Step 4: Use the full Django Web Project template

Article • 12/13/2022

Applies to: ✓ Visual Studio ✗ Visual Studio for Mac ✗ Visual Studio Code

Previous step: Serve static files, add pages, and use template inheritance

Now that you've explored the basics of Django in Visual Studio, you can easily understand the fuller app that's introduced by the "Django Web Project" template.

In this step, you now:

- ✓ Create a fuller Django web app using the "Django Web Project" template and examine the project structure (step 4-1)
- ✓ Understand the views and page templates created by the project template, which consist of three pages that inherit from a base page template and that employs static JavaScript libraries like jQuery and Bootstrap (step 4-2)
- ✓ Understand the URL routing provided by the template (step 4-3)

The template also provides basic authentication, which is covered in Step 5.

Step 4-1: Create a project from the template

1. In Visual Studio, go to **Solution Explorer**, right-click the **LearningDjango** solution created earlier in this tutorial. Then, select **Add** > **New Project**. (If you want to use a new solution, select **File** > **New** > **Project** instead.)

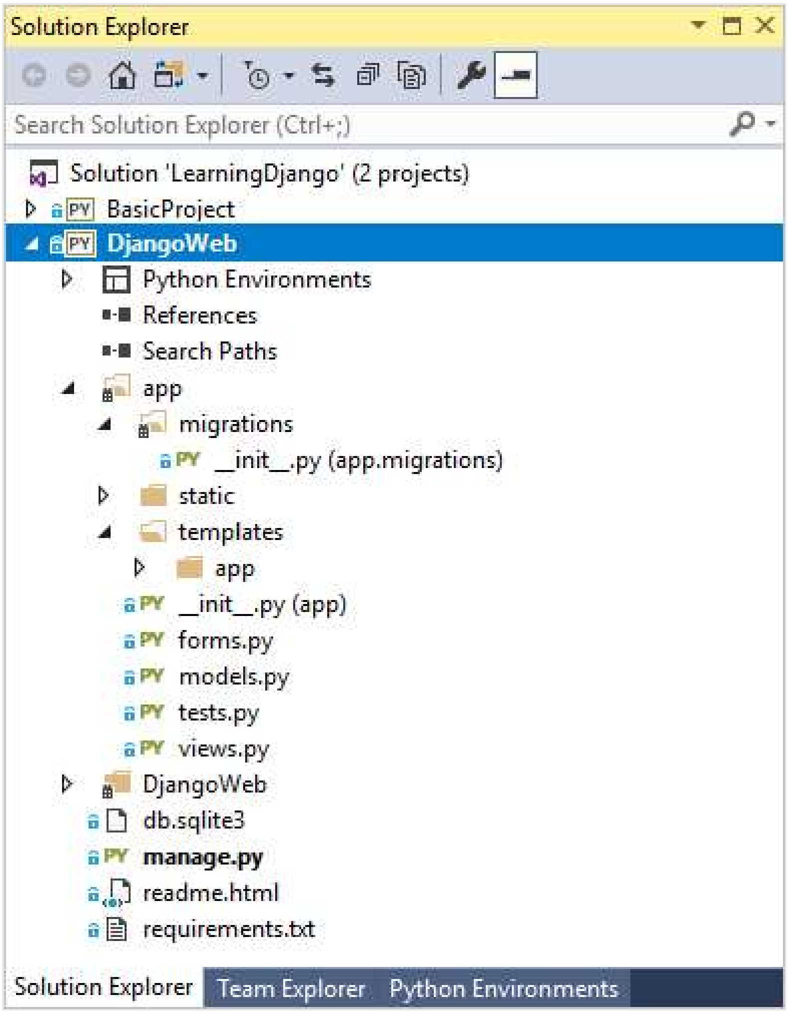
2. In the **New Project** dialog, search for and select the **Django Web Project** template. Call the project "DjangoWeb", and then select **Create**.

3. As the template includes a *requirements.txt* file, Visual Studio prompts for the location to install the dependencies. When prompted, choose the option, **Install into a virtual environment**, and in the **Add Virtual Environment** dialog select **Create** to accept the defaults.

4. When Visual Studio finishes setting up the virtual environment, follow the instructions displayed in the *readme.html* file to create a Django super user (that is, an administrator). Right-click the Visual Studio project and select **Python** > **Django**

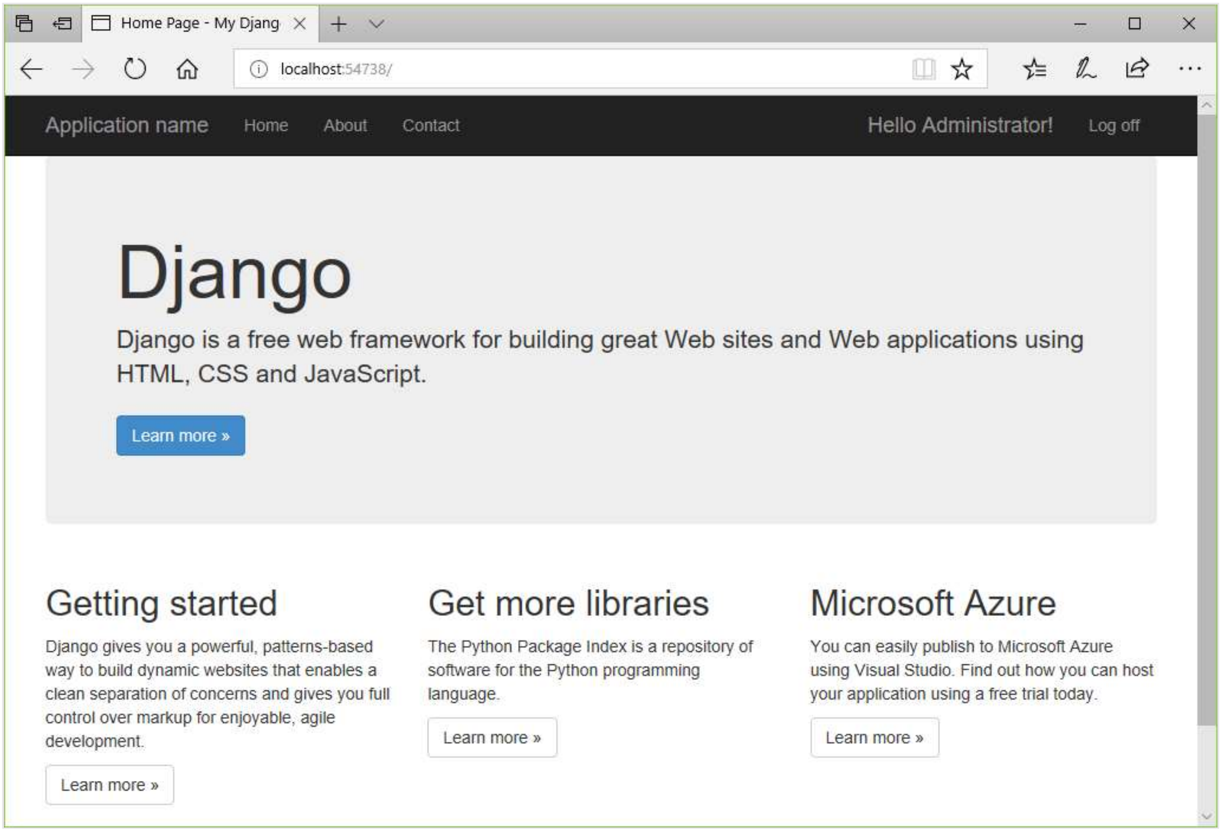
Create Superuser command, then follow the prompts. Ensure that you record your username and password as you use it when exercising the authentication features of the app.

5. Set the **DjangoWeb** project as the default for the Visual Studio solution by right-clicking the project in **Solution Explorer** and selecting **Set as Startup Project**. The startup project, which is shown in bold, is what runs when you start the debugger.



6. Select **Debug** > **Start Debugging** (**F5**) or use the **Web Server** button on the toolbar to run the server.

7. The app created by the template has three pages, Home, About, and Contact. You can navigate between the pages using the navigation bar. Take a minute or two to examine different parts of the app. To authenticate with the app through the **Log in** command, use the superuser credentials created earlier.



Django

Django is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Learn more »

Getting started

Django gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Learn more »

Get more libraries

The Python Package Index is a repository of software for the Python programming language.

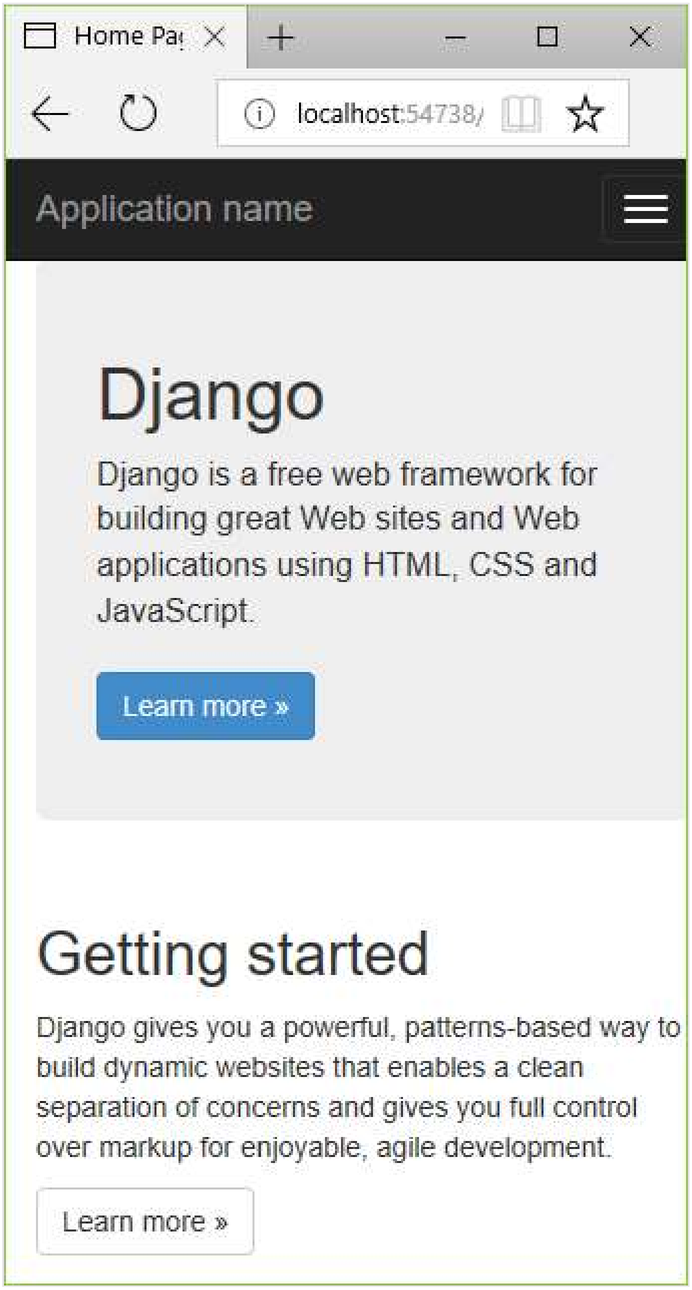
Learn more »

Microsoft Azure

You can easily publish to Microsoft Azure using Visual Studio. Find out how you can host your application using a free trial today.

Learn more »

8. The app created by the "Django Web Project" template uses Bootstrap for responsive layout that accommodates mobile form factors. To see this responsiveness, resize the browser to a narrow view so that the content renders vertically and the navigation bar turns into a menu icon.



Django

Django is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Learn more »

Getting started

Django gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Learn more »

9. You can leave the app running for the sections that follow.

   If you want to stop the app and commit changes to source control, open the **Changes** page in **Team Explorer**, right-click the folder for the virtual environment (probably **env**), and select **Ignore these local items**.

Examine what the template creates

At the broadest level, the "Django Web Project" template creates the following structure:

- Files in the project root:
  - *manage.py*: The Django administrative utility.
  - *db.sqlite3*: A default SQLite database.

  - *requirements.txt*: Contains a dependency on Django 1.x.
  - *readme.html*: A file that's displayed in Visual Studio after creating the project. As noted in the previous section, follow the instructions here to create a super user (administrator) account for the app.
- The *app* folder contains all the app files, including views, models, tests, forms, templates, and static files (see step 4-2). You usually rename this folder to use a more distinctive app name.
- The *DjangoWeb* (Django project) folder contains the typical Django project files: *__init__.py*, *settings.py*, *urls.py*, and *wsgi.py*. The *settings.py* file is already configured for the app and the database file by using the project template. The *urls.py* file is also already set up with the routes to all the app pages, including the sign in form.

Question: Is it possible to share a virtual environment between Visual Studio projects?

Answer: Yes, however, do so with the awareness that different projects likely use different packages over time. Therefore, a shared virtual environment must contain all the packages for all the projects that use it.

Nevertheless, to use an existing virtual environment, follow the steps below:

1. When prompted to install dependencies in Visual Studio, select **I will install them myself** option.
2. In **Solution Explorer**, right-click the **Python Environments** node and select **Add Existing Virtual Environment**.
3. Navigate to and select the folder containing the virtual environment, then select **OK**.

Step 4-2: Understand the views and page templates created by the project template

As you observe when you run the project, the app contains three views: Home, About, and Contact. The code for these views is found in the *views.py* file. Each view function calls `django.shortcuts.render` with the path to a template and a simple dictionary object. For example, the About page is handled by the `about` function:

Python

```python
def about(request):
    """Renders the about page."""
    assert isinstance(request, HttpRequest)
    return render(
        request,
        'app/about.html',
        {
            'title':'About',
            'message':'Your application description page.',
            'year':datetime.now().year,
        }
    )
```

Templates are located in the app's *templates/app* folder (and you usually want to rename *app* to the name of your real app). The base template, *layout.html*, is the most extensive. The *layout.html* file refers to all the necessary static files (JavaScript and CSS). The *layout.html* file also defines a block named "content" that other pages override and provides another block named "scripts." The following annotated excerpts from *layout.html* file show these specific areas:

HTML

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />

    <!-- Define a viewport for Bootstrap's responsive rendering -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ title }} - My Django Application</title>

    {% load staticfiles %}
    <link rel="stylesheet" type="text/css" href="{% static
'app/content/bootstrap.min.css' %}" />
    <link rel="stylesheet" type="text/css" href="{% static
'app/content/site.css' %}" />
    <script src="{% static 'app/scripts/modernizr-2.6.2.js' %}"></script>
</head>
<body>
    <!-- Navbar omitted -->

    <div class="container body-content">

<!-- "content" block that pages are expected to override -->
{% block content %}{% endblock %}
        <hr/>
        <footer>
```

```
            <p>&copy; {{ year }} - My Django Application</p>
        </footer>
    </div>

<!-- Additional scripts; use the "scripts" block to add page-specific scripts.
-->
    <script src="{% static 'app/scripts/jquery-1.10.2.js' %}"></script>
    <script src="{% static 'app/scripts/bootstrap.js' %}"></script>
    <script src="{% static 'app/scripts/respond.js' %}"></script>
{% block scripts %}{% endblock %}

</body>
</html>
```

The individual page templates, *about.html*, *contact.html*, and *index.html*, each extend the base template *layout.html*. The *about.html* template file is the simplest and shows the `{% extends %}` and `{% block content %}` tags:

HTML

```html
{% extends "app/layout.html" %}

{% block content %}

<h2>{{ title }}.</h2>
<h3>{{ message }}</h3>

<p>Use this area to provide additional information.</p>

{% endblock %}
```

The *index.html* and *contact.html* template files use the same structure and provide lengthier content in the "content" block.

In the *templates/app* folder is also a fourth page *login.html*, along with *loginpartial.html* that's brought into *layout.html* using `{% include %}`. These template files are discussed in step 5 on authentication.

Question: Can {% block %} and {% endblock %} be indented in the Django page template?

Answer: Yes. Django page templates work fine if you indent block tags, perhaps to align them within their appropriate parent elements. To clearly view where the block tags are placed, Visual Studio page templates don't indent the block tags.

Step 4-3: Understand the URL routing created by the template

The Django project's *urls.py* file as created by the "Django Web Project" template contains the following code:

Python

```python
from datetime import datetime
from django.urls import path
from django.contrib import admin
from django.contrib.auth.views import LoginView, LogoutView
from app import forms, views


urlpatterns = [
    path('', views.home, name='home'),
    path('contact/', views.contact, name='contact'),
    path('about/', views.about, name='about'),
    path('login/',
         LoginView.as_view
         (
             template_name='app/login.html',
             authentication_form=forms.BootstrapAuthenticationForm,
             extra_context=
             {
                 'title': 'Log in',
                 'year' : datetime.now().year,
             }
         ),
         name='login'),
    path('logout/', LogoutView.as_view(next_page='/'), name='logout'),
    path('admin/', admin.site.urls),
]
```

The first three URL patterns map directly to the `home`, `contact`, and `about` views in the app's *views.py* file. The patterns `^login/$` and `^logout$`, on the other hand, use built-in Django views instead of app-defined views. The calls to the `url` method also include extra data to customize the view. Step 5 explores these calls.

Question: In the project I created, why does the "about" URL pattern uses '^about' instead of '^about$' as shown here?

Answer: The lack of the trailing '$' in the regular expression was a simple oversight in many versions of the project template. The URL pattern works perfectly well for a page named "about." However, without the trailing '$' the URL pattern also matches URLs like "about=django," "about09876," "aboutoflaughter," and so on. The trailing '$' is shown here to create a URL pattern that matches *only* "about."

Next steps

Authenticate users in Django

Go deeper

- Writing your first Django app, part 4 - forms and generic views (docs.djangoproject.com)
- Tutorial source code on GitHub: Microsoft/python-sample-vs-learning-django