Intro To Django Rest Framework

Install

1. Create virtual environment: $ `python -m venv myenv`
2. Install Django: $ `pip install django`
3. Install Django Rest Framework: $ `pip install djangorestframework`
4. Create Django Project: $ `django-admin startproject your_project_name`
5. CD into your-project-name directory: $ `cd your_project_name`
    a. This will be the git repo.
    b. This is where the .gitignore will be.
    c. This is where the requirements.txt will be.
    d. Has a project directory named the same as your project where all the project config happens
6. Create Django App: $ `python manage.py startapp your_app_name`
    a. This has files and configuration that are just for this app.
    b. You can have multiple apps per project.
7. Add Rest Framework and your app to the project config.
    a. Open settings.py
    b. 'INSTALLED_APPS' list
    c. Add `rest_framework,`
    d. Add your app at the end - `your_app_name`
8. Run migrations
    a. $ `python manage.py makemigrations`
    b. $ `python manage.py migrate`

Add File To Run Python Scripts

Any python scripts that we want to run to query the database can be done by running this script.

1. Download `standalone_script.py` from the Slack 'Operations' channel
2. Add file to the your_project_name directory.
3. Rename "your_project_name" on line 6 to the name of your project
4. Run in terminal $ `python standalone_script.py`

What Is Django

Documentation: https://docs.djangoproject.com/en/5.0/intro/overview/

Django is a Python web development framework.

Key Features

Object Relational Mapping (ORM)

A layer between the developer and the database. Allows interacting with the database through Python objects (models). This abstracts away a lot of SQL complexity.

Admin Interface

Has a built in admin interface that automatically generates CRUD for models in the ORM.

Authentication

Has built in support for authentication, permissions, and groups

Templates

An HTML Templating system

Views

Business logic and controls for the templating system

What Is Django Rest Framework

Documentation: https://www.django-rest-framework.org/#

A toolkit that lets you build backend API endpoints with Django.

Key Features

Viewsets

Viewsets are a place for functions and classes that will manage API endpoints for CRUD operations and any business logic surrounding them.

Pagination

DRF includes pagination classes that enable paginated responses for large datasets.

Serialization

Has built-in ‘serializers’ for converting from database query results into JSON and vice versa.

Models

In the `your_app_name` directory, open models.py.

Each Model will have fields attached. These are the equivalent to SQL columns. So each field will have a data type. A full list of Django model field types: https://docs.djangoproject.com/en/5.0/ref/models/fields/

Workflow

Anytime you change a model you have to makemigrations and migrate.

1. $ `python manage.py makemigrations`
    a. This creates migration files - it let's Django track database schema changes
2. $ `python manage.py migrate`
    a. Applies the migration files to the database

Create Model

Creates a table. Equivalent to SQL's CREATE TABLE

```python
class Student(models.Model):
 name = models.CharField(max_length=100)
 age = models.PositiveIntegerField()
```

CRUD

CREATE: Insert Record

In standalone_script.py

1. Import models: `from your_app_name.models import *`
2. Create student: `student = Student(name="John Doe", age=27)`
3. Write to database: `student.save()`

"Save" is a method on models. We inherited this when we created Students: `Student(models.Model)`

A way with .save built in: `Student.objects.create(name="Jane Doe", age=19)`

READ: Read All Records

```python
students = Student.objects.all()
for student in students:
 print(student.name)
```

Add a dunder: __str__

In models.py on class

```python
 def __str__(self):
   return f'{self.name} - {self.age}'
```

And read

```python
students = Student.objects.all()
for student in students:
 print(student)
```

CREATE: Insert Multiple Records

```python
object_list = Student.objects.bulk_create([
       Student(name="Bob Marley", age=57),
       Student(name="Fred Murray", age=12)
])
```

READ: .get

Select a single result

```python
student = Student.objects.get(name='Bob Marley')
```

```
print (student)
```

READ: .filter

Select results matching a field lookup:
https://docs.djangoproject.com/en/5.0/ref/models/querysets/#field-lookups

- field__lt is <
- field__lte is <=
- field__gt is >
- field__gte is >=
- field__in list

```python
students = Student.objects.filter(age__lt=30)
for student in students:
 print (student)
```

UPDATE:

Change stuff and .save()

```python
student = Student.objects.filter(name='Jane Doe').first()
student.name = 'Jane Fonda'
student.save()
```

DELETE:

```python
Student.objects.filter(name='Jane Doe').first().delete()
```

Keys

Primary Key

```python
Django automatically adds id/pk to each record
students = Student.objects.filter(id__gt=3)
for student in students:
 print(student)
```

Foreign Keys

- One to one
- One to many
- Many to many

One To One

Create new class

```python
class MedicalRecord(models.Model):
 student = models.OneToOneField(Student, on_delete=models.CASCADE)
 blood_type = models.CharField(max_length=4)
  def __str__(self):
   return f'{self.student.name}: Blood Type: {self.blood_type}'
```

Note: on_delete - models.CASCADE means that this medical record will be deleted if the student record it is attached to is deleted

Save a record

```python
student = Student.objects.get(id=1)
record = MedicalRecord(student=student, blood_type='A+')
record.save()
```

Read the record

```python
print(student.medicalrecord)
```

One To Many

Create instructor table

```python
class Instructor(models.Model):
 name = models.TextField()
```

Create course table

```python
class Course(models.Model):
  name = models.TextField()
  instructor = models.ForeignKey(Instructor, on_delete=models.SET_NULL,
null=True)
```

NOTE - on_delete SET_NULL means that if the instructor gets deleted, the instructor field on the course will get be set to null

Populate instructors

```python
Instructor.objects.bulk_create([
  Instructor(name='James Bond'),
  Instructor(name='Elmo'),
  Instructor(name='Basil Rathbone')
])

instructors = Instructor.objects.all()
for i in instructors:
  print(i)
```

Populate courses

```python
Course.objects.bulk_create([
  Course(name='Math'),
```

```
  Course(name='History'),
  Course(name='Assassination')
])

courses = Course.objects.all()
for i in courses:
  print(i)
```

Assign instructor to course

```python
course = Course.objects.get(name="Assassination")
course.instructor = Instructor.objects.get(name="Elmo")
course.save()

courses = Course.objects.all()
for i in courses:
  print(i, i.instructor)
```

Assign course to instructor

```python
instructor = Instructor.objects.get(name="James Bond")
course = Course.objects.get(name="Math")
course2 = Course.objects.get(name="History")
instructor.course_set.add(course, course2)

print(instructor.name, instructor.course_set.all())
```

Many To Many

Add courses many to many connection to Student

```python
class Student(models.Model):
  name = models.CharField(max_length=100)
  age = models.PositiveIntegerField()
  courses = models.ManyToManyField(Course)
```

This adds an implicit student_set field to Course

Assign course to student

```python
assassination = Course.objects.get(name='Assassination')
math = Course.objects.get(name='Math')
history = Course.objects.get(name='History')

students = Student.objects.all()

assassination.student_set.add(*students[0:2])
math.student_set.add(*students[3:4])
history.student_set.add(*students)

students = Student.objects.all()
for stud in students:
  print(stud, stud.courses)
```