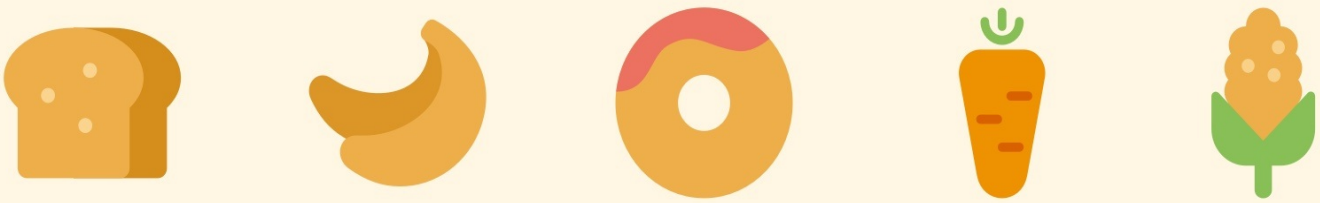
Django Workshop

PyCon PH 2015



Jon Danao

jondanao@gmail.com

Table of Contents

Django Workshop at PyConPH 2015

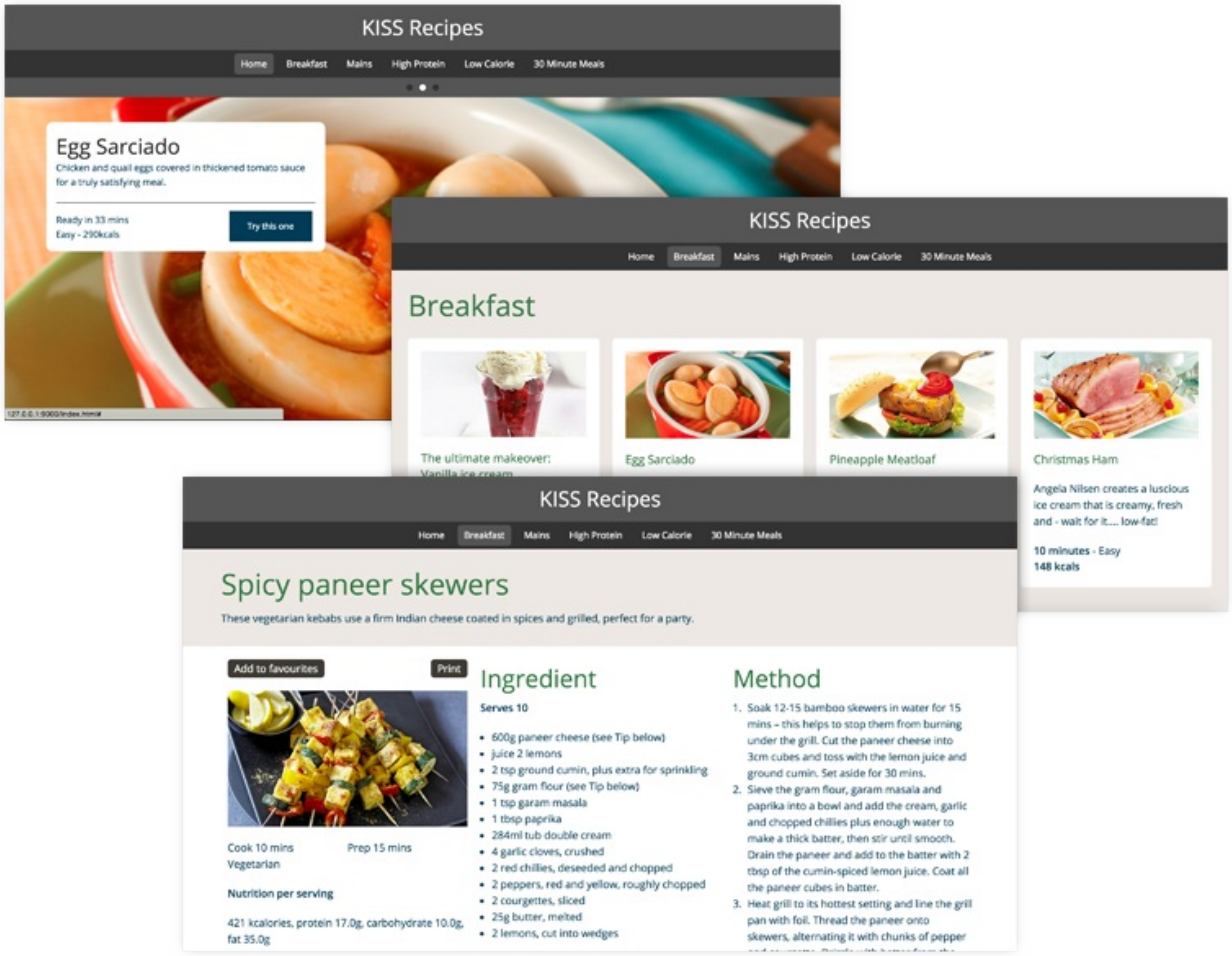
by Jon Danao

Django is a batteries-included Python web framework for creating dynamic web applications and APIs. It is free and open source, has extensive documentation and a large community. In this workshop you will learn how to setup your project, interact with the database and render front-end HTMLs.

- **Expected length:** 3 hours
- **Talk level:** Beginner and Intermediate
- **Equipment:** You may bring a laptop for the workshop.

Jon is the Head Honcho of The App Factory, previously the Head of Technology for Innovations at ABS-CBN. He played lead guitars for a local band in Manila called Bridge. He is strong in blues, rock and roll, and showmanship. Terrible in second voice. He loves to cook and went to culinary school. He is strong in Asian cuisine and any food with patis (fish sauce) and chili.

> ***This workshop has been tested on OSX Yosemite and Windows 7 64-bit***



Topics

1. Setup: How to setup your machine for Django

1. Setup for OSX
2. Setup for Linux
3. Setup for Windows

2. Project: How to start your Django project

1. Setting up the Development Environment
2. Creating the Django Project
3. Configuring the Site

3. Models: How to define your data

1. Creating the Recipe App and its Model
2. Activating the Models in Admin Site

4. URLs and Views: How to access your data

1. Naming the URL for Index Page
2. Querying Data for Index Page

5. Templates: How to render your data

1. Rendering the Index Page
2. Rendering the Category and Recipe Pages

Thank You

- Python PH - https://www.facebook.com/groups/pythonph/
- Jay Gauten - jay@theappfactory.io
- Randell Quitain - randell@theappfactory.io
- Creative Dash for the yummy icons - https://twitter.com/creativedash

Setup: How to setup your machine for Django

Exercise 1.1 - Setup for OSX

The gist of this exercise is to check the dependencies by executing these commands. If all result in some version number, you are ready for this workshop. Otherwise, jump to the first section below:

```
$ python --version
$ pip --version
$ virtualenv --version
$ git --version
```

A. Python 2.7 or 3.4

OSX Mavericks and Yosemite come with a built-in Python 2.7.x. No need to worry about installing a new one.

1. Check if Python is installed or accessible. Open the Terminal. Most likely, you have Python installed:

```
$ python --version
```

2. On Yosemite, you should see:

```
Python 2.7.x
```

3. Unless absolutely necessary, you want to install manually, use Homebrew (optional).

```
$ brew install python
```

4. Check again if Python is installed:

```
$ python --version
```

5. If all goes well, you should see:

```
Python 2.7.x
```

B. Pip

If you have installed Python via Homebrew, its installer already is bundled with Setuptools and Pip.

1. Check if Pip is installed. Open the Terminal and execute:

```
$ pip --version
```

2. If this is what you see, then you don't have Pip installed:

```
-bash: pip: command not found
```

3. Install Pip by downloading this script (https://bootstrap.pypa.io/get-pip.py) and executing this line in the Terminal:

```
$ python get-pip.py
```

4. Check again if Pip is installed. Open the Terminal and execute:

```
$ pip --version
```

5. If all goes well, you should see:

```
pip 1.5.x
```

C. VirtualEnv

1. Install VirtualEnv. Open the Command Prompt and execute:

```
$ sudo pip install virtualenv
```

2. Check if VirtualEnv is already installed. Reopen the Command Prompt and execute:

```
$ virtualenv --version
```

3. If all goes well, you should see

```
virtualenv 1.x.x
```

D. Git (optional)

As a developer, there's no excuse not to use Git. But if you don't have it in

your machine, you don't have to worry about it. It's not really part of Django. Also, it's impractical to install its prerequisite - the Xcode Command Line Tools which has quite a large installer.

Download the installer here (http://git-scm.com/download/mac).

Exercise 1.2 - Setup for Linux

The gist of this exercise is to check the dependencies by executing these commands. If all result in some version number, you are ready for this workshop. Otherwise, jump to the first section below:

```
$ python --version
$ pip --version
$ virtualenv --version
$ git --version
```

A. Python 2.7 or 3.4

The latest Ubuntu and Fedora come with a built-in Python 2.7.x. No need to worry about installing a new one.

1. Check if Python is installed or accessible. Open the Terminal. Most likely, you have Python installed:

```
$ python --version
```

2. You should see:

```
Python 2.7.x
```

B. Pip

1. Check if Pip is installed. Open the Terminal and execute:

```
$ pip --version
```

2. If this is what you see, then you don't have Pip installed:

```
-bash: pip: command not found
```

3. Install Pip by downloading this script (https://bootstrap.pypa.io/get-pip.py) and executing this line in the Terminal:

```
$ python get-pip.py
```

4. Check again if Pip is installed. Open the Terminal and execute:

```
$ pip --version
```

5. If all goes well, you should see:

```
pip 1.5.x
```

C. VirtualEnv

1. Install VirtualEnv. Open the Command Prompt and execute:

```
$ sudo pip install virtualenv
```

2. Check if VirtualEnv is already installed. Reopen the Command Prompt and execute:

```
$ virtualenv --version
```

3. If all goes well, you should see

```
virtualenv 1.x.x
```

D. Git (optional)

As a developer, there's no excuse not to use Git. But if you don't have it in your machine, you don't have to worry about it. It's not really part of Django.

On Fedora

```
$ sudo yum install git
```

On Debian / Ubuntu

```
$ sudo apt-get install git
```

Exercise 1.3 - Setup for Windows

The gist of this exercise is to check the dependencies by executing these commands. If all result in some version number, you are ready for this workshop. Otherwise, jump to the first section below:

```
C:\> python --version
C:\> pip --version
C:\> virtualenv --version
C:\> git --version
```

A. Python 2.7 or 3.4

1. Check if Python is installed or accessible. Open the Command Prompt:

```
C:\> python --version
```

2. If this is what you see, then it means you have no Python installed.

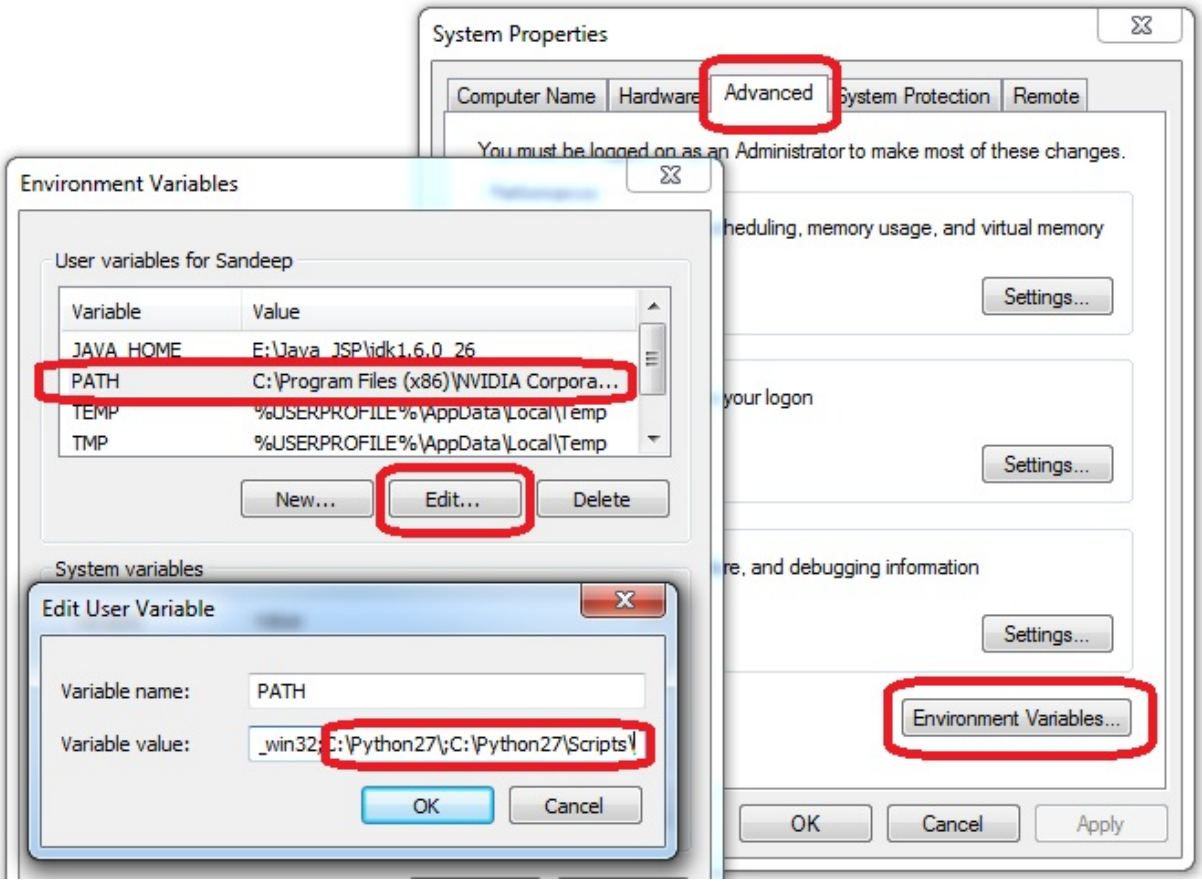
```
'python' is not recognized as an internal or external command, operab
```

3. Download and install the latest Python 2.x or 3.x releases from this location.

```
https://www.python.org/downloads/windows/
```

4. Add the Python path to the system environment variables. To do this, go to `My Computer` > `Properties` > `Advanced System Settings` > `Environment Variables` > `System Variables` > `Path` or `PYTHONPATH` (whichever will work). Click edit. Add this at the *end of the line*:

```
;C:\Python27;C:\Python27\Scripts
```



5. Check again if Python is already accessible. Reopen the Command Prompt and execute:

```
C:\> python --version
```

6. If all goes well, you should see

```
Python 2.7.x
```

B. Pip

1. Check if Pip is installed. Open the Command prompt and execute:

```
C:\> pip --version
```

2. If this is what you see, then you don't have Pip installed:

```
'pip' is not recognized as an internal or external command, operable
```

3. Install Pip by downloading this script (https://bootstrap.pypa.io/get-pip.py) and executing this line in the Terminal:

```
C:\> python get-pip.py
```

4. Check again if Pip is installed. Open the Terminal and execute:

```
C:\> pip --version
```

5. If all goes well, you should see:

```
pip 1.5.x
```

C. VirtualEnv

1. Install VirtualEnv. Open the Command Prompt and execute:

```
C:\> pip install virtualenv
```

2. Check if VirtualEnv is already installed. Reopen the Command Prompt and execute:

```
C:\> virtualenv --version
```

3. If all goes well, you should see

```
virtualenv 1.x.x
```

D. Git (optional)

As a developer, there's no excuse not to use Git. But if you don't have it in your machine, you don't have to worry about it. It's not really part of Django.

Download installer here: http://git-scm.com/download/win

Project: How to start your Django project

Exercise 2.1 - Setting Up the Development Environment

1. Create the project directory using the terminal and CD into it

```
$ mkdir website.com && cd website.com
```

2. Create the virtual environment

```
# On OSX or Linux
website.com $ virtualenv venv --no-site-packages
```

```
# On Windows, do this if package installations below don't work
website.com $ virtualenv venv --system-site-packages
```

3. Activate the virtual environment

```
# On OSX or Linux
website.com $ source venv/bin/activate
(venv)website.com $
```

```
# On Windows
website.com $ venv\Scripts\activate.bat
(venv)website.com $
```

4. Install django

```
(venv)website.com $ pip install django
```

5. Save site packages

```
(venv)website.com $ pip freeze > requirements.txt
```

Challenge: Feeling Advanced? You look advanced. You can do this!

> Why don't you try **VirtualEnvWrapper**? They say it makes working with virtual environments more pleasant.

Exercise 2.2 - Creating the Django Project

1. Start a Django project in the project directory

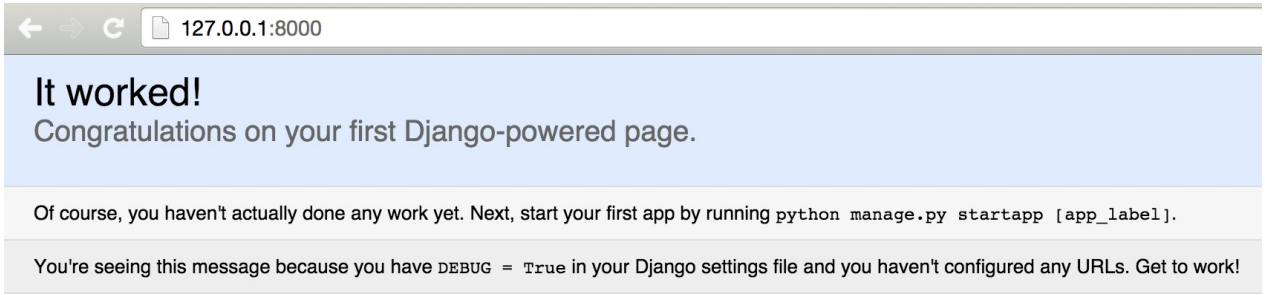
```
# On OSX or Linux
(venv)website.com $ django-admin.py startproject project .
```

```
# On Windows
(venv)website.com $ python -m django-admin startproject project .
```

2. Run the site server and browse the site

```
(venv)website.com $ python manage.py runserver
```

If everything goes well, you should see this:



3. Convert project folder to a Git repo

```
(venv)website.com $ git init .
```

4. Create and edit `.gitignore` file.

```
*.pyc
```

```
venv
```

5. Stage all files and commit!

```
(venv)website.com $ git add .
(venv)website.com $ git commit -m "Created a Django project"
```

6. Verify your last commit. It should say `nothing to commit`.

```
(venv)website.com $ git status
```

Challenge: Feeling Advanced? You look advanced. You can do this!

> Why don't you try streamlining your project using GitFlow or Feature Branch Workflows? They say these workflows are great especially if you're working with a team.

Exercise 2.3 - Configuring the Site Settings

1. Organize our site by creating necessary folders.

```
(venv)website.com $ mkdir apps media static templates
```

2. Edit `/project/settings.py`

```
TEMPLATES = [
    {
        ...
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        ...
    },
]

# Static files (CSS, JavaScript, Images)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]

MEDIA_URL = '/media/'
MEDIA_ROOT = 'media'
```

3. Run the server to check that we did not break anything.

```
(venv)website.com $ python manage.py runserver
```

4. Apply the initial migration and create super user

```
You have unapplied migrations; your app may not work properly until t
Run 'python manage.py migrate' to apply them.
```

```
(venv)website.com $ python manage.py migrate
(venv)website.com $ python manage.py createsuperuser
```

5. Run the server and login to the `/admin/` site.

```
(venv)website.com $ python manage.py runserver
```

6. Your project structure should look like this.

```
website.com
├── apps
├── db.sqlite3
├── manage.py
├── media
├── project
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── requirements.txt
├── static
└── templates
```

7. Stage all files and commit!

```
(venv)website.com $ git add .
(venv)website.com $ git commit -m "Configured the site settings"
```

Challenge: Feeling Advanced? You look advanced. You can do this!

> Why don't you try using **PostgreSQL** as your database instead of SQLite? They say it's more predictable if you use the same type of

> database in your dev machine and your production server.

Models: How to define your data

Exercise 3.1 - Creating the Recipes App and its Models

1. Start an app inside the `/apps/` folder

```
# On OSX or Linux
(venv)website.com $ cd apps
(venv)website.com/apps $ django-admin.py startapp recipes
```

```
# On Windows
(venv)website.com $ cd apps
(venv)website.com/apps $ python -m django-admin startapp articles
```

2. Edit `/apps/recipes/models.py`

```
from django.db import models

class Category(models.Model):
    title = models.CharField(max_length=200)
    slug = models.SlugField(unique=True)

    class Meta:
        verbose_name_plural = 'Categories'

    def __unicode__(self):
        return self.title

class Recipe(models.Model):
    title = models.CharField(max_length=200)
    slug = models.SlugField(unique=True)
    category = models.ForeignKey(Category)
    description = models.TextField()
    photo = models.FileField()
    ingredients = models.TextField()
```

```
    method = models.TextField()

    cooktime = models.IntegerField()
    fat = models.FloatField()
    carbs = models.FloatField()
    protein = models.FloatField()
    calories = models.FloatField()

    def __unicode__(self):
        return self.title
```

3. Edit `/project/settings.py`

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'apps.recipes',
)
```

4. Create `/apps/__init__.py` and run the migrations

```
(venv)website.com $ python manage.py makemigrations
(venv)website.com $ python manage.py migrate
```

5. Inspect the database and commit!

```
(venv)website.com $ git add .
(venv)website.com $ git commit -m "Created the recipes app and its mo
(venv)website.com $ git status
```

Exercise 3.2 - Activating Models in the Admin

1. Edit `/apps/recipes/admin.py`

```
from django.contrib import admin
from .models import Category, Recipe

admin.site.register(Category)
admin.site.register(Recipe)
```

2. Run the site server and browse admin site

```
(venv)website.com $ python manage.py runserver
```

3. In your admin site, populate your database with `2 categories` and `5 recipes`. Don't forget to upload photos!

Suggested category names:

Title	Slug
Breakfast	breakfast
Mains	mains

4. Stage all files and commit!

```
(venv)website.com $ git add .
(venv)website.com $ git commit -m "Activated models in Admin site"
(venv)website.com $ git status
```

Challenge: Feeling Advanced? You look advanced. You can do this!

> The slug is usually the same as the article title but only supports dashes instead of spaces. Why don't you try implementing an autofill feature for the slug field? **Hint:** use ModelAdmin's `prepopulated_fields` option.

URLs and Views: How to access your data

Exercise 4.1 - Naming the URL for Index Page

1. Edit `/project/urls.py`

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

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

2. Create and edit `/project/views.py`

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')
```

3. Create and edit `/templates/home.html`

```
<h1>Hello Django!</h1>
```

4. Run the site server and browse the index page. You should see “Hello Django!”.

```
(venv)website.com $ python manage.py runserver
```

5. Stage all files and commit!

```
(venv)website.com $ git add .
(venv)website.com $ git commit -m "Wrote the URL for index page"
(venv)website.com $ git status
```

Exercise 4.2 - Querying Data for Index Page

1. Edit project/views.py

```
from django.shortcuts import render
from apps.recipes.models import Recipe, Category

def home(request):
   recipes = Recipe.objects.all()[:3]
   print recipes

   return render(request, 'home.html', {
      'recipes': recipes,
   })
```

2. Stage all files and commit!

```
(venv)website.com $ git add .
(venv)website.com $ git commit -m "Queried data for index page"
(venv)website.com $ git status
```

Challenge: Feeling Advanced? You look advanced. You can do this!

> Why don't you try querying 3 recipes and sort them randomly? Hint: you are going to use a questions mark (?) somewhere in your query.

Templates: How to render your data

Exercise 5.1 - Rendering the Index Page

1. Place all HTML files into `/templates/` folder and all static files into `/static/` folder.

2. Edit `/project/urls.py`

```
from django.conf import settings # add this
from django.conf.urls.static import static # add this
from django.conf.urls import include, url
from django.contrib import admin
from . import views

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', views.home),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT) # a
```

3. Edit `/templates/home.html`

```
<!-- RECIPES -->
<div class="carousel">
    <ul>
        {% for recipe in recipes %}
            <li>
                <img src="{{ recipe.photo.url }}" alt="">
                <div class="slide-info">
                    <h3>{{ recipe.title }}</h3>
                    <p>{{ recipe.description }}</p>
                    <p class="meta">
                        <span class="time">Ready in {{ recipe.cooktim
                        <span class="difficulty">{{ recipe.calories }
                        <a href="/{{ recipe.category.slug }}/{{ recip
                    </p>
                </div>
```

```
            </li>
        {% endfor %}
    </ul>
</div>
<!-- RECIPES -->
```

4. Run the site server and browse the index page.

```
(venv)website.com $ python manage.py runserver
```

5. Stage all files and commit!

```
(venv)website.com $ git add .
(venv)website.com $ git commit -m "Rendered the index page"
(venv)website.com $ git status
```

Challenge: Feeling Advanced? You look advanced. You can do this!

> If you haven't noticed, these article links are really tedious to construct and quite messy! Why don't you try applying **DRY** to the URLs? Hint: there is a method in Models called `get_absolute_url`. Do the simplest implementation and your templates will be much cleaner.

Exercise 5.2 - Rendering the Category and Recipe Pages

1. Edit `/project/urls.py`

```
from django.conf import settings
from django.conf.urls.static import static
from django.conf.urls import include, url
from django.contrib import admin
from . import views

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', views.home),
    url(r'^(?P<category_slug>[-\w]+)/$', views.category), # add this
    url(r'^(?P<category_slug>[-\w]+)/(?P<recipe_id>\d+)/([-\w]+)/$',
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

2. Edit `/project/views.py` and add these at the bottom

```
def category(request, category_slug):
    recipes = Recipe.objects.filter(category__slug=category_slug)

    return render(request, 'category.html', {
      'recipes': recipes,
    })

def recipe(request, category_slug, recipe_id):
    recipe = Recipe.objects.get(id=recipe_id)

    return render(request, 'recipe.html', {
      'recipe': recipe,
    })
```

3. Edit `/templates/category.html`

```
<!-- RECIPES -->
{% for recipe in recipes %}
    <div class="recipe-item">
        <article>
            <a class="recipe-image" href="/{{ recipe.category.slug }}
                <img src="{{ recipe.photo.url }}" alt="{{ recipe.titl
                <h2 class="recipe-title">{{ recipe.title }}</h2>
            </a>

            <p>{{ recipe.description|truncatewords:20 }}</p>
            <div class="recipe-info">
                <span class="total-time">{{ recipe.cooktime }} minute
                <span class="kcals">{{ recipe.calories }} kcals</span
                <span class="icons icons vegetarian-icon"></span>
            </div>
        </article>
    </div>
{% endfor %}
<!-- RECIPES -->
```

4. Edit `/apps/recipes/models.py`

```
class Recipe(models.Model):

    ...

    def __unicode__(self):
        return self.title

    def ingredients_list(self):
        return self.ingredients.split('\n')

    def method_list(self):
        return self.method.split('\n')
```

5. Edit `/templates/article.html`

```
<!-- RECIPE -->
<article class="recipes">
    <header class="recipe-header">
        <h1 class="title">{{ recipe.title }}</h1>
        <p>{{ recipe.description }}</p>
        <a class="buttons close" href="#">Close</a>
    </header>
    <div class="recipe-content">
        <div class="details">
            <div class="actions">
                <a class="buttons" href="#">Add to favourites</a>
                <a class="buttons" href="#">Print</a>
            </div>
            <img src="{{ recipe.photo.url }}" alt="Spicy Paneer Skewe
            <ul class="recipe-info">
                <li><span class="icons cooktime-icon">Cook {{ recipe.
                <li><span class="icons vegetarian-icon">Vegetarian</s
            </ul>
            <p class="per-serving">Nutrition per serving</p>
            <p>{{ recipe.calories }} kcalories, protein {{ recipe.pro
        </div>
        <div class="ingredients">
            <h2 class="title">Ingredient</h2>
            <p><strong>Serves 10</strong></p>
            <ul>
                {{ recipe.ingredients_list|unordered_list }}
            </ul>
        </div>
        <div class="method">
            <h2 class="title">Method</h2>
            <ol>
                {{ recipe.method_list|unordered_list }}
            </ol>
        </div>
    </div>
</article>
<!-- RECIPE -->
```

6. Run the site server and browse all the pages.

```
(venv)website.com $ python manage.py runserver
```

7. Stage all files and commit!

```
(venv)website.com $ git add .
(venv)website.com $ git commit -m "Rendered the category and article
(venv)website.com $ git status
```

Challenge: Feeling Advanced? You look advanced. You can do this!

> Why don't you try adding highlight to your navigation's active state when browsing a category or an article? **Hint:** use template inheritance to implement this. Add `class="active"` to the nav's `<li>` element.