Programmable Web Project

RESTful Web APIs with Django

NOTE: Updated in February 2013

Iván Sánchez

Django

- Django is a web framework written in Python.
- They claim that Django is designed to make common Web-development tasks fast and easy
- Links:
  - Project Web page: http://www.djangoproject.com/
  - Lot of documentation: https://docs.djangoproject.com/en/1.4/
  - Tutorial: https://docs.djangoproject.com/en/1.4/intro/tutorial01/
- The VM machine has version 1.4

Iván Sánchez

Django

- Main characteristics:
  - Design according to MVC principles.
    - Model – View –Controller
  - Dynamic admin interface.
  - Support for URI templates.
  - Caching and syndication framework.
  - Dynamic and customizable middleware that allows preprocessing and postprocessing HTTP requests and responses:
    - Similar to Filters in Java Servlet
    - Several layers: SessionMiddleware, Authentication Middleware

Django MVC

- **Model**: contains the essential fields and behaviors of the data you're storing.
  - ***Resource state*** talking in RESTful terms
  - Django has an ORM (Object Relational Mapper) that allows storing Models in Database.
  - Each model is a Python class that is derived from django.db.models.Model.
    - **We are using our own models in this exercsies; not use django.Model as BaseClass!!!**

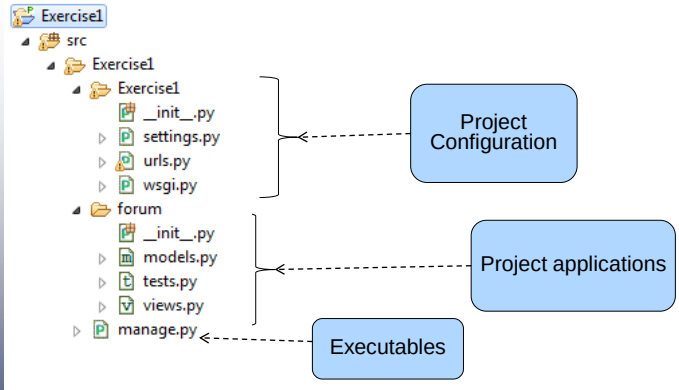
Django MVC

- **View**: a Python function that takes a HTTP request and returns a HTTP response.
  - Is the ***resource representation*** talking in RESTful terms
- Django has its own template language to generate dynamic Web pages
  - template is a text document marked-up using the Django template language.
    - A template can contain block tags or variables.
    - Similar to JSP in Java or PHP
  - **NOT used in this exercise**

Django project structure

- Django is divided in projects
  - Each project runs an instance of the server (must listen requests in a different port)
  - Each project defines its own settings and middleware layers
  - Each project is formed by different applications

Django project structure



You can create a project using the executable from the django/bin or Python/Scripts folder:

```
django-admin.py startproject "mysite"
```


Django project structure

- Files at the root of the project
  - **manage.py** -> Server exectuble.
    - Default port: 8000

```
python manage.py runserver port
```

- Common configuration files in the **project** folder
  - **settings.py** -> Configure your project
    - Define yoDefine database settings
    - ur middleware classes
    - Define the location of ROOT_URLCONF
    - Define the installed applications: INSTALLED_APPS
  - **urls.py** -> The URL declarations for this Django project
    - Tells which is the View which must process a request to a specific URI

Django project structure

- Files at the application level:
  - **models.py** -> Defines the model classes
  - **views.py** -> Defines the views classes
  - **urls.py** -> Urls declaration of this Django application
    - Defines which View must process each URI
    - THIS IS NOT MANDATORY, this information could be in the project urls.py

Django URL dispatcher

- urls.py at the application level has the following structure:

```
urlpatterns = patterns('',
    url(r'forum/messages/(?P<message_id>msg-\d+)$',
       'app2.messageView', name="message"),
    url(r'forum/messages$',
       'app2.messagesView',name="messages"),)
```

- `urlpatterns` contains a list of **patterns**:
  - Each **pattern** is a url, that is a triple formed by:
    - regular expression of the URL path **(URI template)**
    - view class that process the request and generate the response
    - name keyword which contains a unique identifier for this pattern (useful for reverse)
  - Each **URI template** might contains URI template variables
    - The are passed to the view as a keyword argument

Django URL dispatcher

- URI templates variables

```
r'forum/messages/(?P<message_id>msg-\d+)$'
```

  - `/(?P<message_id>`
    - defines a URI template variable named message_id
  - `msg-\d+`
    - defines the regular expression for this variable, that is, a regular expression that restricts the possible values.
  - Example: `forum/messages/msg-1`
    - is a URI that meets this URI template
    - *"message_id"* variable is set to "msg-1"
- You can define regular expression at any point of the URI template
  - Regular expressions in Python: http://docs.python.org/library/re.html

Django URL dispatcher

- `django.core.urlresolvers.reverse ()` function allows to create URI templates dynamically, setting the values for the URI template variables
  - Syntax:

```
reverse('URL_name',(variable1_value,variable2_value,...))
```

Name of the url as defined in the urls.py

values for each one of the URI templates variables defined for that URI template

  - Example, you want to create a link that:
    - points to the URI which identifies '`app2.messageView`'
    - with the message id= msg-1,
    - that is: `/forum/messages/msg-1`

```
reverse("message",("msg-1",))
```

Django REST framework

- Library that applies the REST principles to Django frameworks
  - Substitute the View class by APIView class
  - You can still use Django Models
  - Expand Django model data in formats such as XML, JSON and YAML
  - More info:
    http://django-rest-framework.org/
- urls.py changes to:

```
urlpatterns = patterns('',
    url(r'messages/(?P<message_id>msg-\d+)$',
Message.as_view(),      name="message"),
    url(r'messages$', Messages.as-view(),name="messages"),)
```

where `Message` and `Messages` are derived class from `APIView` class.

Django REST framework

- Skeleton

```
class Message (APIView):

 #DELETE
   def delete(self,request,*uritemplatevariables):
       pass
  #GET
   def get(self,request,*uritemplatevariables):
       pass
  #PUT
   def put (self,request,*uritemplatevariables):
       pass
  #POST
    def post(self,request,*uritemplatevariables):
        pass
```

Methods:
- Each one maps to one HTTP method

Django REST framework

- The method receive extra arguments if the associated uri template contains template variables
- Given the following url definition:

```
url(r'^messages/(?P<message_id>msg-\d+)$',
Message.as_view(), name="message"),
```

- You have to define the methods for Message() as follows:

```
def get(self,request,message_id):
        pass
```


Django REST framework. Request object

- Extends Django HttpRequest, adding support for REST framework request parsing and authentication.
- Data attributes:
  - `.DATA`=> Parsed content of the request body. Data is stored as a set of native python datatypes (See later)
  - `.QUERY_PARAMS`=> A dictionary
  - `.META:` returns a dictionary containing all available HTTP headers. The header name is modified:
    - converting all characters to uppercase
    - replacing any hyphens with underscores
    - adding an HTTP_ prefix to the name

Django REST framework. Response object

- You can generate a HTTP response using the following syntax:

```
response=Response("serialized_body_content",
   status="response_status-code",
   headers="{headername: value,}")
```

- It is very important that the body is serialized into a set of native python datatypes (generaly a dictionary)
- Django REST framework transform the serialized body content into a format that the client accepts.
  - Checks the Accept HTTP request Header
  - Django REST framework choosed adequate renderer. More in exercise 3.

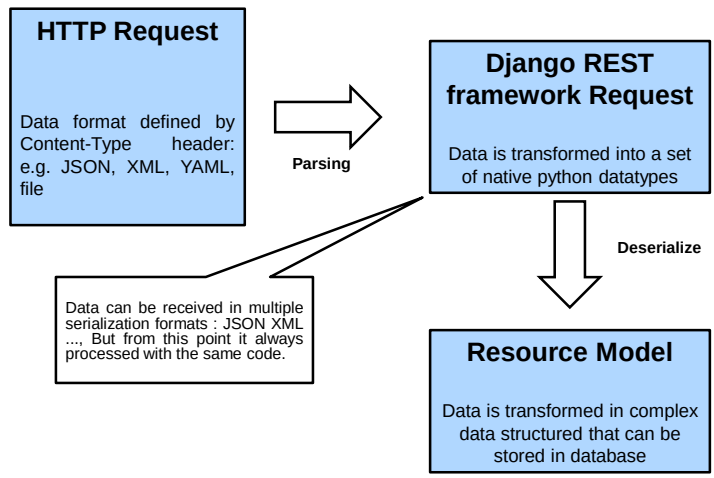
Django REST framework cycle (I)

1. When Django receives a HTTP request for a Django-rest-framework application, it is processed as follows:
2. Generates a rest-framework.Request object.
3. Store the headers in the request.META attribute.
4. Stores the query parameters of the URL in the request.QUERY_PARAMS attribute.
5. Parses the information in the HTTP request entity body and translates it into a native python structure (a dictionary). Stores this dictionary in the request.DATA attribute.
6. Consults the urls.py file to check the resource (class which extends the rest-framework.APIView) in charge of processing the request.

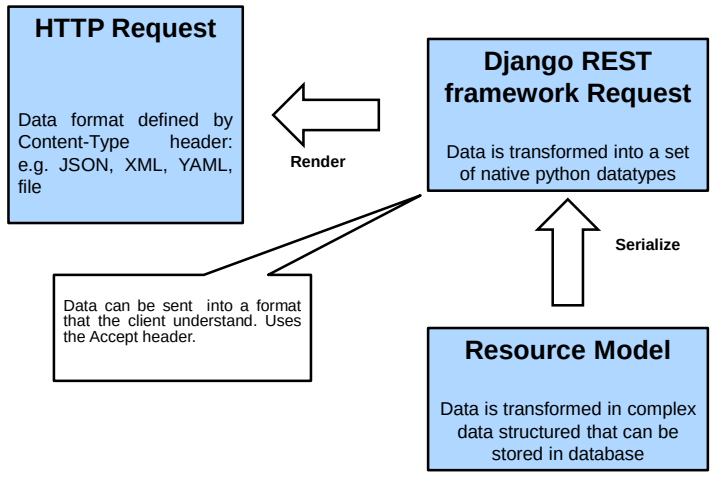
Django REST framework cycle (II)

7. Checks the HTTP method and calls the equivalent method of the class. The method receives as parameters the request object and a keyword with the values of the regular expressions variables.
8. The method deserializes the native python structure into a database module.
9. The method accesses the database and extracts necessary information.
10. The database models are serialized into a python native structure (a dictionary).
11. The method generates the response including the serialized database models, the status code and the headers.
12. Django-rest-framework renders this model to a representation that the client understands (in this application is always JSON).

Parsing/Deserialize/Serialize/ Render

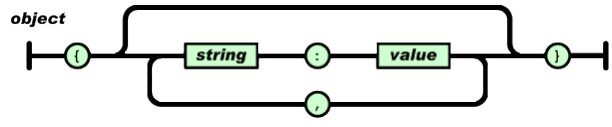


Parsing/Deserialize/Serialize/ Render

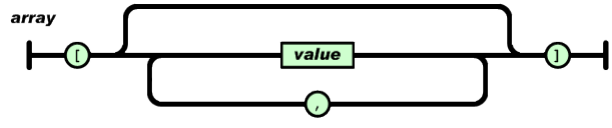


JSON

- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format.



A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes.



CHECK: http://json.org/

Some links

- Python API**:** http://docs.python.org/library/index.html
- Django documentation: https://docs.djangoproject.com/en/1.4/
  - HTTPRequest/response: https://docs.djangoproject.com/en/1.4/ref/request-response/
  - URLConf: https://docs.djangoproject.com/en/1.3/topics/http/urls/
- Django-rest-framework: http://django-rest-framework.org/
- List of HTTP status codes: http://en.wikipedia.org/wiki/List_of_HTTP_status_codes