**Create and run your first Django project**

**Enable the Django plugin﻿**

This functionality relies on the Django plugin, which is bundled and enabled in PyCharm by default. If the relevant features are not available, make sure that you did not disable the plugin.

1. Press Ctrl+Alt+S to open settings and then select **Plugins**.
2. Open the **Installed** tab, find the *Django* plugin, and select the checkbox next to the plugin name.

In this tutorial, we will create a basic to-do application.

**Before you start﻿**

Make sure that the following prerequisites are met:

* You are working with PyCharm version 2022.2 or later. If you still do not have PyCharm, download it from [this page](https://www.jetbrains.com/pycharm/download/). To install PyCharm, follow the instructions, depending on your platform.

This tutorial has been created with the following assumptions:

* Python 3.11
* Django 4.2

**Creating a new project﻿**

1. Go to File | New Project or click the New Project button in the Welcome screen. The New Project Dialog opens

A screenshot of a computer

AI-generated content may be incorrect.

1. In the **New Project** dialog, do the following:
   * Specify project type *Django*.
   * If required, change the default project location.
   * Keep the default **Project venv** interpreter type.
2. Click > **More Settings** and specify *todo* in the **Application name** field.
3. Click **Create**.

**Exploring project structure﻿**

The newly created project contains Django-specific files and directories.

The structure of the project is visible in the [Project tool window](https://www.jetbrains.com/help/pycharm/project-tool-window.html) (Alt + 1):

A screenshot of a computer

AI-generated content may be incorrect.

* **myDjangoProject** directory is a container for your project. It is denoted with bold font.
* The nested directory **myDjangoProject** is the actual Python package for your project.
* **myDjangoProject/\_\_init\_\_.py**: This empty file tells Python that this directory should be considered a Python package.
* **myDjangoProject/settings.py**: This file contains [configuration for your Django project](https://docs.djangoproject.com/en/4.2/topics/settings/).
* **myDjangoProject/urls.py**: This file contains the [URL declarations for your Django project](https://docs.djangoproject.com/en/4.2/topics/http/urls/).
* **myDjangoProject/wsgi.py**: This file defines an entry-point for WSGI-compatible web servers to serve your project. For more information, refer to [How to deploy with WSGI](https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/).
* **templates** directory is empty by now. It will contain Django templates.
* The nested directory **todo** contains all the files required for developing a Django application:
  + Again, **todo/\_init\_.py** tells Python that this directory should be considered a Python package.
  + **todo/models.py**: In this file, we'll [create models](https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-django-project.html#creating_models) for our application.
  + **todo/views.py**: In this file, we'll [create views](https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-django-project.html#creating_views).
* The nested directory **migrations** contain by now only the package file **\_init\_.py**, but will be used in the future to propagate the changes you make to your models (adding a field, deleting a model, and so on) into your database schema. Read the migrations description [here](https://docs.djangoproject.com/en/4.2/topics/migrations/).
* **manage.py** is a command-line utility that lets you interact with your Django project. For more information, refer to the [Django documentation](https://docs.djangoproject.com/en/4.2/ref/django-admin/)

**Launching Django server﻿**

The [Django server run/debug configuration](https://www.jetbrains.com/help/pycharm/run-debug-configuration-django-server.html) is created automatically. If required, you can edit it by selecting the Edit Configurations command in the run/debug configuration list on the main toolbar:

A screenshot of a computer

AI-generated content may be incorrect.

For example, you can choose to open a browser window automatically when the configuration is launched:

A screenshot of a computer

AI-generated content may be incorrect.

Run the **myDjangoProject** configuration by clicking . If a browser window does not open automatically, click the link in the **Run** tool window.

The following page opens:

A screenshot of a computer

AI-generated content may be incorrect.

**Creating a model﻿**

Django *models* define the fields and behaviors of your data. They are represented by Python classes, which are subclasses of the django.db.models.Model class.

Let's create a ToDoItem model for our to-do app. To do that, open **todo/models.py**, and fill it with the following code:

from django.db import models

from django.utils import timezone

class ToDoItem(models.Model):

text = models.CharField(max\_length=100)

due\_date = models.DateField(default=timezone.now)

The model has two class variables represented by instances of field classes:

* text: an instance of the CharField class, we will use it to store the description of what should be done.
* due\_date: an instance of the DateField class, we will use it to store the deadline for the to-do.

For more information about model fields, refer to the [Django documentation](https://docs.djangoproject.com/en/4.2/ref/models/fields/).

**Running migrations﻿**

By default, PyCharm automatically creates an SQLite database for a Django project. We need to create tables in the database for the todo application and the ToDoItem model. In Django, that's done by using *migrations*. Migrations are human-editable files, in which changes to data models are stored.

* 1. Do one of the following:
     + In the main menu, go to **Tools** | **Run manage.py** task
     + Press Ctrl+Alt+R
     + Open the terminal (Alt+F12), click  **New Predefined Session** in the **Terminal** tool window toolbar and select **manage.py**.

A screenshot of a computer

AI-generated content may be incorrect.

The **manage.py** utility starts in a terminal tab

* 1. Type **makemigrations** followed by Enter.
     + You should see Process finished with exit code 0 in the console output. The **todo/migrations** directory now contains the migration file **0001\_initial.py:**

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Type **migrate** and press **Enter** to apply changes and create tables in the database for the two new models:

A screenshot of a computer program

AI-generated content may be incorrect.

* 1. To change from SqlLite to MySQL
     + Replace the DATABASES setting under settings.py

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'MyDB',  
 'USER': 'Student',  
 'PASSWORD': 'P@$$w0rd',  
 'HOST': 'localhost',  
 'PORT': '3306',  
 'OPTIONS': {  
 'init\_command': "SET sql\_mode='STRICT\_TRANS\_TABLES'",  
 }  
 }  
}

**Note:** Assume that you already have a database MyDB in local MySQL, and the admin user Student.

* + - Run **pip install mysqlclient** in the terminal window
    - Rerun Step #2 and #3
    - Check you MySQL, there should some table generated:

A screenshot of a computer

AI-generated content may be incorrect.

**Writing a view﻿**

Django *views* are functions or classes that specify how web requests are processed and which web responses are returned. By convention, views are defined in **<app\_dir>/views.py**, where *<app\_dir>* is the Django application directory.

Open **todo/views.py** and fill it with the following code:

from django.views.generic import ListView

from .models import ToDoItem

class AllToDos(ListView):

model = ToDoItem

template\_name = "todo/index.html"

Here we've defined the AllToDos class which inherits from the Django ListView basic view class. We'll use this view to display all available to-do's.

**Creating a template﻿**

A Django template is basically an html file with the elements of the [Django template language](https://docs.djangoproject.com/en/4.2/topics/templates/#the-django-template-language) in it. Templates are used to generate html pages according to the context provided by views.

You can see that todo/index.html in **views.py** is highlighted as an unresolved reference:

A screenshot of a computer

AI-generated content may be incorrect.

PyCharm suggests a [quick-fix](https://www.jetbrains.com/help/pycharm/intention-actions.html): if you hover over index.html or press AltEnter, you can choose to create the corresponding template file in the **templates** folder:

A computer screen shot of a computer screen

AI-generated content may be incorrect.

PyCharm also creates the directory **todo** where this template should reside. Confirm this operation:

A screenshot of a computer error

AI-generated content may be incorrect.

PyCharm creates and opens **todo/index.html**. To fill it with some basic html code, type html:5 followed by Tab:

A screenshot of a computer

AI-generated content may be incorrect.

Let's add the content with the Django template language elements to the template. Paste the following code between the body tags:

<h1>Things to do:</h1>

<table>

{% for item in object\_list %}

<tr>

<td>{{ item.text }}</td>

<td>{{ item.due\_date }}</td>

</tr>

{% endfor %}

</table>

There's an h1 heading and a table. We're using a for cycle to iterate through the items of object\_list which will be passed from the AllToDos view. For each item, Django will render a row in the table containing the values of text and due\_date.

**Configuring urls﻿**

Now we need to configure the way we will access the AllToDos view in the browser.

1. In the **todo** directory, [create the file](https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-python-project.html#create-file) **urls.py** and fill it with the following code:

from django.urls import path

from . import views

urlpatterns = [

path("", views.AllToDos.as\_view(), name="index")

]

1. Next, open the file **myDjangoProject/urls.py** (which PyCharm has already created for you) and configure it to include the paths from **todo/urls.py**. You should end up with the following code:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path("todo/", include("todo.urls")),

path("admin/", admin.site.urls),

]

Don't forget to import django.urls.include!

1. Open the page <http://127.0.0.1:8000/todo/> in your browser. You should see the following text:

A screenshot of a computer

AI-generated content may be incorrect.

There's a heading, but no to-do's yet. Let's use the Django admin interface to add tasks.

## **Using Django admin﻿**

Admin sites allow adding, editing, and otherwise managing the content. By default, PyCharm enables Django admin interface for a new Django project automatically.

**Setting up an admin site﻿**

1. Firstly, we need to create a superuser. To do that, type **createsuperuser** in the manage.py console (Ctrl+Alt+R), specify your email address, and password.
2. Now go to /admin/ on your Django server, for example <http://127.0.0.1:8000/admin/>. You should see the following login page:

A screenshot of a login

AI-generated content may be incorrect.

After you log in, the administration page is displayed. It has the Authentication and Authorization (Groups and Users) section, but there's no way to add to-do's. To be able to do that, we must register the ToDoItem model in the admin interface.

**Adding content﻿**

1. Open the file **todo/admin.py**, and type the following code:

from django.contrib import admin

from .models import ToDoItem

admin.site.register(ToDoItem)

1. Refresh the page in the browser. The **TODO** section with **To do items** should appear:

A screenshot of a computer

AI-generated content may be incorrect.

1. Click **Add** to create a to-do:

A screenshot of a computer screen

AI-generated content may be incorrect.

When you a ready, click **SAVE**.

The newly created to-do appears in the list as ToDoItem object (1). Such naming makes content management complicated, as you must open each task to read its description.

Let's fix that by adding a \_\_str\_\_() method to the ToDoItem model. Open **todo/models.py** and add the following:

def \_\_str\_\_(self):

return f"{self.text}: due {self.due\_date}"

You should end up with the following:

from django.db import models

from django.utils import timezone

class ToDoItem(models.Model):

text = models.CharField(max\_length=100)

due\_date = models.DateField(default=timezone.now)

def \_\_str\_\_(self):

return f"{self.text}: due {self.due\_date}"

The list of to-do's now provides all the required information:

A screenshot of a computer

AI-generated content may be incorrect.

**Providing features﻿**

Now, as you've added some to-do's, go to <http://127.0.0.1:8000/todo/> again. The list of tasks should be there:

A screenshot of a computer

AI-generated content may be incorrect.

What about filtering to display only the tasks that should be completed today? Let's add this feature to our application.

Go to **views.py** and add the TodayToDos class with the following code:

class TodayToDos(ListView):

model = ToDoItem

template\_name = "todo/today.html"

def get\_queryset(self):

return ToDoItem.objects.filter(due\_date=date.today())

This class is very similar to AllToDos, except that it uses another template (which we will create later) and implements the get\_queryset method to return only those ToDoItem objects whose due\_date is today.

date should be highlighted with a red squiggly line. Hover over it, click Import this name, and select datetime.date to add the corresponding import statement at the beginning of the file.

We need to create the **today.html** template. To avoid copy-pasting from **index.html** and having duplicated code in the project, we'll use [Django template inheritance](https://docs.djangoproject.com/en/4.2/ref/templates/language/#template-inheritance).

1. Create **base.html** in **myDjangoProject/templates** by right-clicking **templates** in the Project tool window and selecting New | HTML File from the context menu.
2. Copy the code from **index.html** into **base.html**, and then replace everything between the <body> tags with the following:

{% block content %}{% endblock %}

This is what **myDjangoProject/templates/base.html** should look like:

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport"

content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

{% block content %}{% endblock %}

</body>

</html>

1. Go to **index.html**, remove everything except the contents of the <h1> and <table> tags, and then enclose them in the Django template language elements as follows:

{% extends "base.html" %}

{% block content %}

<h1>Things to do:</h1>

<table>

{% for item in object\_list %}

<tr>

<td>{{ item.text }}</td>

<td>{{ item.due\_date }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

1. Create **myDjangoProject/templates/todo/today.html** and fill it with the following code:

{% extends "base.html" %}

{% block content %}

<h1>Things to do today:</h1>

<table>

{% for item in object\_list %}

<tr>

<td>{{ item.text }}</td>

</tr>

{% endfor %}

</table>

{% endblock %}

Note that we don't need due dates here, because this page display only those to-do's whose due dates are today.

We've created a base template **base.html** which contains all common html tags. **index.html** and **today.html** are *child templates* which inherit the common elements, but have specific contents.

We also need to configure the way we will access the today's tasks in the browser. Go to **myDjangoProject/todo/urls.py** and add path("today/", views.TodayToDos.as\_view(), name="today") to urlpatterns to get the following:

from django.urls import path

from . import views

urlpatterns = [

path("", views.AllToDos.as\_view(), name="index"),

path("today/", views.TodayToDos.as\_view(), name="today")

]

Let's make sure that everything works as expected. Go to <http://127.0.0.1:8000/admin/> and add a couple of to-do's setting the Due date to today. Then go to <http://127.0.0.1:8000/todo/today/> and make sure that you see the newly added to-do's:

A screenshot of a computer

AI-generated content may be incorrect.

**Improving the experience﻿**

At this stage our to-do application does what it was meant to:

* Displays the list of all to-do's
* Show what should be done today
* Allows adding new tasks by using Django admin interface

But it doesn't look very nice and is not very comfortable to use. Let's fix that!

The easiest way to improve the look of an html page is to implement CSS. For example, we can use [Simple CSS](https://simplecss.org/). Open **base.html** and put the following line anywhere between the <head> tags:

<link rel="stylesheet" href="https://cdn.simplecss.org/simple.min.css">

Go to <http://127.0.0.1:8000/todo/> in your browser to see the new look of our application:

A screenshot of a computer

AI-generated content may be incorrect.

Since we are using a base template, the look of <http://127.0.0.1:8000/todo/today/> changes as well:

A screenshot of a computer

AI-generated content may be incorrect.

Now let's add some navigation elements to avoid typing addresses in the browser's address bar. In **base.html**, paste the following code after the opening <body> tag:

<div>

<a class="button" href='{% url "index" %}'>All tasks</a>

<a class="button" href='{% url "today" %}'>Today</a>

</div>

A screenshot of a calendar

AI-generated content may be incorrect.

**Testing the application﻿**

You might have noticed that a few to-do's on the application's "All tasks" page are due on May 12. These tasks are past due, and we don't need to display them. Such oversights can be avoided by introducing tests while developing applications.

In the **myDjangoProject/todo** directory, there is the **tests.py** file. It is intended for Django tests. Let's write a test to check if to-do's are rendered correctly on the home page of our application depending on their due dates:

from datetime import date, timedelta

from django.test import TestCase

from django.urls import reverse

from .models import ToDoItem

def create\_todo(todo\_text, days):

return ToDoItem.objects.create(text=todo\_text, due\_date=date.today() + timedelta(days=days))

class AllToDosViewTest(TestCase):

def test\_today(self):

todo = create\_todo("To be done today", 0)

response = self.client.get(reverse("index"))

self.assertQuerySetEqual(

response.context["todoitem\_list"],

[todo]

)

def test\_last\_week(self):

todo = create\_todo("This task is past due", -7)

response = self.client.get(reverse("index"))

self.assertQuerySetEqual(

response.context["todoitem\_list"],

[]

)

def test\_next\_week(self):

todo = create\_todo("Still have some time", 7)

response = self.client.get(reverse("index"))

self.assertQuerySetEqual(

response.context["todoitem\_list"],

[todo]

)

Here, create\_todo is a shortcut function to create a to-do and AllToDosViewTest is a test class with 3 methods: test\_today, test\_last\_week, test\_next\_week. Each method creates a task with the corresponding due date, and then checks if the task is rendered on the application's home page. The task whose due date was 7 days before the current date shouldn't be displayed.

To run this test, right-click the background of the file **tests.py** in the editor, choose the option **Run**, or just press Ctrl+Shift+F10.

The test results are show in the [Test Runner](https://www.jetbrains.com/help/pycharm/test-runner-tab.html) tab of the **Run** tool window:

A screenshot of a computer

AI-generated content may be incorrect.

You can see that test\_last\_week has failed, because the created to-do is added to the list of items that will be displayed on the home page despite being past due.

Let's fix it by adding the get\_queryset method to the AllToDos class in views.py:

def get\_queryset(self):

return ToDoItem.objects.filter(due\_date\_\_gte=date.today())

The method will filter the objects, so that the view returns only those tasks whose due dates are greater than or equal to today's date (this is what \_\_gte stands for).

Rerun the test by clicking  on the toolbar of the **Run** tool window. Now all tests pass successfully:

A screenshot of a test

AI-generated content may be incorrect.

**Summary﻿**

This tutorial is over. You have successfully created, launched, and tested a simple Django application. Let's sum up what you have done with the help of PyCharm:

* Created a Django project with an application
* Launched a Django server
* Created models, views and templates
* Configured urls
* Launched the application
* Created and executed a test