INFO
Département Informatique
IUT Belfort-Montbéliard

Gendarmerie nationale

Réalisation d'une application web visant à la gestion des ruches de la D.G.G.N.

RAPPORT TECHNIQUE

Du 08 avril au 14 juin 2019

Dimitri BURVENIQUE

2e année du DUT informatique

Direction Générale de la Gendarmerie Nationale (DGGN)

Guillaume PEPIN, chef de la section Mobilité

Mourad HAKEM, enseignant

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

IUT Belfort-Montbéliard

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD
Tél. +33 (0)3 84 58 77 00 • iut-bm.univ-fcomte.fr

Sommaire

Introduction

Ce rapport technique a pour but de vous expliquer les points techniques de différentes parties du code de ce projet.

Ce projet a pour objectif de simplifier les apiculteurs dans leurs tâches administratives et leurs gestions de ruches. Ce projet a été réalisé du 8 Avril 2019 au 14 juin 2019 pour la Direction Générale de la Gendarmerie Nationale (DGGN). Ce projet a été codé en Python avec le framework Django.

Ce rapport est donc composé d'une partie avec les prérequis, le cahier des charges, une partie permettant la création d'URL, d'autres montrant comme créer la base de données, mettre en place des formulaires, sur le système de connexion et inscription, un exemple de gestion, comment exporter dans différents formats ou encore l'envoie de mail.

1. Prérequis

Pour ce projet, il faut tout d'abord un environnement Linux, avoir de préférence Python 3 même si il est fourni dans le projet comme PIP (module d'installation des librairies Python). Pour cela, il faut se placer dans le répertoire du projet, et faire les commandes « alias python=venv/bin/python3.6 » et « alias pip=venv/bin/pip3.6 ».

Pour les imports de libraires, si besoin, il faut se rendre dans le fichier views.py dans le dossier « ruches » et importer les librairies qui ne viennent pas de Django et des fichiers du projet, c'est-à-dire, celles qui ne contiennent pas « django.* », « .forms » et « ruches.models ».

La base de données est en SQLite3.

Pour créer un superutilisateur, il faut entrer la ligne de commande « python manage.py createsuperuser » et suivre les instructions.

Pour lancer l'application ensuite, il faut écrire « python manage.py runserver » et se rendre sur le lien donné ensuite. Si vous voulez le lancer sur un autre serveur dont on a l'adresse (et un port différent), il faut alors faire « python manage.py runserver *adresse(*:*port)* ».

2. Cahier des charges

Le but de cette application est d'informatiser et d'automatiser le travail des apiculteurs concernant les visites, les traitements, les nourrissements, les récoltes ou encore les pesées. On doit ainsi pouvoir ajouter, modifier, supprimer et consulter des ruchers et ruches. L'application doit aussi permettre de générer des registres d'élevages de façon automatique mais aussi de pouvoir télécharger les feuilles de visites afin de simplifier l'affichage sur le site.

De plus, on doit pouvoir intégrer les données de capteurs implantées sur les ruches tels que la température, l'humidité, le pourcentage de la batterie et la diffusion de vidéos.

Pour cela, l'application devait être réalisées en Python avec le framework Django.

3. Les URL

Pour configurer les URLs de l'application, il faut se rendre dans le fichier « urls.py » qui se situe dans le dossier « ruches ». Dans ce ficher, il faut importer la fonction « path » depuis « django.urls » et le fichier « views.py » qui se trouve dans le répertoire courant.

Nous avons juste à remplir le tableau nommé « urlpatterns ». Pour cela, il faut donc utiliser la fonction « path » et mettre en argument une chaîne de caractère qui va représenter l'URL, la fonction associée qui se situe dans le fichier « views » et le nom de l'URL.

```
1   from django.urls import path
2
3   from . import views
4
5   urlpatterns = [
6       # vues communes
7       path('', views.home, name='home'),
8       path('home', views.home, name='homeUser'),
9       path('informationsUser', views.informationsUser, name='infosUser'),
10      path('detailCapteurUser/<str:idCapteur>/<str:rucher>/<str:colonie>', views.capteurUser, name='detailsUser'),
```

Illustration 1: Exemple de plusieurs choix d'URL

On peut donc voir sur ce bout de code à la ligne 9 où nous avons un URL classique, sans paramètres ajoutés. À la ligne 10, on peut voir un URL avec trois arguments de type chaîne de caractères ou encore à la ligne 12 où nous avons un exemple dans lequel il y a un paramètre de type entier et deux autres de type chaîne de caractères.

4. La base de données (Modèles)

Pour la base de données, elle se situe dans le fichier « models.py » dans le répertoire « ruches ». Dans ce fichier nous avons sous forme de « class » la base de données créée à partir du MCD ci-contre.

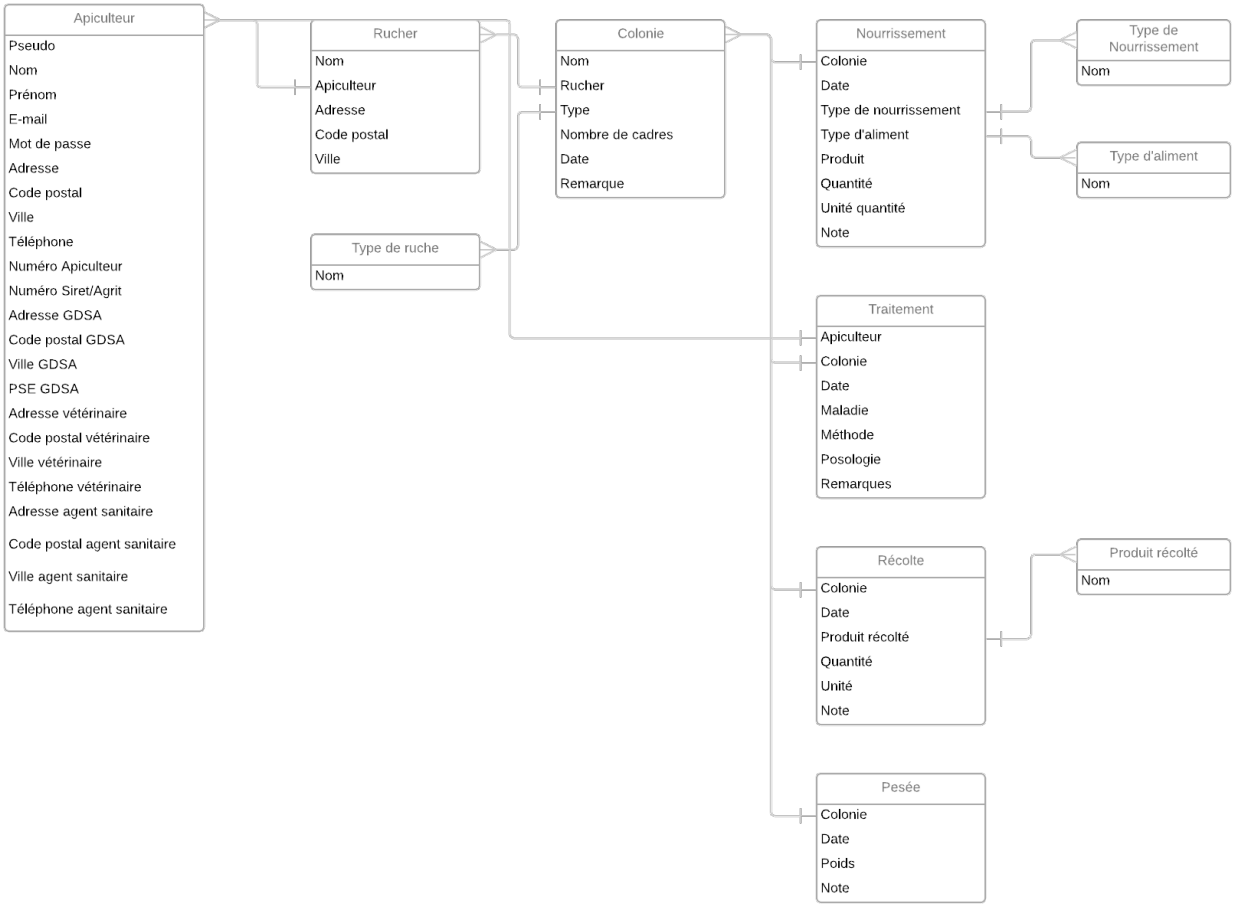

Illustration 2: MCD de la base de données (partie 1)

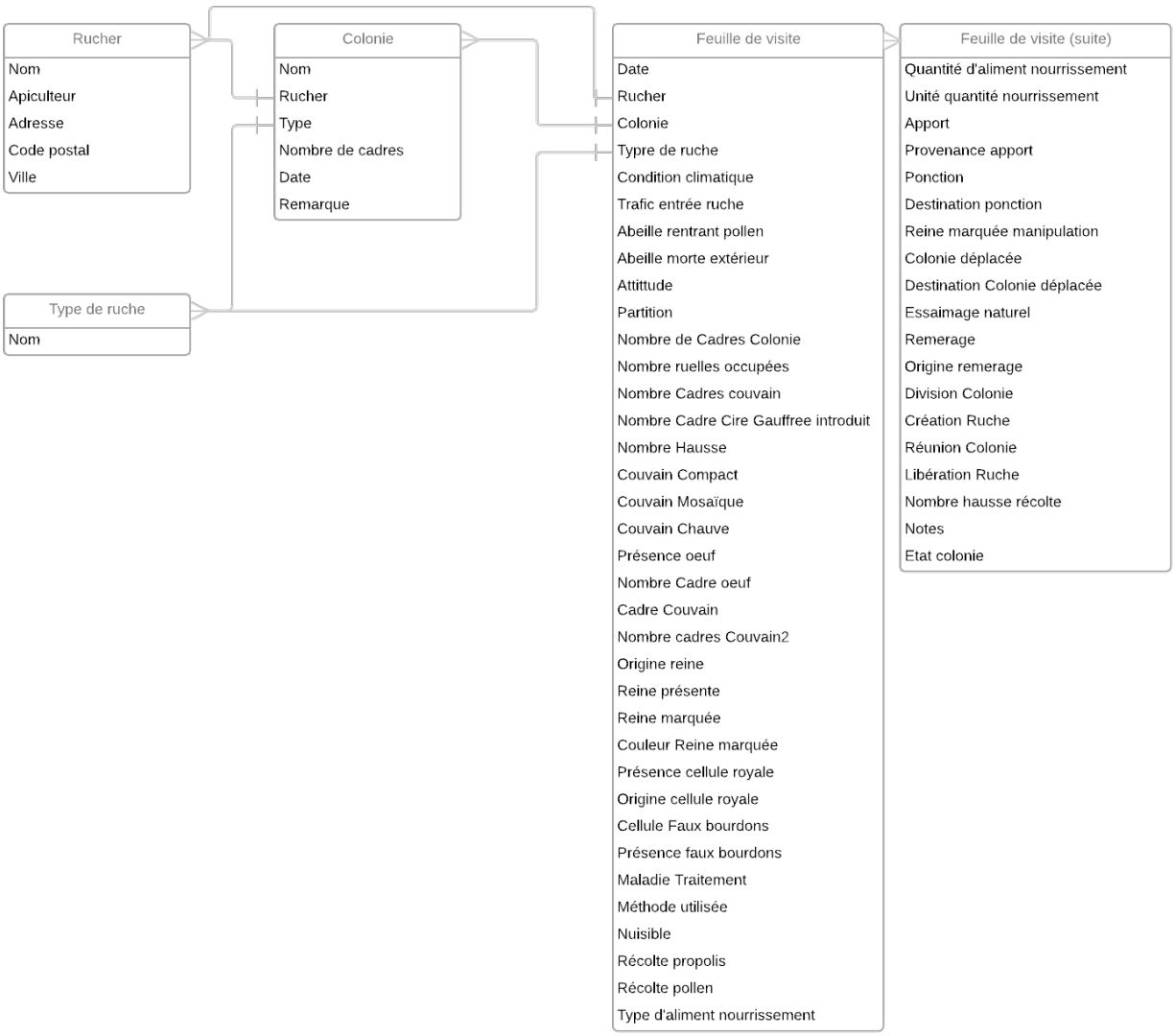

Illustration 3: MCD de la base de données (partie 2)

Pour cela, on nomme la classe avec le nom de la table voulue à laquelle on l'associe à la base de données. Pour chaque attribut, on spécifie le type ainsi que des options si besoin telles que la longueur maximale, si il doit être unique, si il peut être nulle ou vide, ou encore la valeur par défaut. On peut aussi définir le texte qui apparaîtra lors de la sélection de l'objet.

```
52  class Rucher(models.Model):
53      nom = models.CharField(max_length=200, unique=True)
54      user = models.ManyToManyField(User, null=True)
55      adresseP = models.CharField(max_length=200, default='')
56      adresseS = models.CharField(max_length=200, null=True, default='')
57      codePostal = models.CharField(max_length=5, default='')
58      ville = models.CharField(max_length=200, default='')
59
60      def __str__(self):
61          return self.nom
62
63
64  class TypeRuche(models.Model):
65      nom = models.CharField(max_length=200)
66
67      def __str__(self):
68          return self.nom
69
70
71  class Colonie(models.Model):
72      nom = models.CharField(max_length=200)
73      rucher = models.ForeignKey(Rucher, on_delete=models.CASCADE)
74      type = models.ForeignKey(TypeRuche, on_delete=models.CASCADE, null=True)
75      nombre_de_cadres = models.IntegerField(null=True)
76      date = models.DateTimeField(default=datetime.datetime.now, null=True)
77      remarque = models.CharField(max_length=1024, null=True)
78
79      def __str__(self):
80          return self.nom
81
```

Illustration 4: script montrant la base de données

À coté de « class » on retrouve le nom de la table et entre parenthèses la partie du code qui permet de l'associer à la base de données. On retrouve ensuite les attributs. Chaque attribut est associé à un type, par exemple, il peut être de type chaîne de caractères (ligne 53, 55, 57 , …) avec le code « models.CharField(…) » , de type date (ligne 76), de type entier (ligne 75), … ; il peut être aussi de type d'une autre entité comme à la ligne 54 où on signifie une relation plusieurs à plusieurs (mais on peut définir un à plusieurs ou encore un à un) dans laquelle on spécifie l'entité en question puis d'autres options (ici on autorise à ce qu'elle soit nulle). Un attribut peut avoir aussi une clé étrangère comme à la ligne 73 ou 74, où on spécifie donc l'entité associée et le type de suppression (ici en cascade). Concernant le nom associé quand on sélectionne un objet d'une entité, on le définit avec la fonction « __str__(self) » où on retourne le texte qu'on souhaite.

Quand on opère des changements sur ce fichier, il faut ensuite faire deux commandes dans le terminal. On doit se trouver dans le répertoire du projet puis faire les lignes suivantes : « python manage.py makemigrations » et « python manage.py migrate ».

Pour pouvoir modifier et afficher la base de données dans la partie administration de Django, il faut enregistrer les tables voulues dans le fichier « admin.py » dans le répertoire « ruches ».

5. Les formulaires

Pour créer des formulaires, il faut se rendre dans le ficher « forms.py » qui se situe dans le répertoire « ruches ». On crée ensuite une classe pour le formulaire voulu. On peut avoir des formulaires pour enregistrer de nouveaux objets d'entités de la base de données et d'autres non.

On peut spécifier le choix d'affichage des éléments (checkbox, boutons, liste déroulante, zone de texte, …), définir des choix à choisir, la longueur maximale à rentrer, définir si un champ est requis ou pas, le type d'entrée.

```
101 class NourrissementForm(ModelForm):
102     UNITE_CHOICES = [
103         ('gr', 'gr'),
104         ('L', 'L'),
105     ]
106 
107     uniteQuantite = forms.ChoiceField(choices=UNITE_CHOICES)
108     note = forms.CharField(widget=forms.Textarea, max_length=1024, required=False)
109     date = forms.DateTimeField(
110         input_formats=['%d/%m/%Y %H:%M'],
111         widget=forms.DateTimeInput(attrs={
112             'class': 'form-control datetimepicker-input',
113             'data-target': '#datetimepicker1'
114         })
115     )
116 
117     class Meta:
118         model = Nourrissement
119         fields = ['date', 'typeNourrissement', 'typeAliment', 'produit', 'uniteQuantite', 'quantite', 'note']
```

Illustration 5: Un des scripts permettant un formulaire d'enregistrement sur la base de données

Ici, nous avons le cas où nous avons un formulaire qui enregistre dans la base de données. Nous spécifions ceci grâce au « ModelForm » situé entre parenthèses à la ligne 101. On doit ensuite créer une sous-classe « Meta » dans laquelle on spécifie la table associée (ligne 118) et les champs qu'on souhaite remplir avec le formulaire à la ligne 119. si on veut personnaliser l'affichage ou le choix des attributs à remplir, on peut le faire comme indiqué entre les lignes 107 à 115. On peut mettre une liste déroulante (ligne 107) avec des choix (lignes 102 à 105), spécifier qu'on veut un affichage sous forme de zone de texte pour les « notes », la longueur maximale du texte et si le champ est requis (ligne 108) ou encore définir le format d'entrée pour la date et lui ajouter des attributs HTML aux lignes 109 à 115.

```python
class EmailForm(forms.Form):
    ADMIN_CHOICES = []
    usersObj = User.objects.all()
    for u in usersObj:
        if u.is_staff:
            ADMIN_CHOICES.append((u.id, u.username))

    DEMANDE_CHOICES = [
        ('transfert dans rucher(s)', 'Mouvement apiculteur'),
        ('suppression dans rucher(s)', 'Suppression dans rucher(s)'),
        ('suppression de compte', 'Suppression de compte'),
        ('Autre', 'Autre')
    ]

    ruchersObj = Rucher.objects.all()
    RUCHERS_CHOICES = []
    for r in ruchersObj:
        RUCHERS_CHOICES.append((r.nom, r.nom))

    demande = forms.ChoiceField(choices=DEMANDE_CHOICES)
    admins = forms.ChoiceField(choices=ADMIN_CHOICES, required=False)
    ruchers = forms.MultipleChoiceField(widget=forms.CheckboxSelectMultiple, choices=RUCHERS_CHOICES, required=False)
    message = forms.CharField(widget=forms.Textarea, required=False)
```

Illustration 6: Un des scripts qui permet l'affichage d'un formualaire sans enregistrement dans la base de données

Dans ce script on utilise « forms.Form » pour informer Django qu'on fait un formulaire qui n'utilise pas la base de données. On retrouve ici les mêmes options que dans le précédent formulaire pour les choix d'affichage des champs. Cependant, ici on ne crée pas de sous-classe « Meta » vu qu'on n'associe pas ce formulaire à la base de données.

6. Connexion / inscription

Pour l'inscription, il faut vérifier si l'appel de la fonction « inscription » dans le fichier « views.py » soit « POST ». Si elle ne l'est pas, on crée une variable dans laquelle on instancie le formulaire d'utilisateur basique intégré à Django, puis on instancie une autre variable avec le formulaire d'inscription d'apiculteur.

Si l'appel est de type « POST », on crée une variable qu'on instancie avec le formulaire d'inscription d'utilisateur basique de Django avec l'objet reçu qui est « request.POST ». On vérifie après si le formulaire est valide, on le sauvegarde en tant qu'entité « User », on récupère le pseudo et mot de passe entré afin de l'authentifier directement. On crée par la suite un objet de type Apiculteur qui a pour lien l'entité « User » qu'on vient d'ajouter, on instancie ensuite une variable avec le formulaire d'apiculteur avec l'objet « request.POST » et l'instance de l'objet « Apiculteur » qu'on vient de créer. On teste si le formulaire est valide, on écrase alors l'objet « Apiculteur » précédent, on connecte l'utilisateur et on le fait aller sur la page d'accueil.

```
1263    def inscription(request):
1264        if request.method == 'POST':
1265            user_form = UserForm(request.POST)
1266            if user_form.is_valid():
1267                user_form.save()
1268                username = user_form.cleaned_data.get('username')
1269                raw_password = user_form.cleaned_data.get('password1')
1270                user = authenticate(username=username, password=raw_password)
1271                obj = Apiculteur.objects.create(user=user)
1272                api_form = ApiForm(request.POST, instance=obj)
1273                if api_form.is_valid():
1274                    api_form.save()
1275                    login(request, user)
1276                    return redirect('home')
1277            else:
1278                print("test invalid form")
1279                form_errors = user_form.errors
1280                print(form_errors)
1281                erreurs = []
1282                for error in form_errors:
1283                    if error == "password2":
1284                        erreurs.append("Erreur mot de passe")
1285                    if error == "username":
1286                        erreurs.append("Erreur Username")
1287                return render(request, 'registration/inscription.html',
1288                              {'user_form': user_form, 'errorsForm': erreurs})
1289        else:
1290            print("test error post")
1291            user_form = UserForm()
1292            api_form = ApiForm()
1293        return render(request, 'registration/inscription.html', {'user_form': user_form, 'api_form': api_form})
1294
```

Illustration 7: Script permettant l'inscription

Dans ce script, les variables avec les formulaires pour l'affichage sont aux lignes 1291 et 1292, pour les variables après que l'utilisateur ait appuyé sur le bouton inscrit, l'instanciation se fait aux lignes 1265 et et 1272.

La récupération des pseudos et mot de passe se situe aux lignes 1268 et 1269, l'authentification est à la ligne 1970, le retour à la page d'accueil est à la ligne 1976.

7. Gestions

7.1. Affichages

Pour afficher les ruchers, on appelle donc la fonction « afficherRuchers » dans le fichier « views.py ». Cette fonction appelle tous les objets de type « Ruchers » mais aussi tous les objets de type « Colonie » afin de d'enlever les feuilles de visites qui ne sont pas complètements remplies mais aussi pour compter le nombre de colonies par ruchers. Quand on appelle tous les ruchers, ils sont déjà triés automatiquement par ordre alphabétique.

Pour les ruches il faut aussi les trier par ruchers, et afficher les traitements, feuilles de visite, récoltes, pesées.

```
508  def afficherRuchers(request):
509      ruchers = Rucher.objects.all()
510      colonies = Colonie.objects.all()
511      for c in colonies:
512          colonieObj = Colonie.objects.get(nom=c, rucher=c.rucher)
513          feuillesObj = FeuilleVisite.objects.all().filter(colonie=colonieObj)
514          for f in feuillesObj:
515              if f.notes is None:
516                  f.delete()
517      nombreColoRuchers = []
518
519      nombre = 0
520      for r in ruchers:
521          for c in colonies:
522              if c.rucher == r:
523                  nombre += 1
524          nombreColoRuchers.append({'rucher': r, 'nombre': nombre})
525          nombre = 0
526
527      print(nombreColoRuchers)
528
529      return render(request, 'Apiculteurs/affichage/afficherRuchers.html', {'ruchers': ruchers, 'nombreColo': nombreColoRuchers})
```

Illustration 8: Script pour afficher les ruchers

Dans ce script, aux lignes 509 et 510 on importe tous les objets de type « Rucher » et « Colonie » (ruches). Ensuite on a le code qui permet de supprimer les feuilles de visites incomplètes. On calcule ensuite le nombre de colonies par ruchers (lignes 519 à 525). Enfin, on retourne la vue avec le fichier HTML associé et les arguments en paramètres pour les afficher dans la page HTML.

7.2. Ajouts

Pour ajouter, on a sensiblement les mêmes étapes que pour s'inscrire. On vérifie si on appelle la méthode en « POST » Si ce n'est pas le cas, on appelle le formulaire dont on a besoin. Puis après l'appel en « POST » on le rempli avec « request.POST », on teste si le formulaire est valide puis on le sauvegarde et on redirige l'utilisateur vers une autre page.

```
532  def ajouterColonie(request):
533      if request.method == 'POST':
534          print("true post")
535          rucheForm = RucheForm(request.POST)
536          if rucheForm.is_valid():
537              print("true valid")
538              rucheForm.save()
539              return redirect('afficherColonies')
540          else:
541              print("else valid")
542              form_errors = rucheForm.errors
543              print(form_errors)
544      else:
545          print("else post")
546          user = request.user
547          rucheForm = RucheForm()
548      return render(request, 'Apiculteurs/creation/createColonie.html',
549                    {'form': rucheForm})
```

Illustration 9: exemple de script pour un ajout (ici de Colonie)

Ici nous avons le script qui nous permet d'ajouter une colonie à la base de données. On voit aux lignes 535 et 547 les appels au formulaire de l'objet pour la création d'un objet « ruche » et aux lignes 536 et 538, le test si le formulaire est valide et la sauvegarde du formulaire. Enfin nous avons la redirection à la page « afficherColonies » à la ligne 539.

7.3. Modifications

Concernant les modifications, c'est la même chose que pour ajout à un détail près. En effet, on doit récupérer l'Id de l'objet à modifier et importer l'objet. Puis, on utilise le formulaire de l'objet et on l'instancie avec l'objet en question afin d'avoir déjà les informations préremplies quand on a l'affichage du formulaire de modification. Une fois que l'utilisateur valide sa ou ses modification(s), on ajoute là aussi l'instance de l'objet en question afin que le formulaire enregistre dans cet objet (voir exemple dans la partie « Incsription » pour la partie instanciation de l'objet dans un formulaire.

7.4. Suppression

Pour supprimer un objet, on a besoin de l'Id de l'objet, de récupérer l'objet et applique la fonction « delete() » à l'objet dans un « try ».

8. Exports

8.1. PDF / QR Code

Dans cette partie, nous allons voir comment créer un QR Code mais aussi comment transformer en PDF un objet.

```
317
318     # qrcode
319     def render_png_to_pdf(request, c_id):
320         link_to_post = "127.0.0.1:8000/afficherColonieId/{}".format(c_id)
321         qr = qrcode.QRCode(
322             version=1,
323             error_correction=qrcode.constants.ERROR_CORRECT_L,
324             box_size=5,
325             border=4,
326         )
327         qr.add_data(link_to_post)
328         qr.make(fit=True)
329
330         img = qr.make_image()
331         img.save('static/ProjetRuches/images/qrcode.png')
332
333         image_data = Image.open("static/ProjetRuches/images/qrcode.png")
334         filename = 'static/ProjetRuches/images/qrcode.png'
335
336         response = HttpResponse(content_type='application/pdf')
337         response['Content-Disposition'] = 'attachment; filename="qrcode.pdf"
338
339         # Create the PDF object, using the response object as its "file."
340         p = canvas.Canvas(response)
341
342         # Draw things on the PDF. Here's where the PDF generation happens.
343         # See the ReportLab documentation for the full list of functionality
344         p.drawImage(filename, 0, 650)
345
346         # Close the PDF object cleanly, and we're done.
347         p.showPage()
348         p.save()
349         return response
```

Illustration 10: script permettant la génration de QR Code et exemple d'export en PDF

Dans ce script à la ligne 320, on crée une variable dans laquelle on associe le lien URL que l'on veut, ici on décide de faire un lien vers la page de la colonie qu'on scanne. Dans les lignes suivantes (jusqu'à la ligne 326) on instancie un objet QR Code avec les arguments associés. L'argument qui nous intéresse est « box_size » (ligne 324) qui nous permet de définir la longueur d'un côté du QR Code en centimètres. On associe ensuite le lien URL au QR Code, puis on crée l'image et on l'enregistre.

Viens ensuite la partie génération de PDF. On va d'abord chercher l'image où on l'a enregistré. On crée ensuite un objet « HTTPReponse » de type PDF (on peut m'être tous

les types MIME). On définit ensuite le nom du fichier PDF. On crée un objet PDF, on « dessine » ensuite l'image et on définit en même temps la position sur la page (base en bas a gauche de la page). Et on sauvegarde les modifications. Et on retourne la réponse.

8.2. XLS

Pour créer un fichier XLS, on doit créer un objet de type « Workbook », on crée ensuite une feuille avec un nom en option, on peut définir si l'affichage doit être en mode portrait ou pas. Ensuite, on écrit dans la cellule qu'on veut (définie avec des nombres et pas de lettres) et si on veut fusionner des cellules on écrit la ligne de départ puis la ligne de fin et la colonne de début puis la colonne de fin. On ajoute ensuite le texte souhaité puis le style que l'on veut (en option).

Pour exporter le fichier créé, une deuxième solution légèrement différente est utilisée et est utilisée pour générer le PDF de feuille de visite.

```
1249        classeur.save(path)
1250
1251        fs = FileSystemStorage('/tmp')
1252        with fs.open('mycla.xls') as xls:
1253            response = HttpResponse(xls, content_type='application/vnd.ms-excel')
1254            name = str("registre elevage {}-{}.xls".format(rucher.nom, annee))
1255            print(name)
1256            response['Content-Disposition'] = 'attachment; filename="{}"'.format(name)
1257            return response
```

Illustration 11: Un des scripts permettant l'export (ici en XLS)

Dans un premier temps, on récupère le répertoire dans lequel est sauvegardé le fichier, on ouvre le ficher avec la fonction « open », puis comme pour l'exemple précédent on choisit le type de la réponse, on définit le nom du fichier et on retourne la réponse.

9. E-mail

Pour envoyer un mail aux administrateurs, on a donc créé une adresse e-mail Gmail qui va servir à envoyer les mails. Pour cela, il faut configurer l'adresse e-mail dans le fichier « settings.py » qui se trouve dans le répertoire « projet ». Dans le code suivant, on a configuré pour un serveur de mail Gmail. Pour un autre serveur mail, il faut donc une configuration légèrement différente.

```
134  EMAIL_HOST = 'smtp.gmail.com'
135  EMAIL_HOST_USER = 'projetruches@gmail.com'
136  EMAIL_HOST_PASSWORD = 'projetruches1!'
137  EMAIL_PORT = 587
138  EMAIL_USE_TLS = True
```

Illustration 12: Paramètres pour l'envoi d'e-mail

Ici, nous rentrons le nom du serveur en premier, l'adresse e-mail, le mot de passe associé, le port pour le serveur et enfin l'autorisation l'utiliser le protocole TLS essentiel au serveur Gmail.

Pour la partie code dans le fichier « views », on choisit le ou les administrateurs à qui on veut envoyer le mail, puis on rédige le texte du mail sous forme HTML. On crée ensuite le mail en utilisant la méthode « EmailMultiAlternative » qui nous permet de renseigner le sujet du mail, le texte du mail, l'adresse utilisée pour envoyer le mail et enfin sous forme de tableau, la ou les adresses destinataires. On indique ensuite le type de message envoyé (ici HTML) et on l'envoie avec la fonction « send ».

```
1528            if u.is_staff:
1529                admins.append(u.email)
1530        html_content = "Bonjour,<br>" \
1531                       "{} {} vous envoie ce message : <br> {} <br>" \
1532                       "Vous pouvez lui repondre à cette adrresse : {} <br>" \
1533                       "Cordialement,<br>" \
1534                       "Le site des Ruches".format(nom, prenom, message, adresseMail)
1535        print(admins)
1536        msg = EmailMultiAlternatives(
1537            "Demande de renseignements",
1538            html_content,
1539            "projetruches@gmail.com",
1540            admins,
1541        )
1542        msg.content_subtype = "html"
1543        msg.send()
```

Illustration 13: Exemple de script envoyant un mail

10. Les évolutions possibles

Ce projet peut avoir plusieurs évolutions possibles. Notamment avec la partie capteurs où on pourrait créer des alertes aux demandes des apiculteurs. On peut imaginer permettre aux administrateur un plus large choix d'administration comme changer les icônes ou le texte de manière plus directe et éviter de passer par le code.

Table des illustrations

Table des matières