests d'acceptation (ou tests fonctionnels) sont présents dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Les tests fonctionnels permettant de valider la User Story sont :
US029-AT01 : Réinitialisation de mot de passe réussie
US029-AT02 : Échec de réinitialisation de mot de passe avec un email erroné
US029-AT03 : Échec de réinitialisation de mot de passe avec deux nouveaux mots de passe différents

Ils nécessitent que l'utilisateur ne soit pas connecté au préalable, ainsi les tests sont des méthodes de la classe `GeneralUserStories`**`Anonymous`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing`.

5.3.9. US030 : déconnexion de l'application

5.3.9.1. Titre de la User Story

US030 - En tant que Toto, je veux me déconnecter de l'application.

5.3.9.2. Spécifications fonctionnelles : documentation pour les utilisateurs

A partir de l'écran d'accueil du site (cf. US024), cliquer sur le lien « Déconnexion » présent dans la barre de navigation en hait de la page. Ainsi, on est redirigé vers la page confirmant la déconnexion (https://team-spirit.herokuapp.com/users/logout/).

Il est également possible de se déconnecter à partir de n'importe quelle autre page du site.

5.3.9.3. Spécifications techniques : documentation technique pour les développeurs

Application django :
"teamspirit.users"

URL de départ :
N'importe laquelle, par exemple l'accueil "/"

URL d'arrivée:
"users/logout/"

Vue (view) :
La vue est fondée sur la classe `CustomLogoutView`, qui hérite de la classe `LogoutView` du module `django.contrib.auth.views`.

```
class CustomLogoutView(LogoutView):


    template_name = 'users/logout.html'



custom_logout_view = CustomLogoutView.as_view()
```

Formulaire :
N/A

Modèle :
N/A

Gabarit (template) :
RAS

5.3.9.4. Tests d'acceptation

Le test d'acceptation (ou test fonctionnel) est présent dans les modules `test_us022_to_030.py`, situés à la fois dans les répertoires `tests.functional.chrome/` et `tests.functional.firefox/`.

Le test fonctionnel permettant de valider la User Story est :
US030-AT01 : Déconnexion réussie

Il nécessite que l'utilisateur soit connecté au préalable, ainsi le test est une méthode de la classe `GeneralUserStories`**`Authenticated`**`TestCase` qui hérite de la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing`.

6. Plan de tests

Le plan de tests concerne le traitement **côté serveur**, c'est-à-dire le code Python écrit avec le framework Django.

6.1. Périmètre des tests

Le périmètre des tests couvre les aspects suivants :
- Tests unitaires
- Tests d'intégration
- Tests fonctionnels (également appelés tests d'acceptation)

Les tests unitaires sont exécutés **de façon isolée** sur les modèles (et les éventuels modules qui les gèrent, appelés *managers*), les vues, les urls, les formulaires, les commandes et les éventuelles fonctions ou classes utilitaires.

Les tests d'intégration complètent les tests unitaires **lorsque des interactions surviennent** entre les différentes entités (telles que les applications Django).

Les tests fonctionnels **servent à valider les User Stories**, en vérifiant que les fonctionnalités sont opérationnelles, pour les navigateurs *Google Chrome* et *Mozilla Firefox*. Ils sont indiqués dans chaque User Story mentionnée dans la partie 5 du présent document.

6.2. Limites du périmètre des tests

Le périmètre des tests **ne couvre pas** les aspects suivants :
- Tests de performances
- Tests de charge
- Tests de sécurité

De plus, les tests fonctionnels **ne valident pas** le fonctionnement avec des navigateurs différents de *Chrome* et *Firefox*, comme par exemple *Safari*, *Microsoft Edge* et *Internet Explorer*.

6.3. Outils de tests

Les outils utilisés pour les tests sont les suivants :
- `unittest` et `django.test` pour les tests unitaires et les tests d'intégration
- la classe `StaticLiveServerTestCase` du module `django.contrib.staticfiles.testing` pour les tests fonctionnels
- *coverage* pour mesurer la couverture des tests

6.4. Organisation des tests

Tous les tests sont regroupés dans le package `tests/` placé à la racine du projet.
Ce répertoire contient trois sous-packages selon le type de tests :

```
tests/
|---  functional/
|---  integration/
|---  unit/
```

Les tests fonctionnels sont séparés en fonction de l'émulateur de navigateur web, puis regroupés dans des modules par package, selon la liste des User Stories établie dans le paragraphe 3.4.2 :

```
tests/
|---  functional/
      |---  chrome/
      |---  firefox/
|---  ...
```

Les tests unitaires sont organisés par application Django : chaque application Django est un sous-package. En fonction des besoins, des modules sont créés pour les modèles (et leurs éventuels *managers*), les vues, les urls, les formulaires, les commandes et les éventuelles fonctions ou classes utilitaires. L'arborescence est donc la suivante :

```
tests/
|---  unit/
      |---  app1/
            |---  test_commands.py
            |---  test_forms.py
            |---  test_managers.py
            |---  test_models.py
            |---  test_urls.py
            |---  test_views.py
      |---  app2/
            |---  test_commands.py
            |---  test_forms.py
            |---  test_managers.py
            |---  test_models.py
            |---  test_urls.py
            |---  test_views.py
      |---  ...
      |---  utils/
            |---  test_context_processors.py
|---  ...
```

Les tests d'intégration sont organisés, tout comme les tests unitaires, par application Django :

```
tests/
|---  integration/
      |---  app1/
      |---  app2/
      |---  ...
|---  ...
```

6.5. Rédaction et exécution des tests

6.5.1. Méthode

Les tests unitaires et les tests d'intégration sont écrits, dans la mesure du possible, avant la rédaction du code qu'ils valident, selon la méthode du développement mené par les tests (ou *TDD* : *Test Driven Development*). Ils sont exécutés avant et après la rédaction du code qu'ils valident, afin de vérifier que le dernier test écrit échoue initialement, et réussit ensuite.

Les tests fonctionnels sont écrits et exécutés après la rédaction du code des fonctionnalités, afin de valider les User Stories.

6.5.2. Rapport

Avec ce découpage en trois catégories, on recense 233 tests :

- 166 tests unitaires
- 50 tests fonctionnels (25 avec Chrome et 25 avec Firefox)
- 17 tests d'intégration

En exécutant la commande suivante à la racine du projet

```
python -m manage test
```

on obtient le résultat suivant :

```
Ran 233 tests in 73.034s

OK
```

Résultat de l'exécution des tests :

L'ensemble des tests réussit.

On utilise le module *coverage* pour vérifier la couverture de tests du projet. Afin d'obtenir un résultat consistant, certains répertoires importés depuis des libraires externes sont exclues du rapport. Pour cela, on utilise le fichier de configuration *.coveragerc* dont voici le contenu :

```
# .coveragerc to control coverage.py
[run]
command_line = "manage.py" test
source = .

[report]
# Regexes for lines to exclude from consideration
omit =
   tests*
   *migrations*
   runtests.py
   *apps.py
   *wsgi*
   *asgi*
   *site-packages*
   merge_production_dotenvs_in_dotenv.py
   config/*

ignore_errors = True
```

Ainsi, dans la console, l'exécution successive des commandes « coverage run », « coverage report » puis « coverage html » permet d'obtenir le rapport suivant (un certain nombre de lignes est masqué ici) :

Coverage report: 94%

Module ↓	*statements*	*missing*	*excluded*	*coverage*
docs/__init__.py	0	0	0	100%
docs/conf.py	8	8	0	0%
manage.py	16	6	0	62%
teamspirit/__init__.py	2	0	0	100%
teamspirit/budgets/__init__.py	0	0	0	100%
teamspirit/budgets/admin.py	0	0	0	100%
teamspirit/budgets/models.py	0	0	0	100%
teamspirit/budgets/views.py	0	0	0	100%
	[…]			
teamspirit/users/__init__.py	0	0	0	100%
teamspirit/users/adapters.py	0	0	0	100%
teamspirit/users/admin.py	15	0	0	100%
teamspirit/users/forms.py	35	6	0	83%
teamspirit/users/managers.py	28	4	0	86%
teamspirit/users/models.py	25	4	0	84%
teamspirit/users/urls.py	4	0	0	100%
teamspirit/users/views.py	36	4	0	89%
teamspirit/utils/__init__.py	0	0	0	100%
teamspirit/utils/context_processors.py	3	0	0	100%
Total	**860**	**54**	**0**	**94%**

Conclusion : avec un résultat de **94%**, le taux de couverture de tests du projet est **satisfaisant**.

7. Glossaire

Back end	Côté serveur. Un langage de programmation back end (ex : Python) est exécuté sur le serveur.
Framework	Ensemble de composants logiciels servant de socle au développement d'applications
Front end	Côté client (c'est-à-dire la machine de l'utilisateur). Un langage de programmation front end (ex : HTML) est exécuté sur le navigateur web de l'utilisateur.
N/A	Non applicable
RAS	Rien à signaler

A. Annexe : les *Personas*

Voici les *personas* qui ont permis de mieux cerner les utilisateurs de l'application à développer :

Président

QUI	Emmanuel, 33 ans, président de l'association et coureur
POURQUOI CE SITE ?	Emmanuel aimerait pouvoir centraliser la gestion des documents de l'association, et limiter l'accès à son contenu en fonction du rôle joué par chaque adhérent : présidence, trésorerie, secrétariat ou simple adhésion. Il aimerait également simplifier la communication autour des événements organisés par l'association.
OBJECTIFS	- Déléguer les tâches administratives aux membres du bureau - Superviser les tâches administratives - Connaître la liste des adhérents
COMPORTEMENT	Ludique : 20% Techno : 90% Implication : 90% Curiosité : 70% Disponibilité : 30%

Trésorier

QUI	Maxime, 35 ans, trésorier de l'association et coureur
POURQUOI CE SITE ?	Maxime gère parfaitement la trésorerie de l'association avec ses tableurs et son ordinateur, mais ce n'est pas très pratique pour travailler en collaboration avec son adjoint et le président. Il aimerait disposer d'un tableau de bord pour pérenniser la gestion de sa trésorerie.
OBJECTIFS	- Gérer le budget - Suivre le paiement de la cotisation des membres - Suivre le paiement de la participation des membres aux événements payants organisés par l'association - Suivre le paiement des tenues de sport officielles de l'association
COMPORTEMENT	Ludique : 20% Techno : 80% Implication : 90% Curiosité : 70% Disponibilité : 30%

Secrétaire

QUI	Axel, 36 ans, secrétaire de l'association et coureur
POURQUOI CE SITE ?	Axel rédige les compte-rendus de réunions, mais se trouve parfois en difficulté lorsqu'il s'agit de les diffuser à l'intégralité des adhérents.
OBJECTIFS	- Mettre à jour l'agenda des événements organisés par l'association - Diffuser les compte-rendus de réunions - Récupérer les documents administratifs des adhérents
COMPORTEMENT	Ludique : 10% Techno : 80% Implication : 50% Curiosité : 50% Disponibilité : 30%

Figure A.1 : personas 1/2 (package « gestion de l'association »)

Adhérent

QUI	Laurent, 45 ans, coureur régulier, y compris en compétition
POURQUOI CE SITE ?	Laurent s'entraîne très régulièrement. Il participe à des compétitions, individuellement ou avec d'autres membres de l'association. Il aimerait simplifier ses démarches administratives lorsque des événements sont organisés par l'association. Coureur aguerri, il aimerait également proposer ses services pour proposer des entraînements à d'autres membres.
OBJECTIFS	- Simplifier les démarches administratives - Proposer des créneaux d'entraînement
COMPORTEMENT	Ludique : 80% Techno : 90% Implication : 30% Curiosité : 80% Disponibilité : 30%

Adhérente

QUI	Claire, 42 ans, mère de famille, pratique la course à pied en loisir
POURQUOI CE SITE ?	Claire n'a pas beaucoup de temps à consacrer à ses loisirs, mais dès qu'elle le peut, elle aime sortir pratiquer la course à pied dans son quartier. Elle aime beaucoup l'état d'esprit de l'association, même si elle ne peut malheureusement pas souvent être présente aux entraînements et à certaines réunions.
OBJECTIFS	- Être informée des événements proposés par l'association - Lire les compte-rendus de réunions auxquelles elle n'a pas pu participer
COMPORTEMENT	Ludique : 10% Techno : 30% Implication : 20% Curiosité : 10% Disponibilité : 10%

Adhérent

QUI	Pierre, 36 ans, célibataire, aimant pratiquer la course à pied en groupe
POURQUOI CE SITE ?	Pierre est un nouvel adhérent de l'association. Très sociable, il désire connaître d'autres adhérents pour s'entraîner avec eux.
OBJECTIFS	- Rencontrer d'autres coureurs - S'entraîner en groupe - Commander la tenue de sport officielle de l'association
COMPORTEMENT	Ludique : 20% Techno : 80% Implication : 20% Curiosité : 50% Disponibilité : 30%

Figure A.2 : personas 2/2 (package « vie associative »)

B. Annexe : l'*impact mapping*

Voici les deux parties de l'*impact mapping* qui a permis de définir les fonctionnalités à développer :

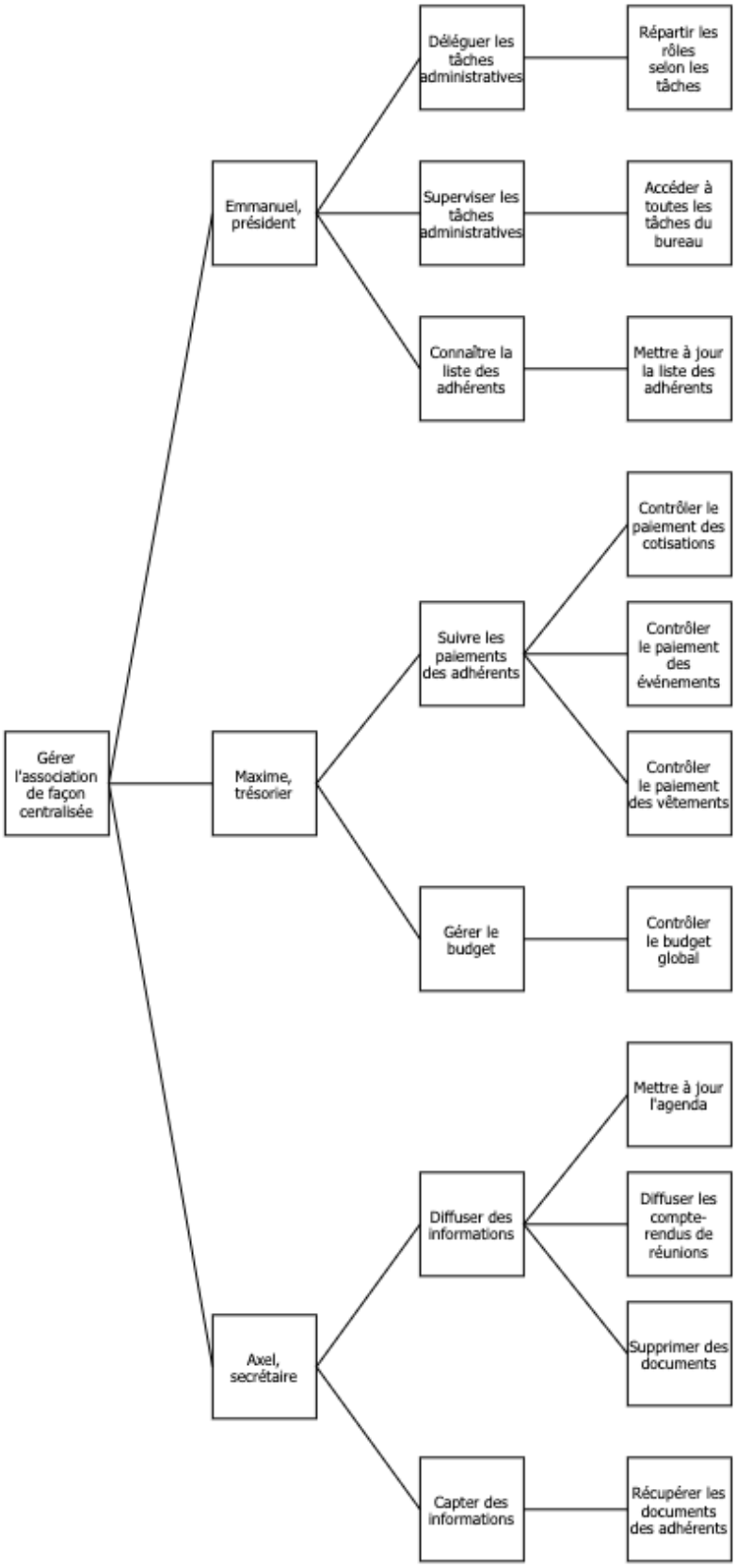

Figure B.1 : impact mapping 1/2 (package « gestion de l'association »)

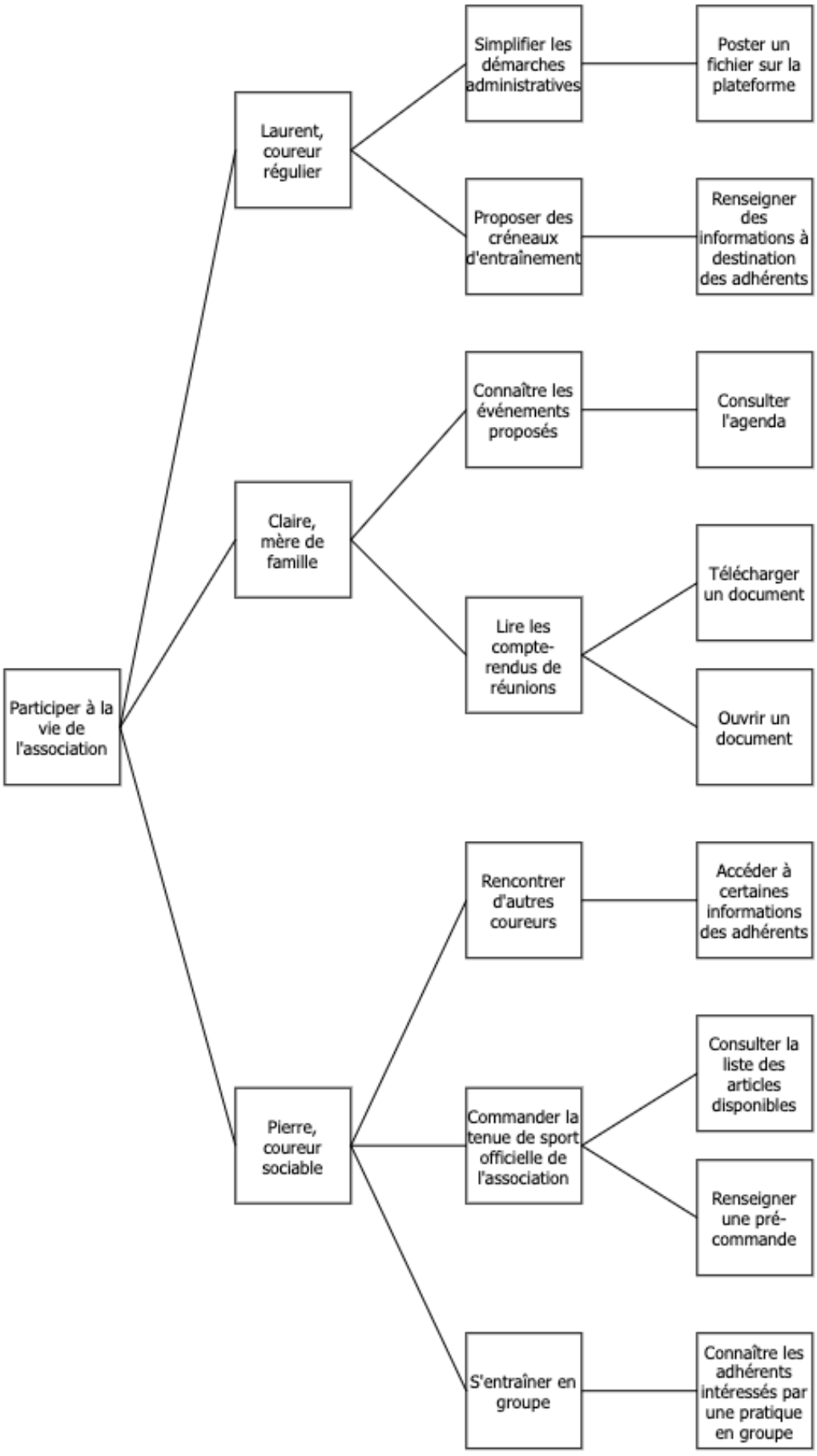

Figure B.2 : impact mapping 2/2 (package « vie associative »)