Documentation: dashboard professeur et développement dirigé par les tests

Version 0.1

Bryan Oberson

23 March 2015

Table des matières

1 Fonctionnalités du dashboard **1**
- 1.1 Ajouter un groupe . . . 1
- 1.2 Gérer un groupe . . . 1
- 1.3 Voir ses exercices . . . 3
- 1.4 Changer de mot de passe . . . 4

2 Guide du développeur **7**
- 2.1 Utilité des différents fichiers de mon projet . . . 7
- 2.2 Modèles . . . 8
- 2.3 Vues . . . 10
- 2.4 Urls . . . 15
- 2.5 Navigation . . . 15

3 Développement dirigé par les tests **17**
- 3.1 Les tests fonctionnels . . . 17
- 3.2 Les tests unitaires . . . 17
- 3.3 Le cycle du développement dirigé par les tests . . . 18
- 3.4 Gain de temps ? . . . 18

4 Débuter un projet **19**
- 4.1 Eléments requis . . . 19
- 4.2 Premier test . . . 19

CHAPITRE 1

Fonctionnalités du dashboard

1.1 Ajouter un groupe

La fonctionnalité de base de ce dashboard est la création de groupe. En créant un groupe, le professeur sera par la suite capable de le gérer en gérant les membres qui s'y trouvent mais aussi en y assignant des devoirs.

Pour créer un groupe, il suffit de se rendre sur Nouveau groupe, tout en bas dans le menu de gauche. Cette action fera apparaître le formulaire de création de groupe.

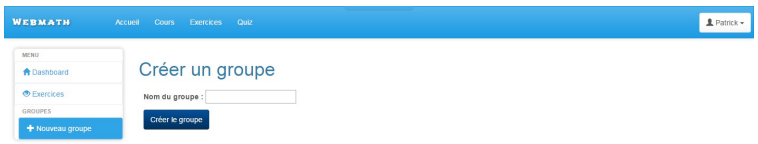

FIGURE 1.1 – Formulaire de création de groupe

La seule exigence présente lors de la création d'un groupe est le nom. Une fois le groupe créé, l'utilisateur actuel est automatiquement défini en tant que professeur pour le groupe.

Le groupe précédemment créé sera désormais affiché en permanence dans le menu de gauche du professeur, ce qui lui permet d'accéder à ses informations.

1.2 Gérer un groupe

1.2.1 Gérer les membres du groupe

Depuis cette page, le professeur peut gérer les membres qui sont actuellement enregistrés dans le groupe.

Il peut tout d'abord rajouter les élèves ou professeurs qu'il souhaite en entrant leur nom d'utilisateur dans le champ à disposition.

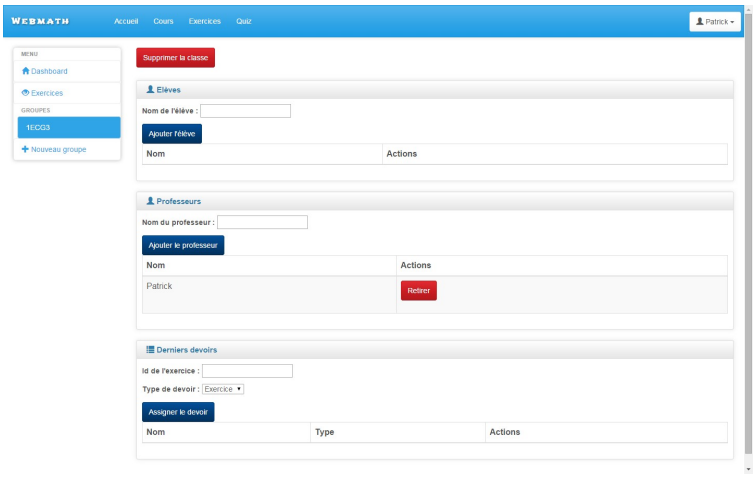

FIGURE 1.2 – Page d'administration d'un groupe

Si le nom d'utilisateur rentré correspond bien à un étudiant ou à un professeur, cet utilisateur sera rajouté dans la liste des membres.

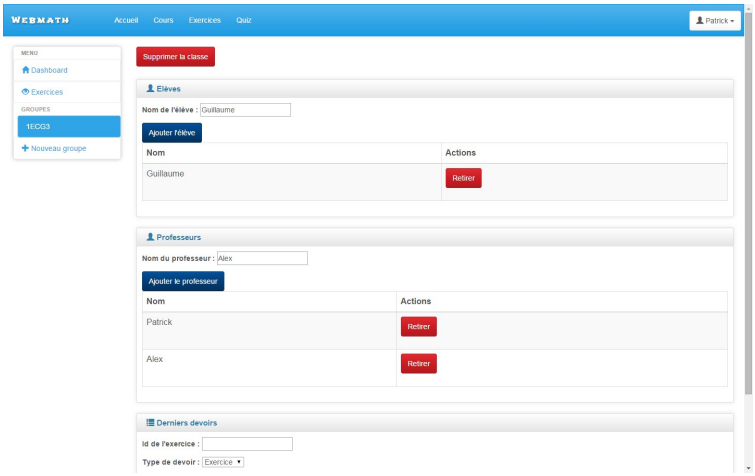

FIGURE 1.3 – Ce à quoi ressemble la page une fois que des membres ont été rajoutés

Au contraire, si aucun utilisateur n'a été trouvé ou si l'utilisateur ne correspond pas au rôle qu'il lui est donné (par exemple si c'est un professeur et qu'il a été ajouté aux étudiants), le site renvoiera un message d'erreur.

Une fois ajouté, un membre peut facilement être retiré du groupe grâce au bouton Retirer qui se trouve à côté de son nom.

1.2.2 Gérer un devoir

Un professeur peut bien évidemment donner des devoirs à son groupe.

Un devoir peut-être un exercice, un quiz ou un cours, tous les trois pouvant avoir été créé par un autre utilisateur.

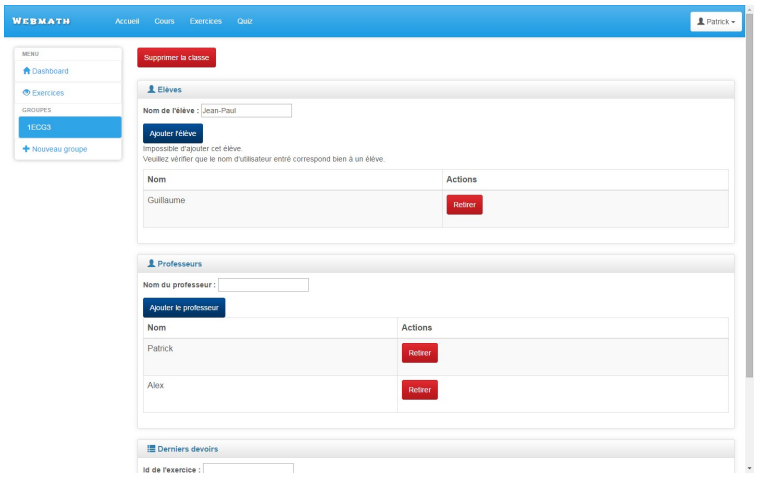

FIGURE 1.4 – Message d'erreur retourné si l'utilisateur n'est pas valable

Pour assigner un devoir, il suffit de savoir l'id de l'exercice, quiz ou cours, et de préciser grâce au menu à choix de quel type de devoir il s'agit.

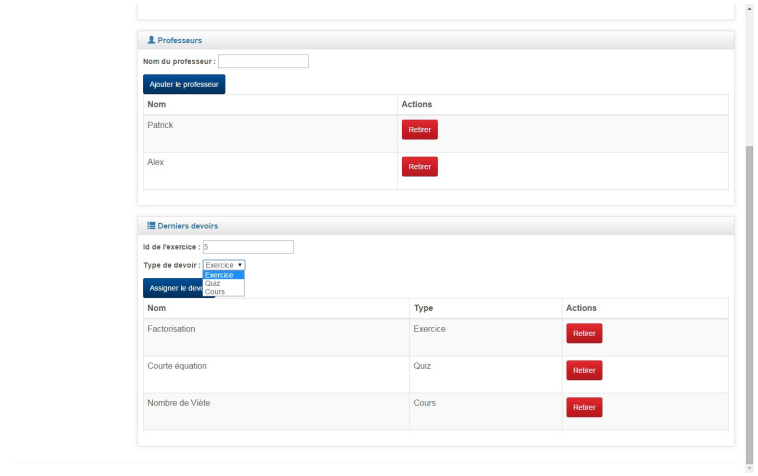

FIGURE 1.5 – Différents champs à compléter pour assigner un devoir

Comme pour les fonctionnalités précédentes, si aucun exercice, quiz ou cours n'a pu être associé à l'id entrée, un message d'erreur sera renvoyé.

Un devoir peut être à tout moment retiré grâce au bouton Retirer à sa droite.

1.3 Voir ses exercices

Dans le menu de gauche, il y a un bouton nommé Exercices. C'est depuis cette page que le professeur pourra voir ses exercices, ses quiz et ses cours.

Pour chaque activité que le professeur aura créé, il pourra voir le titre qu'il lui a donné, la date à laquelle il l'a créé et l'id qui lui sera utile s'il veut l'assigner en tant que devoir à un de ses groupes.

Il peut bien évidemment supprimer une activité en utilisant le bouton Supprimer se trouvant dans la dernière colonne du tableau.

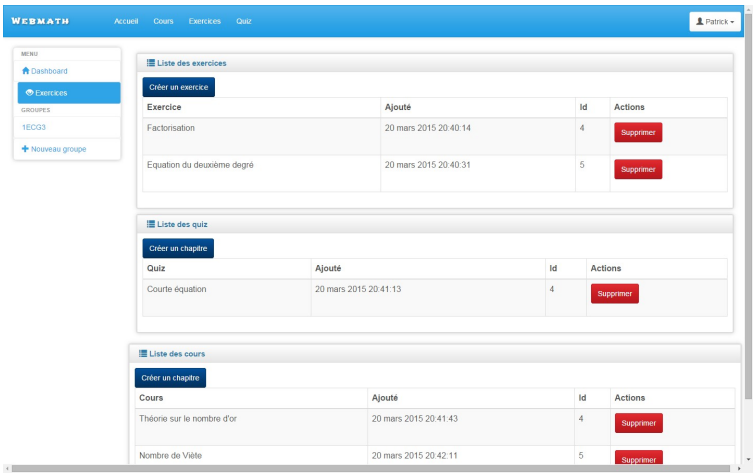

FIGURE 1.6 – Ce à quoi ressemble la page Exercices

Si le professeur souhaite créer une nouvelle activité, il n'a qu'à utiliser le bouton Créer en haut du tableau qui le redirigera directement au formulaire de création.

1.4 Changer de mot de passe

Peu importe sur quelle page il se trouve, le professeur peut accéder à un menu déroulant en haut à droite de cette page.

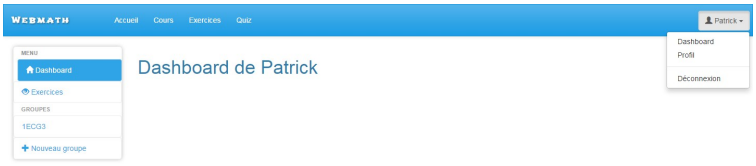

FIGURE 1.7 – Apparence du menu déroulant

Dashboard amène le professeur sur l'accueil de son dashboard, Déconnexion le déconnecte et Profil l'amène sur un formulaire de changement de mot de passe.

Pour le modifier, le professeur n'a qu'à remplir les deux champs et à valider. Si tout a été rentré correctement, le mot de passe sera correctement modifié.

Au contraire, s'il y a une erreur, un message d'erreur sera retourné.

FIGURE 1.8 – Message pour confirmer que le changement de mot de passe a correctement eu lieu

FIGURE 1.9 – Message d'erreur retourné si les champs n'ont pas correctement été remplis

CHAPITRE 2

Guide du développeur

2.1 Utilité des différents fichiers de mon projet

2.1.1 Dossier `static`

Le dossier static est utilisé pour garder tous les fichiers tel que les images, les fichiers CSS ou les fichiers Javascript. Dans cette application, il contient les dossiers suivants :

— `bower_components` : ce dossier contient tous les éléments du front-end qui possèdent des dépendances, comme les fichiers `bootstrap` ou des fichiers de base pour `jquery`.
— `css` : dans ce dossier se trouvent tous les fichiers css qui sont nécessaires pour le design du site.
— `fonts` : le dossier `fonts` de l'application dashboard contient toutes les informations relatives aux petits signes (`glyphicons`) qui sont utilisés dans le dashboard, comme le + devant «Nouveau groupe».

2.1.2 Dossier `templates`

Dans le dossier `templates` se trouvent tous les fichiers `html` qui sont utilisés dans l'application et retournés par les différentes vues.

2.1.3 Fichiers propres à l'application

Les applications Django possèdent les fichiers de base suivants :

— `models.py` qui est utilisé pour créer les différents modèles et leur attribuer des champs
— `admin.py` est utilisé pour signaler à Django quels sont les modèles qui doivent apparaître dans l'application admin. Une fois qu'ils y apparaissent, il est possible de créer, modifier ou supprimer n'importe quel object depuis cette application.application
— Le fichier `forms.py` est celui dans lequel on peut entrer les différents formulaires dont l'on a besoin pour l'application.
— C'est dans `views.py` que l'on peut stocker des variables nécessaires dans certains templates, mais aussi réaliser certaines actions comme la suppression d'un objet. A la fin d'une vue, on retourne souvent un fichiers `html` ou on redirige vers une autre vue.
— Le fichier `urls.py` contient les informations concernant les différentes urls accessibles par l'utilisateur et quelles vues sont censées être utilisées.

2.1.4 Fichiers uniques de Django

On peut modifier le fichier `settings.py` afin de définir la zone temporelle dans laquelle on se trouve, mais aussi les applications qu'un projet doit gérer ou encore l'emplacement du fichier `static`. Il sert donc de configuration de

base pour un projet.

Il y a aussi un autre fichiers `urls.py` qui, lui, est très utile si l'on doit s'occuper de plusieurs applications à la fois. En effet, on peut définir le début de l'url et rediriger vers un autre fichier `urls.py`.

2.2 Modèles

2.2.1 Modèles utilisés pour le dashboard

```
from django.db import models
from django.contrib.auth.models import User


#Profile de base découlant de User

class BaseProfile(models.Model):
    user = models.OneToOneField(User) #Donne les attributs de User à BaseProfile
    avatar = models.ImageField(null=True, blank=True, upload_to="avatars/")

    class Meta:
        abstract = True


#Les deux modèles héritent de BaseProfile et donc de User

class Teacher(BaseProfile):

    def __str__(self):
        return "Professeur {0}".format(self.user.username)

class Student(BaseProfile):

    def __str__(self):
        return "Etudiant {0}".format(self.user.username)


#
# Modèle de Keran pour les cours
#

class Course(models.Model):
    title = models.CharField(max_length=30, unique=True)
    description = models.TextField()
    difficulty = models.IntegerField()
    published = models.BooleanField(default=False)

    author = models.ForeignKey(Teacher)
    #chapter = models.ForeignKey('teachers.Chapter', related_name="courses")
    favorites = models.ManyToManyField(User, related_name="favorite_courses", blank=True, null=True)
    # videos = models.ManyToManyField(Video)
    # images = models.ManyToManyField(Image)
    # definitions = models.ManyToManyField(Definition)

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

```

```
48      def __str__(self):
49        return self.title
50
51  #
52  # Modèle de Florian pour les exercices
53  #
54
55  class Exercise(models.Model):
56
57      owner = models.ForeignKey(Teacher)  # créateur de l'exercice
58      created_on = models.DateTimeField(auto_now_add=True) # Date de création
59      updated_on = models.DateTimeField(auto_now=True)
60      title = models.CharField(max_length=30) # C'est le titre de l'exercice ( factorisation ou dévelop
61      equation = models.CharField(max_length=50) # C'est l'équation entrée par le professeur
62      grade = models.CharField(max_length=60) # donnée une note de difficulté à l'exercice
63      correction = models.CharField(max_length = 200) # Ceci est le corrigé de l'exercice ( obligatoire
64      def __str__(self):
65          return self.title
66
67  #
68  # Modèle de Benoit pour les quiz
69  #
70
71  class Quiz(models.Model): #Infos générales sur le quiz
72      title = models.CharField(max_length=100)
73      creation_date = models.DateTimeField(auto_now_add=True)
74      code = models.CharField(max_length=1000) #Format texte du quiz
75      author = models.ForeignKey(Teacher)
76      #id_chapter = models.ForeignKey('teachers.Chapter')
77
78      def __str__(self):
79          return self.title
80
81
82
83
84  #Modèle pour les groupes
85  class Group(models.Model):
86      name = models.CharField(max_length=30)
87      teacher = models.ManyToManyField(Teacher, through='GroupMembers')
88      student = models.ManyToManyField(Student, through = 'GroupMembers')
89      homeworkExercise = models.ManyToManyField(Exercise, through = 'AssignHomework') #uniquement les d
90      homeworkCourse = models.ManyToManyField(Course, through = 'AssignHomework') #uniquement les devo
91      homeworkQuiz = models.ManyToManyField(Quiz, through = 'AssignHomework') #uniquement les devoirs d
92      created_on = models.DateTimeField(auto_now=True)
93
94      def __str__(self):
95          return"Classe {0}".format(self.name)
96
97
98  #Table intermédiaire pour affecter un membre à un groupe
99
100 class GroupMembers(models.Model):
101     teacher = models.ForeignKey(Teacher, null = True)
102     student = models.ForeignKey(Student, null = True)
103     group = models.ForeignKey(Group)
104     added_on = models.DateTimeField(auto_now=True)
105
```

```
106 #Table intermédiaire pour assigner un devoir à un groupe
107
108 class AssignHomework(models.Model):
109     group = models.ForeignKey(Group)
110     exercise = models.ForeignKey(Exercise, null = True)
111     quiz = models.ForeignKey(Quiz, null = True)
112     course = models.ForeignKey(Course, null = True)
113     assigned_on = models.DateTimeField(auto_now=True)
```

Il y a tout d'abord le modèle `BaseProfile` qui découle de `User` et qui, comme son nom l'indique, va servir de profil de base pour le modèle `Teacher` et `Student`.

L'utilisateur Django possède de base les caractéristiques suivantes [1] :

- `username` : nom d'utilisateur
- `first_name` : prénom
- `last_name` : nom
- `email`
- `password` : mot de passe
- `group` : les relations avec le modèle `Group` de Django
- `user_permissions` : les relations avec le modèle `Permission` de Django
- `is_staff` : si l'utilisateur peut accéder l'application admin
- `is_active` : définit si l'utilisateur doit être considéré comme actif ou non
- `is_superuser` : définit si l'utilisateur à tous les droits
- `last_login` : dernière connexion de l'utilisateur
- `date_joined` : date de création de l'utilisateur

Car un professeur a besoin de voir ses exercices, quiz et cours, et devra les assigner en tant que devoirs à un groupe, les modèles Exercise, Quiz et Course ont tous les trois été apportés.

Il y a ensuite le modèle `Group`, qui n'est pas le même que celui implémenté de base avec Django, car c'est celui qui a été utilisé pour les groupes d'un professeur. Les membres sont ajoutés par le biais du modèle `Groupmembers` qui sert de table intermédiaire entre `Student` ainsi que `Teacher` et `Group`. Le modèle `AssignHomework`, qui est aussi une table intermédiaire, sert à l'affectation de devoirs entre `Exercise`, `Quiz`, `Course` et `Group`.

2.2.2 Diagramme UML

2.3 Vues

```
1  from django.shortcuts import render, redirect
2  from django.contrib.auth import authenticate, login, logout
3  from dashboard.forms import NewGroupForm, NewStudentForm, NewTeacherForm, AddHomeworkForm, NewPasswo
4  from django.core.urlresolvers import reverse
5  from common.models import Group, Teacher, GroupMembers, Student, AssignHomework, Exercise, Quiz, Cou
6  from django.contrib.auth.models import User
7  from django.http import HttpResponse
8
9  #Accueil du dashboard
10 def home(request):
11     voyelle = 'aeiouyàäâéèëêîïíìôöõòûüùúAEIOUY' #Pour déterminer si le template affiche De ou D'
12     user = Teacher.objects.get(user = request.user)
13     firstLetter = request.user.username[0]#Idem
14     return render(request, 'dashboard/templates/dashboard/index.html', locals())
15
16 #Exercices, quiz et cours
```

1. «django.contrib.auth», consulté le 23.03.2015, https://docs.djangoproject.com/en/1.7/ref/contrib/auth/

```
17  def exercises(request):
18      user = Teacher.objects.get(user = request.user)
19      return render(request, 'dashboard/templates/dashboard/exercises.html', locals())
20
21  #Création de groupe
22  def newgroup(request):
23      success = False
24      user = Teacher.objects.get(user = request.user)
25      if request.method == "POST":
26          form = NewGroupForm(request.POST)
27          if form.is_valid():
28              group_name = form.cleaned_data["group_name"]
29
30              newGroup = Group.objects.create(name = group_name)
31              newGroup.save()
32              teacherToGroup = GroupMembers(teacher = user, group = newGroup) #Lie le Teacher et le gr
33              teacherToGroup.save()
34              success = True #Pour retourner le message de confirmation
35      else:
36          form = NewGroupForm()
37      return render(request, "dashboard/templates/dashboard/newclass.html", locals())
38
39  #Changement de mot de passe
40  def profil(request):
41      user = Teacher.objects.get(user = request.user)
42      success = ''
43      if request.method == "POST":
44          userProfile = request.user
45          form = NewPasswordForm(request.POST)
46          if form.is_valid():
47              password = form.cleaned_data["password"]
48              passwordConfirm = form.cleaned_data["passwordConfirm"]
49              if password != passwordConfirm:
50                  success = False #Message d'erreur
51              else:
52                  success = True #Message de confirmation
53                  u = request.user
54                  u.set_password(password)
55                  u.save()
56      else:
57          form = NewPasswordForm()
58      return render(request, 'dashboard/templates/dashboard/profile.html', locals())
59
60  def group(request, group_id):
61      user = Teacher.objects.get(user = request.user)
62      group = Group.objects.get(id = group_id)
63
64      studentList = group.student.all()
65      teacherList = group.teacher.all()
66      homeworkExList = group.homeworkExercise.all()#
67      homeworkQuList = group.homeworkQuiz.all()     # Pour avoir la liste des devoirs selon les genres
68      homeworkCoList = group.homeworkCourse.all()  #
69
70      deleteConfirmation = False #Pour supprimer une classe
71
72      if request.method == "POST":
73
74          #Ajouter un professeur au groupe
```

```
        if 'addTeacher' in request.POST:
            erreurTeacher = False
            formTeacher = NewTeacherForm(request.POST)
            if formTeacher.is_valid():
                newTeacher = formTeacher.cleaned_data["nickname"]
                try:
                    try:
                        teacherUser = User.objects.get(username = newTeacher)
                        teacher = Teacher.objects.get(user = teacherUser)
                        newTeacherToGroup = GroupMembers(teacher = teacher, group = group)
                        newTeacherToGroup.save()
                    except User.DoesNotExist:
                        erreurTeacher = True #Message d'erreur
                except Teacher.DoesNotExist:
                    erreurTeacher = True #Idem

        #Ajouter un élève au groupe
        elif 'addStudent' in request.POST:
            formStudent = NewStudentForm(request.POST)
            erreurStudent = False
            if formStudent.is_valid():
                try:
                    try:
                        newStudent = formStudent.cleaned_data["nickname"]
                        studentUser = User.objects.get(username = newStudent)
                        student = Student.objects.get(user = studentUser)
                        newStudentToGroup = GroupMembers(student = student, group = group)
                        newStudentToGroup.save()
                    except User.DoesNotExist:
                        erreurStudent = True #Message d'erreur
                except Student.DoesNotExist:
                    erreurStudent = True #Idem


        #Assigner un devoir
        elif 'assignHomework' in request.POST:
            formHomework = AddHomeworkForm(request.POST)
            erreur = False
            if formHomework.is_valid():
                homeworkid = formHomework.cleaned_data["homeworkid"]
                genre = formHomework.cleaned_data["genre"]

                #Cherche l'activité selon le genre choisi
                if genre == "exercise":
                    try:
                        exercise = Exercise.objects.get(id = homeworkid)
                        newHomework = AssignHomework(exercise = exercise, group = group)
                        newHomework.save()
                    except Exercise.DoesNotExist:
                        erreur = True #Message d'erreur

                if genre == "quiz":
                    try:
                        quiz = Quiz.objects.get(id = homeworkid)
                        newHomework = AssignHomework(quiz = quiz, group = group)
                        newHomework.save()
                    except Quiz.DoesNotExist:
                        erreur = True #Idem
```

```
133
134                 if genre == "course":
135                     try:
136                         cours = Course.objects.get(id = homeworkid)
137                         newHomework = AssignHomework(course = cours, group = group)
138                         newHomework.save()
139                     except Course.DoesNotExist:
140                         erreur = True #Idem
141         elif 'deleteClass' in request.POST:
142             deleteConfirmation = True #Fait apparaître le deuxième bouton de confirmation
143
144         #Supprime la classe
145         elif 'deleteClassConfirm' in request.POST:
146             group = Group.objects.get(id = group_id)
147             group.delete()
148             return redirect('home')
149
150
151         formStudent = NewStudentForm()
152         formTeacher = NewTeacherForm()
153         formHomework = AddHomeworkForm()
154
155     else:
156         formStudent = NewStudentForm()
157         formTeacher = NewTeacherForm()
158         formHomework = AddHomeworkForm()
159     return render(request, 'dashboard/templates/dashboard/classe.html', locals())
160
161 #Retirer d'un groupe
162 def deleteFromGroup(request, member_id, group_id):
163     if request.method == "POST":
164
165         #Selon élève ou professeur
166         if 'deleteStudent' in request.POST:
167
168             student = Student.objects.get(id = member_id)
169             group = Group.objects.get(id = group_id)
170             studentToGroup = GroupMembers.objects.get(student = student, group = group)
171             studentToGroup.delete()
172
173
174         elif 'deleteTeacher' in request.POST:
175             teacher = Teacher.objects.get(id = member_id)
176             group = Group.objects.get(id = group_id)
177             teacherToGroup = GroupMembers.objects.get(teacher = teacher, group = group)
178             teacherToGroup.delete()
179
180     return redirect('group_view', group_id = group_id)
181
182 #Supprimer une activité
183 def deleteActivity(request, activity_id):
184     if request.method == "POST":
185
186         #Selon exercice, quiz ou cours
187         if 'deleteExercise' in request.POST:
188             exercise = Exercise.objects.get(id = activity_id)
189             exercise.delete()
190         if 'deleteQuiz' in request.POST:
```

```
191             quiz = Quiz.objects.get(id = activity_id)
192             quiz.delete()
193         if 'deleteCourse' in request.POST:
194             course = Course.objects.get(id = activity_id)
195             course.delete()
196     return redirect('exercises')
197 
198 #Retirer un devoir
199 def deleteHomework(request, group_id, homework_id):
200     if request.method == "POST":
201 
202         #Selon exercice, quiz ou cours
203         if 'deleteHomeworkEx' in request.POST:
204             exercise = Exercise.objects.filter(id = homework_id)
205             exercise = exercise[0]
206             group = Group.objects.get(id = group_id)
207             assignedHomework = AssignHomework.objects.filter(group = group, exercise = exercise)
208             assignedHomework = assignedHomework[0]
209             assignedHomework.delete()
210         if 'deleteHomeworkQu' in request.POST:
211             quiz = Quiz.objects.filter(id = homework_id)
212             quiz = quiz[0]
213             group = Group.objects.get(id = group_id)
214             assignedHomework = AssignHomework.objects.filter(group = group, quiz = quiz)
215             assignedHomework = assignedHomework[0]
216             assignedHomework.delete()
217         if 'deleteHomeworkCo' in request.POST:
218             course = Course.objects.filter(id = homework_id)
219             course = course[0]
220             group = Group.objects.get(id = group_id)
221             assignedHomework = AssignHomework.objects.filter(group = group, course = course)
222             assignedHomework = assignedHomework[0]
223             assignedHomework.delete()
224     return redirect('group_view', group_id = group_id)
```

Toutes les vues vont devoir chercher le professeur correspondant à l'utilisateur actuellement connecté. Cela permettra à chaque fois d'aller chercher les données correspondantes.

La vue `home` sert uniquement à distinguer la première lettre du nom d'utilisateur pour qu'apparaisse dans le template «de» ou «d'».

La vue `exercises`, elle, ne cherche rien de plus. L'utilisateur nous permettra d'accéder aux exercices, quiz et cours qui lui sont associés mais tout ceci est directement recherché dans le template.

C'est grâce à la vue `newgroup` qu'un professeur peut créer un groupe. S'il veut créer un groupe, la vue se contentera de créer un groupe associé au nom et de créer un lien entre le professeur et le groupe grâce à la table intermédiaire `GroupMembers`. La variable `success` a pour utilité d'afficher un message de confirmation dans le template `newclass.html` une fois le groupe correctement créé.

La vue `profil` est celle utilisée pour le changement de mot de passe. Elle compare les deux mots de passe entrés. Si les deux mots de passe correspondent, le mot de passe est attribué à l'utilisateur et, grâce au template `profile.html` et à la variable `success`, un message est retourné pour confirmer le changement. Dans le cas contraire, un message d'erreur est retourné.

La vue `groupe`, elle, est composée de plusieurs actions qui dépendent de la forme qui a été remplie.

— Il y a tout d'abord `addTeacher` qui, quand l'utilisateur entre le nom d'utilisateur d'un autre professeur pour l'ajouter dans un groupe existant, va créer un object `GroupMembers` entre ce professeur et le groupe actuel pour qu'il fasse parti de ce groupe. Il se passe la même chose pour `addStudent` si le même utilisateur décide d'ajouter un élève.

— Pour assigner un devoir à un groupe, la vue va utiliser `assignHomework` qui, selon le genre d'activité et l'id qui ont été sélectionnés par l'utilisateur, va chercher l'activité et créer un objet `AssignHomework` qui va lier l'exercice, le quiz ou le cours au groupe.
— Pour supprimer un groupe, il y a d'abord l'utilisation de `deleteClass` qui va uniquement servir à l'apparition d'un deuxième bouton qui activera `deleteClassConfirm`, qui supprimera le groupe et donc tous les objets `AssignHomework` et `GroupMembers` dont il était le groupe.

Cette vue va par la suite retourner le template `classe.html` avec les variables définies au début qui apparaîtront sur la page.

Finalement, quelques vues ont été réalisées pour des actions plus complexes. Par exemple, `deleteFromGroup` avait besoin de deux variables, `member_id` et `group_id`. Cette vue a donc été liée à une url nécessitant ces deux variables. La vue `deleteFromGroup`, composée de `deleteStudent` et `deleteTeacher`, servent à retirer les membres d'un groupe en supprimant l'objet `GroupMembers` qui les liait. La vue `deleteActivity`, qui est elle composée de `deleteExercise`, `deleteQuiz` et `deleteCourse` sert à supprimer une activité depuis son dashboard. Enfin, `deleteHomework` permet au professeur de retirer un devoir précédemment assigné selon le type d'activité auquel il correspond.

2.4 Urls

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
from dashboard.views import *


urlpatterns = patterns('teachers.views',
    url(r'^home/$', home, name='home'),
    url(r'^classe/(?P<group_id>\d+)/$', group, name='group_view'),
    url(r'^exercices/$', exercises, name='exercises'),
    url(r'^nouveau_groupe/$', newgroup, name='newgroup'),
    url(r'^profil/$', profil, name = 'profil'),
    #Pour retirer d'un groupe
    url(r'^enlever_groupe/(?P<group_id>\d+)/(?P<member_id>\d+)/$', deleteFromGroup, name = "deleteFro
    #Pour supprimer une activité
    url(r'^enlever_activité/(?P<activity_id>\d+)/$', deleteActivity, name = "deleteActivity"),
    #Pour retirer un devoir
    url(r'^enlever_devoir/(?P<group_id>\d+)/(?P<homework_id>\d+)/$', deleteHomework, name = "deleteHo
)
```

Les urls `home`, `group_view`, `exercises`, `newgroup` et `profil` redirigent simplement aux vues du même nom.

Les urls `deleteFromGroup`, `deleteActivity` et `deleteHomework`, elles, sont reliées aux vues du même nom qui permettent certaines actions dépendantes de variables très précises. Pour réaliser ceci, j'ai créé des formes dans mes templates redirigeant à ces urls et possédant les variables nécessaires afin que mon programme puisse aller chercher les objets souhaités et permettre, par exemple, la suppression d'une activité.

2.5 Navigation

Il est important de noter que le menu déroulant ainsi que les pages Exercices, Nouveau groupe et la page d'une classe peuvent être atteintes depuis n'importe quelle page du dashboard.

Note de bas de page

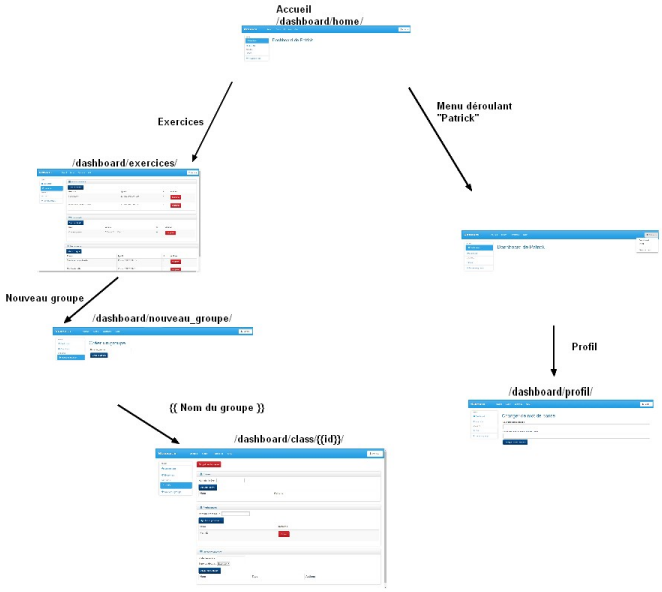

FIGURE 2.1 – Schéma de navigation du site

CHAPITRE 3

Développement dirigé par les tests

Le développement dirigé par les tests, grossièrement traduit de l'anglais Test Driven Development, est une technique de développement utilisée par beaucoup de programmeurs.

En effet, c'est grâce à cette technique que l'on peut le mieux s'assurer de la fonctionnalité du site et de la simplicité du code. Dans un travail de longue haleine, cette méthode devient nécessaire pour ne pas être redondant dans son code et pour le rendre le plus clair possible.

Dans cette première partie, nous allons nous intéresser au côté théorique de ce type de développement avant de se lancer dans un réel projet.

Le développement dirigé par les tests est composé de deux tests importants et bien différents.

3.1 Les tests fonctionnels

Le test fonctionnel, comme son nom l'indique, cherche à tester la fonctionnalité du site. Plus précisément, il se met du côté de l'utilisateur du site. Il va par exemple vérifier que les titres et les textes apparaissent, mais il va aussi regarder si les différents boutons ou champs de textes fonctionnent.

Voici un exemple de test fonctionnel :

Mettre un test fonctionnel plus tard

Nous pouvons déjà voir que les commentaires sont énormément présents dans ce test. En effet, la convention est que l'on crée un scénario pour expliquer exactement ce que l'on teste. Dans cet exemple, nous avons le professeur Jean-Paul qui entend parler d'un site pour apprendre les mathématiques où il y a un dashboard qui lui permet de gérer son travail. Il va donc essayer tous les boutons et toutes les fonctionnalités.

3.2 Les tests unitaires

Le test unitaire, lui, se base plus sur le point de vue du programmateur. Si on pouvait considérer le test fonctionnel comme un test externe, le test unitaire, lui, serait le test interne. Ce qu'il teste ne sera jamais vu, car il permet de vérifier que le code fonctionne comme prévu.

Voici un exemple :

Test unitaire A rajouter un jour

3.3 Le cycle du développement dirigé par les tests

Quand on veut programmer à l'aide du développement dirigé par les tests, on tente de suivre un certain cycle :

1. Tout d'abord, il est **impératif** d'écrire un test avant même d'écrire n'importe quelle ligne de code. En effet, l'idée du Test Driven Development est d'être sûr qu'un test échoue avant d'écrire notre code. Le test peut être fonctionnel ou unitaire, cela dépendant de la partie de votre application que l'on souhaite développer. Le test va évidemment retourner un message négatif, mais c'est normal étant donné que rien n'a été codé concernant la fonctionnalité testée.
2. Ensuite seulement, le but est d'écrire un minimum de code possible pour que le test précédemment lancé fonctionne. Il faut donc réussir à ce que le test, une fois relancé, retourne un résultat positif. Il faut faire attention à ne pas développer une fonctionnalité qui n'est pas dans le test
3. Une fois le code écrit, on peut relancer ce test. Si le résultat est positif, on peut passer à l'étape suivante. Dans le cas contraire, il faudra refaire les étapes 2 et 3 jusqu'à ce que le test fonctionne.
4. Finalement, il ne reste qu'à restructurer le code précédemment écrit pour qu'il soit plus lisible. Il faut faire très attention durant cette étape à ne rien ajouter ou enlever. Le code doit garder le même résultat.

Une fois que ces 4 étapes ont été effectuées, il ne reste qu'à recommencer avec une autre fonctionnalité de l'application, jusqu'à que celle-ci soit finie.

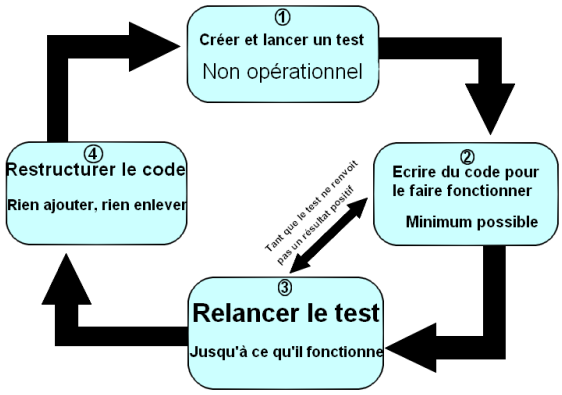

FIGURE 3.1 – Schéma résumant les 4 étapes du Développement dirigé par les tests

3.4 Gain de temps ?

En lisant ces 4 étapes répétitives, on ne peut que se demander si le Test Driven Development et son cycle compliqué est réellement un atout et un gain de temps pour le programmeur.

Il est clair que, sur un travail de petite taille, tout coder n'aurait pas énormément de sens, car tout peut être facilement essayable par soi-même. Dans le cas d'un travail d'une certaine consistance, ce n'est pas pareil. C'est uniquement en testant que l'on peut être sûr de son code, car cela signifie qu'il est valide et devrait le rester.

CHAPITRE 4

Débuter un projet

Pour comprendre grâce à la pratique, nous allons au cours de ce travail établir un dashboard permettant à un professeur de gérer des classes et des exercices sur un site d'e-learning pour les mathématiques grâce à Django, un framework fonctionnant avec Python.

4.1 Eléments requis

Pour pouvoir parfaitement suivre ce guide, il nous faudra les éléments suivants afin de réaliser les tests :

— **Django 1.7 :** en effet, il va vous falloir django, car c'est le framework que l'on va utiliser. Pour le télécharger, il suffit de taper la commande suivante avec pip :

```
sudo pip3 install django==1.7
```

Si vous utilisez Windows, vous pouvez enlever *sudo*.

— **Selenium :** Selenium est un outil permettant de gérer les navigateurs avec des commandes. Ceci nous sera utile pour les tests fonctionnels (ce terme sera expliqué plus tard). Encore une fois, il est possible de le télécharger grâce à pip :

```
sudo pip3 install --upgrade selenium
```

Encore une fois, le *sudo* n'est pas nécessaire sur Windows.
Note : il est important de toujours utiliser la dernière version de Selenium. En effet, une version dépassée peut facilement se comporter de façon non désirée. [1]

Une fois les éléments nécessaires installés, vous pouvez passer à la suite.

4.2 Premier test

Le principe de base du Test Driven Development est d'écrire un test avant même de coder ce que le test doit vérifier. Pour notre exemple, on devrait vérifier qu'il y ait le plus basique des éléments sur notre site : Django. On va donc créer un test fonctionnel pour voir s'il y a bien le titre de Django sur la page d'accueil. Le test fonctionnel permet de nous assurer que notre site fonctionne et possède la fonctionnalité la plus optimale qu'il soit.

Commençons donc par écrire ce code :

```
1 from selenium import webdriver
2
3 browser = webdriver.Firefox()
4 browser.get("http://localhost:8000")
```

1. PERCIVAL, Harry J.W., «Test Driven Development With Python», publié le 19 juin 2014

```
5
6 assert "Django" in browser.title
7
8 browser.quit()
```

Regardons une par une les lignes qui pourrait poser des problèmes de compréhensions :

1. Nous permet d'importer webdriver qui nous sera utile pour gérer les navigateurs web (dans ce cas, Firefox), nous permettant de les ouvrir, d'aller à une URL ou de les fermer.

6. Va basiquement nous dire si la page que l'on vient de charger (http://localhost:8000) contient le mot "Django" dans le titre.

Comme on peut s'y attendre, du moins si on se rappelle du but d'un test, le test ne marchera pas. En effet, comme dit précédemment, les tests sont faits pour évaluer quelque chose que l'on n'a pas encore fait, et pour nous aider à les faire le plus simplement possible.

Quand on test, il ne faut pas avoir les tests qui échouent comme quelque chose de mal. Dans certains cas (comme celui-ci), ces tests sont attendus et recevoir un *False* à la fin est donc un bon signe : notre test marche !

Note de bas de page