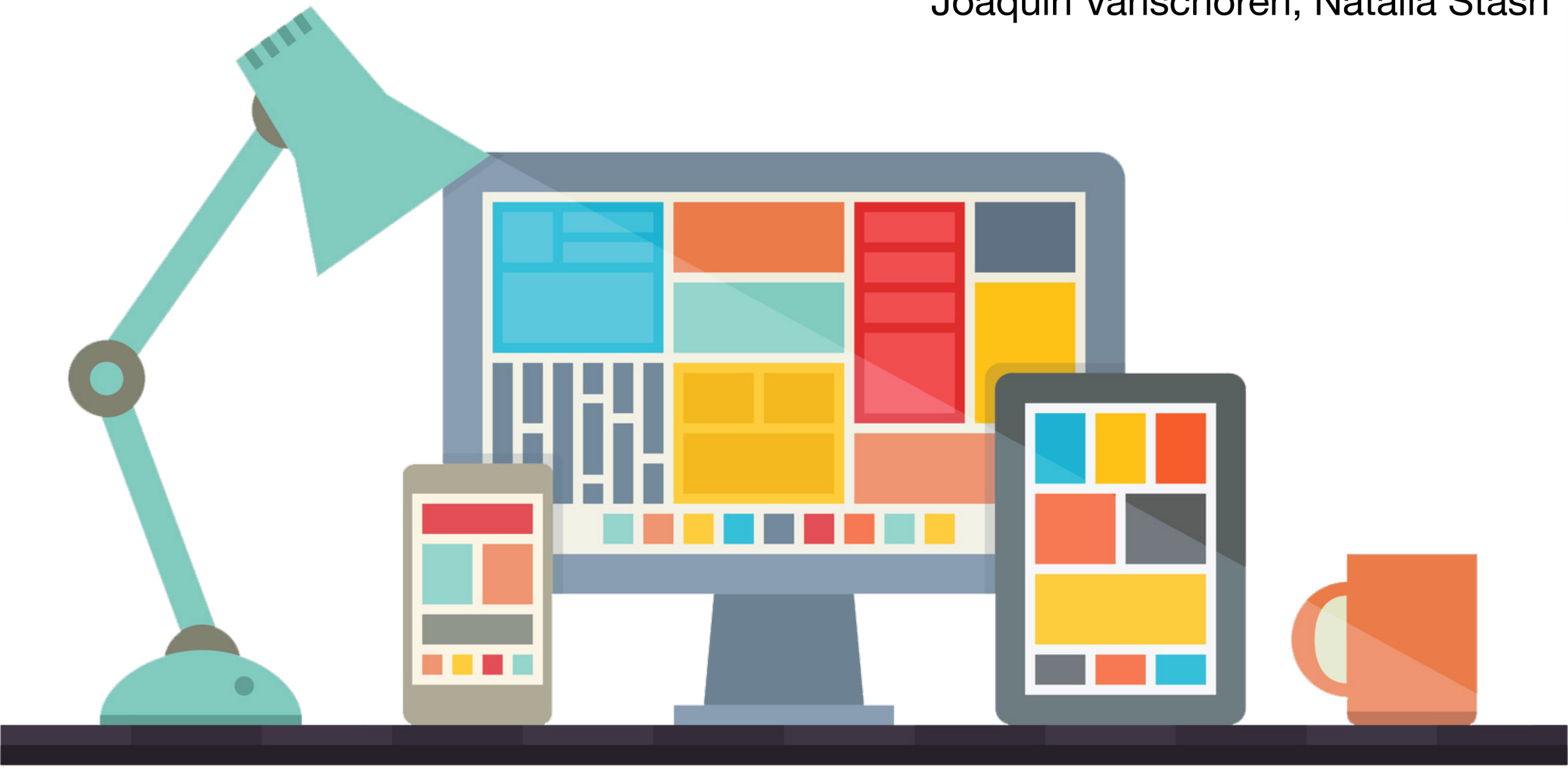
Technische Universiteit Eindhoven University of Technology

Joaquin Vanschoren, Natalia Stash

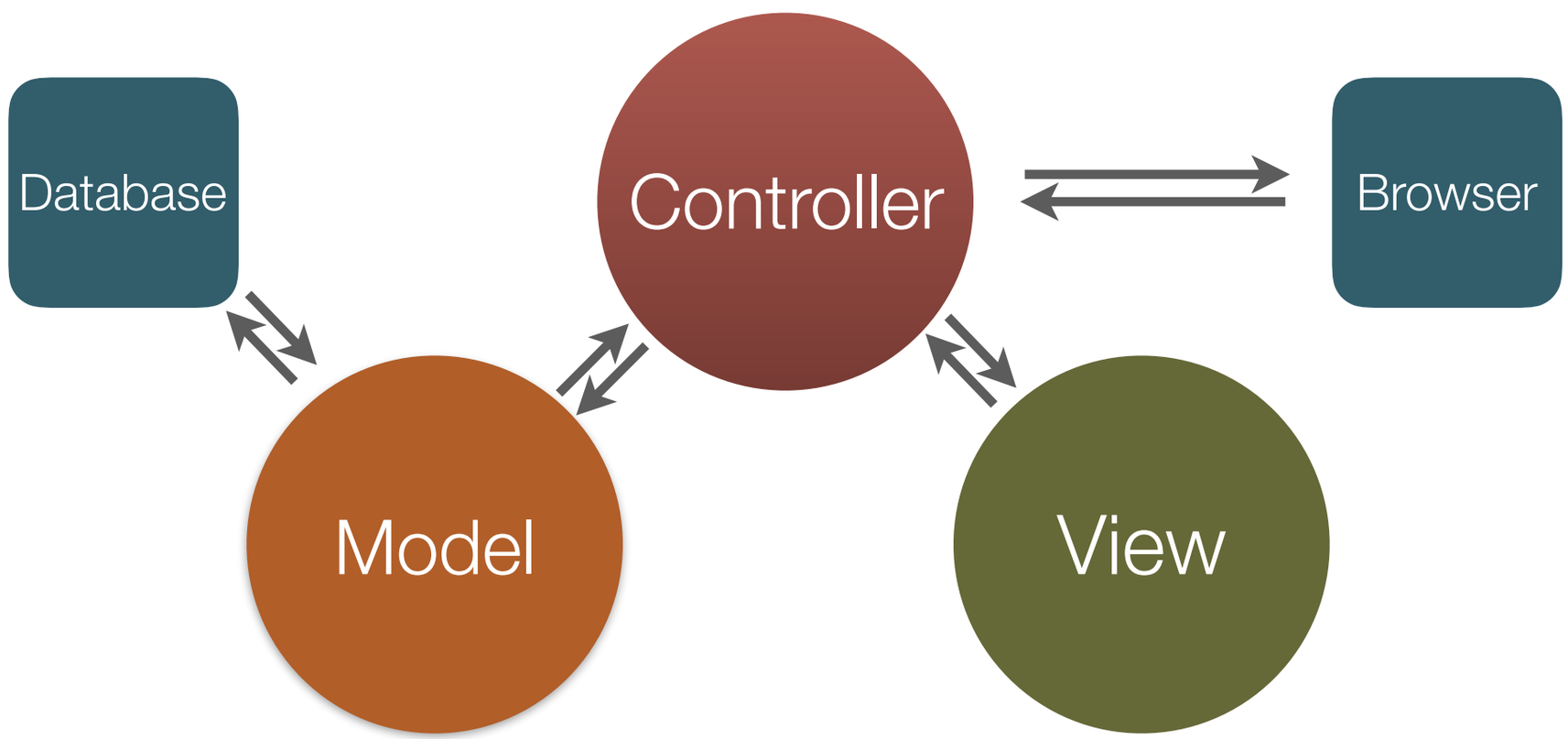


Web Technology

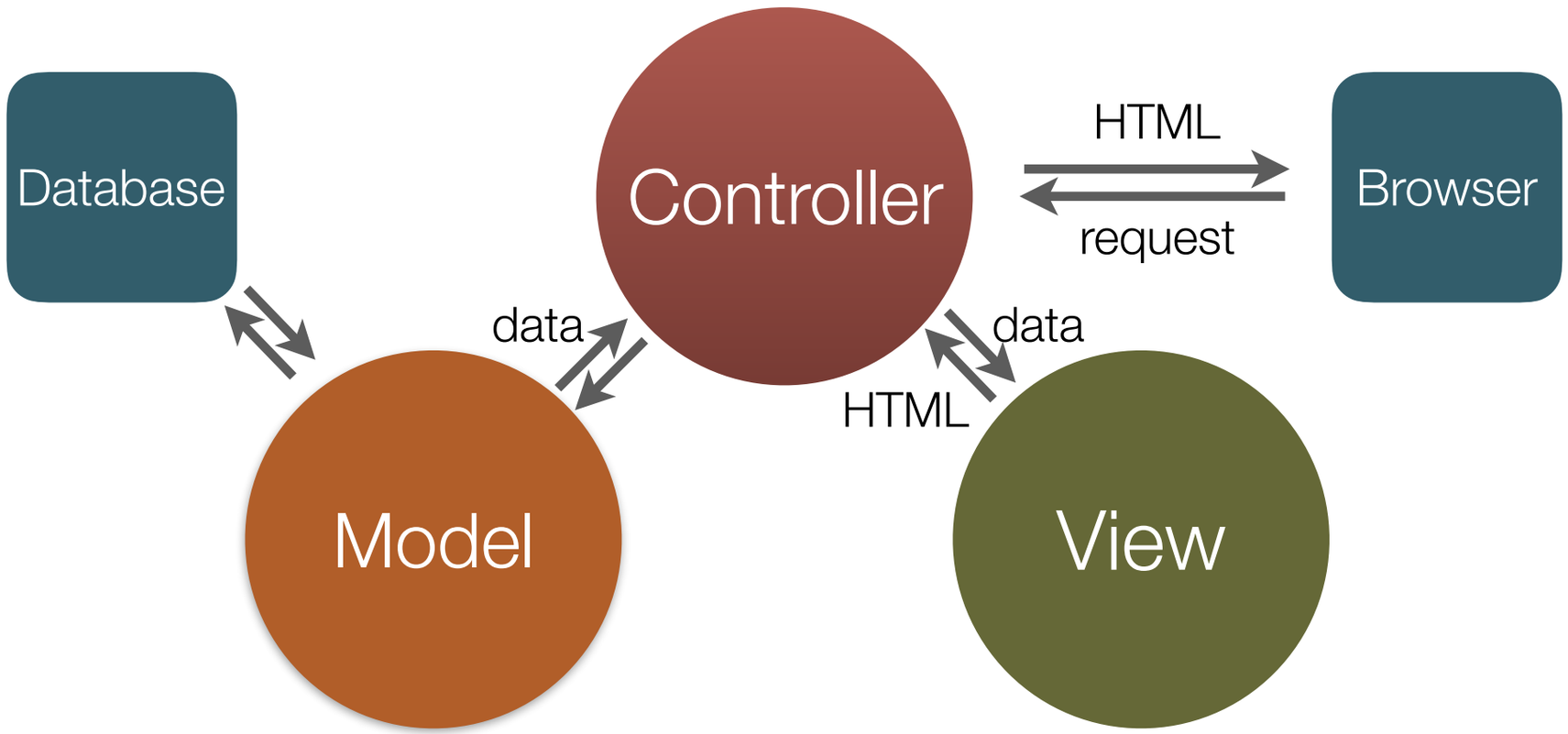
Django

Back-end frameworks

- Backend reads HTTP request, fetches data, generates HTML
- Many back-end processes are repetitive and error-prone (e.g. retrieving data from a database)
- Backend frameworks provide code to hide most complexity
- Architecture: Model (data) - View (HTML template) - Controller (logic)

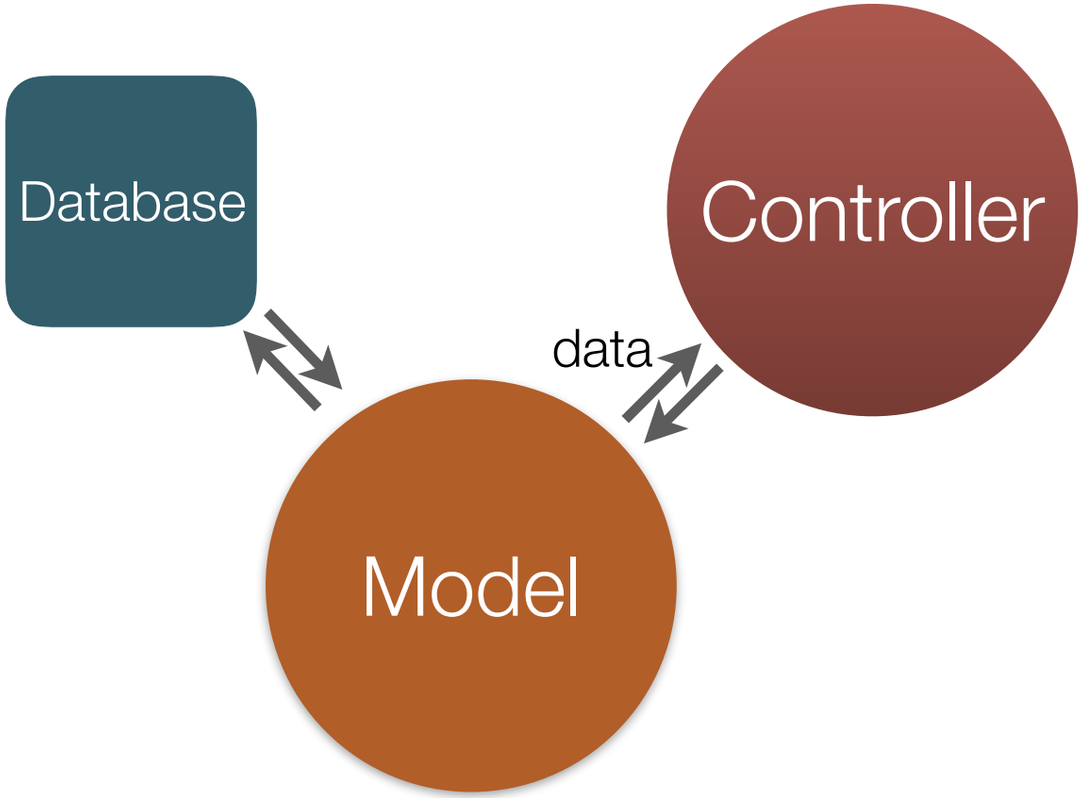


Model-View-Controller



- Browser (client) requests a web page (HTTP request)
- Controller interprets the request, asks the Model for the data
- The Model retrieves the data from a database, passes it to Controller
- Controller passes data to the View, View returns HTML
- Controller sends HTML to client

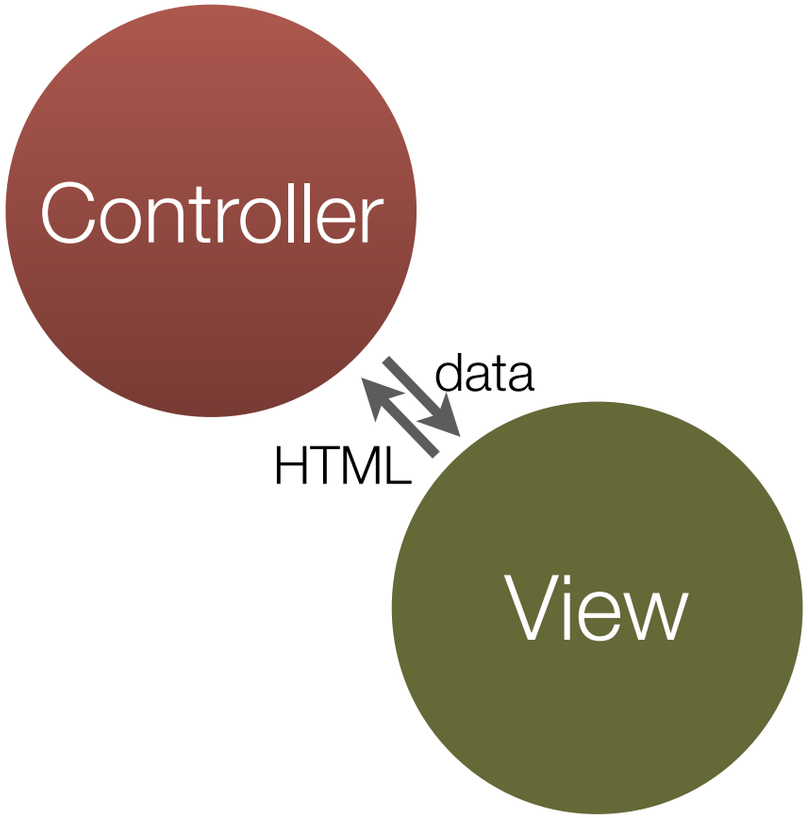
Model



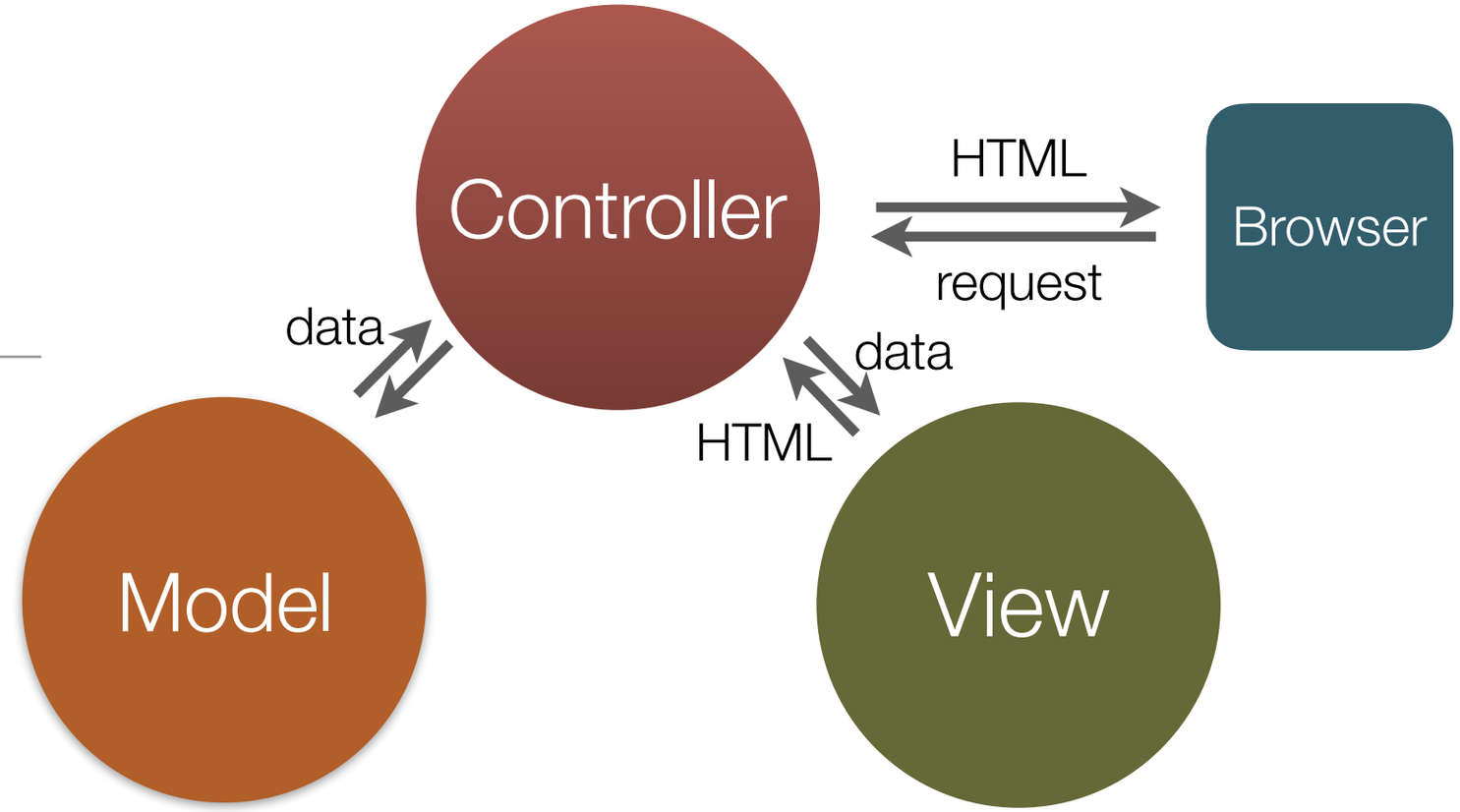
- Receives request from Controller
  - eg. getUserProfile(12345)
- **Talks to the right database(s)**
  - MySQL, MongoDB,…
  - Can be remote (on other server)
- Adds/retrieves data from DB
  - Knows DB schema
- **Returns data in requested format**
  - eg. JSON object
- Only talks to Controller

View

- Receives request from Controller
  - eg. getUserProfilePage(data)
- **Contains HTML templates**
  - Jena, Handlebars,…
- Fills out templates with right data
- **Returns HTML (+CSS,JS)**
  - All the user gets to see
- Only listens to Controller



Controller



- **App has multiple controllers**
  - **Route processor** calls right controller based on URL
    - Route: /user/profile/1 -> call user pages controller
- Controller **processes GET/POST/DELETE requests**
  - Extracts information from URL, HTML headers
    - Route: /user/profile/1
    - URL parameters: ?attr=bla
- Talks to Model to get necessary data
- Talks to View to explain data to the user

django

The Web framework for perfectionists with deadlines.

Fallbacks

- Great online tutorial:
  - https://tutorial.djangogirls.org/
- More complete guide (yet slightly outdated):
  - http://www.gettingstartedwithdjango.com/
  - https://www.youtube.com/watch?v=KZHXjGP71kQ&t=609s
- The official tutorial:
  - https://www.djangoproject.com/start/

Note on naming

- In Django,
  - Models are called models
  - Views are called templates
  - Controllers are called views

Tip: Virtual Environment

- If you run multiple Python projects, use a virtual environment
  - So your project doesn't interfere with other projects (and vice versa)
- Installation:
  - Linux/Mac: http://roundhere.net/journal/virtualenv-ubuntu-12-10/
  - Win: http://pymote.readthedocs.io/en/latest/install/windows_virtualenv.html

```
$ mkvirtualenv django
(django) $ deactivate
$ workon django
(django) $ python
Python 3.5.2 |Continuum Analytics, Inc.
>>> <Ctrl-D>
(django) $
```

Alternative install

- If that didn't work, see:
  - https://tutorial.djangogirls.org/en/django_installation/
  - Offers fallbacks for linux/mac/windows

```
$ virtualenv --python=python3.5 django
$ . myvenv/bin/activate
(django) $
```

Create a Django project

- Follow the slides to create your first simple Django project
- All the code is available on GitHub:
  - https://github.com/WebTech2016/django-todolist

- More detailed installation instructions:
  - https://tutorial.djangogirls.org/en/django_installation/

Create a Django project

- Install the latest Django version
- Create a project (here, we're making a todo-list app)

```
(django) $ pip install Django==1.10.3
(django) $ django-admin.py startproject todolist
```

- These files are created automatically:
  - `manage.py` - access to all django functions
  - project folder with:
    - `settings.py` - settings file
    - `urls.py` - route processor
    - `wsgi.py` - establishes web server communication

Break out Atom (or another good editor)

urls.py — ~/Code/django-tutorial/todolist

- todolist
  - todolist
    - __init__.py
    - settings.py
    - urls.py
    - wsgi.py
  - manage.py

urls.py

```
"""todolist URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/1.10/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  url(r'^$', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  url(r'^$', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.conf.urls import url, inclu
    2. Add a URL to urlpatterns:  url(r'^blog/', include('blog.urls'))
"""
from django.conf.urls import url
from django.contrib import admin

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

```

Adjust settings.py

- Set correct time:
  - `TIME_ZONE = 'Europe/Berlin'`
- Under STATIC_URL, add
  - `STATIC_ROOT = os.path.join(BASE_DIR, 'static')`
- Allow access from host:
  - ALLOWED_HOSTS = ['127.0.0.1', 'localhost', '.pythonanywhere.com', '.herokuapp.com']

Database

- Default database is SQLite (in-file database)
  - Don't use this for real projects (more options later)
- Already set up in settings.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

- *Create* the database with the **`migrate`** command

```
(django) $ python manage.py migrate
```

- You'll now have a **`db.sqlite3`** file in your project

Database: PostgreSQL (Optional)

- For production systems, PostgreSQL is a better choice.
- First, install Postgres: https://www.postgresql.org/
- First, create a database and user, adjust some settings

```
(django) $ psql -h localhost
psql (9.3.0)
joa=# CREATE DATABASE djangotodo;
CREATE DATABASE
joa=# CREATE USER django WITH PASSWORD 'django';
CREATE ROLE
joa=# ALTER ROLE django SET client_encoding TO 'utf8';
ALTER ROLE
joa=# ALTER ROLE django SET default_transaction_isolation TO 'read committed';
ALTER ROLE
joa=# ALTER ROLE django SET timezone TO 'Europe/Berlin';
ALTER ROLE
joa=# GRANT ALL PRIVILEGES ON DATABASE djangotodo TO django;
GRANT
```

Database: PostgreSQL (Optional)

- Install PostgreSQL Python adapter

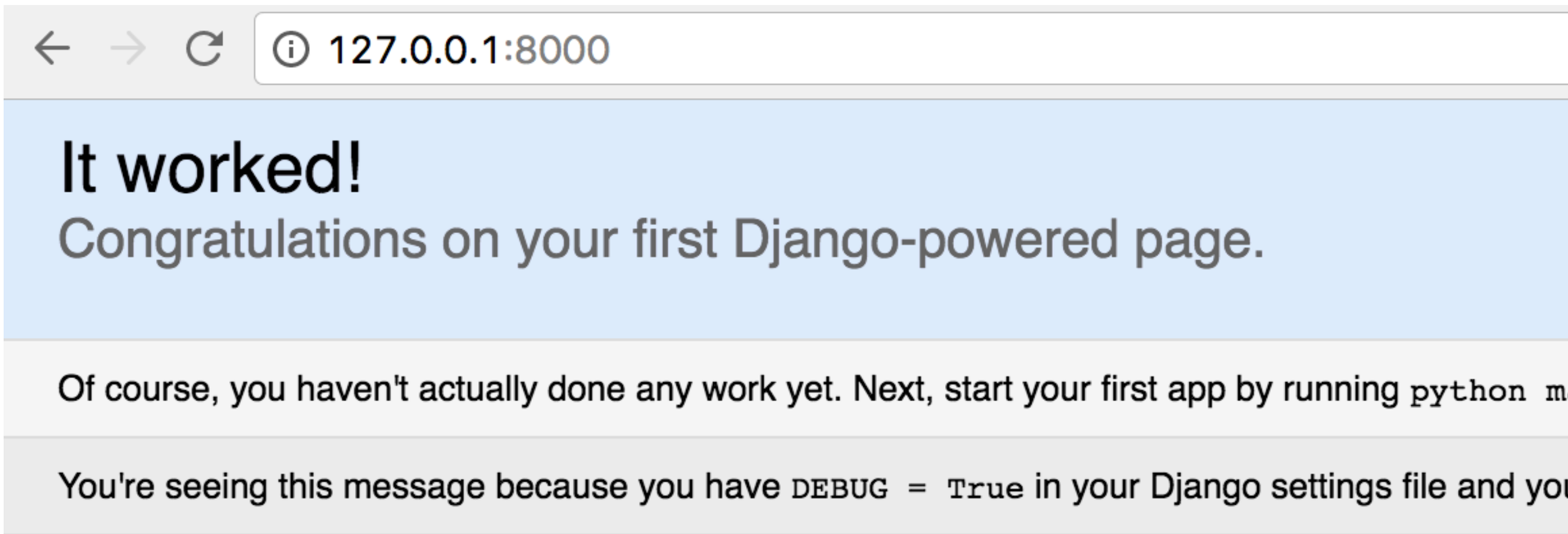
```
(django) $ pip install django psycopg2
```

- Update settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'djangotodo',
        'USER': 'django',
        'PASSWORD': 'django',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

Test webserver

```
(django) $ python manage.py runserver
```



On windows: `$ python manage.py runserver 0:8000`

Run webserver on different port

- Django runs on port 8000 by default
- On a shared server (e.g. University server), only one service can run on port 8000
- Specify your port in the run server command

```
(django) $ python manage.py runserver 8080
```

Applications

- An **app** is a combination of models, views, templates,…
- A Django project can have multiple apps and an app can be reused in multiple projects
- Create an app with `startapp`
  - Our app is called ‘list’

```
(django) $ python manage.py startapp list
```

- New directory ‘list’ with new files appears:
  - `models.py` - create models
  - `views.py` - create views
  - `admin.py` - administration, e.g. registering models
  - `tests.py` - for unit tests

New folder structure

apps.py — ~/Code/django-tutorial/todo

- todolist
  - .git
  - list
    - migrations
      - __init__.py
    - __init__.py
    - admin.py
    - apps.py
    - models.py
    - tests.py
    - views.py
  - todolist
    - __pycache__
    - __init__.py
    - settings.py
    - urls.py
    - wsgi.py
  - .gitignore
  - db.sqlite3
  - manage.py

settings.py | manage.py | admin.py | apps.py

```python
from django.apps import AppConfig


class ListConfig(AppConfig):
    name = 'list'

```

Register your app

- Add the ‘list’ app to INSTALLED_APPS list in settings.py

```
# Application definition

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

Create models in models.py

```
class Todo(models.Model):
    author = models.ForeignKey('auth.User')
    title = models.CharField(max_length=200)
    text = models.TextField()
    deadline_date = models.DateTimeField(default=timezone.now)
    completed_date = models.DateTimeField(blank=True, null=True)
    completed = False

    def complete(self):
        self.completed_date = timezone.now()
        self.completed = True
        self.save()

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

tells Django that this is a model

link to other model

different data types

operations on our model

returns string

Create models in models.py

- Full list of data types:
  - https://docs.djangoproject.com/en/1.9/ref/models/fields/#field-types

Create tables in your database

- Add the model to the database
  - Create migration file with `makemigrations:`

```
(django) $ python manage.py makemigrations list
Migrations for 'list':
  0001_initial.py:
  - Create model Todo
```

  - Apply migration file with `migrate:`

```
(django) $ python manage.py migrate list
Running migrations:
  Rendering model states... DONE
  Applying list.0001_initial... OK
```

Create tables in your database

- Migration folds are kept in the migrations folder

0001_initial.py — ~/Code/django-tutorial/todolist

```
11  class Migration(migrations.Migration):
12
13      initial = True
14
15      dependencies = [
16          migrations.swappable_dependency(settings.AUTH_USER_MODEL),
17      ]
18
19      operations = [
20          migrations.CreateModel(
21              name='Todo',
22              fields=[
23                  ('id', models.AutoField(auto_created=True, primary_k
24                  ('title', models.CharField(max_length=200)),
25                  ('text', models.TextField()),
26                  ('deadline_date', models.DateTimeField(default=djang
27                  ('completed_date', models.DateTimeField(blank=True,
```

Django admin

- Django has a built-in admin module
- First, register your models in admin.py:

settings.py | manage.py | admin.py

```
1  from django.contrib import admin
2  from .models import Todo
3
4  # Register your models here.
5  admin.site.register(Todo)
```

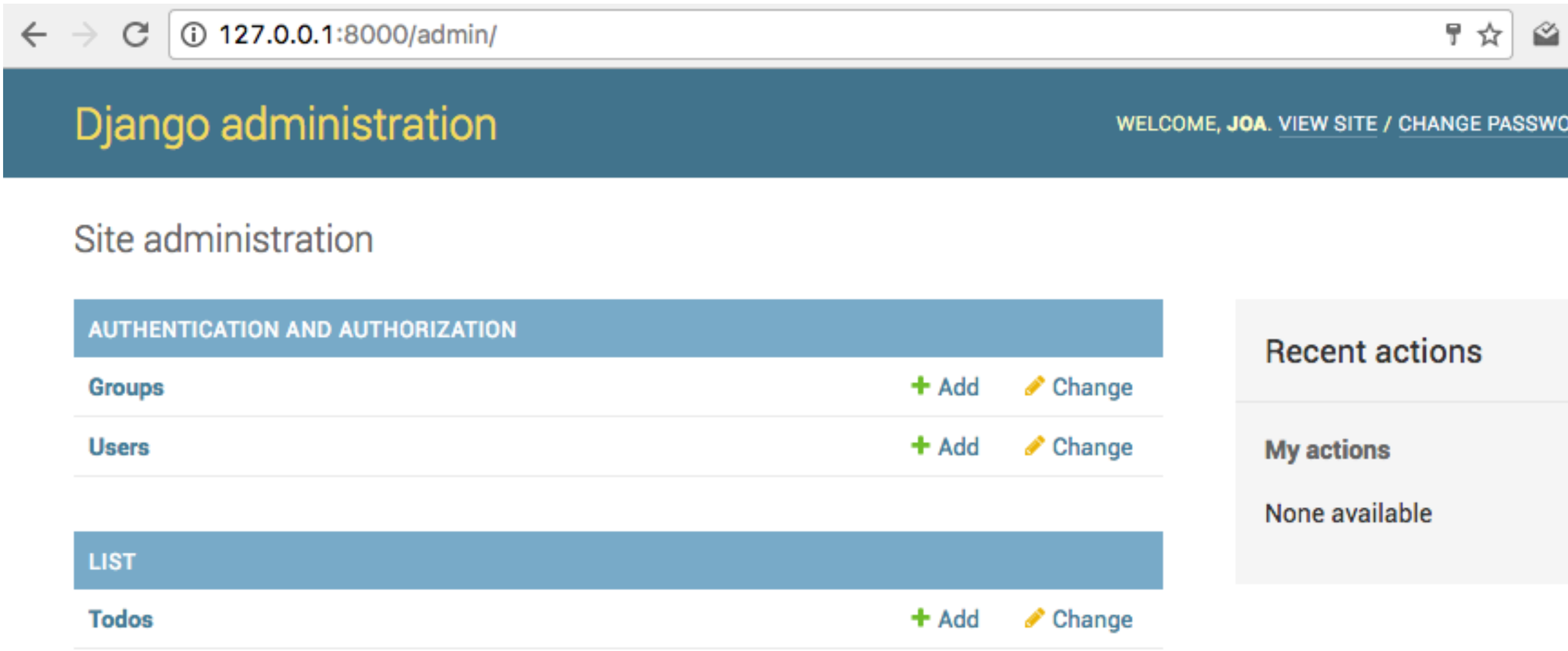
Django admin

- Django has a built-in admin module
- Register your models in admin.py.
- Create admin account with `createsuperuser:`

```
(django) $ python manage.py createsuperuser
Username (leave blank to use 'joa'):
Email address: j.vanschoren@tue.nl
Password:
Password (again):
Superuser created successfully.
```

Django admin

- Admin panel after logging in



Enter some todo's manually

Django administration

WELCOME, **JOA**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › List › Todos › Add todo

Add todo

Author: joa

Title: finish presentation

Text: then go to sleep

Deadline date: Date: 2016-12-02 Today | Time: 22:00:00 Now |

Completed date: Date: Today |

Django admin

- More info on Django Admin:
  - https://docs.djangoproject.com/en/1.9/ref/contrib/admin/

Deploying: hosting your app online

- Several services allow you to host your site for free
  - Heroku
  - PythonAnywhere
  - OpenShift
  - University server
- See the 1st week's instructions
- Many of them allow easy deployment via GitHub
  - Let's push our files to GitHub first

Pushing files to GitHub

- Initialize git (in our top project directory)

```
(django) $ git init
```

- Create .gitignore file with all the files you don't want to upload

todolist
- .git
- list
  - __pycache__
  - migrations
  - __init__.py
  - admin.py
  - apps.py
  - models.py

.gitignore | settings.py | manage.py

```
1  *.pyc
2  db.sqlite3
3  __pycache__
4  .DS_Store
5
6
```

Pushing files to GitHub

- Initialize git (in our top project directory)

```
(django) $ git init
```

- Create .gitignore file with all the files you don't want to upload
- Check, add, and commit your files

```
(django) $ git status
(django) $ git add -A
(django) $ git commit -m "First app draft"
```

Pushing files to GitHub

- Create a repo on GitHub

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner **Repository name**

WebTech2016 / django-todolist

Great repository names are short and memorable. Need inspiration? How about **improved-telegram**.

Description (optional)

Example Django app

Public

Anyone can see this repository. You choose who can commit.

Pushing files to GitHub

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH git@github.com:WebTech2016/django-todolist.git

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# django-todolist" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:WebTech2016/django-todolist.git
git push -u origin master
```

Run this

...or push an existing repository from the command line

```
git remote add origin git@github.com:WebTech2016/django-todolist.git
git push -u origin master
```

Pushing files to GitHub

WebTech2016 / **django-todolist**

Unwatch 1 St

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Setting

Example Django app — Edit

1 commit | 1 branch | 0 releases

Branch: master | New pull request | Create new file | Upload files | Find file

joaquinvanschoren First app draft — Latest commi

list	First app draft
todolist	First app draft
manage.py	First app draft

Help people interested in this repository understand your project by adding a README.

Deployment example: Heroku

- First, we'll need some special files to tell Heroku more about our project
  - Procfile, requirements.txt, runtime.txt

```
(django) $ pip install gunicorn
(django) $ pip freeze > requirements.txt
(django) $ cat Procfile
web: gunicorn todolist.wsgi:application --log-file -
(django) $ cat runtime.txt
python-3.5.2
```

Deployment example: Heroku

- Second, a buildpack needs to be created
  - Install command-line tools:
    - https://devcenter.heroku.com/articles/heroku-command-line

```
(django) $ heroku create --buildpack heroku/python
```

  - Push all to GitHub

```
(django) $ git status
(django) $ git add -A
(django) $ git commit -m "Heroku files"
```

Deployment example: Heroku

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Create New App

App Name (optional)

Leave blank and we'll choose one for you.

djangotodos

djangotodos is available

Runtime Selection

Your app can run in your choice of region in the Common Runtime.

Europe

Create App

Deploy via GitHub (allow access)

Deployment method

Heroku Git
Use Heroku CLI

GitHub
Connect to GitHub

Dropbox
Connect to D

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

View your code diffs on GitHub

Connect your app to a GitHub repository to see commit diffs in the activity log.

Deploy changes with GitHub

Connecting to a repository will allow you to deploy a branch to your app.

Automatic deploys from GitHub

Select a branch to deploy automatically whenever it is pushed to.

Create review apps in pipelines

Pipelines connected to GitHub can enable **review apps**, and create apps for new pu

Connect to GitHub

Deploy via GitHub

Deployment method

Heroku Git
Use Heroku CLI

GitHub
Connect to GitHub

Dropbox
Connect to Dropbox

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

WebTech2016

django-todolist

Search

Missing an organization? Ensure Heroku Dashboard has organization access.

Deploy via GitHub

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

master

Deploy Branch

Receive code from GitHub

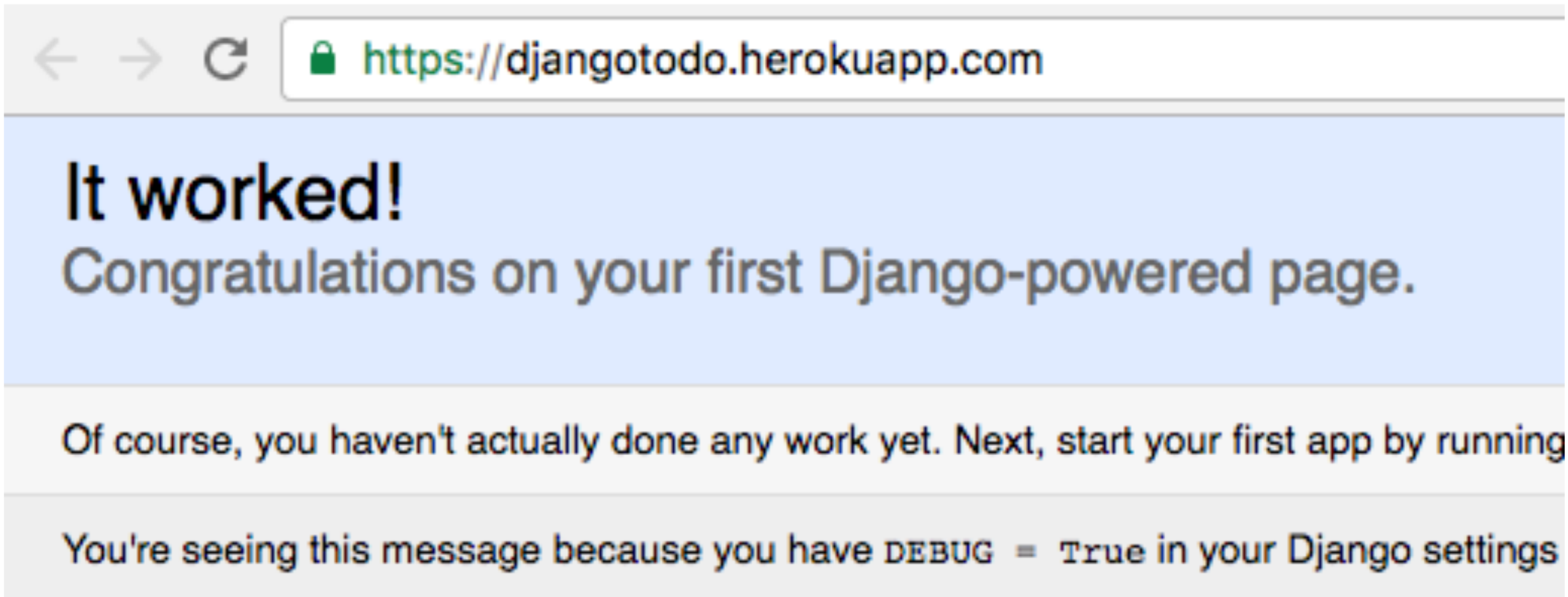
Build master Show build log

Deploy to Heroku

Your app was successfully deployed.

View

Deploy via GitHub



Deployment on PythonAnywhere

- PythonAnywhere instructions:
  - https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/
  - https://tutorial.djangogirls.org/en/deploy/

Django URLs

- Take a look at **`urls.py`**
  - Every url starting with (^) **`admin/`** is linked to the corresponding *view*

```
from django.conf.urls import url
from django.contrib import admin

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

Regex

- A note on regular expressions
  - `^ - matches beginning of a string`
  - `$ - matches end of a string`
  - `\d - matches a digit`
  - `+ - item is repeated at least once`
  - `() - part of a pattern`
- See:
  - http://regexlib.com/ > Cheat Sheet

URL files

- Instead of keeping all urls in the global urls.py file, we can create separate files per app
- Create an empty urls.py file in your app folder
- Update the global urls.py to import it:

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

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

URLs for your own views

- We will create a *todo_list* view soon
- Add the following to your app's urls.py file so that it wil be loaded when the url is an empty string (nothing after the first '/'):

```
from django.conf.urls import url
from . import views

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

Create your own views (controllers)

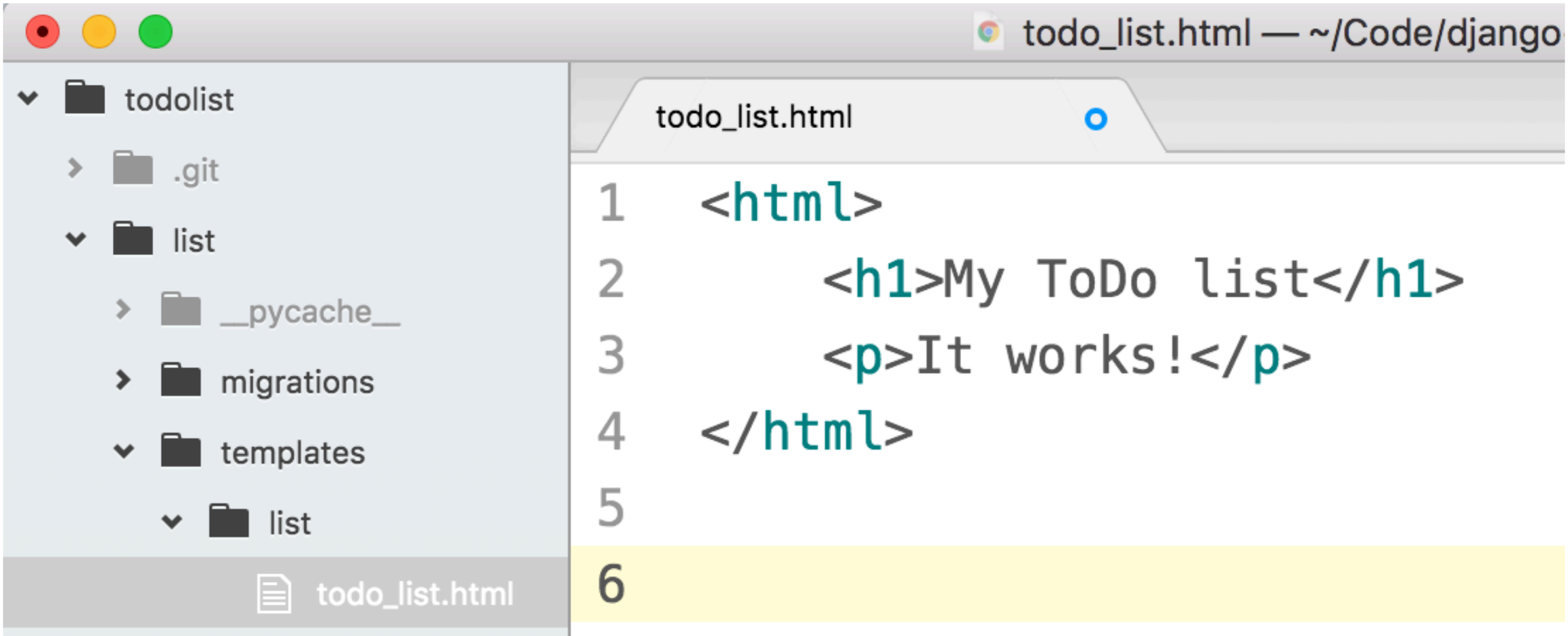
- Time to edit views.py
- Create a view to render the todo list:

```
from django.shortcuts import render

# Create your views here.
def todo_list(request):
    return render(request, 'list/todo_list.html', {})
```

Create your own templates (views)

- In your app folder, create a folder ‘templates’ and, with a subfolder ‘<your app name>’ and a template HTML file, ‘todo_list.html’
- Add some HTML that you want to show on the page



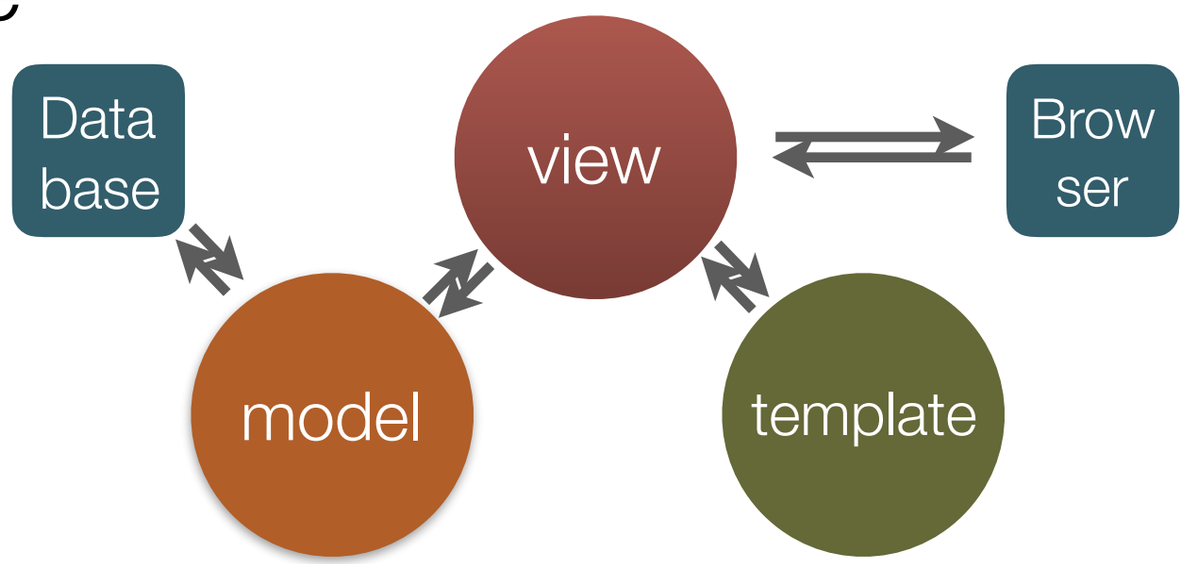
Create your own templates (views)

- Run the server

```
(django) $ python manage.py runserver
```

Connect views (controllers) to models

- Import the model in views.py
- Query for the right data -> QuerySets!
- Send the data to the template




```python
from django.shortcuts import render
from django.utils import timezone
from .models import Todo

# Create your views here.
def todo_list(request):
    todos = Todo.objects.filter(
        deadline_date__lte=timezone.now()).order_by('deadline_date')
    return render(request, 'list/todo_list.html', {'todos': todos})
```

QuerySets: how to query Django models

- QuerySet = list of objects of a given model
- Docs: https://docs.djangoproject.com/en/1.9/ref/models/querysets/
- Test queries with Django interactive console
  - Don't forget to import the models first

```
(django) $ python manage.py shell
(InteractiveConsole)
>>> from list.models import Todo
>>> Todo.objects.all()
<QuerySet [<Todo: finish presentation>, <Todo:
get exercise>]>
>>>
```

QuerySets: create objects

- Create a new todo
- You need a user to do that, get that first

```
>>> from django.contrib.auth.models import User
>>> User.objects.all()
<QuerySet [<User: joa>]>
>>> me = User.objects.get(username='joa')
>>> Todo.objects.create(author=me, title='More
to do', text='It never ends')
<Todo: More to do>
```

QuerySets: filter objects

- Filter on attribute:

```
>>> Todo.objects.filter(author=me)
<QuerySet [<Todo: More to do>, <Todo: get exercise>,
<Todo: finish presentation>]>
```

- Filter on part of text: __contains

```
>>> Todo.objects.filter(title__contains='presentation')
<QuerySet [<Todo: finish presentation>]>
```

- Filter on date

```
>>> from django.utils import timezone
>>> Todo.objects.filter(deadline_date__lte=timezone.now())
<QuerySet [<Todo: finish presentation>, <Todo: More to
do>]>
```

QuerySets: ordering and chaining

- Ordering:

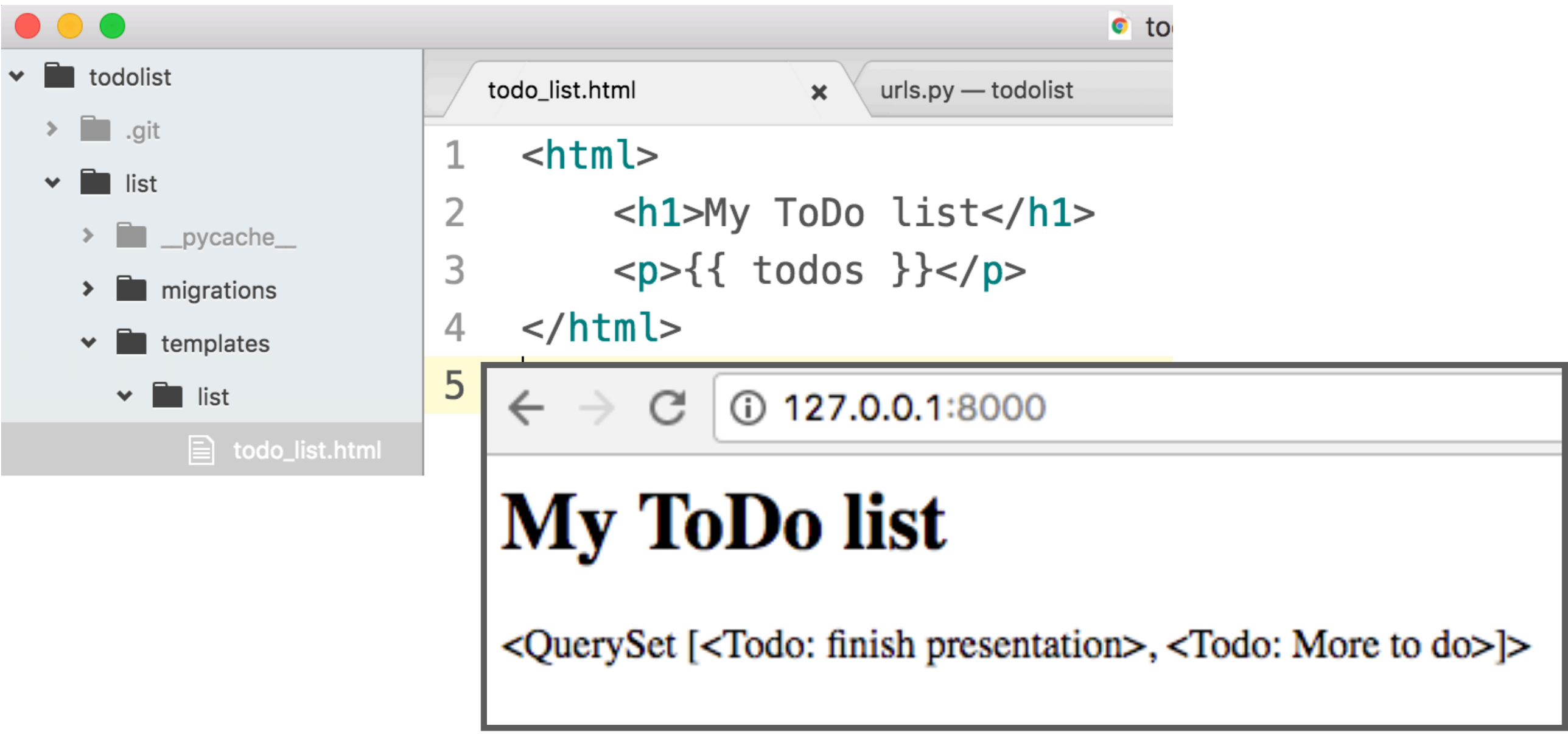
```
>>> Todo.objects.order_by('deadline_date')
<QuerySet [<Todo: finish presentation>, <Todo: More to
do>, <Todo: get exercise>]>
```

- Create complex queries by *chaining*

```
>>> Todo.objects.filter(deadline_date__lte=timezone.now())
        .order_by('deadline_date')
<QuerySet [<Todo: finish presentation>, <Todo: More to do>]>
```

Templating

- Template-tags: placeholders for data that should be fused with the HTML -> default: {{ }}



Templating

- Cheat sheet: https://code.djangoproject.com/wiki/TemplateTagsCheatSheet

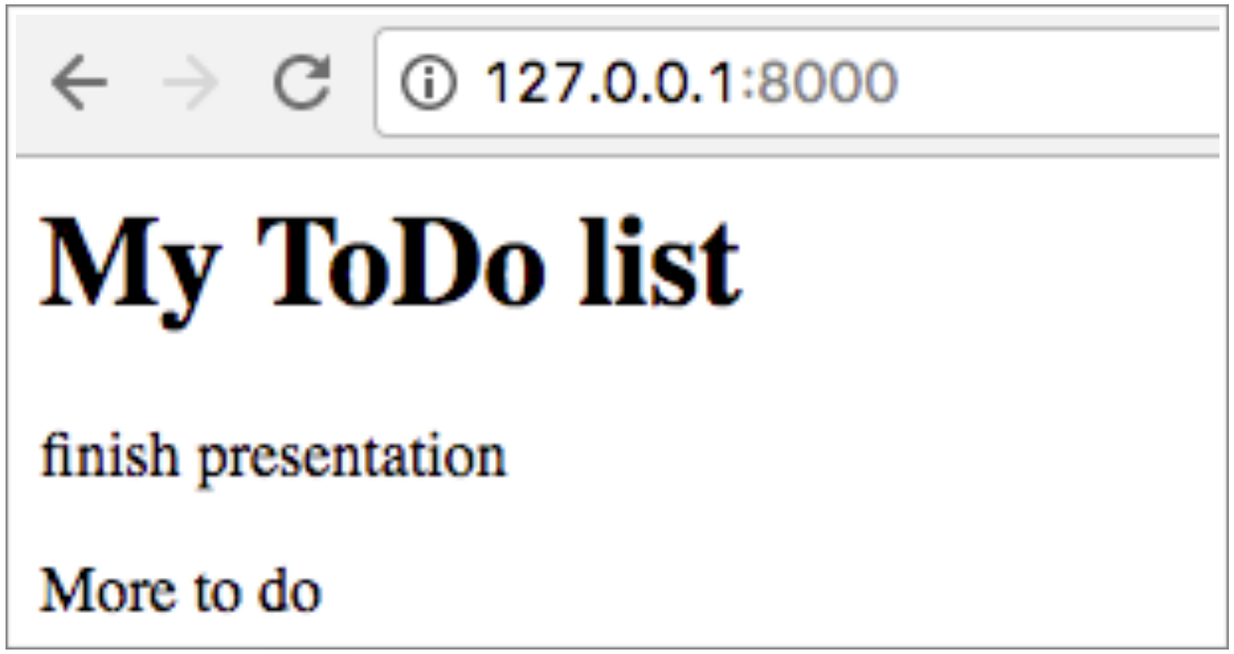
Control

- {% for o in some_list %} ... {% endfor %}
- {% ifchanged %} (Content to check status of) {% endifchanged %}
- {% if <var> %} ... {% endif %}
- {% if[not]equal user.id comment.user_id %} ... [{% else %}] ... {% endif[not]equal %}
- {% cycle row1,row2 %}
- {% cycle row1,row2 as rowcolors %} **{% cycle rowcolors %}**
- {% firstof var1 var2 var3 %}
- {% regroup people by gender as grouped %} ... {% for group in grouped %} {{ group.grouper }} .

Templating

- Let's loop over our todo's:

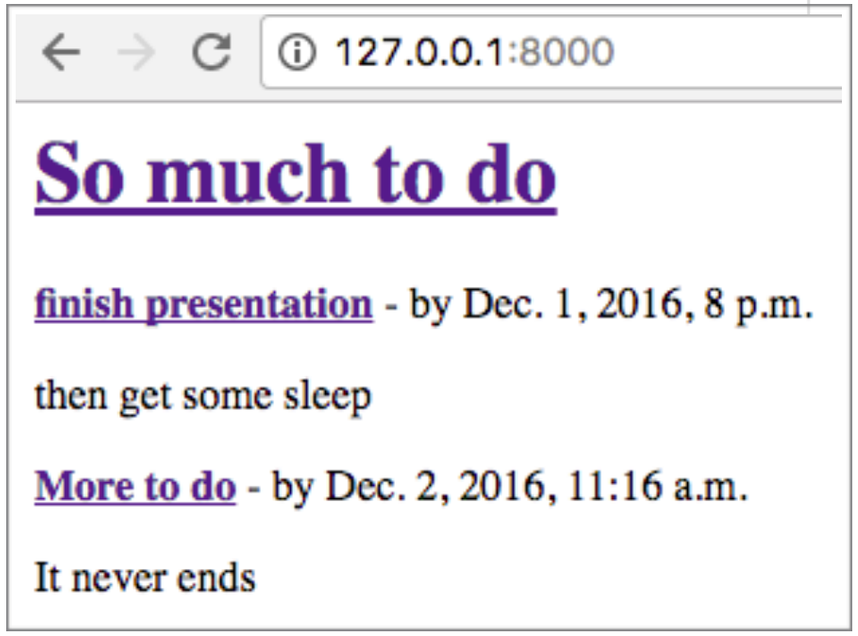
```
<html>
    <h1>My ToDo list</h1>
    {% for todo in todos %}
        <p>{{ todo }}</p>
    {% endfor %}
</html>
```



Templating

- Interleave HTML and template tags

```
<html>
<body>
  <div>
      <h1><a href="/">So much to do</a></h1>
  </div>
  {% for todo in todos %}
    <div>
        <p><b><a href="">{{ todo.title }}</a></b> - by {{ todo.deadline_date }}</p>
        <p>{{ todo.text|linebreaksbr }}</p>
    </div>
  {% endfor %}
</body>
</html>
```

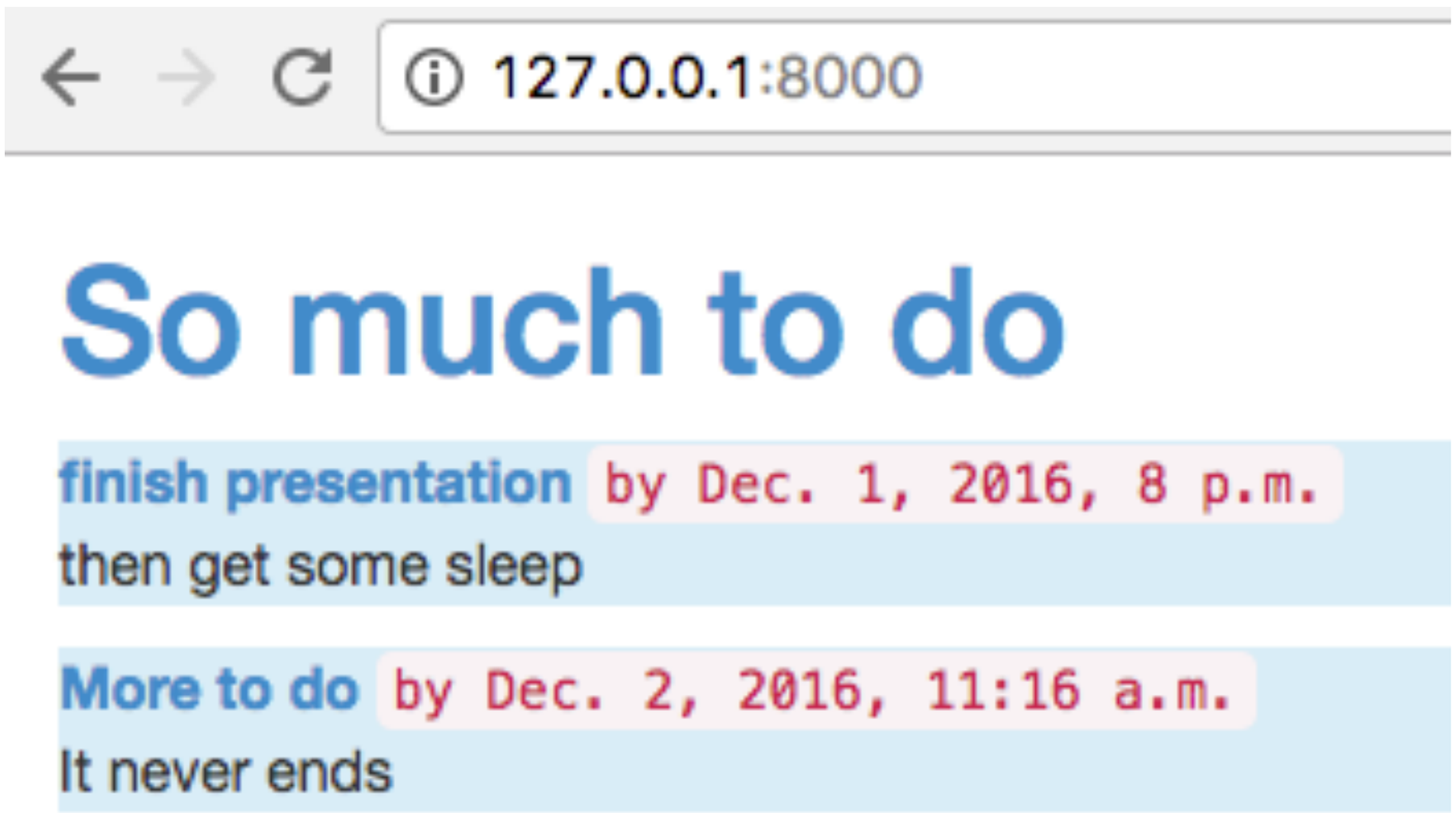


Break out bootstrap

```
<html>
<head>
  <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css
  <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css
</head>
<body>
  <div class="col-lg-12">
      <h1><a href="/">So much to do</a></h1>
      {% for todo in todos %}
        <p class="bg-info">
            <b><a href="">{{ todo.title }}</a></b> <code>by {{ todo.deadline
            {{ todo.text|linebreaksbr }}
        </p>
      {% endfor %}
  </div>
</body>
</html>
```

Break out bootstrap

- Still ugly, more work needed :)



Static files

- Static files (CSS,JS, images) go in their own ‘static’ folder

Static files

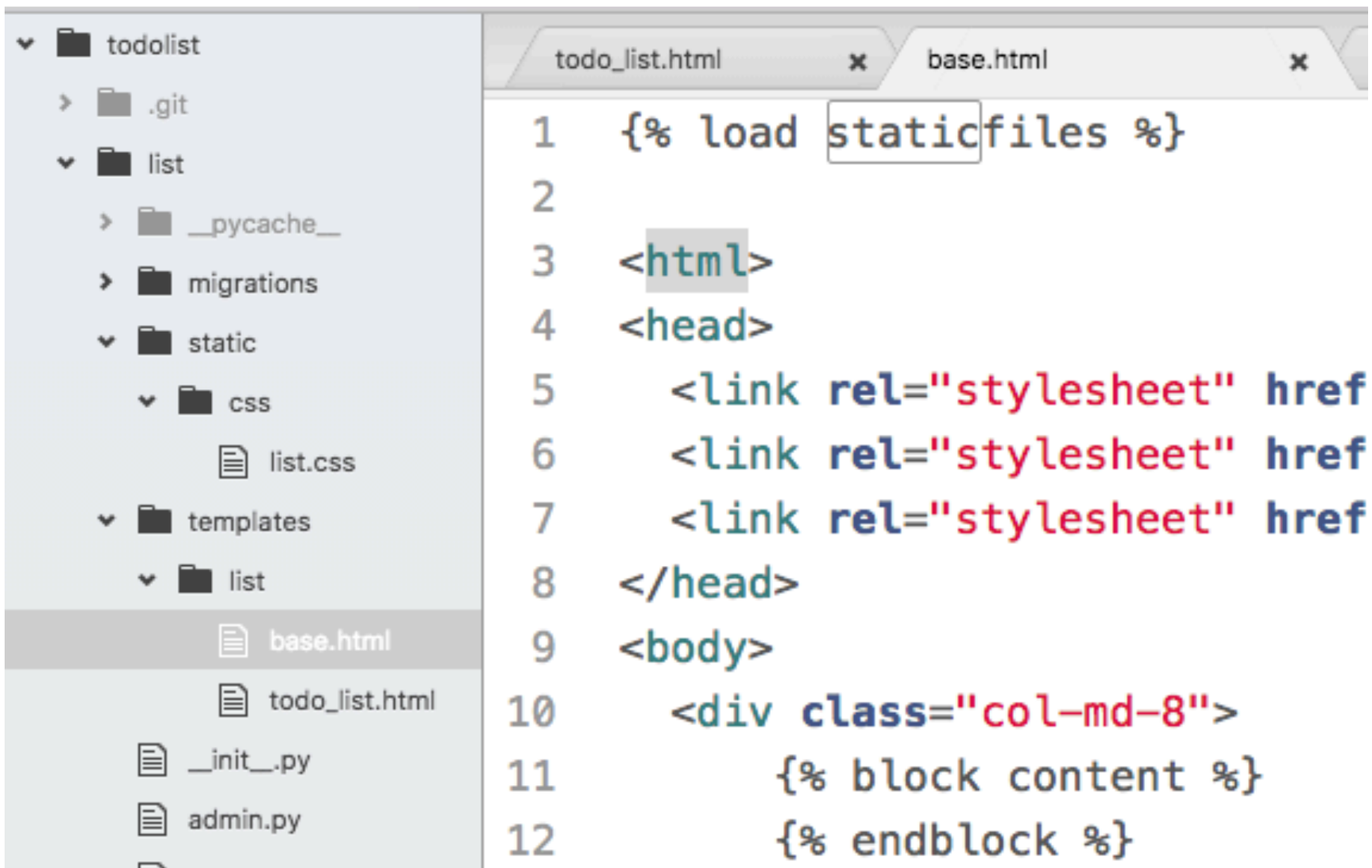
- Load static files in template: `{% load` and `{% static`

```
{% load staticfiles %}

<html>
  <head>
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap
    <link rel="stylesheet" href="{% static 'css/list.css' %}">
  </head>
  <body>
    <div class="col-lg-12">
        <h1><a href="/">So much to do</a></h1>
```

Template extending

- Reuse templates for different pages
  - Create a new base.html template with basic skeleton
  - Add a **`block`** where content should go



Template extending

- In original template, create a block with matching name
- Let it extend the base.html template

todo_list.html | base.html | list.css

```
1   {% extends 'list/base.html' %}
2
3   {% block content %}
4     {% for todo in todos %}
5       <p class="bg-info">
6           <b><a href="">{{ todo.title }}<
7           {{ todo.text|linebreaksbr }}
8       </p>
9     {% endfor %}
10  {% endblock content %}
```

Navigating between templates

- Create a new todo detail page, linked from the list
- `{% url` creates a new url, it needs
  - the path to the view we link to
  - the primary key (pk) of the todo item we want to link ti
  - in todo_list.html:

```
{% for todo in todos %}
  <p class="bg-info">
      <b><a href="{% url 'todo_detail' pk=todo.pk %}">{{ todo.title }}</a>
      {{ todo.text|linebreaksbr }}
  </p>
{% endfor %}
```

New view: route

- Create a new URL route for the view in urls.py
  - E.g. show ToDo 1 on http://myapp.com/todo/1
  - `(?P<pk>\d+)` - matches number and transfers to variable pk

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.todo_list, name='todo_list'),
    url(r'^todo/(?P<pk>\d+)/$', views.todo_list, name='todo_detail'),
]
```

New view: create view

- Add a view to view.py
- Use `get_object_or_404` to get the Todo from the model

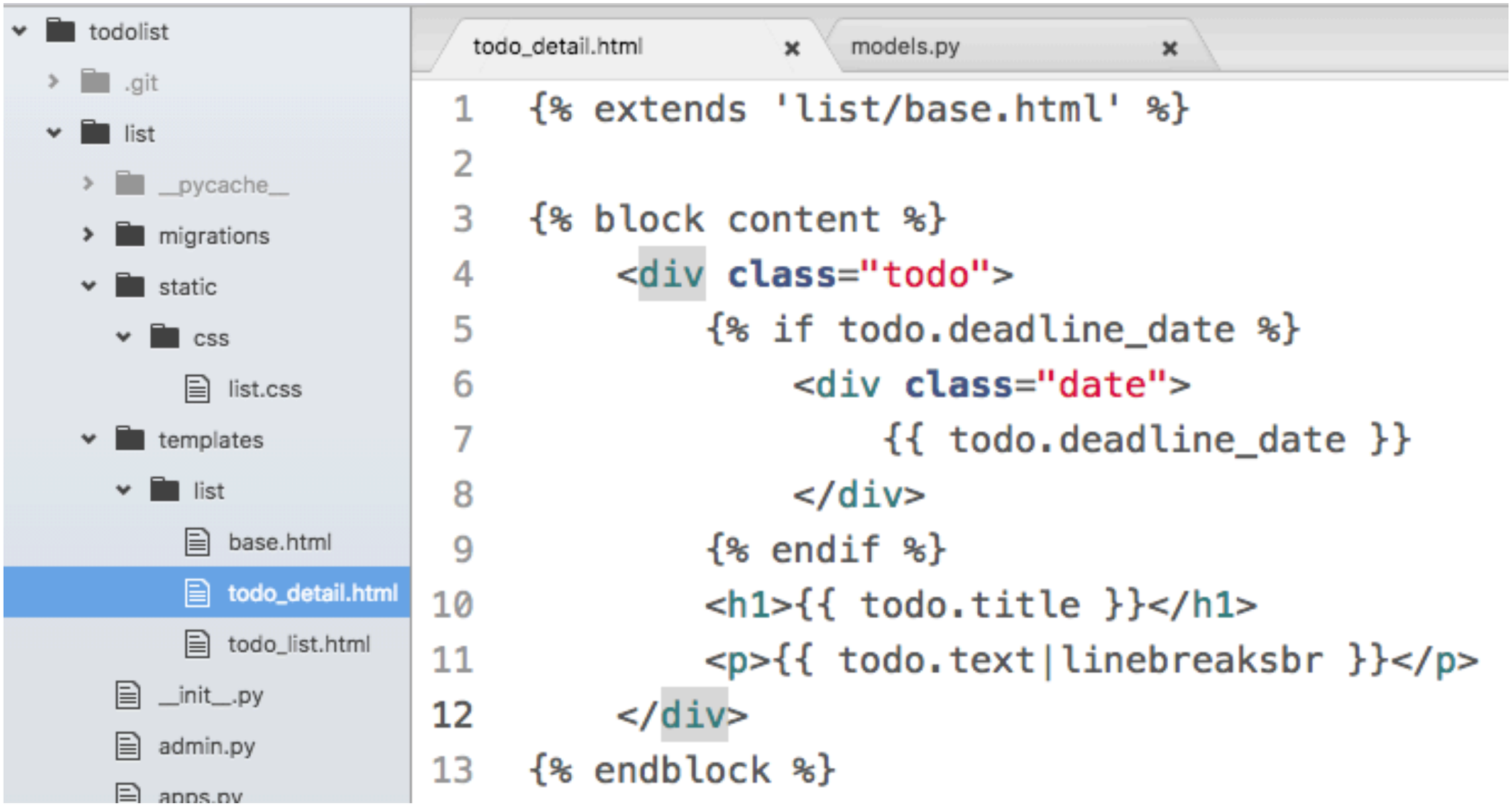
```
from django.shortcuts import render, get_object_or_404
from django.utils import timezone
from .models import Todo

# Create your views here.
def todo_list(request):
    todos = Todo.objects.filter(
        deadline_date__lte=timezone.now()).order_by('deadline_date')
    return render(request, 'list/todo_list.html', {'todos': todos})

def todo_detail(request, pk):
    todo = get_object_or_404(Todo, pk=pk)
    return render(request, 'list/todo_detail.html', {'todo': todo})
```

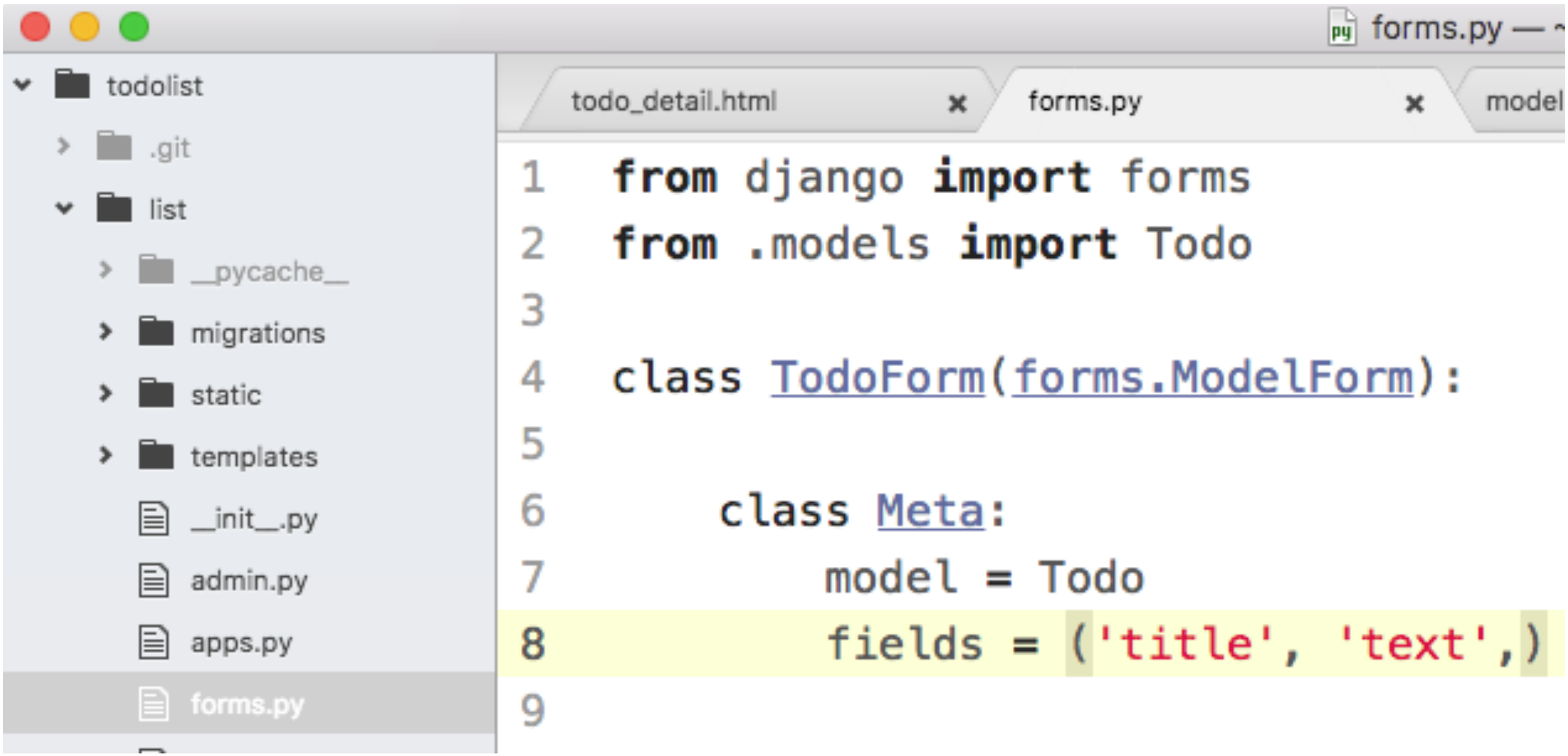
New view: create template

- All that's left is to create the new template



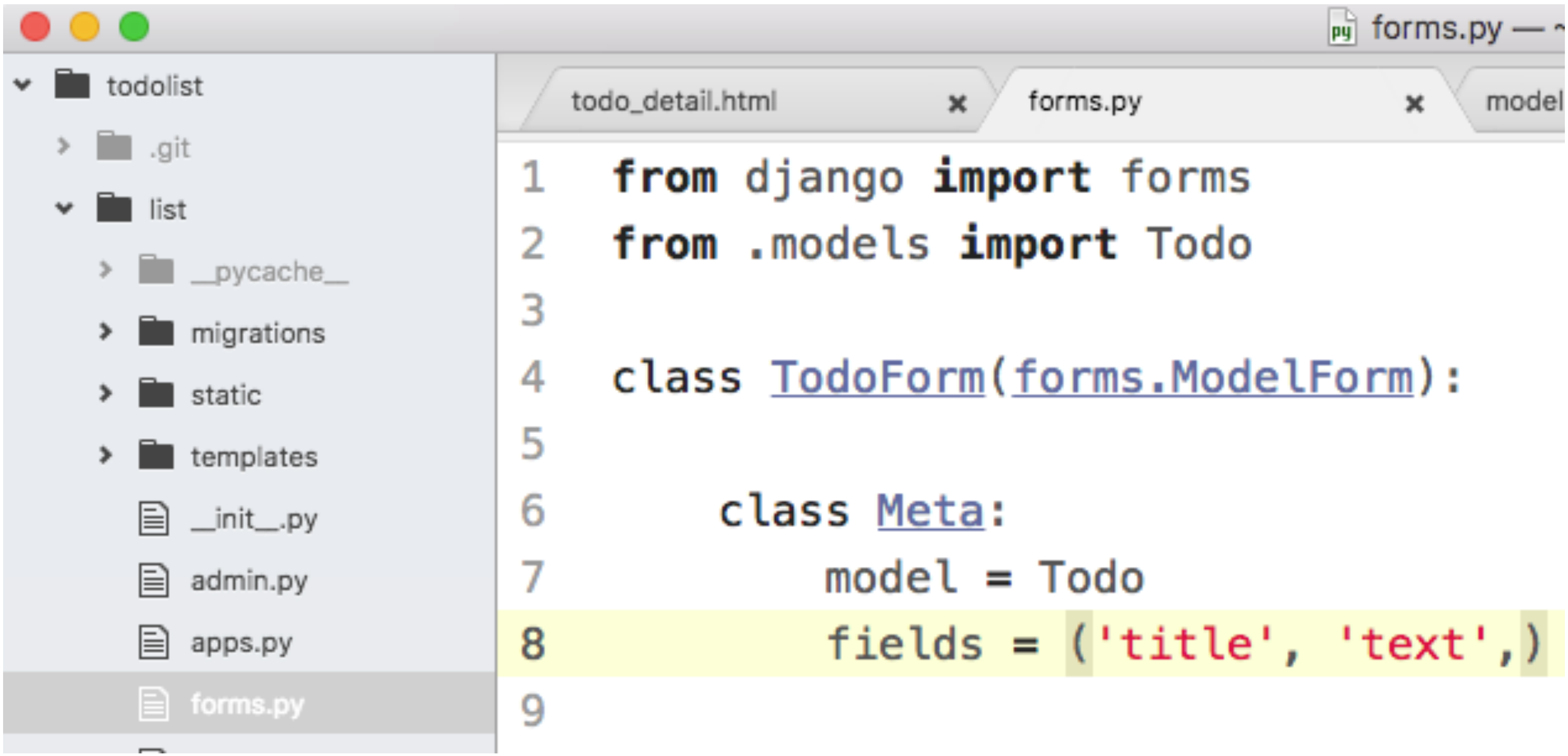
Forms

- Forms have their own file in Django. Create the following:



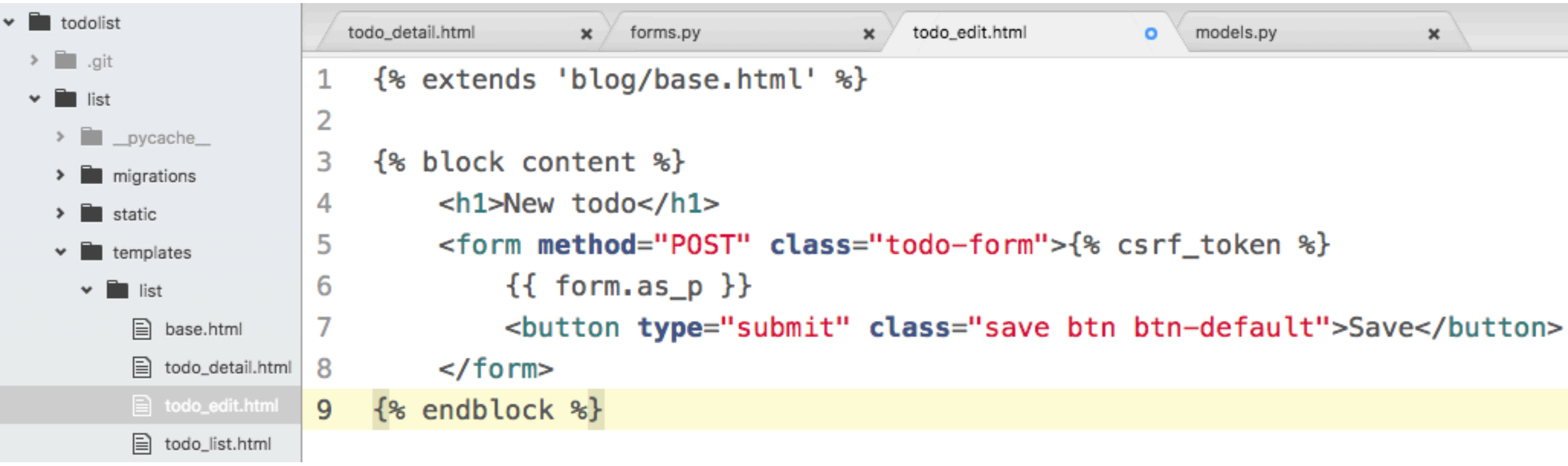
Forms

- Forms have their own file in Django. Create the following:



Forms

- Create a template



- Can you add the missing urls and views?
  - More info on forms: https://tutorial.djangogirls.org/en/django_forms/