
Conception d'une plateforme de e-learning

Développement d'une application de création d'exercices de factorisation et de développement

Collège du sud, Travail de maturité

Florian Genilloud

30 Mars 2015

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	La page de création d'exercice	3
2.2	La page de résolution d'exercice	4
3	Documentation du développeur	5
3.1	Démarrage du projet depuis Cloud9	6
3.2	Les modèles	6
3.3	Les vues	9
3.4	Les urls	12
3.5	Les templates	13
4	Conclusion	18
5	Webographie	19
6	Table des illustrations	20

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 de l'informatique. En effet, depuis mon plus jeune âge, le monde de l'informatique et de la technologie font partie intégrante de ma vie. De plus, le sujet de ce travail de maturité étant *Développement d'une plateforme Web d'e-learning* concernant surtout les mathématiques a joué un rôle important dans mon choix car les mathématiques sont un domaine complexe et très intéressant. De ce fait, ce travail de maturité était parfait pour moi en ce qui concerne ma curiosité et mon envie d'en apprendre plus sur l'informatique.

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 le *développement d'une application de création d'exercices de factorisation et de développement*. Les objectifs de base pour qu'une application de ce type puisse fonctionner sont bien évidemment qu'un professeur puisse créer un exercice et qu'un élève puisse le résoudre en sachant si sa résolution est juste ou fausse. Relativement à ce dernier point, la méthode la plus simple est de permettre au professeur de créer un corrigé de l'exercice que l'élève pourra voir par la suite. Pour arriver à un tel résultat, la méthode la plus simple est d'utiliser un framework possédant un système de base de données et d'une partie s'occupant des requêtes de l'utilisateur. Pour cela, le framework Django écrit sous le langage de programmation Python est idéal car il possède tous les points précédemment cités.

1.2 Le but de l'application

Voici la documentation de l'application `exercices` présente sur le site [suivant](https://webmath-thirteenfoil8.c9.io/exercises/)¹. Celle-ci permet de pouvoir utiliser la partie création ainsi que la partie résolution des exercices de manière complète et détaillée. Cette application servira par la suite à un professeur de pouvoir créer un exercice de factorisation ou de développement et de pouvoir 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 mathématique dans le domaine de la factorisation et dans celui du calcul. Elle est essentielle au projet pour que les professeurs puissent créer des exercices selon le besoin de leurs élèves et pour pouvoir analyser les erreurs que les élèves font par rapport à ceux-ci. Cela permet aussi à un élève de

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

savoir où sont ses difficultés et de savoir quelles sont les thèmes qu'il doit travailler. 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. 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.

1.3 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 pouvoir 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 pouvoir contenir les liens des exercices qui sont en rapport avec celui-ci.
- La collaboration avec les quiz :
 - Il faudrait pouvoir mettre en relation un exercice ou bien un groupe d'exercices avec un ou plusieurs quiz ayant le même but pédagogique.

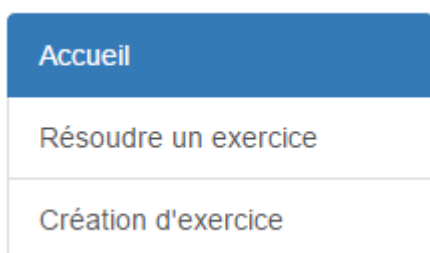
1.4 Intégration de l'application

L'intégration de cette application au reste du projet ne devrait normalement pas poser trop 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.

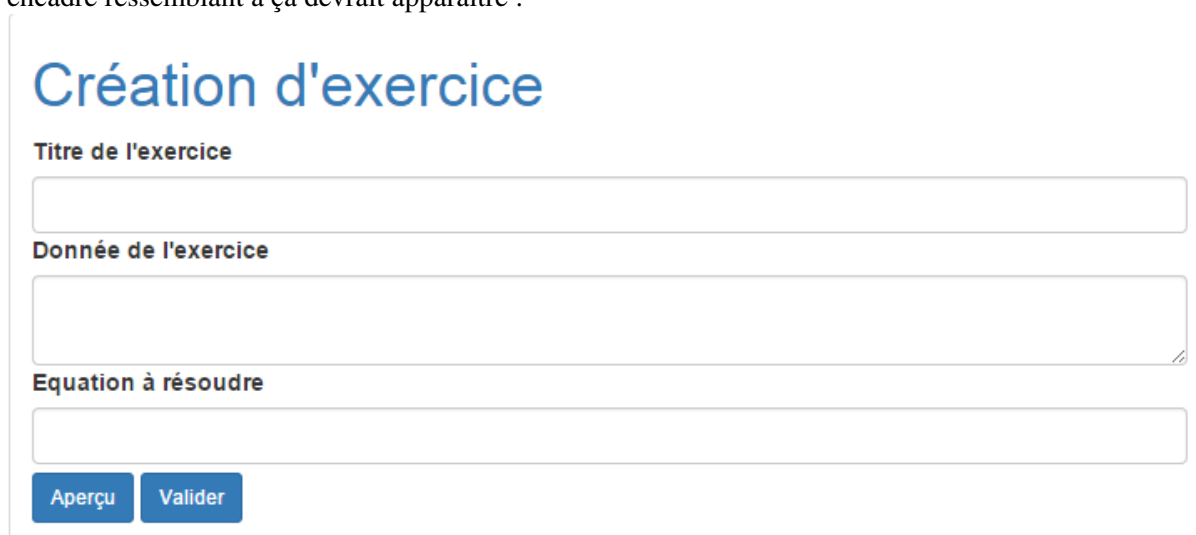
Documentation de l'utilisateur

2.1 La page de création d'exercice

L'application `Exercice` possède un onglet `création d'exercice`, présent sur le menu latéral permettant de renvoyer à la page de création d'exercice.



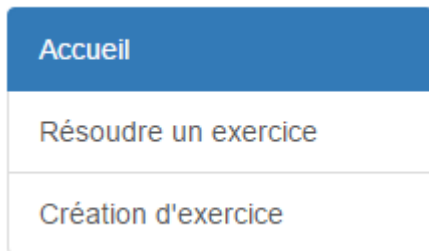
Cliquez sur l'onglet `Création d'exercice` pour aller sur la page de création. Une fois sur la page, un encadré ressemblant à ça devrait apparaître :

La page de création d'exercice est un formulaire encadré. Elle commence par un titre 'Création d'exercice' en bleu. Ensuite, il y a trois sections, chacune avec un titre et un champ de saisie : 'Titre de l'exercice', 'Donnée de l'exercice' et 'Equation à résoudre'. Les champs de saisie sont des rectangles blancs avec une bordure grise. À la fin du formulaire, il y a deux boutons : 'Aperçu' et 'Valider', tous deux en bleu.

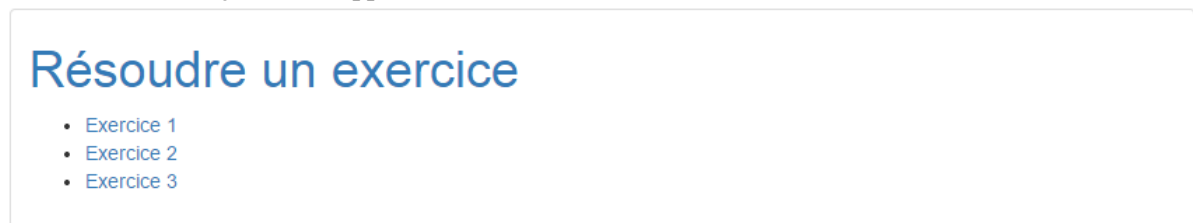
Tout est dit, il suffit juste de spécifier le titre, la donnée ainsi que l'équation à résoudre pour pouvoir créer un exercice. Si vous observez bien, vous remarquez un bouton `aperçu` présent à côté de celui nommé `valider`. Celui-ci, sert une fois avoir insérer les données voulu d'avoir un aperçu de ce qui sera présent dans l'exercice.

2.2 La page de résolution d'exercice

L'application exercice possède un onglet [Résoudre un exercice](#), présent sur le menu latéral permettant de renvoyer à la page de résolution d'exercice.



Cliquez sur l'onglet Résoudre un exercice pour aller sur la page de résolution. Une fois sur la page, une liste ressemblant à ça devrait apparaître :



Comme vous pouvez le constater, tous les exercices créés sont présents sur cette page. Il suffit juste de cliquer sur l'exercice dont vous avez besoin pour être redirigé vers celui-ci.

Documentation du développeur

Cette partie de la documentation est essentiellement destinée au développeur qui aimerait comprendre comment cette application fonctionne. 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 langages de programmation et les frameworks suivant pour comprendre la documentation développeur :

- Les langages 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](https://c9.io/)⁸ est très utile. Cela permet de ne pas surcharger la machine sur laquelle on travaille et le code est accessible depuis n'importe quel ordinateur dans le mode possédant une connexion internet.

Une fois sur Cloud9, il faut créer un *Workspace custom* en cliquant sur l'onglet *create new workspace*. Voici la série de commande 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 manage.py runserver $IP:$PORT
```

Il est à noter que le projet contenant l'entier des fichiers est sur un [dépôt](https://github.com/thirteenfoil8/TM-Code-Doc)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` présent plus bas dans la documentation), l'équation que l'élève devra traiter : `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 une résolution à un exercice fait 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.

2. La relation :

8. Le lien 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/>

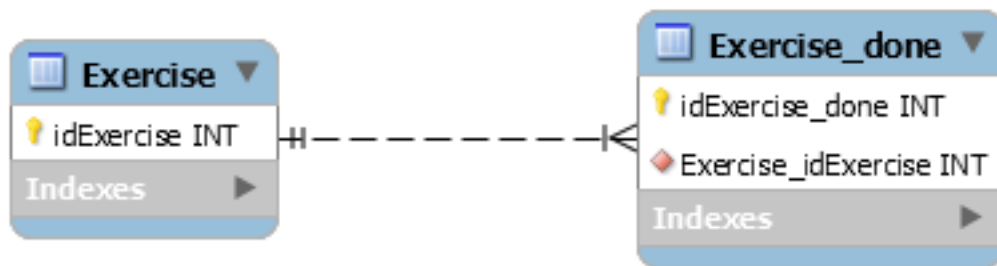


FIGURE 3.1 – *Diagramme UML du modèle relationnel*

C'est deux modèles sont relié entre eux grâce à une `ForeignKey` qui est présente dans la table `Exercice_done`. Cela signifie qu'un exercice peut posséder plusieurs résolution, mais qu'une résolution ne fait partie que d'un exercice.

3. Le code :

```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4
5 class Exercise(models.Model):
6
7     owner = models.CharField(max_length=20) # créateur
8     created_on = models.DateTimeField(auto_now_add=True) # date de création
9     title = models.CharField(max_length=30) # type d'exercice ( choisi
10                                     # dans create.html )
11     equation = models.CharField(max_length=50) # Equation de l'exercice
12     grade = models.CharField(max_length=60) # difficulté ( entre 1 et 5 )
13     correction = models.CharField(max_length = 200) # corrigé de l'exercice
14     def __str__(self):
15         # recherche plus facile dans http://webmath-thirteenfoil8.c9.io/admin/
16         return self.title + " " + self.owner + " " + str(self.pk)
17
18
19 class Exercise_done(models.Model): # Résolutions d'un exercice ( n...1 )
20     student = models.CharField(max_length=20) # Etudiant résolvant l'équation
21     do_on = models.DateTimeField(auto_now_add=True) # date de résolution
22     exercise_done = models.ForeignKey(Exercise) # l'exercice auquel les
23                                     # résolutions seront liées
24     resolution = models.CharField(max_length = 200) # la résolution
25
26     def __str__(self):
27         # recherche plus facile dans http://webmath-thirteenfoil8.c9.io/admin/
28         return self.exercise_done.title + " " + self.exercise_done.cowner/
29         + str(self.exercise_done.pk) + " fait par: " + self.student
30
31     # retourne une liste avec chaque ligne de la résolution.
32     def get_lines(self):
33         return self.resolution.split("\n")
```

4. Utilisation :

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

En premier lieu, il faut récupérer tous les objets déjà 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 enregistrer un objet et le traduire en SQL.

```
1 # ici, on utilise un formulaire.
2 if request.method == 'POST':
3     title = request.POST['type']
4     equation = request.POST['equation']
5     grade = request.POST['grade']
6     correction = request.POST['correction']
7     owner = request.user.username
8     Exercise(title=title, owner=owner, equation=equation, grade=grade, \
9     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/exercices/views.py`. Par la suite, deux points seront assez récurrents :

1. **L'appel `@login_required`** : Cette 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)`** : Cette 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 servent à gérer les authentifications et les permissions d'un utilisateur.

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

```
1 from django.shortcuts import render, HttpResponseRedirect, get_object_or_404, \
2 HttpResponseRedirect
3 from django.core.urlresolvers import reverse
4 from exercices.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, 'exercices/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 données entrées dans les balises du template `create.html` puissent être assignées et enregistrer 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("exercices:index"))
17     else:
18         return render(request, 'exercices/create.html')
```

Dans cette vue, la difficulté se trouve principalement dans l'enregistrement des données. A la ligne 4, la condition `if` permet de différencier si un enregistrement des données est nécessaire et dans le cas contraire, c'est le template `create.html` qui 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 sert à enregistrer 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 permet d'assigner à `latest_exercise_list` une liste comportant tous les exercices appartenant à la table `Exercise` présents dans la base de données. 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, 'exercices/find.html', {"exercices_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 résolutions d'un élève en rapport avec l'exercice `n_exercise` dans la table `Exercise_done`.

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

```
1 @login_required
2 def resolve(request, n_exercise):
3     exercise = get_object_or_404(Exercise, id=n_exercise) # Assigne les Querysets
4     # des objets exercise, 404 si inexistant
5
6     # enregistre les données du formulaire dans la base de données si requête
7     # POST sinon, retourne la page
8     if request.method == 'POST':
9         student = request.user.username
10        resolution = request.POST['response']
11        Exercise_done(exercise_done=exercise, resolution=resolution, \
12            student=student).save() # sauvegarde des données dans la db
13
14        return HttpResponseRedirect(reverse("exercices:correction", \
```

```
15         args=[n_exercice]))
16     else:
17         return render(request, 'exercices/resolve.html', \
18             {"exercice" : exercise, "id" : n_exercice})
```

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_exercice` dans la table `Exercice` puis on affecte cette valeur à la variable `correction`. L'utilisateur entre les étapes de la résolution de l'exercice ligne par ligne. Du coup, 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()`.

```
1 def correction(request, n_exercice):
2     correction = get_object_or_404(Exercice, id=n_exercice)
3     correction_line = correction.correction.split("\n")
4     return render(request, 'exercices/correction.html', locals())
```

3.3.5 La vue done

Cette vue permet à un professeur de voir toutes les résolutions des élèves présentes dans l'exercice `n_exercice`. La fonction `objects.filter()` permet d'affecter à la variable `exercices_done` les valeurs de l'objet `n_exercice` qui se trouvent dans la table `Exercice_done`. Cette dernière est en lien avec l'exercice grâce à une `ForeignKey`. Du coup, `exercices_done` peut contenir plusieurs objets.

```
1 @login_required
2 @user_passes_test(is_teacher)
3 def done(request, n_exercice):
4     exercise = get_object_or_404(Exercice, id=n_exercice)
5     exercices_done = Exercise_done.objects.filter(exercise_done=exercise)
6     return render(request, 'exercices/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 l'input `#search_input` présent dans le template `find.html` de retourner le lien de l'exercice `exercice.pk`. Une méthode Ajax est nécessaire pour éviter de faire recharger la page et rendre les recherches plus rapide.

```
1 def search(request):
2     search_input = request.GET["search"]
3
4     exercise = Exercise.objects.get(pk=search_input)
5
6     pk = exercise.pk
7     url = reverse("exercices:resolve", args=[exercise.pk])
```

```
8
9     json_dict = {
10         "pk" : pk,
11         "url" : url,
12     }
13
14     json_string = json.dumps(json_dict)
15
16     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 d'un des urls suivants, c'est à dire : `admin`, `exercices`, `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 urls présent dans les applications de pouvoir s'ajouter à l'url de base. Les urls de l'application `exercices` 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'^exercices/', include('exercices.urls', namespace='exercices')),
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 exercices

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. Pour les urls suivants, dès qu'il y a la présence de `(\d+)/`, cela appellera 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-dessous) comme valeur de l'argument `n_exercice`. 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` affichera une page d'erreur.

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.

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

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.

```
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 )
```

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` :

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

2. Soit sous forme d'appel du champ présent dans les modèles directement sur l'objet d'`Exercise` ou d'`Exercise_done`. Par exemple :

```
1 {{ exercise.equation }}
2 {{ exercise.id }}
```

De plus, au début de chaque template, on doit intégrer la ligne de code `{% extends "exercises/index.html" %}` pour permettre au template traité d'avoir le même Frontend que le template `index.html` qui est le template de base du site.

3.5.1 Le template de base du site

Pour ce qui est de la barre latéral se trouvant à gauche des pages du site, 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.

Pour ce qui est du Frontend, l'utilisation d'un thème Bootstrap permet de ne pas trop se focaliser sur le design. Pour cette application, Le thème `shop-item`¹² est parfait car il est simple, ergonomique et ne demande que très peu de modifications.

```
1 <div class="list-group">
2   <a href="{% url 'exercices:index' %}" class="list-group-item
3     {% block active-home %}active{% endblock %}">Accueil</a>
4
5   <a href="{% url 'exercices:find' %}" class="list-group-item
6     {% block active-reso %}{% endblock %}">Résoudre un exercice</a>
7
8   <a href="{% url 'exercices:create' %}" class="list-group-item
9     {% block active-create %}{% endblock %}">Création d'exercice</a>
10
11 </div>
```

Les urls de redirection vers les différentes pages du site sont gérés de la manière ci-dessus. On utilise `<a href="{% url 'exercices:<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.

3.5.2 Le template `create.html`

Le template `create.html` est le template utilisé par les professeurs pour créer l'exercice ainsi que son corrigé. Pour pouvoir enregistrer les données entrées par l'utilisateur, la présence de la balise `<form>` est absolument nécessaire. Toutes les données entrées sont traités dans la vue relative à ce template.

Le `<button id="voir">` utilise un script se trouvant sous `exercices/js/create.js`. Ce script est codé en jQuery et permet d'afficher la deuxième partie du formulaire et, grâce à la méthode `MathJax.Hub.Queue(["Typeset", MathJax.Hub])`, de formater l'équation entrée précédemment en la mettant sous une forme mathématique. Pour ce qui est de la documentation de MathJax, elle se trouve [ici](https://www.mathjax.org/#docs)¹³.

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 }
```

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

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


```
19
20     }
21     });
22 });
```

Pour ce qui est du deuxième bouton présent dans le template, il utilise le code javascript présent depuis la ligne 11. En utilisant la condition `if ($("#correction").val() && $("#equation").val())`, on contrôle que tous les champs du formulaire ont été remplis, sinon, on affiche un message d'erreur.

3.5.3 Le template `find.html`

Le template de cette page se trouve sous le fichier `static/exercices/templates/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 :

```
1 <div>
2     <label for="search">Entrez le numéro de l'exercice</label>
3     <input type="text" id="search_input" name="search" class="form-control">
4     <button type="button" id="search" name="search" class="btn btn-warning">Rechercher
5     </button>
6 </div>
7 <div class="alert alert-info" id="true">
8     <strong>Succès!</strong> <span id="lien"></span> de l'exercice en question.
9 </div>
10 <div class="alert alert-info" id="false">
11     <strong>Erreur!</strong> Cet exercice n'existe pas ou n'existe plus,
12     veuillez entrez un autre numéro
13 </div>
14 </div>
```

Grâce au script de cette page se trouvant dans `static/exercices/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>` en utilisant le code suivant :

```
1 $(document).ready(function() {
2     $('#false').hide(); // Cache les divs #false et #true
3     $('#true').hide();
4     $("#search").click(function() {
5         $("#lien").empty(); // Supprime l'éventuelle ancienne valeur
6         var $search = $("#search_input").val(); // enregistre la valeur de
7         //la recherche
8         $('#false').hide();
9         $('#true').hide();
10
11         $.ajax({
12             url: "/exercices/search/",
13             type: "GET",
14             dataType: "json",
15             data : {
16                 search : $search, //récupère les données de la recherche par
17                 //rapport à l'exercice recherché ( $search )
18             },
19         });
20     });
21 });
```

```
19         success : function(response) { // Ajoute le lien de l'exercice si
20           //il existe et l'affiche à l'utilisateur dans la div #true
21             var $url= response["url"];
22             $('#true').show();
23             $('<a>', {
24               "href": $url,
25             }).text("Voici le lien").appendTo("#lien");
26           },
27           error : function() { // Affiche le message d'erreur si l'exercice
28             //n'existe pas
29               $('#false').show();
30             }
31         });
32     });
33 }
```

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.

Pour ce qui est de la mise en page, les panel de Bootstrap sont très clairs et permette de bien différencié la page de résolution de l'exercice et la page contenant les résolutions des élèves. Cette dernière est accessible que par les professeurs.

```
1 <div class="panel panel-success">
2   <div class="panel-heading">
3     <a href="{% url 'exercises:resolve' exercise.id %}">{{ exercise.title }}:
4     {{ exercise.owner }} no{{ exercise.id }} difficulté :{{ exercise.grade }}</a>
5   </div>
6   <div class="panel-body">
7     <a id="resolve" href="{% url 'exercises:done' exercise.id %}">
8     Les résolutions des élèves</a>
9   </div>
10 </div>
```

<div class="panel-heading"> contient le lien de la page de résolutions et <div class="panel-body"> contient la page contenant 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.

```
1 <form id="resolve-form" action="{% url 'exercises:resolve' id %}" method="post">
2   {% csrf_token %}
3   <div>
4     <label for="response">Résoudre l'équation</label>
5     <textarea type="text" id="response" name="response" class="form-control">
6     </textarea>
7   </div>
8   <button type="button" id="submit-resolve" class="btn btn-sm btn-primary">
9   Soumettre et voir le corrigé</button>
10   <a class="btn btn-sm btn-primary" href="{% url 'exercises:find' %}">Retour</a>
11 </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/exercices/js/resolve.js`.

```
1 $(document).ready(function() {
2     $("#submit-resolve").click(function() {
3         // renvoie le formulaire si tous les champs sont remplis
4         if ($("#response").val()) {
5             $("#resolve-form").submit();
6         }
7         else {
8             // Affiche un message d'erreur si tous les champs ne sont pas remplis
9             $("#form-warning").modal("show");
10        }
11    });
12 });
13 });
```

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. Ensuite, on traite cette liste 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"

```
1 <h2>Voici l'équation de l'exercice no{{ exercise.id }}</h2>
2 <h1 class="resolve">$$ {{ exercise.equation }} $$</h1>
3 <h2 id="titre">Résolution des élèves</h2>
4 {% if exercises_done %}
5 {% for exercise in exercises_done %}
6     <div class="thumbnail">
7         <div class="caption-full">
8             <h2>{{ exercise.student }}</h2>
9             {% for element in exercise.get_lines %}
10                <h2 class="resolve">$$ {{ element }} $$</h2>
11            {% endfor %}
12            <p id="date">Fait le : {{ exercise.do_on }}</p>
13        </div>
14    </div>
15 {% endfor %}
16 {% else %}
17 <div class="thumbnail">
18     <div class="caption-full">
19         <h4 class="resolve">Aucune résolution effectuée pour cet exercice</h4>
20     </div>
21 </div>
22 {% endif %}
```

Conclusion

L'application `Exercises` est fonctionnelle, ne comporte aucun bogue et permet à un professeur de créer un exercice et voir les résolutions des élèves. L'élève, quant à lui, peut résoudre un exercice. En prenant du recul, nous nous rendons compte que cette application comporte 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 peut bien évidemment voir le 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 pouvoir faire des exercices englobant des domaines plus vaste que de l'algèbre de base. De plus, un rendu côté client de la véracité d'une résolution serait formidable.

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>>

Table des illustrations

Table des illustrations

3.1 *Diagramme UML du modèle relationnel* 7