Создание REST веб-сервиса.

Вы решили разработать информационный API для текстовой online RPG «Valar Morghulis» («VM») по мотивам «Игры Престолов». Реализацию API вы решили сделать в виде REST -сервиса.

Задание: разработать REST-сервис (и клиента для него *) для игры «Valar Morghulis» («VM»). Прототип сервиса игры должен поддерживать предоставление и редактирование следующей информации:

1. Player – персонаж игры:
    a. id – уникальный идентификатор персонажа (int);
    b. name – Имя персонажа (не более 30 символов);
    c. playerclass – Класс персонажа (Knight, Wizard, Thief, Paladin);
    d. email;
    e. level – уровень персонажа;
    f. position – положение персонажа на карте (идентификатор локации).
2. ItemType - виды предметов в игре:
    a. id - уникальный идентификатор ItemType (INT);
    b. name - название ItemType (меч, шлем ...);
3. Item - отдельные предметы в игре, принадлежащие игрокам:
    a. itemType - тип элемента;
    b. quality - качество предмета (100 - идеальное, 0 - сломан);
    c. owner - игрок-владелец предмета.
4. Location – локация в игре
    a. locationId – краткое название-идентификатор локации (не более 10 символов)
    b. description – текст с описанием локации
    c. locationType – тип локации (Forest, Desert, Dungeon, River, Ocean)
5. Messages – сообщения между игроками
    a. messageId – идентификатор сообщения;
    b. playerFrom – игрок, отправивший сообщение;
    c. playerTo – игрок, получивший сообщение;
    d. messageText – текст сообщения (не более 1000 символов).

Методические рекомендации

1) Перейти на сайт http://www.jetbrains.com/pycharm/download/ и получить пробной выпуск профессиональной версии PyCharm (вы можете использовать Community Edition, но вам придется потратить некоторое время, чтобы настроить работу Python и Django). Также, для работы можно воспользоваться PyCharm, установленным в ПВК.
2) Перейти на сайт http://www.python.org и загрузить релиз 2.6.7 . http://www.python.org/download/releases/2.7.6/
3) Установить все это на вашем ПК
4) Откройте Pycharm и начните новый проект: File->New Project…
5) Укажите имя для вашего проекта: «VM_REST_Service»
6) Выберите «Django project»
7) Нажмите «...» справа от строки “Interpreter”.

8) Если Pycharm не нашел папку с python, помогите ему, нажав на кнопку «+» и показав ему путь к интерпретатору python.
9) Нажмите «установить pip» и «setuptools» в желтой полосе в нижней части
10) После этого необходимо установить все пакеты дополнительные python-пакеты, которые мы будем использовать в наших проектах. Нажмите «Install» и найдите следующие пакеты:
    a. django (https://www.djangoproject.com/)
    b. djangorestframework (http://www.django-rest-framework.org/)
11) Могут возникнуть проблемы с https-соединением. В этом случае вы можете скачать и установить это пакеты отдельно:
    a. Перейти на https://pypi.python.org/simple
    b. Найти пакет «Django» и нажмите на ссылку
    c. Найдите «Django-1.6.tar.gz» найти и скачайте его
    d. Найдите пакет «djangorestframework» и нажмите на ссылку
    e. Найдите «djangorestframework-2.3.9.tar.gz» и загрузите его
    f. Откройте командную строку (cmd) и перейдите в папку C:\Python27\Scripts
    g. Введите: pip install C:\Users\{your_user_name}\Downloads\Django-1.6.tar.gz
    h. Введите: pip install C:\Users\{your_user_name}\Downloads\ djangorestframework-2.3.9.tar.gz
12) Выберете интерпретатор, который вы только что настроили для вашего проекта.
13) Нажмите кнопку «ОК»
14) Выберите имя приложения, «RestService» и нажмите «ОК».
15) Появится дерево проекта по умолчанию для нашего нового проекта.
16) Во-первых мы должны создать небольшую базу данных для нашего проекта. Нажмите Tools -> Run manage.py Task -> syncdb. Если у вас установлен Python v. 3 и выше, вам необходимо ввести имя пользователя и пароль для администратора базы данных отдельно. Для этого необходимо открыть командную строку и перейти в папку, в которой располагается каталог с сайтом, после чего запустить и выполнить там команду “manage.py createsuperuser”
17) Введите имя супер пользователя («user») и пароль («test»), для нашего сайта и нашей БД. Будет автоматически создан файл базы данных «db.sqlite3».
18) Давайте проверим, если он работает: нажмите зеленую стрелку в верху «Run»
19) Внизу должен появиться адрес нашего веб-приложения, что-то вроде http://127.0.0.1:8000/ Давайте откроем эту страницу в веб-браузере.
20) Если мы добавим «admin» в наш адрес, мы можем открыть страницу автоматически сгенерированного интерфейса администрирования: http://127.0.0.1:8000/admin
21) Введите имя пользователя и пароль, который вы только что указали
22) Django предоставляет интерфейс администрирования по умолчанию, так что вы можете редактировать информацию о вещах («models»), которые хранятся в вашей базе данных.
23) Теперь давайте добавим GameMap в модель нашего приложения:
    a. Tools -> Run manage.py Task -> startapp и введите имя для пакета «GameMap»
    b. Укажем следующий код внутри models.py в папке «GameMap»:

```
from django.db import models

Location type = (
    ('Forest', u'Forest'),
    ('Desert', u'Desert'),
    ('Dungeon', u'Dungeon'),
    ('River', u'River'),
    ('Ocean', u'Ocean'),
)

class Location(models.Model):
    locationId = models.CharField(verbose_name=u'Location Id', max_length=10, unique=True, default='0-0')
    description = models.TextField(verbose_name=u'Location Description', blank=True)
```

```
    locationType = models.CharField(verbose_name=u'Location Type', max_length=15, choices=Location_type,
default='Forest')
    def __unicode__(self):
        return self.locationId
```

ВНИМАНИЕ!!! В python отступы – очень важны, так что пожалуйста не игнорируйте их!

Давайте обсудим, что делает этот код.

c. Добавьте следующий код в admin.py внутри папки «GameMap»:

```
from django.contrib import admin
from GameMap.models import Location

class Location_Admin(admin.ModelAdmin):
    list_display = ('locationId',)

admin.site.register(Location, Location_Admin)
```

d. Давайте проверим, работает ли наш сайт и сделаем тестовый запуск: запустите сайт и проверьте страницу «admin».
e. Ничего не изменилось, потому что Джанго пока не знает, что мы создали эту новую модель. Мы должны добавить приложение «GameMap» на сайт Django.
f. Откройте setteings.py в папке «VM_REST_Service\VM_REST_Service»

Прочтите этот файл, это довольно интересно.

g. Добавим модуль GameMap в блок установленных приложений:

```
INSTALLED_APPS = (
     ...

     'RestService',
     'GameMap',
)
```

h. Кроме того, мы должны синхронизировать нашу модель GameMap с базой данных. Запустите команду syncdb (см. пункт 16). Если вы используете Python v 3, вам вместо этого необходимо выполнить 2 команды: makemigrations и migrate.
i. Попробуем перезагрузить наш сервер и открыть страницу «admin» еще раз. Вы должны увидеть там блок «GameMap». Нажмите кнопку «Locations» и попробуйте добавить или изменить местоположения.

24) Добавим также приложения “Player” и “Inventory” (см. пункт 23).
25) Player/models.py

```
from django.db import models

Choices_class = (
    ('Knight', u'Knight'),
    ('Wizard', u'Wizard'),
    ('Thief', u'Thief'),
    ('Paladin', u'Paladin')
)

class Player(models.Model):
    class Meta:
```

```
        app_label = 'Player'
    name = models. ForeignKey(verbose_name=u'User Name', max_length=30, blank=True)
    playerclass = models.CharField(verbose_name=u'Class', max_length=15, choices=Choices_class,
default='Knight')
    email = models.CharField(verbose_name=u'Email', max_length=50, blank=True)
    level =  models.IntegerField(verbose_name=u'Player level', default=0)
    position = models.ForeignKey(Location, default=1)
    def __unicode__(self):
        return self.name
```

26)Player/admin.py

Player/admin.py – заполните самостоятельно

27)Inventory/models.py

```
from django.db import models
from django.core.validators import MinValueValidator, MaxValueValidator
from Player.models import Player

class ItemType(models.Model):
    name = models.CharField (max length=40, unique=True)
    def   unicode  (self):
        return self.type

class Item(models.Model):
    itemType = models.CharField(ItemType)
    quality = models.SmallIntegerField(default=100, validators=[MaxValueValidator(100), MinValueValidator(0)])
    owner = models.ForeignKey(Player, null=True)
    def __unicode__(self):
        return "{0} - {1}".format(self.itemType,self.quality)
```

28)Inventory/admin.py

Inventory/admin.py – заполните самостоятельно

29)Таким образом мы создали модель нашего приложения. Теперь вы можете использовать ваш интерфейс администратора для создания локаций, игроков, объектов в инвентаре. Создадим REST API для доступа к этой информации. Используем платформу django (http://www.django-rest-framework.org) для этого.

30)Добавим в раздел установленных приложений (в файл settings.py) rest_framework

```
INSTALLED_APPS = (
     'django.contrib.admin',
     'django.contrib.auth',
     'django.contrib.contenttypes',
     'django.contrib.sessions',
     'django.contrib.messages',
     'django.contrib.staticfiles',
     'rest_framework',

     'RestService',
     'GameMap',
     'Inventory',
     'Player',
 )
```

31)Попробуем создать REST-доступ к списку ItemType «напрямую», с минимальным использованием средств django (чтобы понять, как это работает внутри)

a. В VM_REST_Service/urls.py добавьте следующую строку в блоке urlpatterns:

```
urlpatterns = patterns('',
     ...

     url(r'^', include('Inventory.urls'))
 )
```

Это позволит нам обрабатывать URL-адреса в нашем модуле Inventory. Прямо сейчас нет кода, который обрабатывает эти URL, так создадим его.

b. Во-первых мы должны создать сериализатор для нашей модели. Создайте файл Inventory/serializers.py. Мы можем использовать немного магии REST-фреймворка и позволить ему создать сериализатор из модели данных Джанго в родную модель классов Python для нас (с использованием базового класса ModelSerializer):

```
from rest_framework import serializers
from Inventory.models import ItemType

class ItemTypeSerializer(serializers.ModelSerializer):
    class Meta:
        model = ItemType
        fields = ('id', 'name')
```

Этот код автоматически создает сериализатор/десериализатор для нашей модели ItemType, включая его скрытое поле «id» и откроe поле «имя».

32)Теперь попробуем использовать родное представление Django для создания службы REST для нашей модели. Добавим следующий заголовок в Inventory/views.py

```
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt
from rest_framework.renderers import JSONRenderer
from rest_framework.parsers import JSONParser
from Inventory.models import ItemType
from Inventory.serializers import ItemTypeSerializer
```

33)Теперь добавим класс отображения JSON, чтобы мы могли предоставить ответ в формате JSON для любого запроса

```
class JSONResponse(HttpResponse):
     """
     HttpResponse, который отображает его содержимое в JSON.
     """
     def __init__(self, data, **kwargs):
         content = JSONRenderer().render(data)
         kwargs['content_type'] = 'application/json'
         super(JSONResponse, self).__init__(content, **kwargs)
```

34)Теперь давайте добавим метод для обработки запросов, который работает со списком ItemTypes

```
@csrf_exempt
 def ItemType_list(request):
     """
     Отобразить все коды ItemTypes, или создать новый ItemType.
```

```
    """
    if request.method == 'GET':
        snippets = ItemType.objects.all()
        serializer = ItemTypeSerializer(snippets, many=True)
        return JSONResponse(serializer.data)

    elif request.method == 'POST':
        data = JSONParser().parse(request)
        serializer = ItemTypeSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return JSONResponse(serializer.data, status=201)
    return JSONResponse(serializer.errors, status=400)
```

Давайте обсудим, что делает этот код.

35)Теперь давайте добавим метод для обработки запросов, которые работают с отдельными ItemType

```
@csrf_exempt
 def ItemType_detail(request, pk):
    """
    Retrieve, update or delete a code snippet.
    """
    try:
        itemType = ItemType.objects.get(pk=pk)
    except ItemType.DoesNotExist:
        return HttpResponse(status=404)

    if request.method == 'GET':
        serializer = ItemTypeSerializer(itemType)
        return JSONResponse(serializer.data)

    elif request.method == 'PUT':
        data = JSONParser().parse(request)
        serializer = ItemTypeSerializer(itemType, data=data)
        if serializer.is_valid():
            serializer.save()
            return JSONResponse(serializer.data)
        return JSONResponse(serializer.errors, status=400)

    elif request.method == 'DELETE':
        itemType.delete()
        return HttpResponse(status=204)
```

Давайте обсудим, что делает этот код.

36)Теперь давайте попробуем проверить наш проект: Нажимаем Пуск (или выполнить «python manage.py runserver») и открываем http://127.0.0.1:8000/Inventory/

37)Что-то не так, что мы забыли? Ах URL-пути!
38)Добавьте следующий код к Inventory/urls.py:

```
from django.conf.urls import patterns, url

 urlpatterns = patterns('Inventory.views',
    url(r'^Inventory/$', 'ItemType_list'),
    url(r'^Inventory/(?P<pk>[0-9]+)/$', 'ItemType_detail'),
```

```
)
```

39) Сейчас все должно заработать. Попробуйте открыть http://127.0.0.1:8000/Inventory/
Вы должны получить что-то вроде этого:

```
[{"id": 1, "name": "dagger"}, {"id": 2, "name": "sword"}, {"id": 3, "name": "helmet"}]
```

40) Попробуйте открыть http://127.0.0.1:8000/Inventory/1/
Вы должны получить что-то вроде этого:

```
{"id": 1, "name": "dagger"}
```

41) Чтобы проверить запрос «POST», можно использовать команду «curl» (родная для linus/macos, Windows версию вы можете найти здесь http://curl.haxx.se/download.html).
Используйте следующую команду:

```
curl -X POST http://127.0.0.1:8000/Inventory/ -d '{"name": "armor"}' -H "Content-Type: application/json"
```

Вы должны получить следующий ответ:

```
{"id": 4, "name": "armor"}
```

Теперь проверьте, http://127.0.0.1:8000/Inventory/ и убедитесь, что запрос «POST» на самом деле добавил и ItemType в нашу базу данных:

```
[{"id": 1, "name": "dagger"}, {"id": 2, "name": "sword"}, {"id": 3, "name": "helmet"},
{"id": 4, "name": "armor"}]
```

Также, для более наглядного взаимодействия можно использовать плагин к браузеру Google Chorme под названием Posman: https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm?hl=en

42) Теперь давайте обратимся к настоящей магии REST для остальной части наших моделей. Давайте использовать приложение RestService в качестве основы для остальной части нашей службы REST. Добавьте RestService/serializers.py:

```
from Player.models import Player
 from GameMap.models import Location
 from rest_framework import serializers

class PlayerSerializer(serializers.HyperlinkedModelSerializer):
     class Meta:
         model = Players
         fields = ('url', 'name', 'email', 'playerclass', 'level', 'position')

class LocationSerializer(serializers.HyperlinkedModelSerializer):
     class Meta:
         model = Location
         fields = ('url', 'locationId', 'description', 'locationType')
```

HyperlinkedModelSerializer обеспечивает гиперссылки вместо простых id.

43) Добавьте следующий код к RestService/views.py:

```
from Player.models import Player
from GameMap.models import Location
from rest_framework import viewsets
from RestService.serializers import LocationSerializer

class PlayerViewSet(viewsets.ModelViewSet):
     """
     Конечная точка API, который позволяет просматривать или редактировать игроков.
     """
     queryset = Player.objects.all()
     serializer_class = PlayerSerializer

class LocationViewSet(viewsets.ModelSerializer):
     """
     Конечная точка API, который позволяет просматривать или редактировать locations.
     """
     queryset = Location.objects.all()
     serializer_class = LocationSerializer
```

Базовый класс rest_framework.viewsets.ModelViewSet предоставляет полное решение для автоматической генерации интерфейса API REST для вашей модели Django.

44)Добавьте следующий код в файл параметров:

```
REST_FRAMEWORK = {
     'DEFAULT_PERMISSION_CLASSES': ('rest_framework.permissions.IsAdminUser',),
     'PAGINATE_BY': 10
 }
```

45)Добавьте следующий заголовок в VM_REST_Service/urls.py:

```
from django.conf.urls import patterns, url, include
from rest_framework import routers
from RestService import views
```

46)Добавьте следующий код к VM_REST_Service/urls.py:

```
router = routers.DefaultRouter()
router.register(r'players', views.PlayerViewSet)
router.register(r'locations', views.LocationViewSet)
```

47)Изменим urlpatterns в VM_REST_Service/urls.py:

```
urlpatterns = patterns('',
     # Examples:
     # url(r'^$', 'VM_REST.views.home', name='home'),
     # url(r'^blog/', include('blog.urls')),

     url(r'^admin/', include(admin.site.urls)),

     url(r'^', include(router.urls)),
     url(r'^', include('Inventory.urls')),
     url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework'))
 )
```

Давайте обсудим, что делает этот код.

48)Перезагрузите сервер и перейдите на страницу http://127.0.0.1:8000 .

49)Нажмите на ссылку http://127.0.0.1:8000/players/ URL
50)Вы можете попытаться добавить нового игрока с помощью нижней формы.
51)Или, вы можете попробовать использовать `curl` для получения списка игроков в формате JSON:

```
curl http://127.0.0.1:8000/players/
```

52)Упс. Нам нужна проверка авторизации для нашего запроса!

```
curl http://127.0.0.1:8000/players/ -u user:password

{"count": 2,
"next": null,
"previous": null,

"results": [
{"url": "http://127.0.0.1:8000/players/1/",
"name": "username",
"email": "adf",
"playerclass": "Knight",
"level": 0,
"position": "http://127.0.0.1:8000/locations/1/"},

{"url": "http://127.0.0.1:8000/players/2/",
"name": "seconduser",
"email": "adf",
"playerclass": "Thief",
"level": 10,
"position": "http://127.0.0.1:8000/locations/1/"}]}
```

Таким образом, мы создали интерфейс REST API для нашего сервиса. Поздравляю!

Пожалуйста, разработайте и внедрите модель сообщений (Messages) и REST API для нее.