deti universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Technologies and Web Programming

Django Framework

Django Framework

RESTful Web Services

Django REST framework

Web Services

- Web services are services offered by electronic devices to send data to another electronic devices, using web technologies.
- Normally, data is transmitted in JSON or XML format.
- One of the main purposes of web services is to provide interoperability and data integration between heterogeneous information systems.

REST *Web Services*

- REST – *Representational State Transfer*
  - It's an architectural model for hypermedia applications, mainly used for web services implementation, which are considered to be light, simple, sustainable and scalable.
  - A service based on this technology is named as RESTful Service.
  - REST services don't depend on any particular protocol, however most of them use HTTP for transporting.
  - Another kind of web services are the SOAP Web Services. These are based on the SOAP protocol, which is very formal, strict and heavy. That's why it's not often used.

Django REST framework (DRF)

- DRF is a python library to create REST Web Services integrated with Django framework.
- It provides an important set of functions for ease programming this kind of services, as:
  - The possibility to publish the provided API;
  - Authentication policies, using OAuth1a and OAuth2 protocols;
  - Data serialization from DBs, through Django ORM or other means;
  - It can use general views if advanced facilities aren’t needed;
  - Currently, it’s used by big organizations (Mozilla, Red Hat, etc.), what proves its credibility.

DRF - Installing

- Installing
  - pip install djangorestframework
  - pip install markdown
  - pip install django-filter
  - pip install django-cors-headers

Configuring (i)

- Add the following text lines “settings.py” file:

settings.py

```
33  INSTALLED_APPS = [
34      'django.contrib.admin',
35      'django.contrib.auth',
36      'django.contrib.contenttypes',
37      'django.contrib.sessions',
38      'django.contrib.messages',
39      'django.contrib.staticfiles',
40      'app.apps.AppConfig',
41      'rest_framework',
42      'corsheaders',
43  ]
44
45  MIDDLEWARE = [
46      'django.middleware.security.SecurityMiddleware',
47      'django.contrib.sessions.middleware.SessionMiddleware',
48      'corsheaders.middleware.CorsMiddleware',
49      'django.middleware.common.CommonMiddleware',
50      'django.middleware.csrf.CsrfViewMiddleware',
51      'django.contrib.auth.middleware.AuthenticationMiddleware',
52      'django.contrib.messages.middleware.MessageMiddleware',
53      'django.middleware.clickjacking.XFrameOptionsMiddleware',
54  ]
```

Configuring (ii)

- Add the following configuration to “settings.py” file:

settings.py

```
129  # DJANGO REST FRAMEWORK Config
130  REST_FRAMEWORK = {
131      'DEFAULT_PERMISSION_CLASSES': [
132          'rest_framework.permissions.AllowAny'
133      ]
134  }
135
136  # CORS (Cross-Origin Resource Sharing) config
137  CORS_ORIGIN_ALLOW_ALL = True
```

Serializers

- Creating serializers to put data from BD in a sending format.
  - Create a file named “serializers.py” in folder “app”.

serializers.py

```
1   from app.models import Author, Publisher, Book
2   from rest_framework import serializers
3
4   class AuthorSerializer(serializers.ModelSerializer):
5       class Meta:
6           model = Author
7           fields = ('id', 'name', 'email')
8
9   class PublisherSerializer(serializers.ModelSerializer):
10      class Meta:
11          model = Publisher
12          fields = ('id','name', 'city', 'country', 'website')
13
14  class BookSerializer(serializers.ModelSerializer):
15      class Meta:
16          model = Book
17          fields = ('id','title', 'date', 'authors', 'publisher')
```

Views (i)

- Creating views to send data
  - Imports:

views.py

```
7
8   from rest_framework import status
9   from rest_framework.decorators import api_view
10  from rest_framework.response import Response
11  from app.serializers import AuthorSerializer
```

Views (ii)

- Configuring urls routes.

urls.py

```
30  urlpatterns = [
31      # web services
32      path('ws/author', views.get_author),
33      path('ws/authors', views.get_authors),
34      path('ws/authorcre', views.create_author),
35      path('ws/authorupd', views.update_author),
36      path('ws/authordel/<int:id>', views.del_author),
```

Views (iii)

- View to get one author.

views.py

```
185  # web service to get specific author
186  @api_view(['GET'])
187  def get_author(request):
188      id = int(request.GET['id'])
189      try:
190          author = Author.objects.get(id=id)
191      except Author.DoesNotExist:
192          return Response(status=status.HTTP_404_NOT_FOUND)
193      serializer = AuthorSerializer(author)
194      return Response(serializer.data)
```

Views (iv)

- View to get a list of authors.

views.py

```
197  # web service to get a list of authors
198  @api_view(['GET'])
199  def get_authors(request):
200      authors = Author.objects.all()
201      if 'num' in request.GET:
202          num = int(request.GET['num'])
203          authors = authors[:num]
204      serializer = AuthorSerializer(authors, many=True)
205      return Response(serializer.data)
```

Views (v)

- View to create an author.

views.py

```
207
208  # web service to create an author
209  @api_view(['POST'])
210  def create_author(request):
211      serializer = AuthorSerializer(data=request.data)
212      if serializer.is_valid():
213          serializer.save()
214          return Response(serializer.data, status=status.HTTP_201_CREATED)
215      return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
216
```

Views (vi)

- View to update an author.

views.py

```
218  # web service to update an author
219  @api_view(['PUT'])
220  def update_author(request):
221      id = request.data['id']
222      try:
223          author = Author.objects.get(id=id)
224      except Author.DoesNotExist:
225          return Response(status=status.HTTP_404_NOT_FOUND)
226      serializer = AuthorSerializer(author, data=request.data)
227      if serializer.is_valid():
228          serializer.save()
229          return Response(serializer.data)
230      return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
231
```

Views (vii)

- View to delete an author.

views.py

```
233  # web service to delete an author
234  @api_view(['DELETE'])
235  def del_author(request, id):
236      try:
237          author = Author.objects.get(id=id)
238      except Author.DoesNotExist:
239          return Response(status=status.HTTP_404_NOT_FOUND)
240      author.delete()
241      return Response(status=status.HTTP_204_NO_CONTENT)
```