Conception d'une plateforme de e-learning

Application d'exercices interactifs sur les polynômes du premier et deuxième degré

Collège du sud, Travail de maturité

Florian Genilloud

30 Mars 2015

Table des matières

1 Introduction **1**
1.1 Approche personnelle par rapport à l'application dans le séminaire 1
1.2 Le but de l'application 1
1.3 La collaboration dans le projet 2
1.4 Intégration de l'application 2

2 Documentation de l'utilisateur **3**
2.1 Les fonctionnalités 3
2.2 La navigation sur le site 4
2.3 Fonctionnement des différentes pages 5

3 Documentation du développeur **11**
3.1 Démarrage du projet depuis Cloud9 12
3.2 Les modèles 12
3.3 Les vues 15
3.4 Les urls 19
3.5 Les templates 20

4 Conclusion **27**

5 Webographie **28**

CHAPITRE 1

Introduction

1.1 Approche personnelle par rapport à l'application dans le séminaire

Suite à la lecture des sujets de travail de maturité, je me suis senti particulièrement intéressé par le sujet du *Développement d'une plateforme Web d'e-learning*. En effet, depuis mon plus jeune âge, le monde de l'informatique et de la technologie font partie intégrante de ma vie. De plus, les mathématiques cultivent ma curiosité et me fascinent. De ce fait, ce travail de maturité était parfait pour moi.

Une des premières choses qu'un élève du secondaire II apprend lors de ses cours de mathématiques est l'algèbre de base. C'est pourquoi j'ai choisi la partie du site Web concernant l'*application d'exercices interactifs sur les polynômes du premier et deuxième degré*. L'objectif de base pour qu'une application de ce type puisse fonctionner est bien évidemment qu'un professeur crée un exercice afin qu'un élève le résolve en sachant si sa résolution est correcte ou non. C'est à l'aide de la création de corrigés d'exercices que l'élève pourra se perfectionner. Pour arriver à ce résultat, la méthode la plus simple est d'utiliser un framework possédant un système de stockage et d'une partie s'occupant des requêtes de l'utilisateur. Le framework Django écrit sous le language de programmation Python est idéal pour cette tâche

1.2 Le but de l'application

Voici la documentation de l'application `exercises` présente sur le site suivant [1] . Celle-ci permet d'utiliser la partie création ainsi que la partie résolution des exercices de manière complète et détaillée. Cette application servira par la suite au professeur à réaliser un exercice de factorisation ou de développemennt puis le mettre en ligne. Il suffira de donner le lien de l'exercice à l'élève pour qu'il puisse le résoudre.

Cette application consiste en premier lieu à avoir un support internet sur lequel un élève du Collège du Sud pourra s'entraîner en prévision de ses examens ou alors tout simplement pour perfectionner ses capacités en mathématiques dans le domaine de la factorisation et dans celui du calcul. Celle-ci permet au professeur d'adapter son exercice aux besoins de ses élèves. Cette application aide également l'élève à connaître ses lacunes et s'améliorer Django permet de stocker les données créées par les professeurs dans une base de données et de récupérer celles-ci pour en faire des pages Web. C'est exactement ce dont on a besoin pour cette application car le professeur crée un exercice et la partie backend très développée de Django s'occupe de créer la page web contenant les données entrées précédemment.

1. Le lien de la page d'accueil : https://webmath-thirteenfoil8.c9.io/exercises/

1.3 La collaboration dans le projet

Pour ce qui est de la collaboration avec les autres applications du projet, il faudrait au minimum que les fonctionnalités suivantes soit disponible :

- La collaboration avec le dashboard élève :
  - L'élève doit pouvoir ajouter les liens des exercices qu'il a trouvé compliqués.
  - Il doit avoir un Feedback des exercices.
  - Il doit pouvoir mettre les liens des exercices à faire pour les devoirs ou autres dans un dossier.
- La collaboration avec le dashboard professeur :
  - Le professeur doit pouvoir faire des dossiers avec les exercices qu'il a créés.
  - Il doit également pouvoir prendre des exercices d'autres professeurs pour les intégrer dans un dossier.
- La collaboration avec les cours :
  - Un cours doit contenir les liens des exercices qui sont en rapport avec celui-ci.
- La collaboration avec les quiz :
  - Il faudrait mettre en relation un exercice ou bien un groupe d'exercices avec un ou plusieurs quiz ayant le même but pédagogique.

1.4 Intégration de l'application

L'intégration de cette application au reste du projet ne devrait normalement pas poser de problèmes. La manière la plus simple de faire correspondre les exercices à des cours est d'utiliser les liens des exercices pour pouvoir y accéder.

CHAPITRE 2

Documentation de l'utilisateur

En premier lieu, une explication de comment fonctionne le site du côté des utilisateurs est nécessaire.

2.1 Les fonctionnalités

Il est a noté que ce site internet possède un système d'authentification. Il est donc indispensable de se connecter avant de pouvoir utiliser les fonctionnalités du site. Pour ce connecter, il suffit de cliquer sur *Rechercher un exercice* ou *Création d'exercice* pour être rediriger vers la page d'inscription ou de connection. Il existe deux types de compte pour cette application, un compte professeur ou un compte élève. Cela veut dire que les fonctionnalités qu'un compte professeur possède ne sont pas les mêmes qu'un compte élève. En effet, un professeur peut accéder à toutes les pages du site, cependant, les droits de élève sont moindres.

2.1.1 Les fonctionnalités du professeur

Les fonctionnalités qu'un professeur possède sont les suivantes :

1. **Créer un exercice.** L'apport principal d'un professeur dans cette application est la création des exercices. En effet, un professeur peut créer des exercices soit en cliquant sur l'onglet *Création d'exercice*, soit en entrant l'url http://webmath-thirteenfoil8.c9.io/exercises/create/ dans la barre de recherche du navigateur utilisé. En parallèle à l'exercice créé, il doit créer son corrigé.
2. **Contrôler la résolution des élèves :** Pour que l'exercice puisse avoir un suivi de son créateur, un page s'occupant d'afficher toutes les résolutions faites par les élèves est disponible. Sur cette page , le professeur voit l'équation de l'exercice qu'il a fait mais également les résolutions accompagnées du nom de l'élève et de la date de résolution.

2.1.2 Les fonctionnalités de l'élève

Les fonctionnalités qu'un élève peut utiliser sont les suivantes :

1. **Résoudre un exercice :** Grâce au système de recherche présent sur la page http://webmath-thirteenfoil8.c9.io/exercises/find/ , il est simple pour un élève de trouver un exercice à résoudre.
2. **Voir le corrigé d'un exercice :** Suite à la résolution d'un exercice, l'élève est redirigé sur le corrigé de celui-ci. Tout le développement de la résolution y figure. Cela permet à l'élève de bien comprendre ses fautes en prévisions d'une amélioration de ses capacités mathématiques.

2.2 La navigation sur le site

2.2.1 Les urls

La manière la plus simple de se rendre directement sur une page est de rentrer l'url exact dans le navigateur.

Les urls de cette application fonctionne comme ci-dessous :

- Page d'accueil : http://webmath-thirteenfoil8.c9.io/exercises/
- Page de création d'un exercice : http://webmath-thirteenfoil8.c9.io/exercises/create/
- Page de recherche des exercices : http://webmath-thirteenfoil8.c9.io/exercises/find/
- Page de résolution d'un exercice : http://webmath-thirteenfoil8.c9.io/exercises/resolve/<numéro de l'exercice>/
- Page du corrigé d'un exercice : http://webmath-thirteenfoil8.c9.io/exercises/correction/<numéro de l'exercice>/
- Page des résolutions d'un exercice : http://webmath-thirteenfoil8.c9.io/exercises/done/<numéro de l'exercice>/

2.2.2 Le schéma de navigation des pages

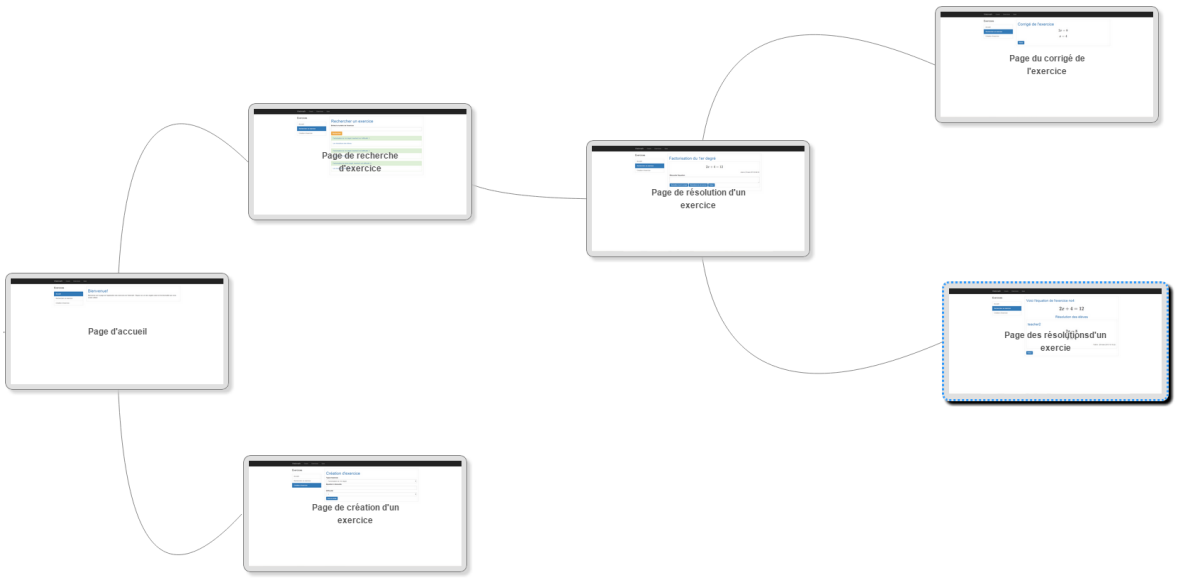

FIGURE 2.1 – *Schéma de navigation*

2.3 Fonctionnement des différentes pages

2.3.1 La page d'accueil

La page d'accueil sert simplement à l'utilisateur d'accéder au site sans avoir à se connecter.

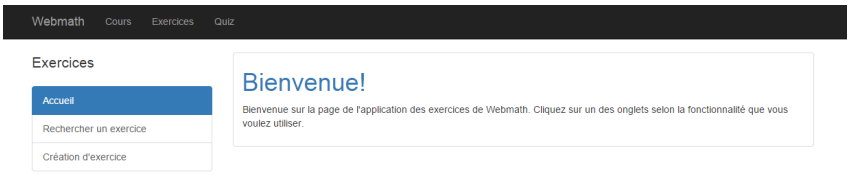

FIGURE 2.2 – *La page d'accueil*

Sur cette page, il y a différents onglets cliquables :

- ***Webmath*** Cet onglet renvoie sur la page d'accueil.
- ***Cours*** Celui-ci ne renvoie vers aucune page pour le moment (en développement).
- ***Exercices*** Celui-ci renvoie vers la page d'accueil de l'application exercice.
- ***Quiz*** Cet onglet redirige l'utilisateur vers la page d'accueil de l'application Quiz.
- ***Accueil*** Celui-ci renvoie vers la page d'accueil de l'application exercice.
- ***Rechercher un exercice*** Cet onglet redirige l'utilisateur vers la page de recherche d'exercice.
- ***Création d'exercice*** Et finalement, l'onglet *Création d'exercice* renvoie vers la page de création d'exercice.

2.3.2 La page de création d'exercice

La page de création d'exercice n'est accessible que par les professeurs. Elle se présente de la manière suivante :

FIGURE 2.3 – *La page de création*

On remarque que sur cette page, on demande à l'utilisateur d'entrer des données :

- ***Type d'exercice*** On fait le choix entre les quatres possibilités de la liste déroulante.
- ***Equation à résoudre*** L'utilisateur entre l'équation qui sera résolue par un élève.
- ***Difficulté*** De nouveau, on choisit entre les possibilités de la liste déroulante. Cela donne une indication de la difficulté de l'exercice à l'élève. Cette difficulté est croissante de 1 jusqu'à 5.

De plus, le bouton *Faire le corrigé* permet d'afficher la suite du formulaire que l'utilisateur doit remplir.

Voici un exemple de comment remplir un formulaire. Pour ce qui est du corrigé, il faut entrer chaque étape de l'équation ligne par ligne. Une fois le corrigé fini, le bouton *Soumettre l'exercice* enregistre les données du formulaire et redirige le professeur vers la page d'accueil.

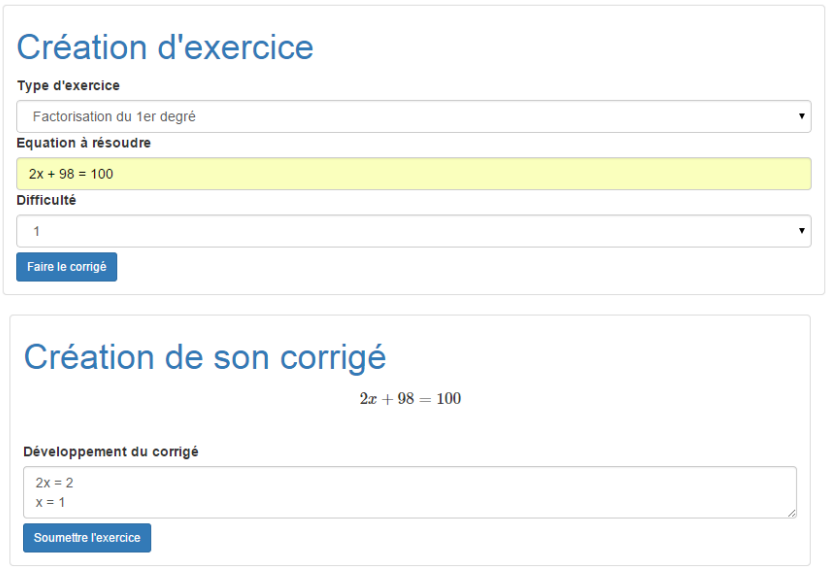

FIGURE 2.4 – *Remplir le formulaire de création*

Si il manque une donnée, le message suivant s'affiche.

FIGURE 2.5 – *Message d'erreur*

2.3.3 La page de recherche

Sur la page de recherche, le bouton *Rechercher* permet de trouver l'exercice correspondant au numéro entré dans la barre de recherche juste au-dessus. Cependant, on peut également faire une recherche manuelle en faisant défiler la page. Chaque fois qu'un professeur crée un nouvel exercice, il s'ajoute à la liste. Le lien le plus foncé permet de se rendre à la page de résolution de l'exercice. Le lien le plus clair, quant à lui, redirige l'utilisateur, qui doit être un professeur, vers la page contenant l'ensemble des résolutions pour un exercice.

FIGURE 2.6 – *La page de recherche*

Pour ce qui est de la recherche, si l'exercice existe, un message s'affiche avec le lien de l'exercice.

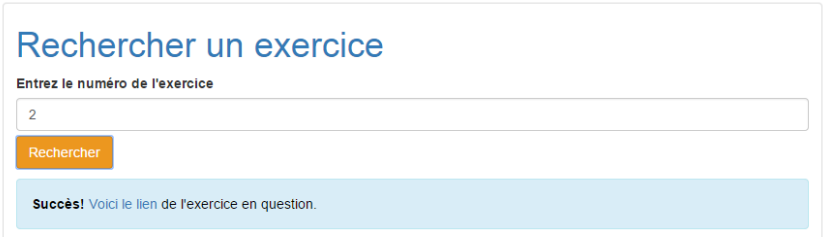

FIGURE 2.7 – *Message positif de recherche*

Dans le cas où l'exercice n'existe pas, un message d'erreur apparaît.

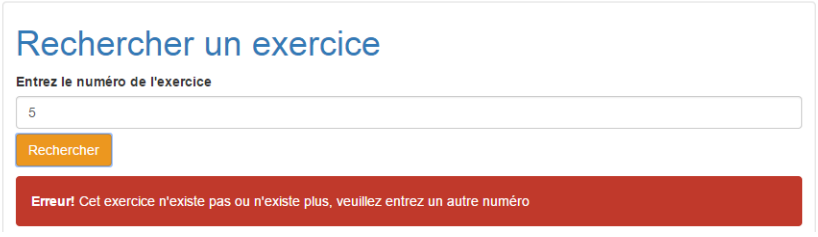

FIGURE 2.8 – *Message négatif de recherche*

2.3.4 La page de résolution d'exercice

La page de résolution d'exercice est assez similaire à la page de création. En effet, il y a également la présence d'un formulaire que l'élève doit remplir. Le seul champ à remplir est la résolution de l'exercice en prenant en compte d'écrire les étapes ligne par ligne. Sur cette page, trois boutons sont disponibles :

- ***Soumettre et voir le corrigé*** Ce bouton enregistre la résolution et renvoie l'élève vers le corrigé.
- ***Résolutions de cet exercice*** Celui-ci renvoie l'utilisateur qui doit être un professeur vers la page contenant les résolutions des élèves.
- ***Retour*** Et finalement, le bouton *Retour* redirige l'utilisateur à la page de recherche.

FIGURE 2.9 – *La page de résolution*

Voici un exemple de résolution :

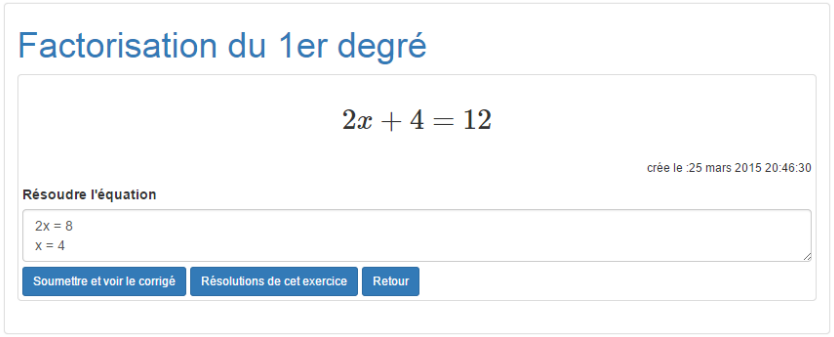

FIGURE 2.10 – *Exemple de résolution*

Si il manque une donnée, le message suivant s'affiche.

FIGURE 2.11 – *Message d'erreur*

2.3.5 La page des résolutions

Cette page n'est accessible que par les professeurs. Elle sert uniquement à afficher les résolutions faites par les élèves. Les résolutions contiennent le nom de l'élève, sa résolution et la date de résolution. Un bouton *Retour* permet de retourner à la page de recherche.

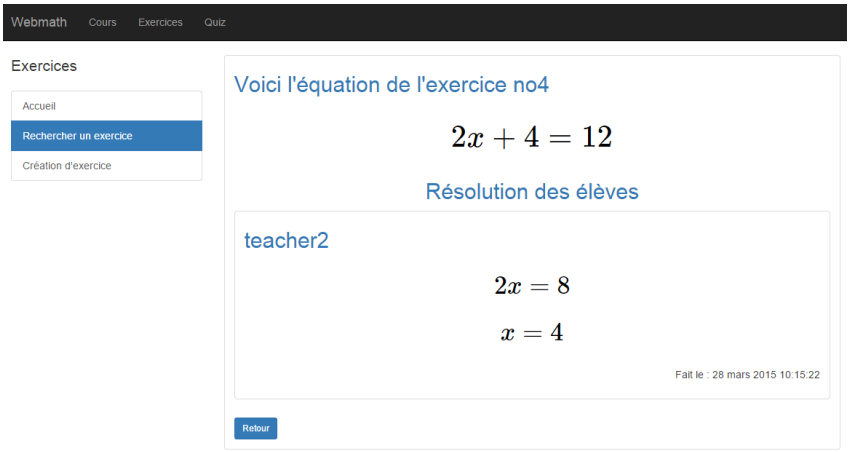

FIGURE 2.12 – *La page des résolutions d'un exercice*

De plus, si un exercice possède encore aucune résolution, le message suivant s'affiche.

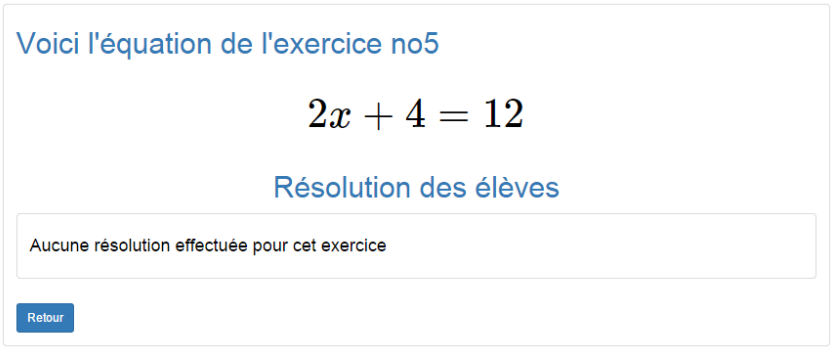

FIGURE 2.13 – *Aucune résolution*

2.3.6 La page du corrigé

Ceci est la dernière page du site. Elle permet à un élève de voir le corrigé de l'exercice qu'il vient de résoudre. Le bouton *Retour* redirige l'utilisateur vers la page de recherche.

FIGURE 2.14 – *La page du corrigé*

CHAPITRE 3

Documentation du développeur

Cette partie de la documentation est essentiellement destinée au développeur pour mieux comprendre le fonctionnement de l'application. Tout ce qui concerne les modèles, les vues, les urls, les templates, ... est affiché ci-dessous. Le code est accompagné de quelques annotations mais celles-ci sont là que pour donner quelques précisions quant à celui-ci. Il est donc nécessaire de connaître les languages de programmation et les frameworks suivant pour comprendre la documentation du développeur :

- Les languages de programmation :
  - Python [1]
  - Html [2]
  - Css [3]
  - Javascript [4]
- Les FrameWorks :
  - Bootstrap [5]
  - jQuery [6]
  - Django [7]

1. Le lien de la documentation de Python : https://docs.python.org/3/
2. Le lien de la documentation d'Html : http://overapi.com/html/
3. Le lien de la documentation de CSS : http://overapi.com/css/
4. Le lien de la documentation de Javascript : http://overapi.com/javascript/
5. Le lien de la documentation de Bootstrap : http://getbootstrap.com/getting-started/
6. Le lien de la documentation de jQuery : http://overapi.com/jquery/
7. Le lien de la documentation de Django : https://docs.djangoproject.com/en/1.7/

3.1 Démarrage du projet depuis Cloud9

L'utilisation de l'environnement Web Cloud9 [8] est très utile. Il permet d'éviter la surcharge de la machine sur laquelle on travaille et permet au code d'être accessible de tout ordinateur possédant une connection internet. Une fois sur Cloud9, il faut créer un *Workspace custom* en cliquant sur l'onglet *create new workspace*. Voici la série de commandes à entrer pour pouvoir démarrer le projet :

```
#installer django
sudo pip3 install django

#installer pillow qui gère les images de l'application common
sudo pip3 install pillow

#cloner le dépôt git
git clone https://github.com/thirteenfoil8/TM-Code-Doc

#Lancer le serveur
python3 manange.py runserver $IP:$PORT
```

Il est a noté que le projet contenant l'entier des fichiers est sur un dépôt [9] GitHub. Une fois ces commandes entrées dans le `bash`, l'entier du projet sera présent dans le *workspace*.

3.2 Les modèles

1. **Les modèles de cette application sont les suivants :**
   - `Exercise`

     Ce modèle contient les informations relatives à un exercice en particulier. Il contient le nom du créateur : `owner`, la date de création : `created_on`, le titre de l'exercice : `title` (celui-ci ne possède que 4 choix présents dans le template `create.html` présenté plus bas dans la documentation), l'équation que l'élève devra traîter : `equation`, la difficulté de l'exercice : `grade` (choisi entre 1 et 5 également dans `create.html`), et enfin la correction de l'exercice : `correction`. La fonction `def __str__(self)` sert uniquement à rendre quelque chose de plus propre sur la page [10] prévue pour les admins du site.
   - `Exercise_done`

     Ce modèle contient les informations concernant la résolution d'un exercice traité par un élève. Il contient le nom de l'élève : `student`, la date à laquelle l'élève a fait l'exercice : `do_on`, l'exercice auquel la résolution fait référence : `exercise_done` et la résolution de l'élève : `resolution`. La fonction `def __str__(self)` a le même but que pour la table `Exercise`. Pour ce qui est de la fonction `def get_lines(self)` : nous permet de retourner une liste avec chaque ligne de la résolution de l'élève. Cette fonction sera utile dans le template `done.html` par la suite.

8. Le line vers Cloud9 : https://c9.io/
9. Le lien de la documentation GitHub : https://github.com/thirteenfoil8/TM-Code-Doc
10. Le lien vers la page admin : http://webmath-thirteenfoil8.c9.io/admin/

2. La relation :

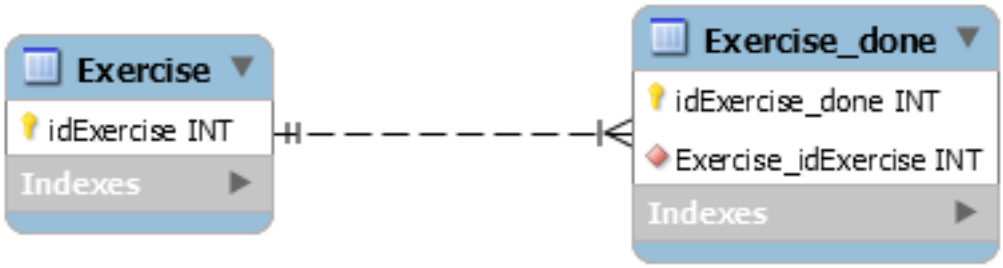

FIGURE 3.1 – *Diagramme UML du modèle relationnel*

Ces deux modèles sont reliés entre eux grâce à une `ForeignKey` qui est présente dans la table `Exercise_done`. Cela signifie qu'un exercice peut posséder plusieurs résolutions, mais qu'une résolution ne répond qu'à un exercice.

3. Le code :

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


class Exercise(models.Model):

    owner = models.CharField(max_length=20)  # créateur
    created_on = models.DateTimeField(auto_now_add=True) # date de création
    title = models.CharField(max_length=30) # type d'exerciCe ( choisi
                                            # dans create.html )
    equation = models.CharField(max_length=50) # Equation de l'exercice
    grade = models.CharField(max_length=60) # difficulté ( entre 1 et 5 )
    correction = models.CharField(max_length = 200) # corrigé de l'exercice
    def __str__(self):
        # recherche plus facile dans http://webmath-thirteenfoil8.c9.io/admin/
        return self.title + " " + self.owner + " " + str(self.pk)


class Exercise_done(models.Model): # Résolutions d'un exercice ( n...1 )
    student = models.CharField(max_length=20) # Etudiant résolvant l'équation
    do_on = models.DateTimeField(auto_now_add=True) # date de résolution
    exercise_done = models.ForeignKey(Exercise) # l'exercice auquel les
                                                # résolutions seront liées
    resolution = models.CharField(max_length = 200) # la résolution

    def __str__(self):
        # recherche plus facile dans http://webmath-thirteenfoil8.c9.io/admin/
        return self.exercise_done.title + " " + self.exercise_done.owner/
        + str(self.exercise_done.pk) + " fait par: " + self.student

    # retourne une liste avec chaque ligne de la résolution.
    def get_lines(self):
        return self.resolution.split("\n")
```

4. Utilisation :

   Pour ce qui est de l'utilisation, lorsqu'on enregistre un formulaire dans la base de données, le code est d'abord traité grâce au méthode offert par Django puis, il est traduit en SQL.

   En premier lieu, il faut récupérer tous les objets déja existant grâce au code suivant :

   ```
   Exercise.objects.all()
   ```

   Ensuite, pour ce qui est de la création d'exercice, la méthode `.save()` de Django sert à enregister un objet et le traduire en SQL.

```
# ici, on utilise un formulaire.
if request.method == 'POST':
    title = request.POST['type']
    equation = request.POST['equation']
    grade = request.POST['grade']
    correction = request.POST['correction']
    owner = request.user.username
    Exercise(title=title, owner=owner, equation=equation, grade=grade, \
    correction=correction).save() # On crée l'exercice
```

3.3 Les vues

Le concept des « vues » est la base de la logique responsable du traitement des requêtes des utilisateurs et le renvoi des réponses vers un template. Toutes les vues en lien avec cette application se trouve dans `MainProject/webmath/exercises/views.py`. Par la suite, deux points seront assez récurrents :

1. **L'appel `@login_required` :** Cet appel là permet de demander à l'utilisateur d'être connecté pour pouvoir aller sur la page en question.
2. **L'appel `@user_passes_test(is_teacher)` :** Cet appel est plus strict et sert à préciser que seul un professeur peut se diriger vers la page.

Ces deux appels viennent des applications common et permission qui gèrent les authentifications et les permissions d'un utilisateur.

Les différents `import` à faire dans la vue du template de base `index.html` sont les suivants :

```
1  from django.shortcuts import render, HttpResponseRedirect, get_object_or_404,\
2  HttpResponse
3  from django.core.urlresolvers import reverse
4  from exercises.models import *
5  import json
6  from common.models import Teacher, Student
7  from common.auth_utils import *
8  from django.contrib.auth.decorators import login_required, user_passes_test
9  # Create your views here.
10 def index(request):
11     return render(request, 'exercises/index.html')
12
13 # @login_required demande à l'utilisateur d'être connecté
14 # @user_passes_test(is_teacher) restreint l'accès seulement au teachers
```

3.3.1 La vue create

Pour ce qui est de la vue fonctionnant derrière `create.html`, la difficulté se trouve surtout dans la sauvegarde des données.

En effet, il faut que chaque donnée entrée dans les balises du template `create.html` puisse être assignée et enregistrée plus tard dans la base de données. Les données seront appliquées à la table `Exercices`. Ces données seront récupérées plus tard dans l'ensemble des vues de l'application.

Le code permettant de faire ça se trouve dans la vue `create`.

```
1  @login_required
2  @user_passes_test(is_teacher)
3  def create(request):
4      # enregistre les données du formulaire dans la base de données si requête
5      # POST sinon, retourne la page
6      if request.method == 'POST':
7          title = request.POST['type']
8          equation = request.POST['equation']
9          grade = request.POST['grade']
10         correction = request.POST['correction']
11         owner = request.user.username # prendre l'username du user dans
12         #la table User de Django
13         Exercise(title=title, owner=owner, equation=equation, grade=grade, \
14         correction=correction).save()
15
16         return HttpResponseRedirect(reverse("exercises:index"))
17     else:
18         return render(request, 'exercises/create.html')
```

A la ligne 4, la condition `if` permet de différencier si un enregistrement des données est nécessaire et dans le cas contraire, le template `create.html` sera affiché à l'utilisateur. Dans le cas où un enregistrement des données est demandé par l'utilisateur, celles-ci sont assignées à différentes variables (`title`, `equation`, `grade`, `correction`, `owner`) puis instanciées au modèle `Exercise` auquel on applique la fonction `.save()` qui enregistre les données dans la base de données SQL proposée par Django.

3.3.2 La vue find

La vue `find` utilise la fonction `objects.all()` qui assigne à `latest_exercise_list` une liste comportant tous les exercices présent dans la table `Exercise`. La fonction `return` retourne ici le template `find.html` mais également un dictionnaire possédant la variable `latest_exercise_list`.

```
1  @login_required
2  def find(request):
3      # Assigne les Querysets des objets exercise
4      latest_exercise_list = Exercise.objects.all()
5      return render(request, 'exercises/find.html', {"exercises_list" : \
6      latest_exercise_list})
```

3.3.3 La vue resolve

La vue `resolve` permet d'afficher un exercice dans son template `resolve.html`. La fonction `get_object_or_404()` assigne à la variable `exercise` toutes les données de l'objet `n_exercise` présent dans la table `Exercise`. Si celui-là est inexistant, la vue renvoie une erreur *404*. La fonction `.save()` est également présente dans ce template et instance la résolution d'un élève en rapport avec l'exercice `n_exercise` de la table `Exercise_done`.

Le return de la condition `if` renvoie l'utilisateur sur la page du corrigé de l'exercice `n_exercise`.

```
@login_required
def resolve(request, n_exercise):
    exercise = get_object_or_404(Exercise, id=n_exercise) # Assigne les Querysets
    # des objets exercise, 404 si inexistant

    # enregistre les données du formulaire dans la base de données si requête
    # POST sinon, retourne la page
    if request.method == 'POST' :
        student = request.user.username
        resolution = request.POST['response']
        Exercise_done(exercise_done=exercise, resolution=resolution, \
        student=student).save() # sauvegarde des données dans la db

        return HttpResponseRedirect(reverse("exercises:correction", \
        args=[n_exercise]))
    else:
        return render(request, 'exercises/resolve.html', \
        {"exercise" : exercise, "id" : n_exercise})
```

3.3.4 La vue correction

L'utilisateur accède au template relatif à cette vue suite à l'envoi de son formulaire dans la vue `resolve`.

Dans cette vue, on récupère le corrigé de l'exercice `n_exercise` de la table `Exercise` puis on affecte cette valeur à la variable `correction`. L'utilisateur entre les étapes de la résolution de l'exercice ligne par ligne. Donc, on utilise la fonction `split("\n") pour créer une liste contenant chaque ligne de la résolution. Cette liste est retournée dans le template grâce à la fonction ``locals()`.

```
def correction(request, n_exercise):
    correction = get_object_or_404(Exercise, id=n_exercise)
    correction_line = correction.correction.split("\n")
    return render(request,'exercises/correction.html', locals())
```

3.3.5 La vue done

Cette vue permet à un professeur de voir toutes les résolutions des élèves présentes pour l'exercice `n_exercise`. La fonction `objects.filter()` affecte à la variable `exercises_done` les valeurs de l'objet `n_exercise` se trouvant dans la table `Exercise_done`. Cette dernière est en lien avec l'exercice grâce à une `ForeignKey`. Du coup, `exercises_done` peut contenir plusieurs objets.

```
@login_required
@user_passes_test(is_teacher)
def done(request, n_exercise):
    exercise = get_object_or_404(Exercise, id=n_exercise)
    exercises_done = Exercise_done.objects.filter(exercise_done=exercise)
    return render(request, 'exercises/done.html', locals())
```

3.3.6 La vue search

Ceci est la dernière vue de l'application. Son rôle est totalement différent de toutes les autres vues. En effet, cette vue ne retourne aucun template visible par l'utilisateur mais elle sert à `#search_input` présent dans le template `find.html` de retouner le lien de l'exercice `exercise.pk`. Une méthode Ajax est nécessaire pour éviter de recharger la page à chaque nouvelle recherche.

```
def search(request):
    search_input = request.GET["search"]

    exercise = Exercise.objects.get(pk=search_input)

    pk = exercise.pk
    url = reverse("exercises:resolve", args=[exercise.pk])

    json_dict = {
        "pk" : pk,
        "url" : url,
    }

    json_string = json.dumps(json_dict)

    return HttpResponse(json_string)
```

3.4 Les urls

3.4.1 Les urls de la racine du projet

Les urls du code suivant servent tout simplement à indiquer les urls de base de l'application. Cela veut dire que suite à l'url http://webmath-thirteenfoil8.c9.io/ [11] , un simple rajout de : `admin`, `exercises`, `common` ou `permission`, amenera l'utilisateur directement à la base d'une des applications du projet. À cela, il faut signaler la présence de la fonction `include()` permet à chaque url présent dans les applications de pouvoir s'ajouter à l'url de base. Les urls de l'application `exercises` sont expliqués dans la rubrique suivante.

```
1  from django.conf.urls import patterns, include, url
2  from django.contrib import admin
3
4  urlpatterns = patterns('',
5
6      url(r'^admin/', include(admin.site.urls)),
7      url(r'^exercises/', include('exercises.urls', namespace='exercises')),
8      url(r'^common/', include('common.urls', namespace="common")),
9      url(r'^permission/', include('permission.urls', namespace="permission")),
10
11 )
```

3.4.2 Les urls de l'application exercises

Tout d'abord, on importe les vues qui seront utilisées dans l'application. Pour cela, on indique dans quel répertoire les vues se trouvent(cf. ligne3). Par convention, on nomme les urls d'un application du même nom que son template et de sa vue.

```
1  from django.conf.urls import patterns, include, url
2  from django.contrib import admin
3  from exercises.views import index, create, find, resolve, correction, search, done
4
5  urlpatterns = patterns('',
6      url(r'^$', index, name="index"),
7      url(r'^create/$', create, name="create"),
8      url(r'^find/$', find, name="find"),
9      url(r'^done/(\d+)/$', done, name="done"),
10     url(r'^resolve/(\d+)/$', resolve, name="resolve"),
11     url(r'^correction/(\d+)/$', correction, name='correction'),
12     url(r'^search/', search, name="search"),
13 )
```

11. Le lien vers la page de base du projet : http://webmath-thirteenfoil8.c9.io/

Dès qu'il y a la présence de `(\d+)/`, cela appelera la vue sur laquelle l'url dirige en utilisant le nombre entré à la suite de `/exercices/X` (ou X est un des urls situés ci-dessus) comme valeur de l'argument `n_exercise`. Par exemple, `/exercices/done/1` retournera la page des résolutions de l'exercice numéro 1, si l'exercice n'existe pas, la fonction `get_object_or_404` retournera une erreur 404.

1. L'`url(r'^$', index, name="index")` renvoie la page d'accueil du site.
2. L'`url(r'^create/$', create, name="create"),` renvoie la page de création d'exercices, accessible que par les professeurs.
3. L'`url(r'^find/$', find, name="find"),` renvoie la page de recherche des exercices.
4. L'`url(r'^done/(\d+)/$', done, name="done"),` renvoie la page comportant les résolutions des élèves par rapport à un exercice.
5. L'`url(r'^resolve/(\d+)/$', resolve, name="resolve"),` renvoie la page de résolutions d'un exercice.
6. L'`url(r'^correction/(\d+)/$', correction, name='correction'),` renvoie la page de correction d'un exercice.
7. L'`url(r'^search/', search, name="search"),` ne renvoie aucune page visible par l'utilisateur mais sert à afficher les données qui seront récupérées par la requête Ajax pour la recherche d'un exercice.

3.5 Les templates

Dans les templates de cette application, on utilise les données présentes dans la base de deux manières différentes :

1. Soit sous forme de boucle `for` :

```
{% for line in correction_line %}
    <p>$$ {{ line }} $$</p>
{% endfor %}
```

2. Soit sous forme d'appel du champ nécessaire de l'objet. Par exemple :

```
{{ exercise.equation }}
{{ exercise.id }}
```

De plus, au début de chaque template, on doit intégrer la ligne de code `{% extends "exercises/index.html" %}` pour permettre au template traîté d'avoir les mêmes attributs que le template de base `index.html`

3.5.1 Le template de base du site

Pour ce qui est des trois onglets présents sur toutes les pages, il faut mettre des liens vers les différents templates. Pour cela, on utilise une formule Django simple qui permet, si il y a un changement d'url par la suite dans le fichier `urls.py`, de faire automatiquement le changement pour éviter les erreurs de redirection.

Le *Frontend* est mis en place en utilisant un thème Bootstrap. Pour cette application, Le thème shop-item [12] est parfait car il est simple, ergonomique et ne demande que très peu de modifications.

```
<div class="list-group">
    <a href="{% url 'exercises:index' %}" class="list-group-item
    {% block active-home %}active{% endblock %}">Accueil</a>

    <a href="{% url 'exercises:find' %}" class="list-group-item
    {% block active-reso %}{% endblock %}">Résoudre un exercice</a>

    <a href="{% url 'exercises:create' %}" class="list-group-item
    {% block active-create %}{% endblock %}">Création d'exercice</a>

</div>
```

Les urls de redirection vers les différentes pages du site sont gérés ci-dessus. On utilise `<a href="{% url 'exercises:<nom_du_template>' %}"` pour renvoyer l'utilisateur vers les `templates`. Le bloque {% block active-<home, reso ou create> %}{% endblock %} permet d'activer une classe sur l'onglet actuel.

12. Le lien du thème : http://startbootstrap.com/template-overviews/shop-item/

3.5.2 Le template `create.html`

Le template `create.html` est utilisé par les professeurs pour créer un exercice ainsi que son corrigé. Pour pouvoir enregistrer les données, la présence de la balise `<form>` est obligatoire. Toutes les données entrées sont traités dans la vue du template.

Le `<button id="voir">` utilise un script se trouvant sous `exercises/js/create.js`. Ce script affiche la deuxième partie du formulaire et, grâce à la méthode `MathJax.Hub.Queue(["Typeset", MathJax.Hub])`, formate l'équation entrée précédement en la mettant sous une forme mathématique.

Le voici :

```
1  $(document).ready(function() {
2    $( ".corrigé" ).hide(); // cache la div du corrigé qui sera affiché plus tard
3    $("#voir").click(function() {
4        var $formule = $(".equation").val(); // Récupère la valeur de l'équation
5        $(".formule").text("$$" + $formule + "$$"); // La formate en Latex grâce
6        //à MathJax
7        $(".corrigé").show();
8        MathJax.Hub.Queue(["Typeset", MathJax.Hub]); // permet d'afficher l'équation
9        //en Latex sans avoir à recharger la page
10   });
11   $("#submit-resolve").click(function() {
12       if ($("#correction").val() && $("#equation").val()) {
13               $("#create-form").submit(); // renvoie le formulaire si les
14               // tous les champs sont remplis
15           }
16       else {
17           $("#form-warning").modal("show"); // Affiche un message d'erreur si
18           // tous les champs ne sont pas rempli
19
20       }
21   });
22 });
```

La documentation de Mathjax se trouve ici [13] .

Le deuxième `button` présent dans le template, utilise le code javascript présent depuis la ligne 11. La condition `if ($("#correction").val() && $("#equation").val())` contrôle que tous les champs du formulaire ont été remplis, sinon, le `else` affiche un message d'erreur.

13. Le lien de la documentation MathJax : https://www.mathjax.org/#docs

3.5.3 Le template `find.html`

Ce template comporte tous les exercices déjà présent dans la base de données.

La fonctionnalité permettant la recherche d'un exercice nécessite le code `html` suivant :

```
<div>
    <label for="search">Entrez le numéro de l'exercice</label>
    <input type="text" id="search_input" name="search" class="form-control">
    <button type="button" id="search" name="search" class="btn btn-warning">Rechercher
    </button>
</div>
<div class="alert alert-info" id="true">
    <strong>Succès!</strong> <span id="lien"></span> de l'exercice en question.
</div>
<div class="alert alert-info" id="false">
    <strong>Erreur!</strong> Cet exercice n'existe pas ou n'existe plus,
     veuillez entrez un autre numéro
</div>
<div>
```

Grâce au script de cette page se trouvant dans `static/exercises/js/find.js`, la vue `search` analysée auparavant prend tout son sens car ce script utilise les données trouvées par ajax pour les formater et les mettre en page suite à l'activation du bouton `<button type="button" id="search" name="search" class="btn btn-warning">Rechercher</button>`.

Le code est le suivant :

```
$("#search").click(function() {
    $("#lien").empty(); // Supprime l'éventuelle ancienne valeur
    var $search = $("#search_input").val(); // enregistre la valeur de
    //la recherche
    $('#false').hide();
    $('#true').hide();

    $.ajax({
        url: "/exercises/search/",
        type: "GET",
        dataType: "json",
        data : {
            search : $search, //récupère les données de la recherche par
            //rapport à l'exercice recherché ( $search )
        },
        success : function(response) { // Ajoute le lien de l'exercice si
        //il existe et l'affiche à l'utilisateur dans la div #true
            var $url= response["url"];
            $('#true').show();
            $("<a>", {
            "href": $url,
            }).text("Voici le lien").appendTo("#lien");
        },
        error : function() { // Affiche le message d'erreur si l'exercice
        //n'existe pas
            $("#false").show();
        }
    });
});
```

Les commentaires parlent d'eux même. Si l'id de l'exercice existe, on affiche la div :`<div id="true">` contenant le lien de l'exercice en question sinon, on affiche la div : `<div id="false">` indiquant que l'exercice n'existe pas.

Les `panel` de Bootstrap sont très clairs et permette de bien différencier la page de résolution de l'exercice et la page contenant les résolutions des élèves. Cette dernière est accessible que par les professeurs.

```
<div class="panel panel-success">
    <div class="panel-heading">
        <a href="{% url 'exercises:resolve' exercise.id %}">{{ exercise.title }}:
        {{ exercise.owner }} no{{ exercise.id }} difficulté :{{ exercise.grade }}</a>
    </div>
    <div class="panel-body">
        <a id ="resolve" href="{% url 'exercises:done' exercise.id %}">
        Les résolutions des élèves</a>
    </div>
</div>
```

`<div class="panel-heading">` redirige vers la page de résolutions et `<div class="panel-body">` vers les résolutions des élèves.

3.5.4 Le template `resolve.html`

`resolve.html` permet à un élève de résoudre un exercice. Du coup, un formulaire doit être présent dans le template. Pour cela, on utilise la balise `<form>` à laquelle il faut ajouter la commande `{% csrf_token %}` permettant de sécuriser les données qui seront entrées par l'utilisateur.

```
<form id="resolve-form" action="{% url 'exercises:resolve' id %}" method="post">
{% csrf_token %}
    <div>
        <label for="response">Résoudre l'équation</label>
        <textarea type="text" id="response" name="response" class="form-control">
        </textarea>
    </div>
    <button type="button" id="submit-resolve" class="btn btn-sm btn-primary">
    Soumettre et voir le corrigé</button>
    <a class="btn btn-sm btn-primary" href="{% url 'exercises:find' %}">Retour</a>
</form>
```

Le bouton `<button type="button" id="submit-resolve" class="btn btn-sm btn-primary">` renvoie la même fonction javascript que pour le template `find.html`. Cela renvoie un message d'erreur si l'utilisateur n'a pas rempli tout le formulaire et envoie les données à la vue `resolve.html` si le formulaire est complet.

Le fichier javascript se trouve dans `static/exercises/js/resolve.js`.

```
$(document).ready(function() {
  $("#submit-resolve").click(function() {
    // renvoie le formulaire si tous les champs sont remplis
    if ($("#response").val()) {
        $("#resolve-form").submit();
    }
    else {
        // Affiche un message d'erreur si tous les champs ne sont pas remplis
        $("#form-warning").modal("show");

    }
  });
});
```

3.5.5 le template `done.html`

Le template `done.html` utilise la fonction `get_lines` présent dans `models.py` pour créer une liste contenant toutes les résolutions faites pour un exercice. LA liste est traité à l'aide d'une boucle `for` pour séparer les résolutions et rendre la page plus claire. Si l'exercice ne comporte aucune résolution, on affiche le texte suivant : "Aucune résolution effectuée pour cet exercice"

```
<h2>Voici l'équation de l'exercice no{{ exercise.id }}</h2>
<h1 class="resolve">$$ {{ exercise.equation }} $$</h1>
<h2 id="titre">Résolution des élèves</h2>
{% if exercises_done %}
{% for exercise in exercises_done %}
    <div class="thumbnail">
        <div class="caption-full">
            <h2>{{ exercise.student }}</h2>
            {% for element in exercise.get_lines %}
            <h2 class="resolve">$$ {{ element }} $$</h2>
            {% endfor %}
            <p id="date">Fait le : {{ exercise.do_on }}</p>
        </div>
    </div>
{% endfor %}
{% else %}
<div class="thumbnail">
    <div class="caption-full">
        <h4 class="resolve">Aucune résolution effectuée pour cet exercice</h4>
    </div>
</div>
{% endif %}
```

CHAPITRE 4

Conclusion

L'application `Exercises` est fonctionnelle, elle ne comporte aucun bogue. Elle permet à un professeur d'élaborer un exercice et connaître les résultats des élèves. Cette application comporte néanmoins des limites pédagogiques.

En effet, bien qu'un professeur puisse créer un exercice de factorisation ou de développement, il est cependant impossible pour lui de toucher à d'autres domaines mathématiques. De plus, un élève accède au corrigé de l'exercice mais rien ne lui indique concrètement si sa résolution est correcte ou non.

Nous pourrions imaginer continuer le développement de cette application en y ajoutant la possibilité de faire des exercices regroupant des domaines plus vastes que de l'algèbre de base. De plus, un rendu côté client de la véracité d'une résolution serait formidable.

CHAPITRE 5

Webographie

TUTORIEL DE HTML5, consulté la dernière fois en septembre 2014 ; <http://www.codecademy.com/fr/courses/web-beginner-fr-FR>

TUTORIEL DE PYTHON 3, consulté la dernière fois en septembre 2014 ; <http://www.codecademy.com/fr/tracks/python-fr-FR>

TUTORIEL DE JQUERY, consulté la dernière fois en septembre 2014 ; <http://www.eclaireur.net/technique/scripts-exemples-ressources-developper-jquery/>

DOCUMENTATION DE PYTHON 3, consulté la dernière fois le 27.03.15 ; <https://docs.python.org/3/>

EUILLE DE TRICHE D'HTML5, consulté la dernière fois le 27.03.15 ; <http://overapi.com/html/>

FEUILLE DE TRICHE DE CSS3, consulté la dernière fois le 27.03.15 ; <http://overapi.com/css/>

FEUILLE DE TRICHE DE JAVASCRIPT, consulté la dernière fois le 27.03.15 ; <http://overapi.com/javascript/>

DOCUMENTATION DE BOOTSTRAP, consulté la dernière fois le 27.03.15 ; <http://getbootstrap.com/gettihng-started/>

FEUILLE DE TRICHE DE JQUERY, consulté la dernière fois le 27.03.15 ; <http://overapi.com/jquery/>

DOCUMENTATION DE DJANGO, consulté la dernière fois le 27.03.15 ; <https://docs.djangoproject.com/en/1.7/>

DOCUMENTATION DE MATHJAX, consulté la dernière fois le 27.03.15 ; <https://www.mathjax.org/#docs>