TA Section Week 9

11/9/2018 8-9PM

Zhenyu Zhao

DevOps Engineer

Infrastructure Technology Services

Harvard University IT

Email: Zhenyu_zhao@Harvard.edu

Agenda

- ❑Midterm Questions
- ❑Django REST Framework
- ❑Demo
- ❑Q&A

Midtem Exam Questions

#25: Declarative stems

Suppose we have a python module with the following class:

```
from nltk.stem import PorterStemmer, SnowballStemmer
from nltk.tokenize import word_tokenize as nltk_tokenizer
from pset_utils import tokenizer as pset_tokenizer


class DocumentProcessor:
    def __init__(self, tokenizer, stemmer=None):
        """
        :param tokenizer: callable, tokenizer(string) returns a list of words
        :param stemmer: Stemmer class instance
        """
        self.tokenizer = tokenizer
        self.stemmer = stemmer or SnowballStemmer()

    def stem(self, tokenized):
        return [stemmer.stem(token) for token in tokens]
```

Create an instance that matches the following variable name:

```
pset_porter = DocumentProcessor(...)
```

#25: Declarative stems (cont.)

Now, write a `DocumentProcessor` subclass that can be used for declarative classes; include a child class using the NLTK tokenizer and create an instance.

```
class BaseDeclarativeDocumentProcessor(DocumentProcessor):
    ...

# an example subclass
class NLTKDocumentProcessor(BaseDeclarativeDocumentProcessor):
    ...

nltk_snowball = NLTKDocumentProcessor(...)
```

#25: Declarative stems (cont.)

- What is a declarative class in Python?
- Solution

```
# creating an instance of DocumentProcessor
pset_porter = DocumentProcessor(pset_tokenizer, PorterStemmer())

# defining declarative classes

class BaseDeclarativeDocumentProcessor(DocumentProcessor):
    # defaults
    tokenizer = pset_tokenizer

    stemmer = SnowballStemmer()

    # don't set attributes from init args
    def __init__(self):
        pass

class NLTKDocumentProcessor(BaseDeclarativeDocumentProcessor):
    tokenizer = nltk_tokenizer

# create an instance of NLTKDocumentProcessor
nltk_snowball = NLTKDocumentProcessor()
```

#26: Descriptors for backward compatibility

Let's say we have a bunch of classes that use github id's, eg:

```
class MyClass:
    def __init__(self, github_id):
        """
        :param str github_id: an id
        """
        self.github_id = git
```

We decide that plaintext github id's are a security risk and we want to use hashed id's everywhere instead. However, we want to remain backwards compatible to the extent possible.

Write a descriptor class that ensures any github id is hashed immediately before it is stored on an instance. You may assume that id's of type `bytes` (instead of `str`) are already hashed, and need no further processing. Then, show how MyClass must be modified to use the descriptor.

```
def hash_str(some_text, salt=CSCI_SALT):
    """Returns the digest of a string
    :rtype: bytes
    """
    # You may call this function without implementing it

class HashedGithubId:
    # hint: use 'if isinstance(something, bytes)' to check if already hashed
    ...

class MyClass:
    # hint: do not modify __init__
    ...
```

Question:
Can you still change the value of declarative class instance?

#26: Descriptors for backward compatibility (cont.)

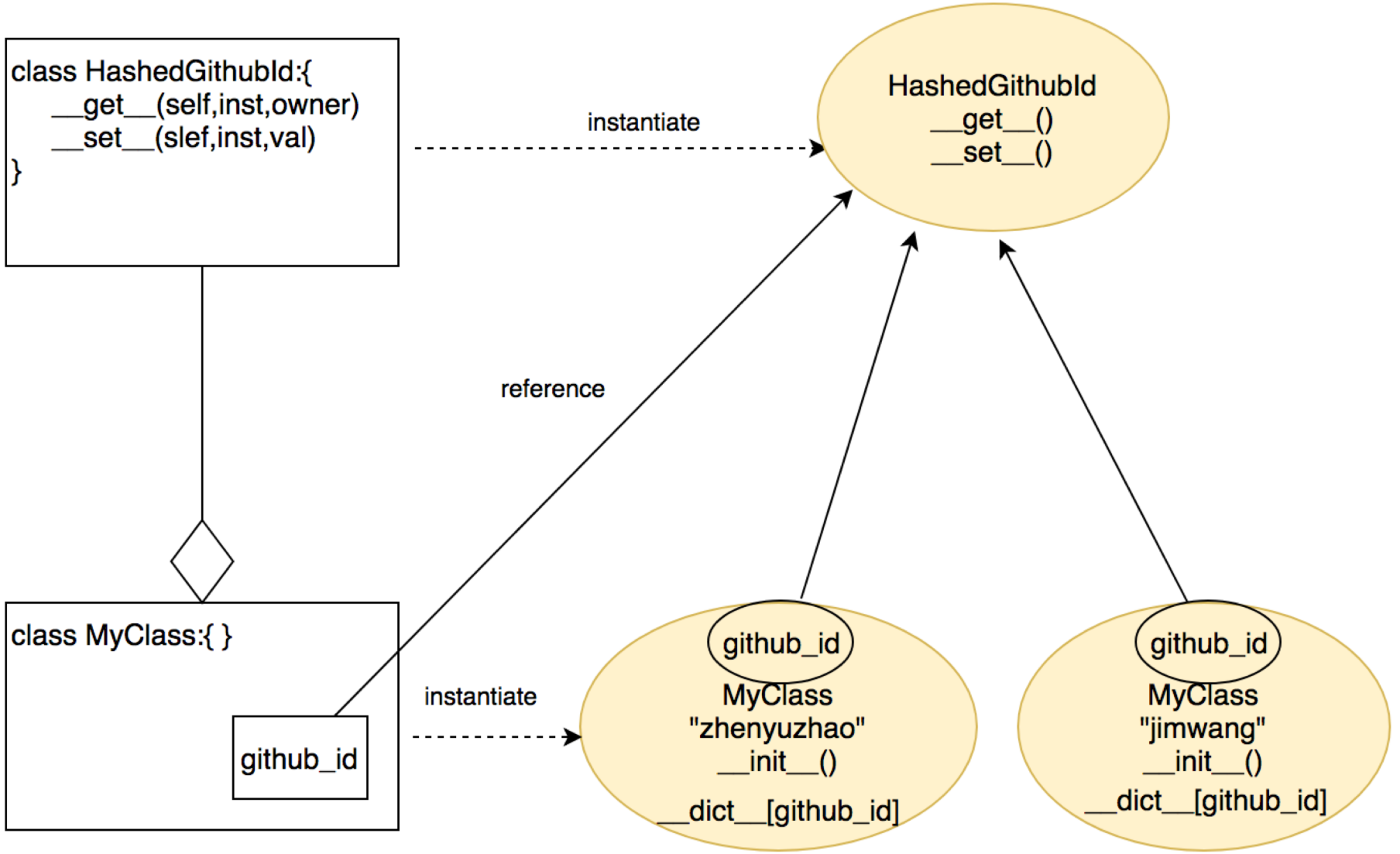
- Difference between storing attributes at instance level and at descriptor level
- LEGB scope: Local=>Enclosing=>Global=>Built-in
- Solution

```
class HashedGithubId:
    def __get__(self, instance, owner):
        return instance._github_id

    def __set__(self, instance, value):
        instance._github_id = value if isinstance(value, bytes) else hash_str(value)

class MyClass:
    github_id = HashedGithubId()

    # init and other methods would be defined below
    ...
```

```
def __init__(self, id):
    self.github_id = id
```

Question:
Which method is executed the first, __new__() or __init_() ?

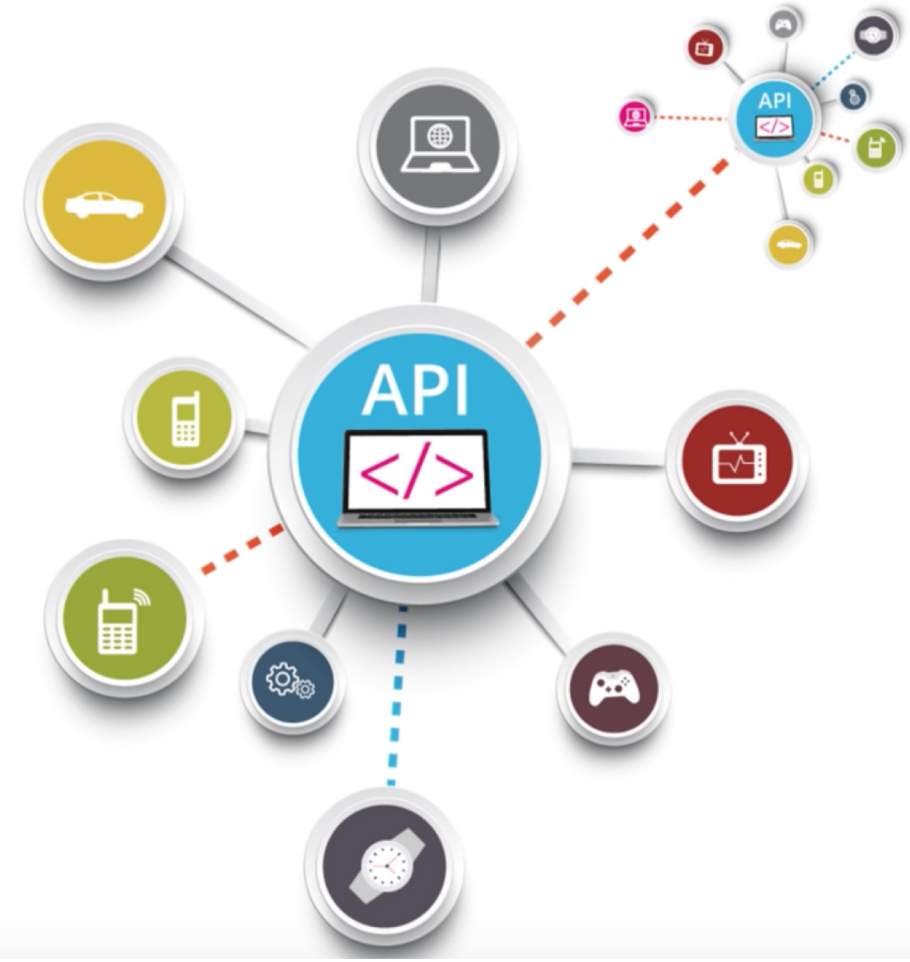
#26: Descriptors for backward compatibility (cont.)



Django REST Framework

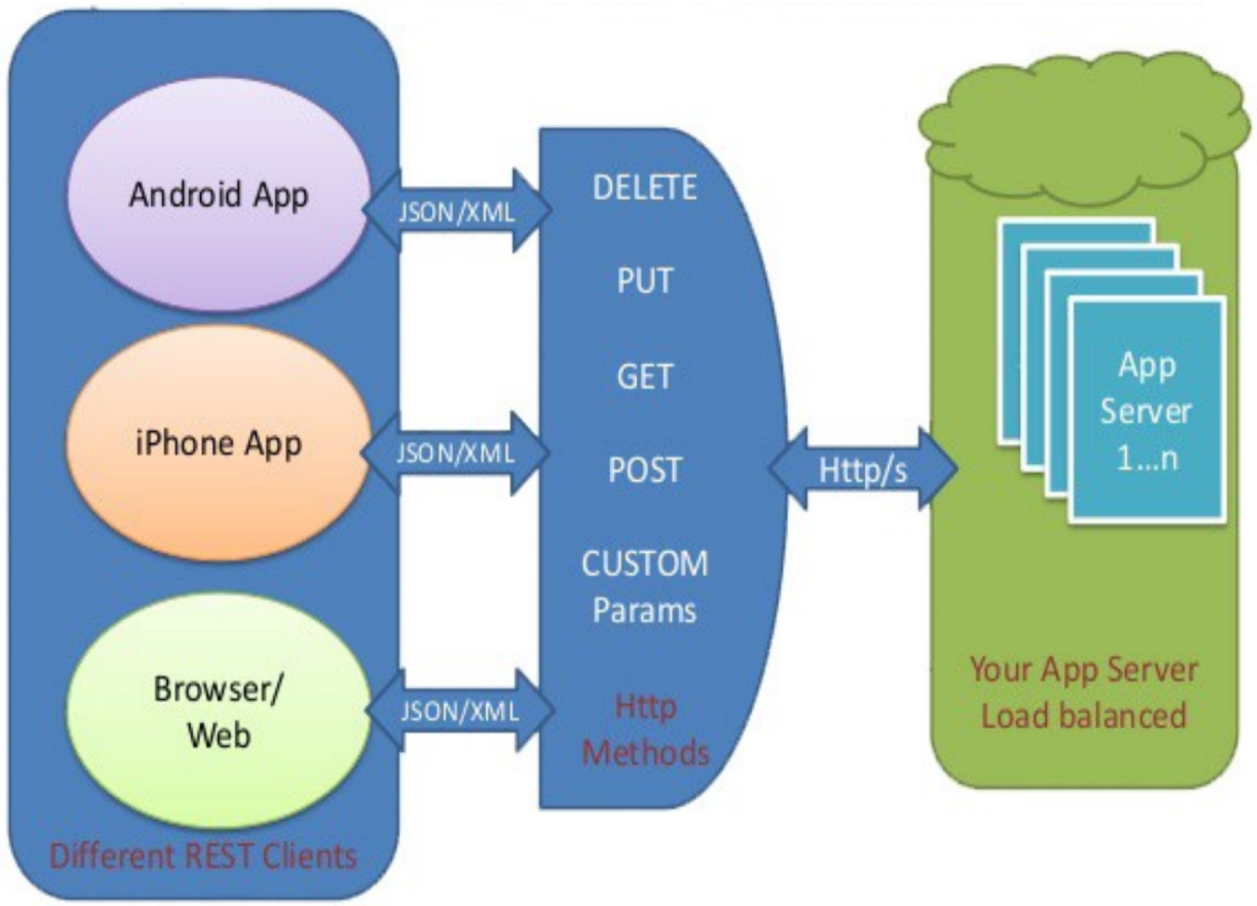
What is API?

- API stands for Application Programming Interface
- There are many types of API:
  - C/C++ header files and shared libraries at OS level
  - SOAP (Simple Object Access Protocol) and XML over HTTP
  - REST (REpresentational State Transfer) and JSON over HTTP



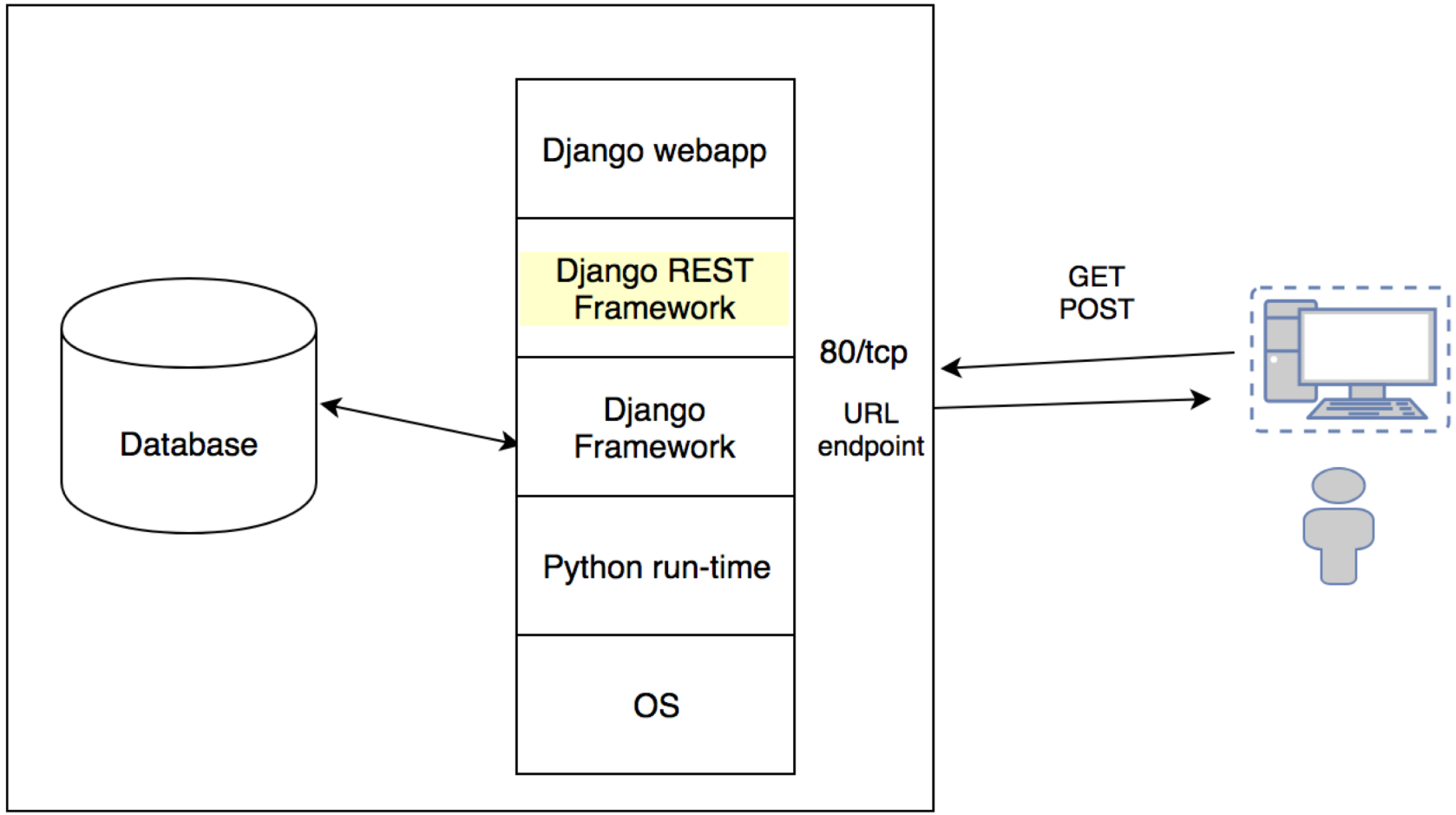
What is REST API

- REST stands for REpresentational State Transfer
- Is an architecture
- Over HTTP
- Object resources



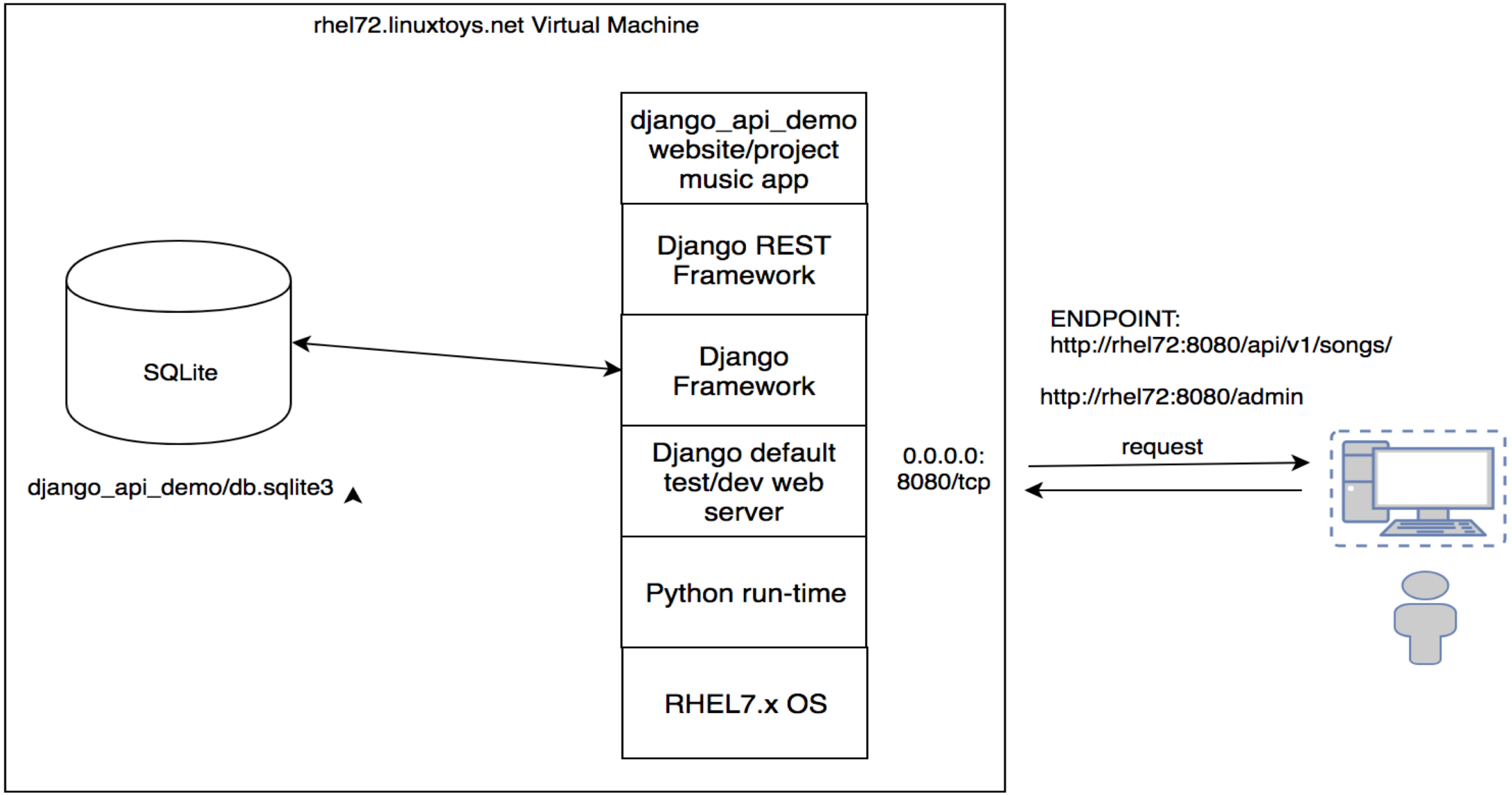
What is Django REST Framework?

- REST stands for REpresentational State Transfer
- Transmitting object resources
- Over HTTP



Demo

Architecture



File System

- **./db.sqlite3**: SQLite3 database
- **./django_api_demo/urls.py**: main URL dispatcher
- **./django_api_demo/settings.py**: site-wide config
- ./django_api_demo/wsgi.py: *C* in MVC
- **./music/*:** *web application*
- **./music/urls.py**: *application-level URL dispatcher; linked to main dispacther*
- ./music/admin.py: view for admin page
- ./music/apps.py
- ./music/views.py
- ./music/models.py:
- ./music/serializers.py

Question:
Why do people say Django conforms to MVT instead of MVC?

```
[zzhao@rhel72 django_api_demo]$ pwd
/home/zzhao/myprojects/django_demo_prj/django_api_demo
[zzhao@rhel72 django_api_demo]$ tree
.
├── db.sqlite3
├── django_api_demo
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-36.pyc
│   │   ├── settings.cpython-36.pyc
│   │   ├── urls.cpython-36.pyc
│   │   └── wsgi.cpython-36.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── manage.py
└── music
    ├── admin.py
    ├── apps.py
    ├── __init__.py
    ├── migrations
    │   ├── 0001_initial.py
    │   ├── __init__.py
    │   └── __pycache__
    │       ├── 0001_initial.cpython-36.pyc
    │       └── __init__.cpython-36.pyc
    ├── models.py
    ├── __pycache__
    │   ├── admin.cpython-36.pyc
    │   ├── __init__.cpython-36.pyc
    │   ├── models.cpython-36.pyc
    │   ├── serializers.cpython-36.pyc
    │   ├── tests.cpython-36.pyc
    │   ├── urls.cpython-36.pyc
    │   └── views.cpython-36.pyc
    ├── serializers.py
    ├── tests.py
    ├── urls.py
    └── views.py
```

Basic Django & SQLite3 Commands

❑Create a new Django site/project

$ Django-admin startupproject django_api_demo

❑Start a built-in test/dev web server

$ python manage.py startserver 0.0.0.0:8080

❑Create a new application

$ python manage.py startup music

❑Create new migration

$ python manage.py makemigrations

❑Start migration

$ python manage.py migrate

❑Connect to SQLite3 database

$ sqlite3 /path/to/mydb

❑Show databases

sqlite3> .databases

❑Show tables

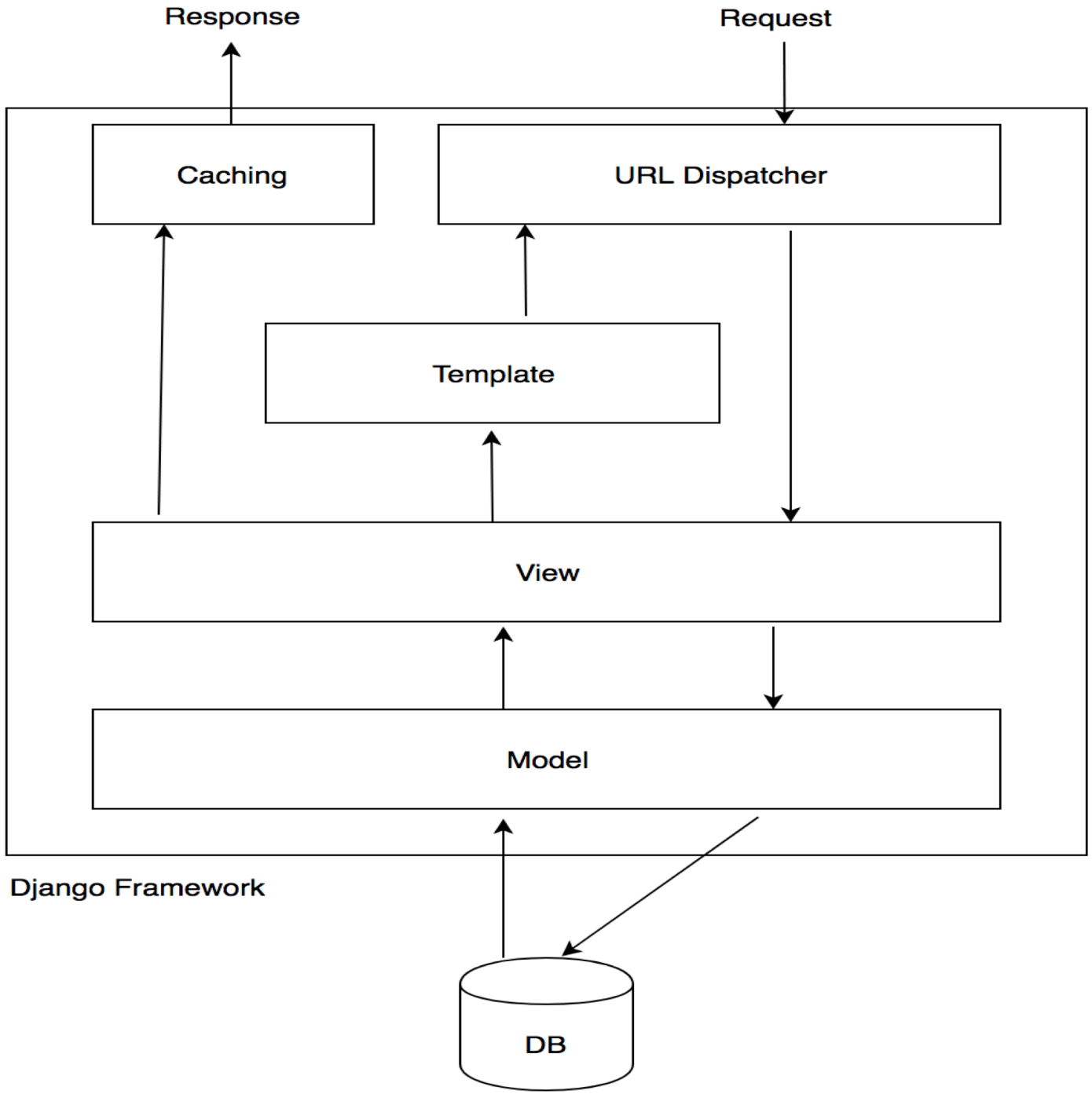
sqlite3> .tables

❑Query table

sqlite3> select * from music_songs;

❑Exit current session

sqlite3> .exit


Response
Request
Caching
URL Dispatcher
Template
View
Model
Django Framework
DB

Q&A