Build a Web App with Django

Slides & Code:

https://github.com/noah-dev/pykc_django_talk

What we will cover

What is a Website Application (Web App)? Why build a Web App?

What tools are available? Why use Python & Django?

Building a basic Todo web app with Django:

- Understand Django's built-in utilities to setup the structure
- Setup the Routes, Views, and Model
- Create HTML Templates to use, including styling
- Process POST requests for user input

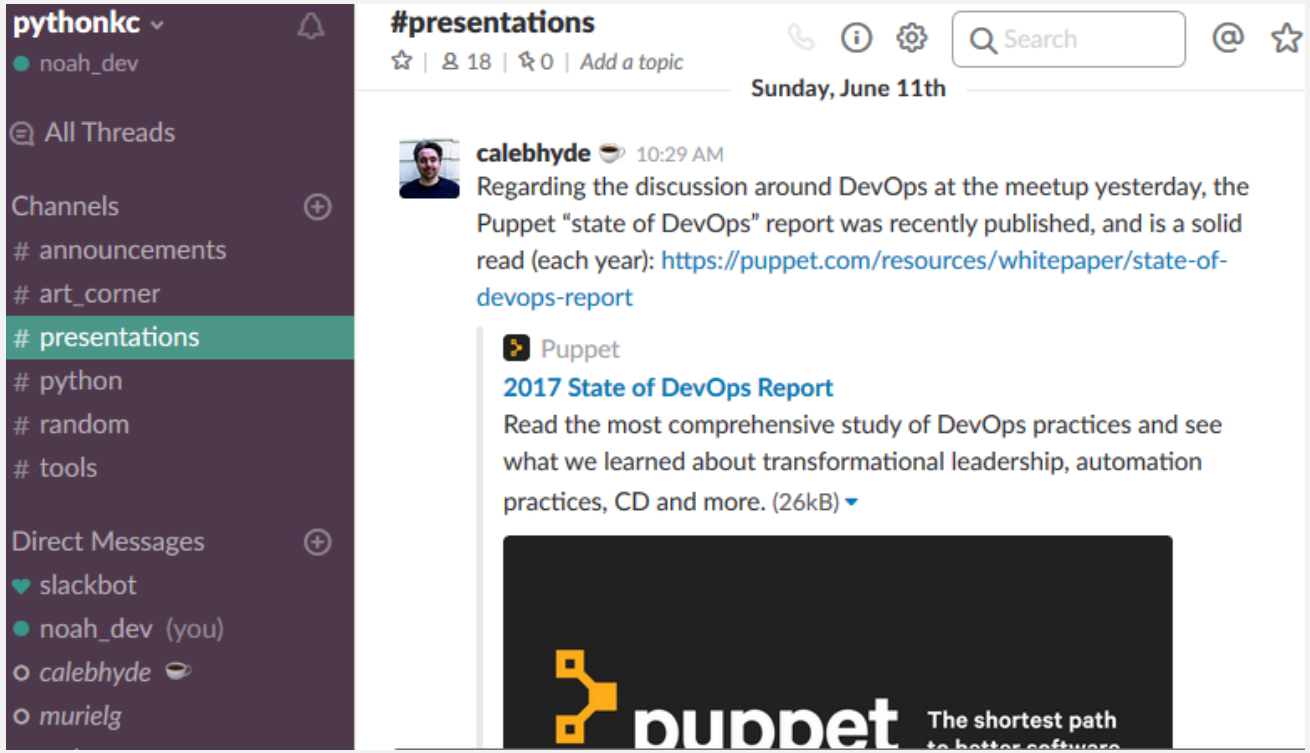
Crash Course

- We will be hitting these topics quickly at a high level
- If you are interested, see resources at end of presentation
- Feel free to reach out to me on Slack

What is a Web App?

Like Slack, Web Apps are:

- Accessible with a Browser
- Accepts user input
- Processes user input
- Save to a database
- Provide dynamic, user-specific content



Why Build A Web App?

Advantages (Short-list)

- Easy to use & share! Only needs a browser
- Already cross platform, welcoming OS X, PC, Linux, iOS, Android & more!
- Accelerate development with great tools, cloud architecture, and one primary development version!*

Disadvantages (Short-list)

- Usually needs internet to start & run app
- Need to request user permission for certain features.

* Some features may not be supported on all browsers

What Tools are Available?

Python (Short-list)

- Flask – quick turn-around for small apps
- Dash – offers great looking front-end for data driven & analytical apps.
- Django – offers built-in utilities, boilerplate code & admin panel.

Other Tools (Non-exhaustive)

- Ruby on Rails (Ruby), Spring (Java), Play (Scala / Java), NodeJS (JavaScript), ASP.NET (C#), Drupal (CMS), Laravel (PhP), and more

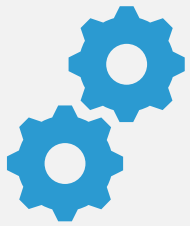
Why use Python & Django?

Why Python?

- Python has great syntax. Good code can often be self-documenting
- Great community. Many obstacles can be overcome with Google

Why Django?

- Offers robust structure that can support small & big apps
- Built-in utilities do the heavy lifting, such as Database Migration
- Pre-built code, such as Admin panel, provides great features quickly



Front-End & Back-End

Broadly speaking, a web app has a two main parts:

- Front-End: What the user sees and is able to interact with
- Back-End: What the user cannot see, and does most business logic

Front-End

Web Page
What user interacts with

Back-End

Server Code
Combines user input with other info to deliver result

Database
Store and retrieve data

Prerequisites

You should have some basic experience with:

- Python
- HTML & CSS
- Using terminal commands (CLI)

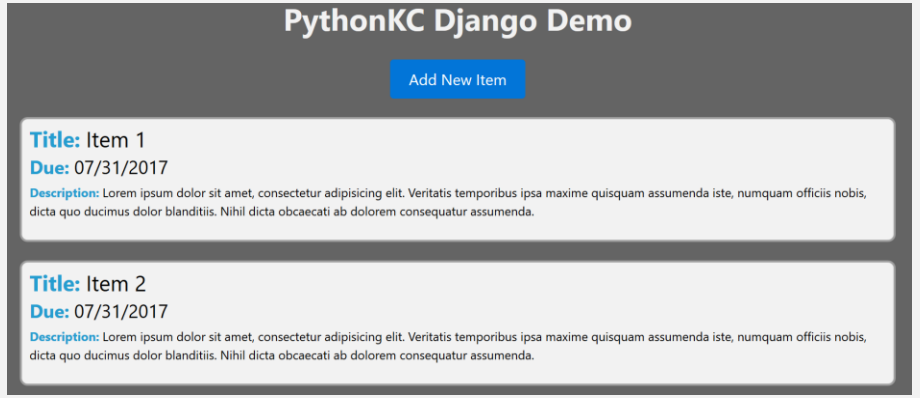
Make sure to install Django:

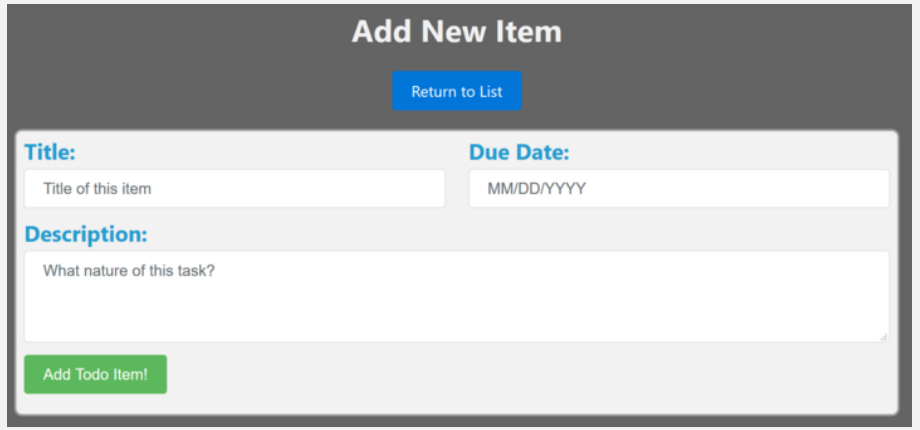
- pip install django

Let's Build!

Agenda: 5 Steps

- Create the structure for app
- Basic routes and view in the app
- Make model and migrate to the database
- Create HTML templates, including styling
- Update the view to handle user input





Create the Structure

Create the needed Folders & Files

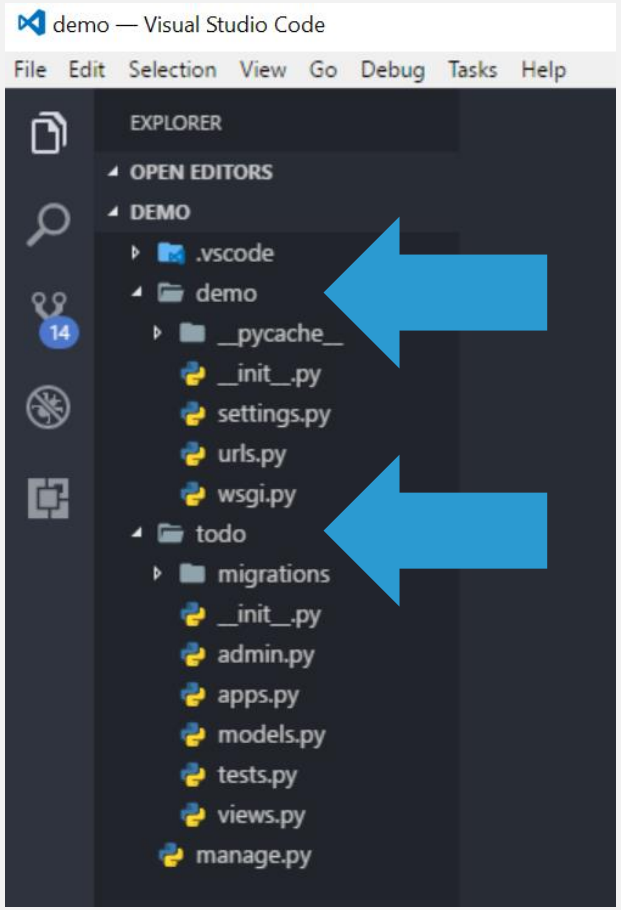
- **django-admin startproject demo**
- **python manage.py startapp todo**

The *demo* folder will contain the main project

The *todo* folder will contain the app, managed by main project.

- Run server: **python manage.py runserver**

In browser, navigate to:
localhost:8000



Settings.py

Holds key configurations for the project

- Include the todo app within the project
- Set time zone to America/Chicago (Central Time)

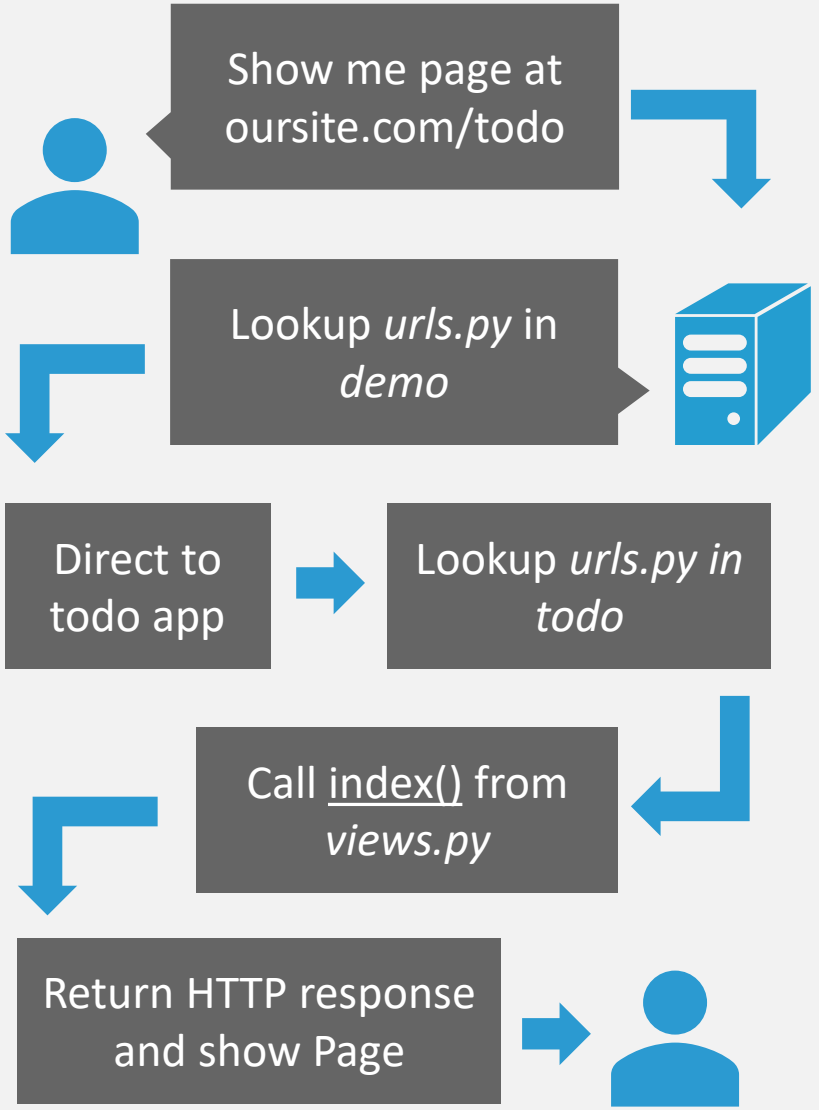
```
INSTALLED_APPS = [
    'todo.apps.TodoConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

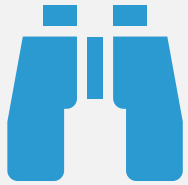
```
LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'America/Chicago'

USE_I18N = True

USE_L10N = True

USE_TZ = True
```

Routes

Which page to load at which URL? Routes!

- *urls.py* in *demo* and *urls.py in todo*
- Direct requested URL to function in *views.py*
- Call a function to provide HTTP Response



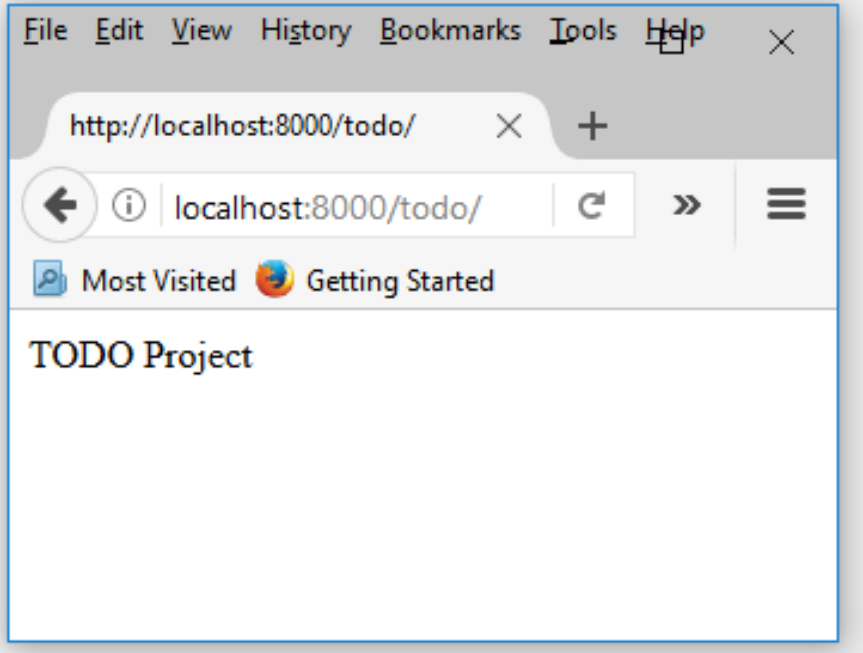


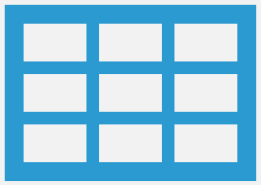
Views

views.py determines content served to user

- Returns an HTTP response which is interpreted by the user's browser
- Many different ways to return, ranging from simple text, HTML template, to JSON.
- Can also accept user input via GET/POST request

```
from django.shortcuts import HttpResponse

def index(request):
    return HttpResponse("TODO Project")
```





Database – What is a model?

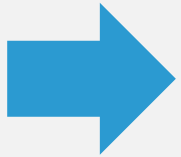
Let’s imagine how the an item would look in our Todo app

- How do we represent something like this in the database?

Title: Get Milk

Due: 8/27

Description:
I’m almost out of milk. I don’t want to eat cereal with water.



Title	Due	Description
Get Milk	8/27	I’m almost out of milk…
Learn new framework	9/4	Learn new tools and build….
Prepare for PyKC Talk	8/25	Need to prep & practice

Model

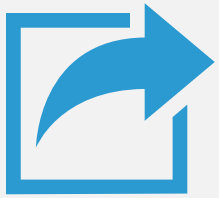
models.py defines the structure, or schema, of table(s) in the database.

- Defines how many fields, what kind of fields, and default value.
- Also defines methods, like magic methods such as __str__

models.py

```python
from django.db import models

class Item(models.Model):
    '''Represents an item on a todo list'''

    title_text = models.CharField(max_length=200)
    desc_text = models.TextField()
    due_date = models.DateTimeField()
    complete = models.BooleanField()
    add_date = models.DateTimeField()

    def __str__(self):
        return self.title_text
```

```python
title_text = models.CharField(max_length=200)
desc_text = models.TextField()
due_date = models.DateTimeField()
complete = models.BooleanField()
add_date = models.DateTimeField()
```



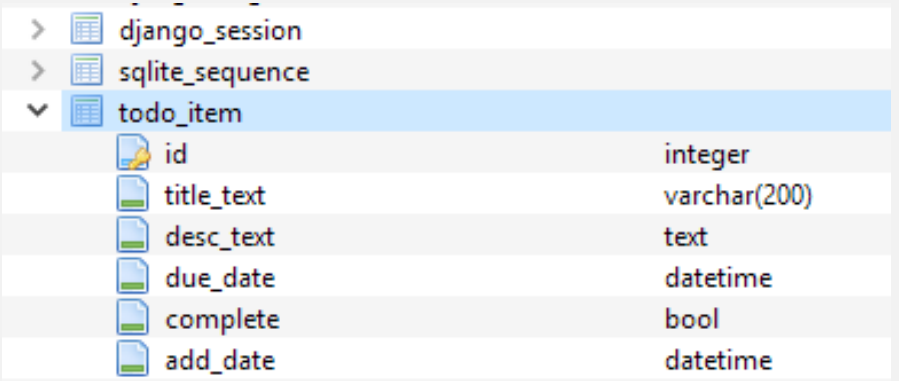
Migrate to Database

Update the database accept new items

- **python manage.py makemigrations todo**
- **python manage.py migrate**

We'll also create a superuser (admin)

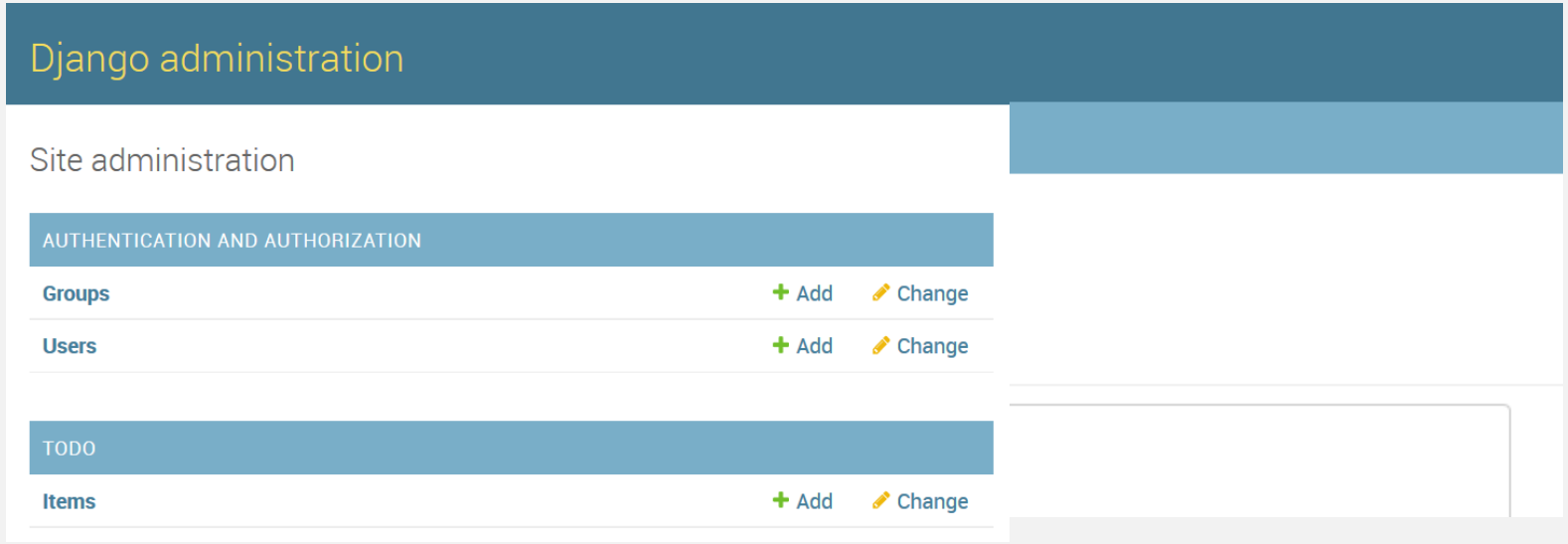
- **python manage.py createsuperuser**
- Enter username & password



Admin Time!

With the database ready, lets add some items to it

- Add code to *admin.py* to enable in admin view
- Using superuser account, log in at localhost:8000/admin



</> HTML Templates

Lets add some structure with HTML Templates

- Double curly braces signify variables (e.g. {{ title }})
- Single curly with percent signify logic (e.g. {% for item in items %})

views.py | base.html | index.html

```
from django.shortcuts import render
from .models import Item

def index(request):
    '''index will populate the page with items from the database'''

    items = Item.objects.order_by('due_date')

    return render(request, 'todo/index.html', {'items': items})
</div>
{% endblock %}
```

Styling

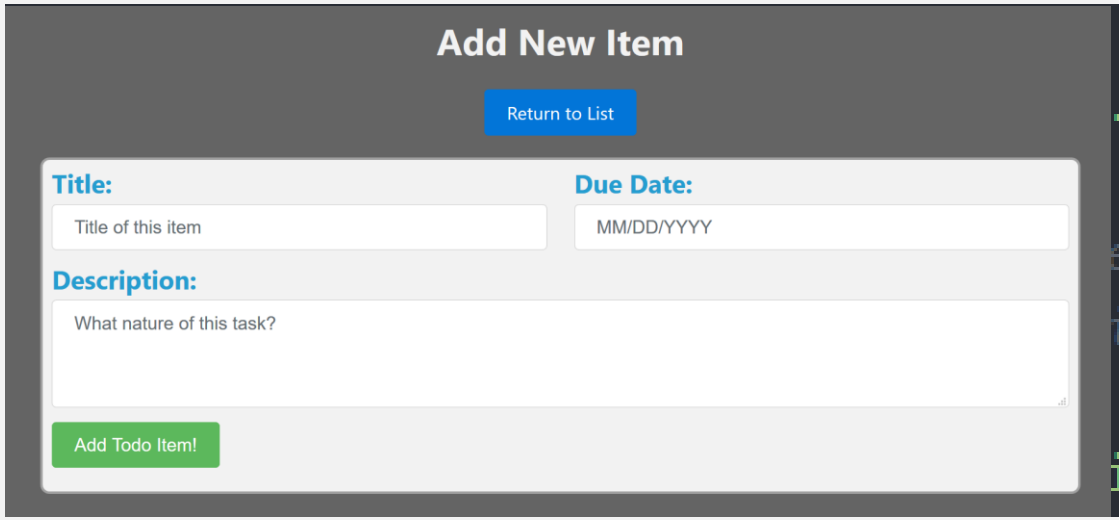
Let’s add some styling to polish it.

- Django stores static assets separate from templates
- To add to HTML template, use { % load static % } block

User Input (POST request)

To accept user input, we'll use an HTTP POST Request

- Create a form in HTML template, set to POST
- Create code in views.py to accept & process the POST request




```
.html')

eate new record.
is not valid
                    g" name = "title"

itle']
```

Lets Recap

- We setup the app using django-admin
- We created routes & views to display a basic page
- We set a model and which setup the table & fields in the database
- We migrated the database, setup the admin, and added items
- We setup the view to show the items, and added styling
- We enabled user input by adding front-end mark-up & back-end code

Next Steps

Consider taking a look at tutorials and official docs

- See resources slide

The “complete” field is practically unused

- Add feature to let user mark items as complete

Items can only be displayed

- Allow for items to be deleted or modified

Resources (1/2)

Official Django Tutorial

- https://docs.djangoproject.com/en/1.11/intro/tutorial01/

Django Girls Tutorial:

- https://tutorial.djangogirls.org/en/

YouTube Tutorial By *thenewboston*

- https://www.youtube.com/watch?v=qgGIqRFvFFk&list=PL6gx4Cwl9DGBlmzzFcLgDhKTTfNLfX1IK

Resources (2/2)

Django Docs:

- https://docs.djangoproject.com/en/1.11/

Deploying to Heroku

- https://devcenter.heroku.com/articles/deploying-python

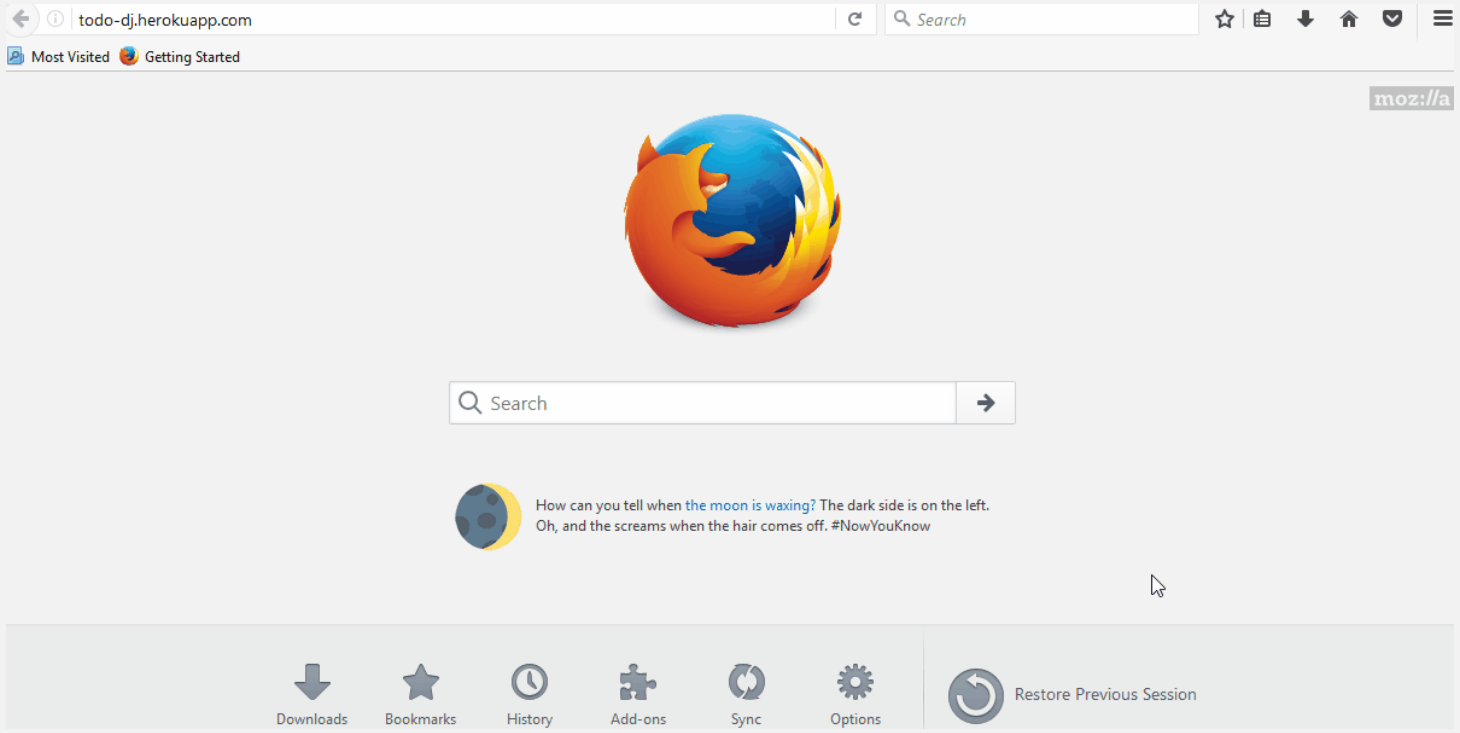
The slides & the demo's code:

- https://github.com/noah-dev/pykc_django_talk

Questions?

Thank you for attending! Have any questions? Fire away!

- I'll be here for the rest of the meetup – please come by



Deploy! To Heroku!

Deploying online will require additional steps. For Heroku:

- Setup a python virtualenv (required for Heroku)
- Install django-toolbelt module and whitenoise modules
- Update *settings.py* to setup Postgres, static collect, & ENV secret key
- Create the a procfile and setup wsgi.py
- Create requirements.txt to list dependencies
- Push to Heroku repo, build, and migrate database. Deployed!

Setup the Virtualenv

Heroku needs it to retrieve and install the different python modules

- Virtualenv creates a specific environment for each project.
- For example, use different versions of a module for different projects

Install virtualenv & virtualenvwrapper (virtualenvwrapper-win for windows)

- Navigate to project folder and do:
  - **mkvirtualenv django_demo**
  - **workon django_demo**

Update settings.py

Update settings to set: secret key, host name, database, & static path

```
# Update database configuration with $DATABASE_URL.
db_from_env = dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(db_from_env)

    DEBUG = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.11/howto/static-files/
PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
STATIC_ROOT = os.path.join(PROJECT_ROOT, 'static')
STATIC_URL = '/static/'


ALLOWED_HOSTS = ['todo-demo-dj.herokuapp.com',
                 'localhost', ]
```

Setup Procfile & wsgi.py

The app will need a WSGI & a static content server

- gunicorn is a Web Server Gateway Interface, which enables python code to process incoming HTTP requests concurrently
- whitenoise is a module that manages static assets, like images & css

Procfile

```
web: gunicorn demo.wsgi --log-file -
```

```
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "demo.settings")

application = get_wsgi_application()
application = DjangoWhiteNoise(application)
```

Installed modules – requirements.txt

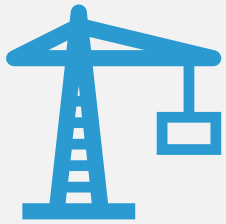
By now, the following modules should be installed:

- pip install django
- pip install django-toolbelt
- pip install whitenoise

Finally, **pip freeze > requirements.txt**

requirements.txt

```
1  dj-database-url==0.4.2
2  dj-static==0.0.6
3  Django==1.11.3
4  django-toolbelt==0.0.1
5  gunicorn==19.7.1
6  psycopg2==2.7.1
7  pytz==2017.2
8  static3==0.7.0
9  whitenoise==3.3.0
```



Setup the Heroku App

With the changes complete, create a new Heroku App

- Setup a free Heroku account, install their CLI, & create an application
- Go to settings and add a pair of config vars (click “Reveal Config Vars”)
- Set a “DJANGO_DEBUG” to “False”
- Set the “SECRET_KEY” to a different secret key
- There are a few utilities that can generate new keys. You can use:

```
import string,random; uni=string.ascii_letters+string.digits+string.punctuation;
print(''.join([random.SystemRandom().choice(uni) for i in range(random.randint(45,50))]))
```

Deploy to Heroku – Push & Migrate

Everything is ready for deployment – let's ship it!

- Set a git remote using the link for the Heroku app
- Push to the Heroku repo and wait for it to build
- Login using **heroku run bash** to remote into the server
- Run **python manage.py migrate**
  - If model is changed, first run **python manage.py makemigrations todo**
- Use **exit** to leave heroku bash

Complete!

Success – the application is now live!

- Visit [app_name].herokuapp.com – or use **heroku open**

Want to try some more?

- Setup a views.py in the project folder to create page at the root url
- Update items using AJAX requests without refreshing the page
- Add user login with Django's prebuilt authentication