Contents

Part I

TUTORIAL

Chapter 1

Installation

1.1 Python

As a Python Web framework, Django requires Python. The recommended installation of Python is to install Anaconda.

1.2 Database

Python includes a lightweight database called SQLite so you won't need to set up a database just yet. This step is only necessary if you'd like to work with a "large" database engine like PostgreSQL, MariaDB, MySQL, or Oracle.

1.3 Django

This is the recommended way to install Django.

1. Create a virtual python environment with `conda` (`conda create -n django`).
2. Activate the virtual environment (`source activate django`).
3. Install Django with `pip` (`python -m pip install Django`)

To test the installation of Django, run the command `python -m django --version`. If Django is installed, you should see the version of your installation. If it isn't, you'll get an error telling "No module named django".

Chapter 2

Request and response

2.1 Creating a project

```
1 django-admin startproject mysite
```

This will create the files shown in Figure 2.1:

2.2 The development server

To start the project server to listen to all IPs on port 8000:

```
1 python manager.py runserver 0:8000
```

This will produce the following website Figure 2.2:

2.3 Creating an app

Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories.

```
1 python manage.py startapp pdfs
```

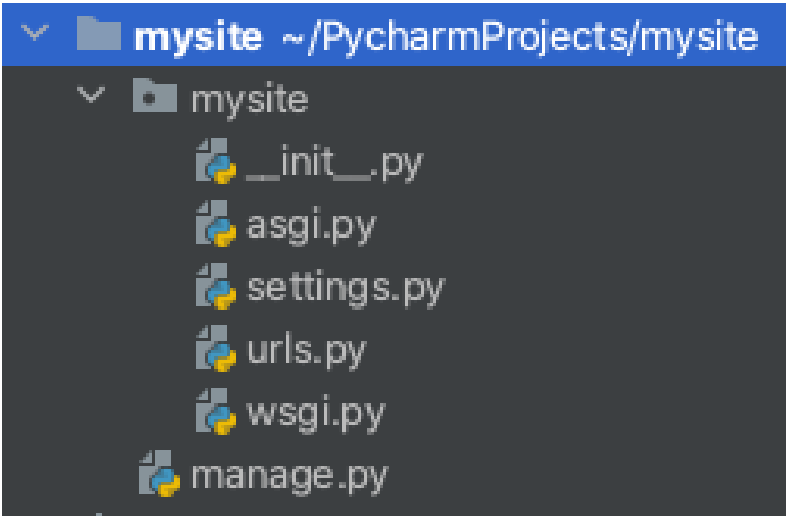

Figure 2.1: Start project

2.4 Write your first view

Open the file **polls/views.py** and put the following Python code in it:

```
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4
5 # Create your views here.
6 def index(request):
7     return HttpResponse("Hello, world. You're at the polls index.")
```

To call the view, we need to map it to a URL - and for this we need a URL conf.

To create a URLconf in the polls directory, create a file called

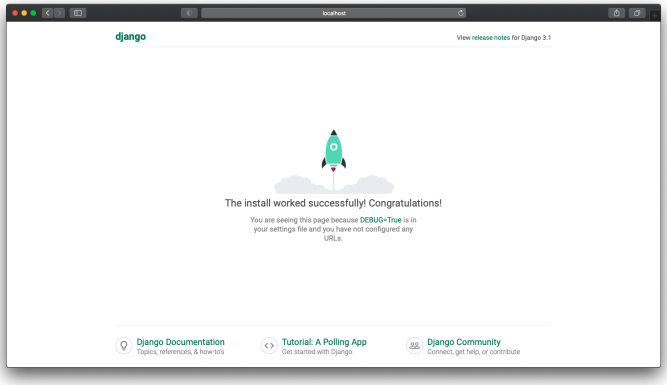

Figure 2.2: Run server

polls/urls.py. In it include the following code:

```
1 from django.urls import path
2
3 from . import views
4
5 urlpatterns = [
6     path('', views.index, name='index'),
7 ]
```

The next step is to point the root URLconf at the **polls.urls** module. In **mysite/urls.py**:

```
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('polls/', include('polls.urls')),
6     path('admin/', admin.site.urls),
7 ]
```

Rerun the server and visit http://localhost:8000/polls.

Chapter 3

Models and the admin site

3.1 Database setup

Now, open up **mysite/settings.py**. It's a normal Python module with module-level variables representing Django settings.

By default, the configuration uses SQLite. SQLite is included in Python, so you won't need to install anything else to support your database. When starting your first real project, however, you may want to use a more scalable database like PostgreSQL, to avoid database-switching headaches down the road.

While you're editing **mysite/settings.py**, set `TIME_ZONE` to your time zone.

```
TIME_ZONE = 'Asia/Shanghai'
```

3.2 Creating models

In our poll app, we'll create two models: **Question** and **Choice**. A Question has a question and a publication date. A Choice has two fields: the text of the choice and a vote tally. Each Choice is associated with a Question.

These concepts are represented by Python classes. Edit the **polls/models.py** file so it looks like this:

```
1  from django.db import models
2
3
4  # Create your models here.
5  class Question(models.Model):
6  question_text = models.CharField(max_length=200)
7  pub_date = models.DateTimeField('data_published')
8
9
10 class Choice(models.Model):
11 question = models.ForeignKey(Question, on_delete=models.CASCADE)
12 choice_text = models.CharField(max_length=200)
13 votes = models.IntegerField(default=0)
```

3.3 Activating models

To include the app in our project, we need a reference to its configuration class in the `INSTALLED_APPS` setting. The **PollsConfig** class is in the **polls/apps.py** file, so its dotted path is **'polls.apps.PollsConfig'**. Edit the **mysite/settings.py** file and add that dotted path to the `INSTALLED_APPS` setting. It'll look like this:

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

Now Django knows to include the **polls** app.

Three steps to make model changes:

1. Change your models (in **models.py**)

2. Run `python manager.py makemigrations` to create migrations for those changes

3. Run `python manage.py migrate` to apply those changes to the database

3.4 Introducing the Django Admin

3.4.1 Creating an admin user

```
1 python manage.py createsuperuser
```

3.4.2 Make the poll app modifiable in the admin

We need to tell the admin that **Question** objects have an admin interface. To do this, open the **polls/admin.py** file, and edit it to look like this:

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

The registered website is shown in Figure 3.1:

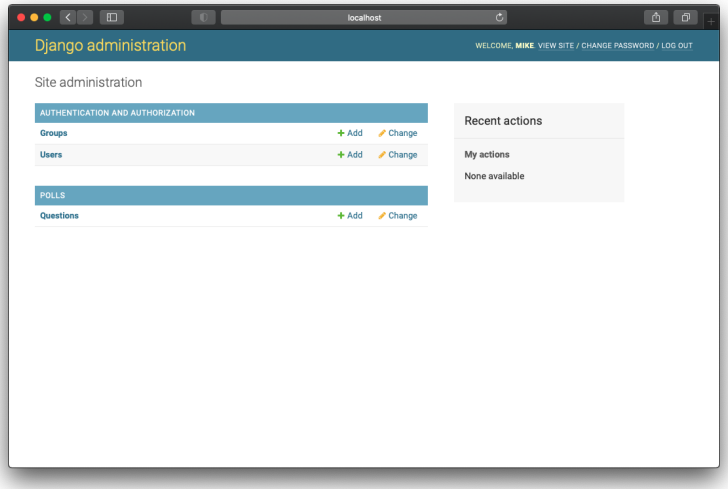

Figure 3.1: Register

Chapter 4

Views and templates

4.1 Overview

A view is a “type” of Web page in your Django application that generally serves a specific function and has a specific template.

In Django, web pages and other content are delivered by views. Each view is represented by a Python function (or method, in the case of class-based views). Django will choose a view by examining the URL that’s requested (to be precise, the part of the URL after the domain name).

To get from a URL to a view, Django uses what are known as ’URLconfs’. A URLconf maps URL patterns to views.

4.2 Writing more views

Let’s add a few more views to **polls/views.py**.

```
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4
5 # Create your views here.
6 def index(request):
7     return HttpResponse("Hello, world. You're at the polls index.")
8
```

```
9
10 def detail(request, question_id):
11     return HttpResponse("You're looking at question %s." % question_id)
12
13
14 def results(request, question_id):
15     response = "You're looking at the results of question %s."
16     return HttpResponse(response % question_id)
17
18
19 def vote(request, question_id):
20     return HttpResponse("You're voting on question %s." % question_id)
```

Wire these new views into the **polls.urls** module by adding the following **path()** calls:

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.index, name='index'),
6     path('<int:question_id>/', views.detail, name='detail'),
7     path('<int:question_id>/results/', views.results, name='results'),
8     path('<int:question_id>/votes/', views.vote, name='vote')
9 ]
```

Visite the website `localhost:8000/polls/1/` or `localhost:8000/polls/1/results` or `localhost:8000/polls/1/votes`, you will get the corresponding view.

See Figure 4.1:

4.3 Write views that actually do something

Each view is responsible for doing one of two things: returning an **HttpResponse** object containing the content for the requested page, or raising an exception such as **Http404**. The rest is up to you.

```
1 def index(request):
2     # return HttpResponse("Hello, world. You're at the polls index.")
3     latest_question_list = Question.objects.order_by('-pub_date')[:5]
4     output = ', '.join([q.question_text for q in latest_question_list])
5     return HttpResponse(output)
```

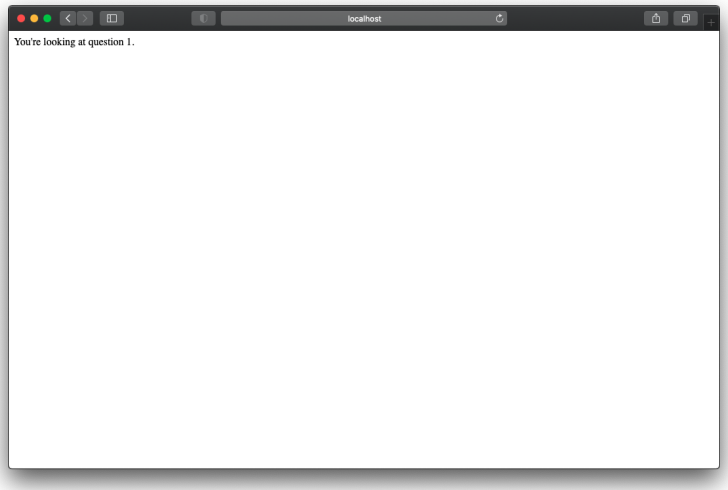

Figure 4.1: View

There's a problem here, though: the page's design is hard-coded in the view. If you want to change the way the page looks, you'll have to edit this Python code. So let's use Django's template system to separate the design from Python by creating a template that the view can use.

The structure is shown in Figure 4.2:

```
1 from django.shortcuts import render
2
3 from .models import Question
4
5
6 def index(request):
7     latest_question_list = Question.objects.order_by('-pub_date')[:5]
8     context = {'latest_question_list': latest_question_list}
9     return render(request, 'polls/index.html', context)
```

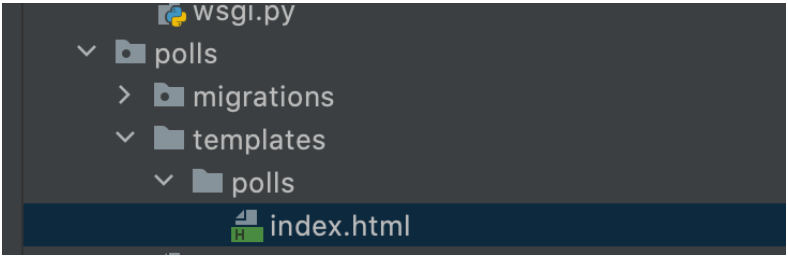

Figure 4.2: Templates

4.4 Raising a 404 error

polls/views.py:

```
from django.shortcuts import get_object_or_404, render

from .models import Question
# ...
def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})
```

polls/templates/polls/detail.html:

```
{{ question }}
```

4.5 Use the template system

polls/templates/polls/detail.html:

```
<h1>{{ question.question_text }}</h1>
```

```
<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }}</li>
{% endfor %}
</ul>
```

4.6 Removing hardcoded URLs in templates

From:

```
1 <li><a href="/polls/{{ question.id }}/">{{ question.question_text }}</a></li>
```

To:

```
1 <li><a href="{% url 'detail' question.id %}">{{ question.question_text }}</a></li>
```

4.7 Namespacing URL names

polls/urls.py:

```
1 app_name = 'polls'
```

To:

```
1 <li><a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a></li>
```

4.8 Forms

polls/tempaltes/polls/detail.html:

```
<h1>{{ question.question_text }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="{%_url_'polls:vote'_question.id_%}" method="post">
{% csrf_token %}
{% for choice in question.choice_set.all %}
    <input type="radio" name="choice" id="choice{{_forloop.counter_}}" value="{{_choice.id
        _}}">
    <label for="choice{{_forloop.counter_}}">{{ choice.choice_text }}</label><br>
{% endfor %}
<input type="submit" value="Vote">
</form>
```

polls/views.py:

```
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import get_object_or_404, render
from django.urls import reverse

from .models import Choice, Question
# ...
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request, 'polls/detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

polls/views.py:

```
from django.shortcuts import get_object_or_404, render


def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/results.html', {'question': question})
```

polls/templates/polls/results.html:

```
<h1>{{ question.question_text }}</h1>

<ul>
{% for choice in question.choice_set.all %}
    <li>{{ choice.choice_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>

<a href="{%_url_'polls:detail'_question.id_%}">Vote again?</a>
```

Chapter 5

Static files

5.1 Customize you app's look and feel

`polls/static/polls/style.css`:

```
li a {
    color: green;
}

body {
    background: white url("images/background.jpeg") no-repeat;
    background-size: 100% 100%;
}
```

Add the following at the top of `polls/static/polls/index.html`:

```
{% load static %}

<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css' %}" >
```

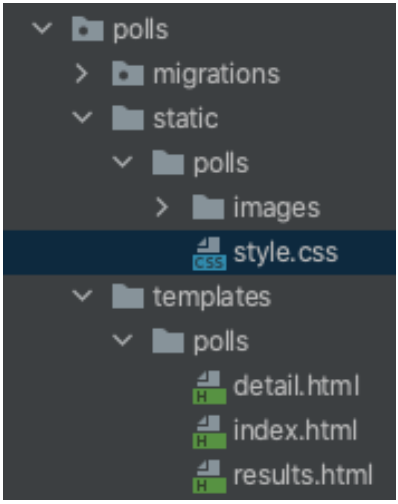

Figure 5.1: Style

Chapter 6

Customizing the admin site

polls/admin.py:

```
from django.contrib import admin

from .models import Question


class QuestionAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question_text']

admin.site.register(Question, QuestionAdmin)
```

```
from django.contrib import admin

from .models import Question


class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date_information', {'fields': ['pub_date']}),
    ]

admin.site.register(Question, QuestionAdmin)
```

6.1 Adding related objects

polls/admin.py:

```
from django.contrib import admin

from .models import Choice, Question


class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3


class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date_information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Question, QuestionAdmin)
```

6.2 Customizing the admin change list

`polls/admin.py`:

```
class QuestionAdmin(admin.ModelAdmin):
    # ...
    list_display = ('question_text', 'pub_date', 'was_published_recently')
```

6.3 Customize the admin look and feel

Open your setting file (`mysite/settings.py`) and add a **DIRS** option:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
```

```
13          },
14      },
15  ]
```

Copy the default setting file `django/contrib/admin/templates/admin/base_site.html` into the new created directory `templates/admin`. Edit the file:

```
1  {% block branding %}
2  <h1 id="site-name"><a href="{%_url_'admin:index'_%}">Polls Administration</a></h1>
3  {% endblock %}
```

Chapter 7

How to write reusable apps

7.1 Your project and your reusable app

The project structure is shown in Figure 7.1

7.2 Packaging your app

Python packaging refers to preparing your app in a specific format that can be easily installed and used. For a small app like polls, this process isn't too difficult.

1. Create a parent directory for **polls**, outside of your Django project. Call this directory **django-polls**.

2. Move the **polls** directory into the django-polls directory.

3. Create a file **django-polls/README.rst** with the following contents:

```
=====
Polls
=====

Polls is a Django app to conduct Web-based polls. For each question,
visitors can choose between a fixed number of answers.

Detailed documentation is in the "docs" directory.

Quick start
-----------

1. Add "polls" to your INSTALLED_APPS setting like this::

    INSTALLED_APPS = [
        ...
        'polls',
    ]

2. Include the polls URLconf in your project urls.py like this::

    path('polls/', include('polls.urls')),

3. Run ``python manage.py migrate`` to create the polls models.

4. Start the development server and visit http://127.0.0.1:8000/admin/
   to create a poll (you'll need the Admin app enabled).

5. Visit http://127.0.0.1:8000/polls/ to participate in the poll.
```

4. Create a **django-polls/LICENSE** file.

5. Create **setup.cfg** and **setup.py** files which detail how to build and install the app.

django-polls/setup.cfg

```
[metadata]
name = django-polls
version = 0.1
description = A Django app to conduct Web-based polls.
long_description = file: README.rst
url = https://www.example.com/
author = Your Name
author_email = yourname@example.com
license = BSD-3-Clause # Example license
classifiers =
Environment :: Web Environment
Framework :: Django
Framework :: Django :: X.Y # Replace "X.Y" as appropriate
Intended Audience :: Developers
License :: OSI Approved :: BSD License
Operating System :: OS Independent
Programming Language :: Python
Programming Language :: Python :: 3
Programming Language :: Python :: 3 :: Only
Programming Language :: Python :: 3.6
Programming Language :: Python :: 3.7
Programming Language :: Python :: 3.8
Topic :: Internet :: WWW/HTTP
Topic :: Internet :: WWW/HTTP :: Dynamic Content

[options]
include_package_data = true
packages = find:
```

django-polls/setup.py

```
from setuptools import setup

setup()
```

6. Only Python modules and packages are included in the package by default. To include additional files, we'll need to create a **MANIFEST.in** file.

```
include LICENSE
include README.rst
recursive-include polls/static *
recursive-include polls/templates *
```

7. It's optional, but recommended, to include detailed documentation with your app. Create an empty directory **django-polls/docs** for future documentation. Add an additional line to **django-polls/MANIFEST.in**

```
recursive-include doc *
```

8. Try building your package with **python setup.py sdist** (run from inside **django-polls**). This creates a directory called **dist** and builds your new package, **django-polls-0.1.tar.gz**.

7.3 Using your own package

1. To install the package

```
python -m pip install --user django-polls/dist/django-polls-0.1.tar.gz
```

2. With luck, your Django project should now work correctly again. Run the server again to confirm this.

3. To uninstall the package, use pip:

```
python -m pip uninstall django-polls
```

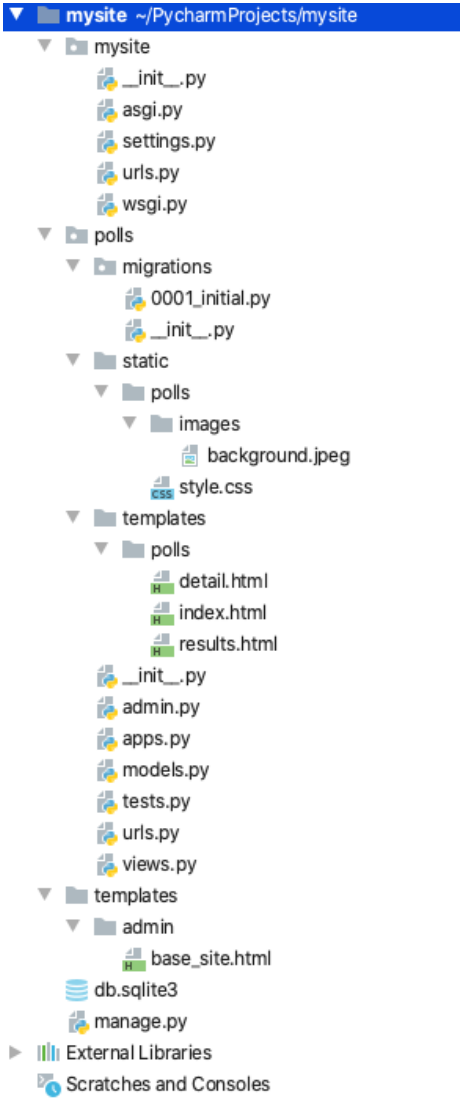

Figure 7.1: Project

Part II

BLOG PROJECT

Chapter 8

Overview

There are five applications:

- home
- pdf
- info
- technology
- plan

8.1 Install applications

settings.py:

```
INSTALLED_APPS = [
    'plan.apps.PlanConfig',
    'technology.apps.TechnologyConfig',
    'info.apps.InfoConfig',
    'home.apps.HomeConfig',
    'pdf.apps.PdfsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

8.2 Include urls

urls.py:

```
urlpatterns = [
    path('plan/', include('plan.urls')),
    path('technology/', include('technology.urls')),
    path('', include('home.urls')),
    path('info/', include('info.urls')),
    path('pdf/', include('pdf.urls')),
    path('admin/', admin.site.urls),
]
```

Chapter 9

Home

The goal of this application is to serve as the homepage.

The structure of home application is shown in Figure 9.1:

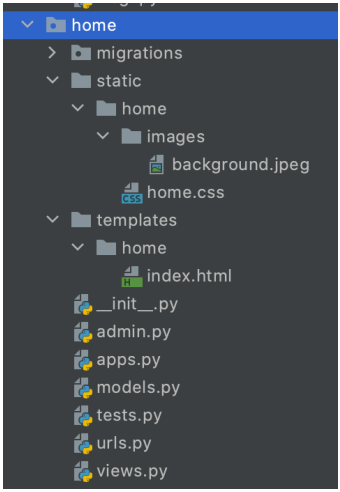

Figure 9.1: Home

9.1 index.html

```
{% load static %}
<link rel="stylesheet" type="text/css" href="{% static 'home/home.css' %}">


<html>
<head>
    <meta name="author" content="Mike Chyson">
    <meta charset="UTF-8">
    <meta name="description" content="Mike Chyson's Blog">
    <meta name="keywords" content="blog, python, ai, deep learning">
    <title>Mike Chyson</title>
    <meta http-equiv="Cache-Control" content="no-cache, no-store, must-revalidate" />
    <meta http-equiv="Pragma" content="no-cache" />
    <meta http-equiv="Expires" content="0" />
</head>

<body>
<h1 class="blog">Mike Chyson's Blog</h1>
<div class="navigator-container">
    <div class="navigator"><a href="{% url 'pdf:index' %}"><h3>PDF</h3></a></div>
    <div class="navigator"><a href="{% url 'plan:index' %}"><h3>Plan</h3></a></div
        >
    <div class="navigator"><a href="{% url 'technology:index' %}"><h3>Technology</h3>
        </a></div>
    <div class="navigator"><a href="{% url 'home:index' %}"><h3>Placeholder</h3></a
        ></div>
    <div class="navigator"><a href="{% url 'info:index' %}"><h4>Author</h4></a></
        div>
</div>

{#<div class="navigator"><a href="{% url 'plan:index' %}">Plan</a></div>#}

<div class="content">
    <div class="introduction">
        <p>
            This website for the following goals:
        </p>
        <ul>
            <li>To review what I have learned</li>
            <li>To help others with similar questions</li>
            <li>To manage my plan</li>
            <li>To visit the most new technologies conveniently</li>
        </ul>
    </div>
    <div class="toolbar">
        The time you visited this website: <br>
        {{ now_time }}<br>
        Asia/Shanghai
    </div>
</div>

```

```
48 <footer>
49     <a href="https://chyson.net"><h3>My Old Website</h3></a>
50 </footer>
51
52 </body>
53 </html>
```

Chapter 10

PDF

10.1 Model

models.py:

```
1  from django.db import models
2
3
4  # Create your models here.
5
6  class Category(models.Model):
7      category_name = models.CharField('Category', max_length=200)
8
9      def __str__(self):
10         return self.category_name
11
12
13 class PDF(models.Model):
14     name = models.CharField(max_length=200)
15     author = models.CharField(max_length=200, null=True, blank=True)
16     keywords = models.CharField(max_length=200, null=True, blank=True)
17     date = models.DateTimeField(blank=True)
18     url = models.CharField(max_length=200)
19     category = models.ManyToManyField(to=Category)
20     introduction = models.CharField(max_length=1000, null=True, blank=True)
21
22     def __str__(self):
23         return self.name
24
25
26 class Comment(models.Model):
27     pdf = models.ForeignKey(PDF, on_delete=models.CASCADE)
28     content = models.CharField('Content', max_length=10000)
29     author = models.CharField('Author', max_length=200)
30     date = models.DateTimeField('Date', auto_now_add=True)
31     email = models.CharField('Email', max_length=200, null=True, blank=True)
32
```

```
33      def __str__(self):
34          return self.content
```

`null=True` is used to allow null value in database. `blank=True` is used to allow null in admin view.

`__str__` is used to show name instead of info table primary key. This is usefull in showing foreign key.

10.2 View

views.py:

```
1  from django.shortcuts import render, get_object_or_404
2  from django.http import HttpResponse, FileResponse, Http404
3  from .models import PDF, Comment
4  from django.template import loader
5  from django.views.generic import DetailView, ListView
6  from django.core.paginator import Paginator
7
8
9  # Create your views here.
10
11 def index(request):
12      pdf_list = PDF.objects.order_by('-name').reverse()
13      paginator = Paginator(pdf_list, 10)
14      page_number = request.GET.get('page')
15      page_obj = paginator.get_page(page_number)
16      context = {
17          'page_obj': page_obj
18      }
19      return render(request, 'pdf/index.html', context)
20
21
22 def detail(request, pk):
23      pdf = get_object_or_404(PDF, pk=pk)
24      comments = Comment.objects.filter(pdf=pk)[:5]
25      context = {
26          'pdf': pdf,
27          'comments': comments
28      }
29      return render(request, 'pdf/detail.html', context)
```

Paginator is easy to use class to provide page function. The corresponding `index.html` is:

```
{% if page_obj %}
    <div class="pdf_list">
        <table>
            <tr>
                <th>PDF NAME</th>
                <th>AUTHOR</th>
                <th>KEY WORDS</th>
            </tr>
            {% for pdf in page_obj %}
                <tr>
                    <th><a href="{%_url_'pdf:detail'_pdf.id_%}">{{ pdf.name }}</a></th>
                    <th>{{ pdf.author }}</th>
                    <th>{{ pdf.keywords }}</th>
                </tr>
            {% endfor %}
        </table>
    </div>

    <div class="pagination">
    <span class="step-links">
        {% if page_obj.has_previous %}
            <a href="?page=1">&laquo; first</a>
            <a href="?page={{_page_obj.previous_page_number_}}">previous</a>
        {% endif %}

        <span class="current">
            Page {{ page_obj.number }} of {{ page_obj.paginator.num_pages }}.
        </span>

        {% if page_obj.has_next %}
            <a href="?page={{_page_obj.next_page_number_}}">next</a>
            <a href="?page={{_page_obj.paginator.num_pages_}}">last &raquo;</a>
        {% endif %}
    </span>
    </div>
{% else %}
    <p>No pdfs are available.</p>
{% endif %}
```

10.3 Url

urls.py:

```
from django.urls import path

from . import views

app_name = 'pdf'
urlpatterns = [
    path('', views.index, name='index'),
```

```
8       path('<int:pk>/', views.detail, name='detail')
9   ]
```

10.4 Install app in admin

admin.py:

```
1   from django.contrib import admin
2   from .models import PDF, Comment, Category
3
4
5   class CommentAdmin(admin.ModelAdmin):
6       list_display = ('content', 'author', 'date', 'pdf')
7       search_fields = ['content', 'author', 'pdf']
8        list_filter = ['pdf']
9
10
11  class PDFAdmin(admin.ModelAdmin):
12      list_display = ('name', 'author', 'date')
13       list_filter = ['date']
14      search_fields = ['name']
15
16
17  admin.site.register(Comment, CommentAdmin)
18  admin.site.register(PDF, PDFAdmin)
19  admin.site.register(Category)
```

Chapter 11

Favicon

This chapter is used to show an icon in my website tab.

11.1 Prepare an icon

The prepared icon is shown in Figure 11.1:

Figure 11.1: logo icon

11.2 Remove the background

You can remove the background in your icon on the website: https://www.remove.bg/upload. The icon with background revmoved is shown in Figure 11.2

Figure 11.2: icon with background removed

11.3 Generate the ico image

Generate the corresponding ico image with the image in Figure 11.2.

11.4 Add icon in html

Add the following code into your html's head.

```
1 <link rel="shortcut_icon" type="image/x-icon" href="{%_static_'home/images/favicon.ico'_%}
      " media="screen"/>
```

Chapter 12

Web

12.1 Common Header and Footer

In developing my blog, there is a need to write common headers and footer for all the index pages. At first, I copy the common header and footer to every index page, because there are only 5 index pages in my blog. But soon I find the weakness this method. If I change the header or footer, I need to copy them every again. It's time consuming and error prone.

Here is another method I take.

1. Extract the header and footer into separate file, say `header.html` and `footer.html`.

2. use the code `{% include header.html %}` to include the html file where you want to include the html episode.

> Note: The `header.html` and the `footer.html` does not contain the all html attributes. It is just the episode of the index page.

Part III

PRODUCT

Chapter 13

Deploy Django project with Apache

You can run Django project separately and it works. Then why did you deploy Django project to Apache or Nginx? Becuase Django is intended only for use while developing. It is in the business of making Web frameworks, not Web servers. Deploying Django with Apache and **mod_wsgi**[1] is a tried and tested way to get Django into production.

13.1 Apache

13.1.1 Install Apache httpd

The operating system is CentOS 8.

```
1 dnf install httpd
```

13.1.2 Start httpd

[1] wsgi: web server gateway interface

```
systemctl start httpd
```

13.1.3 Stop httpd

```
systemctl stop httpd
```

13.1.4 Enable httpd on operationg system start

```
systemctl enable httpd
```

13.2 mod_wsgi

The `mod_wsgi` package implements a simple to use Apache module which can host any Python web application which supports the Python WSGI specification.

To install mod_wsgi in CentOS 8:

```
dnf search mod_wsgi
dnf install python3-mod_wsgi
```

13.3 Configurate Apache

Create a new file `/etc/httpd/conf.d/django.conf`:

```
# you can visit the static file system
Alias /static /home/mike/blog/static
<Directory /home/mike/blog/static>
    Require all granted # permission
</Directory>

# permission to access wsgi.py file
<Directory /home/mike/blog/blog>
    <Files wsgi.py>
```

```
10          Require all granted
11      </Files>
12  </Directory>
13
14  # /home/mike/blog is your Django project
15  # /home/mike/anaconda3/envs/django/lib/python3.8/site-packages is the virtual python environment's library
16  WSGIDaemonProcess blog python-path=/home/mike/blog:/home/mike/anaconda3/envs/django/lib/python3.8/site-packages
17  WSGIProcessGroup blog
18  WSGIScriptAlias / /home/mike/blog/blog/wsgi.py
```

13.4 Permission

Give apache the right to read, write and execute the files in your django project

```
1  setfacl -R -m u:apache:rwx /home/mike/blog
```

13.5 Collect static files

In order to let django can find the static files, add the following configuration into `blog/setting.py`:

```
1  STATIC_ROOT = os.path.join(BASE_DIR, "static")
```

Then run the following command:

```
1  python manage.py collectstatic
```

This command collects all the installed application static files into "static" directory in blog.

Chapter 14

SSL certification

I use aliyun's certification service.

14.1 Generate SSL certificate

1. Login your aliyun account.
2. In Products and Services page, search SSL.
3. Select SSL Certificates.
4. Purchase Certificate. (See Figure 14.1)
5. Purchase the free certificate. (See Figure 14.2)

> Another choice is visite the website `https://letsencrypt.org/` to generate the certificate.

14.2 Download your SSL certificate

Download your SSL certificate as shown in Figure 14.3 and 14.4.

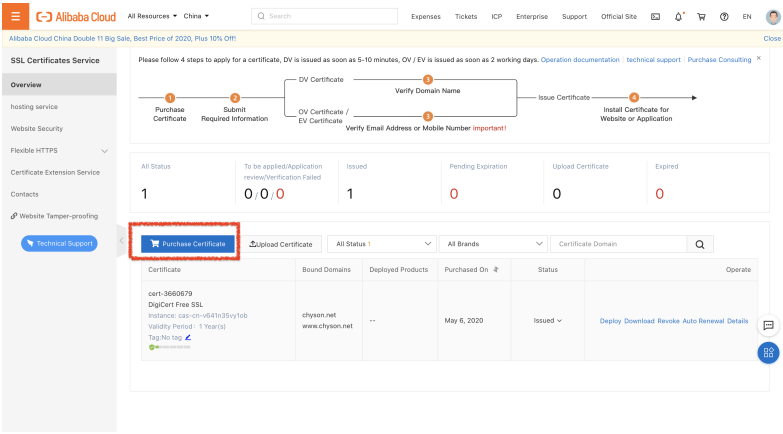

Figure 14.1: Purchase Certificate

14.3 Install SSL module

```
1 dnf install mod_ssl
```

14.4 Configurate SSL certificate in Apache

Alter the configuration file `/etc/httpd/conf/httpd.conf`:

```
1 LoadModule ssl_module modules/mod_ssl.so
```

Create a directory `/etc/httpd/cert` and put all you certificates in it.

`/etc/httpd/cert/chyson.net_public.crt`
`/etc/httpd/cert/chyson.net.key`

Figure 14.2: Purchase Free Certificate

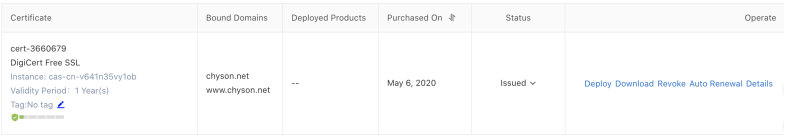

Figure 14.3: Download ssl

`/etc/httpd/cert/chyson.net_chain.crt`

Alter the file `/etc/httpd/conf.d/ssl.conf`:

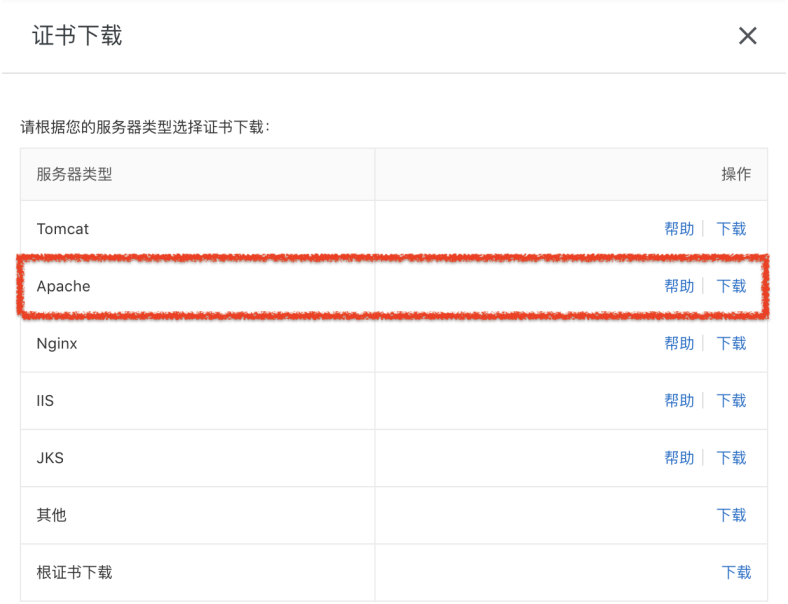

Figure 14.4: Download ssl

```
1 SSLCertificateFile /etc/httpd/cert/chyson.net_public.crt
2 SSLCertificateKeyFile /etc/httpd/cert/chyson.net.key
3 SSLCertificateChainFile /etc/httpd/cert/chyson.net_chain.crt
```

After the configuration, restart the httpd service.

```
1 systemctl restart httpd
```

14.5 Redict http to https

Alter `/etc/httpd/conf/httpd.conf`:

```
<Directory "/var/www/html">
#
# Possible values for the Options directive are "None", "All",
# or any combination of:
#   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
#
# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you.
#
# The Options directive is both complicated and important. Please see
# http://httpd.apache.org/docs/2.4/mod/core.html#options
# for more information.
#
Options Indexes FollowSymLinks

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
AllowOverride None

#
# Controls who can get stuff from this server.
#
Require all granted
</Directory>
```

to

```
AllowOverride All
```

Then restart the httpd service.

Create .htaccess file in `/var/www/html`:

```
RewriteEngine On
RewriteCond %{HTTPS}  !=on
RewriteRule ^/?(.*) https://chyson.net/$1 [R,L]
```

Chapter 15

Virtual host

Sometimes, you want to run several website on one server with only one ip. In this situation, you can use name-based virtual host.

15.1 Virtual host

To make the configuration clear, I create a two virtual hosts configuration `chyson.net.conf` and `chyson.fun.conf` separately. Here is my directory structure:

```
[root@chyson conf.d]# pwd
/etc/httpd/conf.d
[root@chyson conf.d]# ls
README          chyson.fun.conf  django.conf  userdir.conf
autoindex.conf  chyson.net.conf  ssl.conf     welcome.conf
```