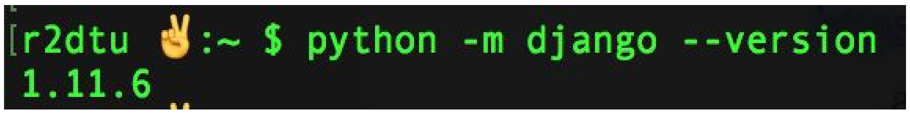
HackXX Website Development Tutorial

By: David Tu & Jerry (Yihui) Yang
IEEE Technical Chairs 2017-2018

Pre-requisites:

1) Install Python.
2) Install `pip`: https://pip.pypa.io/en/latest/installing/#installing-with-get-pip-py.
    a) `curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py`
        i) For Windows users, download: https://bootstrap.pypa.io/get-pip.py
    b) `python get-pip.py`
    c) Upgrading on Mac:
        i) `pip install -U pip`
    d) Upgrading on Windows:
        i) `python -m pip install -U pip`
3) Install Django: https://docs.djangoproject.com/en/2.0/topics/install/#installing-official-release.
    a) Mac:
        i) Check if already installed:
            (1)
            ```
            ~$ python
            >>> import django
            >>> print(django.get_version())
            ```
            or
            ```
            python -m django --version
            ```

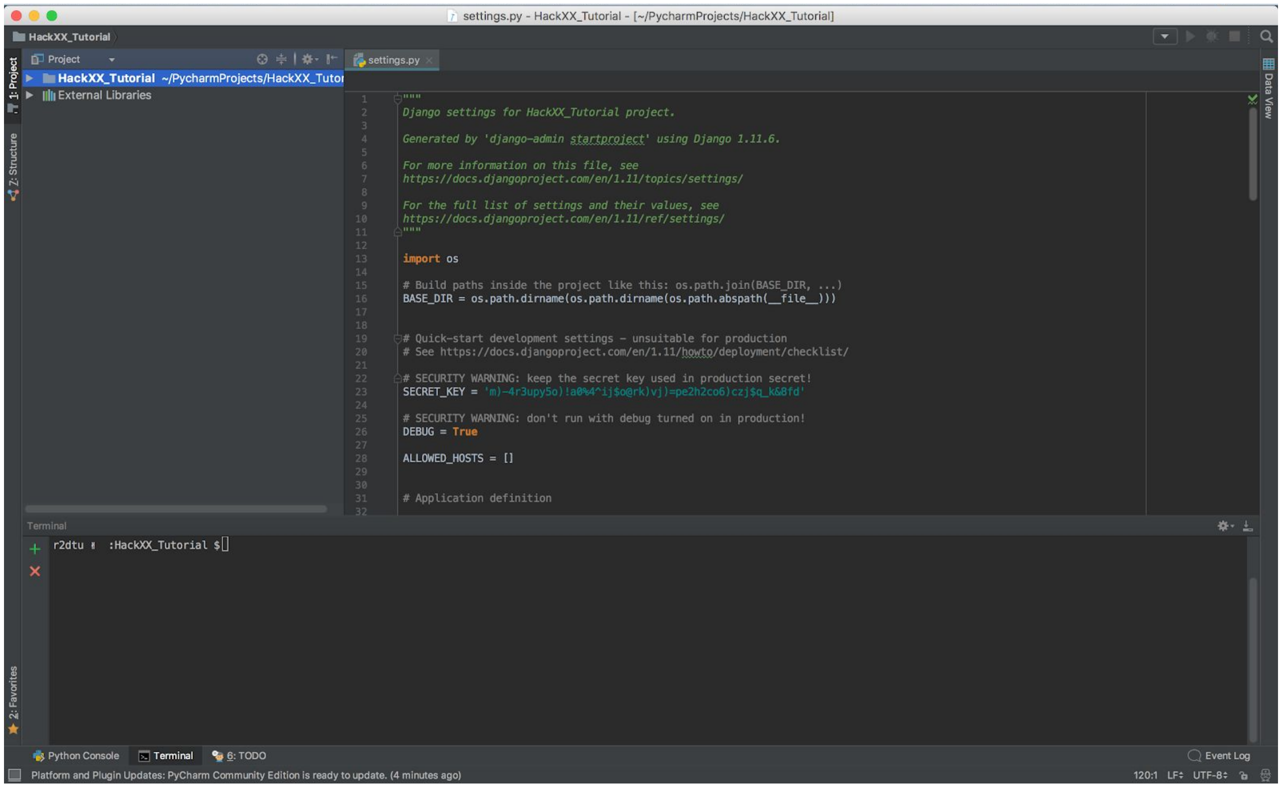


```
[r2dtu ✌:~ $ python -m django --version
1.11.6
```

        ii) If not, `pip install django`
    b) Windows: https://docs.djangoproject.com/en/2.0/howto/windows/ (virtualenv not required)
4) Install PycharmCE (*optional, but highly recommended*).
    a) https://www.jetbrains.com/pycharm/download/
5) Now you are ready for web development!

Getting Started with your First Django Website

1) You can find the entire code here: https://github.com/r2dtu/HackXX_WebDev_Tutorial
2) Open up Pycharm, or your favorite text editor. Make sure you have a terminal / command-line open as well!
3) Install Django and requests libraries:
```
pip install django
pip install requests
```
4) Let's start! Type in `django-admin startproject HackXX_Tutorial`
    a) `manage.py`: A command-line utility for us to interact with Django applications.
    b) `HackXX_Tutorial/__init__.py`: This directory is a Python package.
    c) `HackXX_Tutorial/settings.py`: A file consisting of key-value pairs for configuring your Django application. https://docs.djangoproject.com/en/1.8/topics/settings/.
    d) `HackXX_Tutorial/urls.py`: URL declarations, like a "table of contents". https://docs.djangoproject.com/en/1.8/topics/http/urls/.
    e) `HackXX_Tutorial/wsgi.py`: Not necessary for our demonstration. You can read more about it here: https://docs.djangoproject.com/en/1.8/howto/deployment/wsgi/.

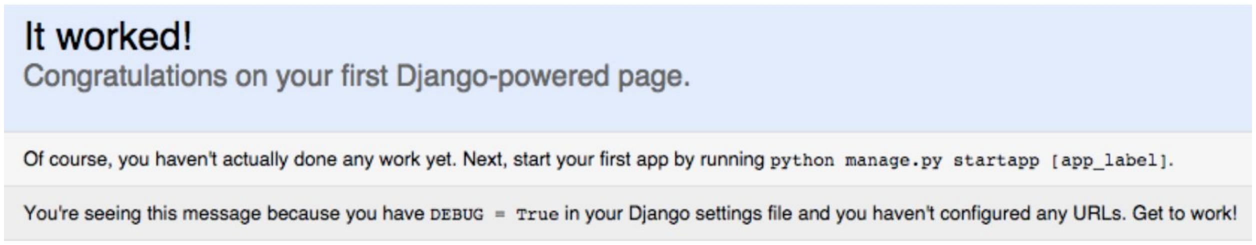


```python
"""
Django settings for HackXX_Tutorial project.

Generated by 'django-admin startproject' using Django 1.11.6.

For more information on this file, see
https://docs.djangoproject.com/en/1.11/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.11/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.11/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'm)-4r3upy5o)!a0%4^ij$o@rk)vj)=pe2h2co6)czj$q_k&8fd'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition
```

```
r2dtu  :HackXX_Tutorial $
```

Platform and Plugin Updates: PyCharm Community Edition is ready to update. (4 minutes ago)

5) Let's start the web server. `python manage.py runserver`
6) You'll see a bunch of weird errors. `Ctrl + C`
7) Run the migration tool. `python manage.py migrate`
8) Start it up again! `python manage.py runserver`
9) Visit http://127.0.0.1:8000/. You should see something like the following:



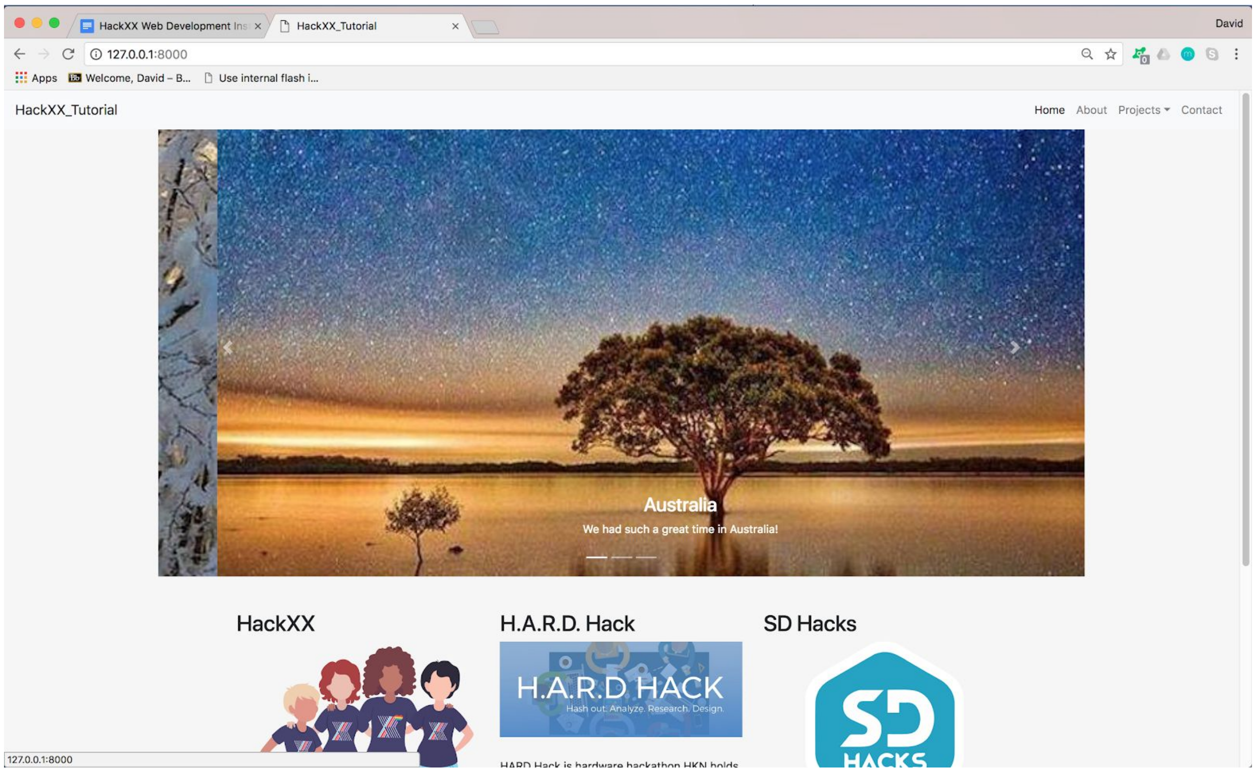
It worked!
Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [app_label]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

Whoo! We have a website! Not much, but it's a start. Alrighty, before we make an application, let's make our site look a little prettier first.

Django Templating



First, a little background and overview. The Django framework utilizes *apps* to help make the site modular. Apps are simply Web applications that do something. Each app has its own set of distinct features and functionality. You can access them via certain URLs, as you will see later on. The first thing to do is generate a basic web "app". This will hold all of our basic apps that do not require special back-end rendering.

1) Get the starter app package from Django. `python manage.py startapp web`
2) Open up `settings.py`. Your `INSTALLED_APPS` variable should look like this.
    a)
```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'web',
)
```

3) Now open up `HackXX_Tutorial/urls.py`:
    a) You'll see a list called `urlpatterns`. Add the following to it:

```
from django.conf.urls import url
url(r'^projects/', views.projects),
url(r'^about/', views.about),
url(r'^$', views.home),
```

    b) What do those mean?
        - When you visit http://127.0.0.1/, it will use the ^$ URL and open up the home page. Visiting http://127.0.0.1/about will open up the about page (render `about.html`), and so on. The Python code for this is in the next step.

4) Now open up `web/views.py`:
    a)

```
from django.http import HttpResponse
from django.template import loader
from django.shortcuts import render

def home(request):
    template = loader.get_template('home.html')
    context = {

    }
    return HttpResponse(template.render(context, request))


def about(request):
    template = loader.get_template('about.html')
    context = {

    }
    return HttpResponse(template.render(context, request))


def projects(request):
    template = loader.get_template('projects.html')
    context = {

    }
    return HttpResponse(template.render(context, request))
```

Sweet! Now, let's do the front-end development.

1) Type these into your terminal, one-by-one. You can create these folders manually, if necessary.

```
mkdir static
mkdir static/css
mkdir static/js
mkdir templates
cd templates
```

a) Let's create our first HTML file. `touch header.html` (you can simply create a new file if you're using an IDE). You can find the HTML code here: https://github.com/r2dtu/HackXX_WebDev_Tutorial.

   What's the purpose of these: {% %}? Well, this helps lessen the clutter (copy/paste errors, too) when we're writing our different HTML files. We're using the same CSS and JS files, so why not just have them in one file we can use to import into others. What? *Import into others*? You'll see. It's Jinja.

Download the images here (download as ZIP, and extract the images), and put them in `web/static/images/`.

What does this all mean?

- `<!DOCTYPE html>` signifies that this document contains HTML code.
- `{% load staticfiles %}`. Duh heck? More on this a bit later.
- `<link rel … >`, `<script src … >`: "imports" (if you will) the JS and CSS we need to render our HTML page.
- `{% block content %}`, `{% endblock %}`, `{% static 'images/...svg' %}`. Okay seriously, what are these? This is Jinja, a Python template engine. You'll see more of this later. The static files are specified by the static directory defined in `settings.py`. We will use `block content` and `endblock` in our other HTML files.

b) Create new file called home.html. `touch home.html`.
   - A ha! There it is. The lovely Jinja. This is how we can "import" `header.html`. We can also reference files in the static directory with `{% static`.

Pretty neat?

c) This one is up for you to customize. Just a template. Again, create a file called `about.html`, or `touch about.html`.
d) Okay. Let's put the CSS in. This you can definitely copy-paste.

```
cd static/css
touch main.css
```

Now go to the GitHub and copy and paste that CSS file.

Checkpoint:

You should have at least the following files, in the correct directories:

```
- HackXX_Tutorial/
    - HackXX_Tutorial/
        - settings.py
        - urls.py
        - wsgi.py
    - web/
        - static/
            - css/
                - main.css
            - images/
                - (a bunch of images)
        - templates/
            - about.html
            - header.html
            - home.html
        - views.py
```

Let's build an app, and connect it to our website!

Let's take advantage of Django's apps. We can make our own! We will query YouTube channel data, and get the most recent 5 videos from it and display it nicely on a table.

1) Start a new app: `python manage.py startapp youtubeapp`
    a) `youtubeapp/admin.py`: Allows us to register our models to view on the Django Admin Site.
    b) `youtubeapp/migrations.py`: Keeps track of our migrations which are essentially Django’s way of propagating changes that are made to our models.
    c) `youtubeapp/models.py`: Stores our representations of objects which will be stored in our database.
    d) `youtubeapp/tests.py`: Our unit tests will live here. Although Django calls these tests “unit tests”, they’re actually closer to integration tests.

    e) `youtubeapp/views.py`: All of our views will live here, which are mapped to URLs.

2) Open up `settings.py`. Add to your `INSTALLED_APPS`: '`youtubeapp`'
3) Let's render our HTML page. Open up `youtubeapp/views.py`:
    a)

```
# views.py
from django.shortcuts import render, HttpResponse

# Create your views here.
def index(request):
    return HttpResponse('Hello World!')
```

4) Open up `HackXX_Tutorial/urls.py`:
    a) Add the new YouTube App URL path to the file.

```
# HackXX_Tutorial/urls.py

from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    ...
    url(r'^youtubeapp/', include('youtubeapp.urls')),
]
```

5) Create a new `urls.py` file in `youtubeapp/`
    a)

```
# youtubeapp/urls.py
from django.conf.urls import url
from app import views

urlpatterns = [
    url(r'^$', views.index),
]
```

Now, when you call http://127.0.0.1:8000/youtubeapp/, it'll render the index.

6) Now, we use the YouTube API:
    a) `python`

```
    >>> import requests
    >>> req =
requests.get('https://www.googleapis.com/youtube/v3/playlis
tItems?part=snippet&playlistId=UUbW18JZRgko_mOGm5er8Yzg&key
=AIzaSyA_dZuW2zI5FXBM63OsE7DIctXwhMAib3o')
    >>> print req.content
```

```
{
'kind': 'youtube#playlistItemListResponse',
'etag': '"RmznBCICv9YtgWaaa_nWDIH1_GM/xbMfmbgsu40Oj_BIiL9HQssiyn0"',
'nextPageToken': 'CAUQAA',
'pageInfo': {'totalResults': 171, 'resultsPerPage': 5},
'items': [
    {
        'kind': 'youtube#playlistItem',
        'etag': '"RmznBCICv9YtgWaaa_nWDIH1_GM/qKI5Yg1Vl2i2-2N6r0fbQoupdS0"',
        'id': 'VVViVzE4SlpSZ2tvX21PR201ZXI4WXpnLlRId1hfbEUtbElJ',
        'snippet': {
            'publishedAt': '2018-01-30T09:35:59.000Z', 'channelId':
'UCbW18JZRgko_mOGm5er8Yzg',
            'title': 'One Direction - One Way or Another (Teenage Kicks)
(Live at the BRIT Awards 2013)',
            'description': '..........................',
            'thumbnails': {
                'default': {
                    'url': 'https://i.ytimg.com/vi/THwX_lE-lII/default.jpg',
                    'width': 120, 'height': 90
                },
                'medium': {'url':
'https://i.ytimg.com/vi/THwX_lE-lII/mqdefault.jpg', 'width': 320, 'height':
180},
                'high': {'url':
'https://i.ytimg.com/vi/THwX_lE-lII/hqdefault.jpg', 'width': 480, 'height':
360},
                'standard': {'url':
'https://i.ytimg.com/vi/THwX_lE-lII/sddefault.jpg', 'width': 640, 'height':
480},
                'maxres': {'url':
'https://i.ytimg.com/vi/THwX_lE-lII/maxresdefault.jpg', 'width': 1280,
'height': 720}
            },
            'channelTitle': 'OneDirectionVEVO',
            'playlistId': 'UUbW18JZRgko_mOGm5er8Yzg',
            'position': 0,
            'resourceId': {'kind': 'youtube#video', 'videoId':
'THwX_lE-lII'}
        }
    }
    ...
```

b) Add this to `youtubeapp/urls.py`:

```
url(r'^videos/$', views.videos)
```

So visiting http://127.0.0.1:8000/youtubeapp/videos will render the page by calling videos(), which is defined in the next step.

c) Add this to `youtubeapp/views.py`:

```
def videos(request):
    API_KEY = "AIzaSyA_dZuW2zI5FXBM63OsE7DIctXwhMAib3o"
    jsonList = []
    req =
requests.get('https://www.googleapis.com/youtube/v3/channels?part=contentDet
ails,statistics&forUsername=Numberphile&key=' + API_KEY)
    jsonList.append(json.loads(req.content))
    userData = []
    for data in jsonList:
        for video in data['items']:
            info = {}
            info['username'] = username
            info['publishedAt'] = video['snippet']['publishedAt']
            info['title'] = video['snippet']['title']
            info['videoId'] = video['snippet']['resourceId']['videoId']
            info['channelId'] = video['snippet']['channelId']
            userData.append(info)

    return HttpResponse(userData)
```

7) Let's make the site look nice:

a) Go to `youtubeapp/` directory.

```
mkdir templates
cd templates
touch videos.html
```

b) Go to the GitHub, find `videos.html`, and copy and paste that code.

c) See that Jinja? Pretty neat, huh? It's taking our dictionary data that will be passed into `render` and format it as a table. Looks just like Python code (for-each loop)!

d) Now let's fix the videos so it does what we want it to (as specified above):

```
def videos(request):
    API_KEY = "AIzaSyA_dZuW2zI5FXBM63OsE7DIctXwhMAib3o"
    userData = []
    if request.method == 'POST':
        username = request.POST.get('user')
        response = 
requests.get('https://www.googleapis.com/youtube/v3/channels?part=contentDet
ails,statistics&forUsername=' + username + '&key=' + API_KEY)
        req = json.loads(response.content)

        if 'items' not in req or len(req['items']) < 1:
            ERROR = 'Could not acquire data.'
            info = {}
            info['username'] = username
            info['publishedAt'] = ERROR
            info['title'] = ERROR
            info['videoId'] = ''
            info['channelId'] = ERROR
            userData.append(info)
            return render(request, 'videos.html', {'data': userData})

        playlistId = 
req['items'][0]['contentDetails']['relatedPlaylists']['uploads']
        followers = req['items'][0]['statistics']['subscriberCount']
        videos = req['items'][0]['statistics']['videoCount']
        req = 
requests.get('https://www.googleapis.com/youtube/v3/playlistItems?part=snipp
et&playlistId=' + playlistId + '&key=' + API_KEY)
        jsonList = []
        jsonList.append(json.loads(req.content))
        while True:

            for data in jsonList:
                for video in data['items']:
                    info = {}
                    info['username'] = username
                    info['publishedAt'] = video['snippet']['publishedAt']
                    info['title'] = video['snippet']['title']
                    info['videoId'] = 
video['snippet']['resourceId']['videoId']
                    info['channelId'] = video['snippet']['channelId']
                    info['thumbnail'] = 
video['snippet']['thumbnails']['default']['url'] # also medium, high, 
standard, maxres

                    userData.append(info)

            if "nextPageToken" in data:
                nextPageToken = data['nextPageToken']
                req = requests.get(
'https://www.googleapis.com/youtube/v3/playlistItems?part=snippet&pageToken=
' + nextPageToken + '&playlistId=' + playlistId + '&key=' + API_KEY)
                jsonList = []
                jsonList.append(json.loads(req.content))
            else:
                break
    return render(request, 'videos.html', {'data': userData})
```

Essentially, the HTML code for this page made a POST request to send the data to the back-end. The Django framework recognizes the web URL and calls the corresponding method. Then the function grabs the information it needs from the POST request, sends it over the YouTube API HTTP request, and parses the response. See the dictionary being passed into `render`? That's what our Jinja from `videos.html` will read from.

8) If we have time, we'll do the contact form!
    a) pip install widget_tweaks
    b) *To be continued...*

That's all folks! We hope you learned something about web development with us. Happy hacking!