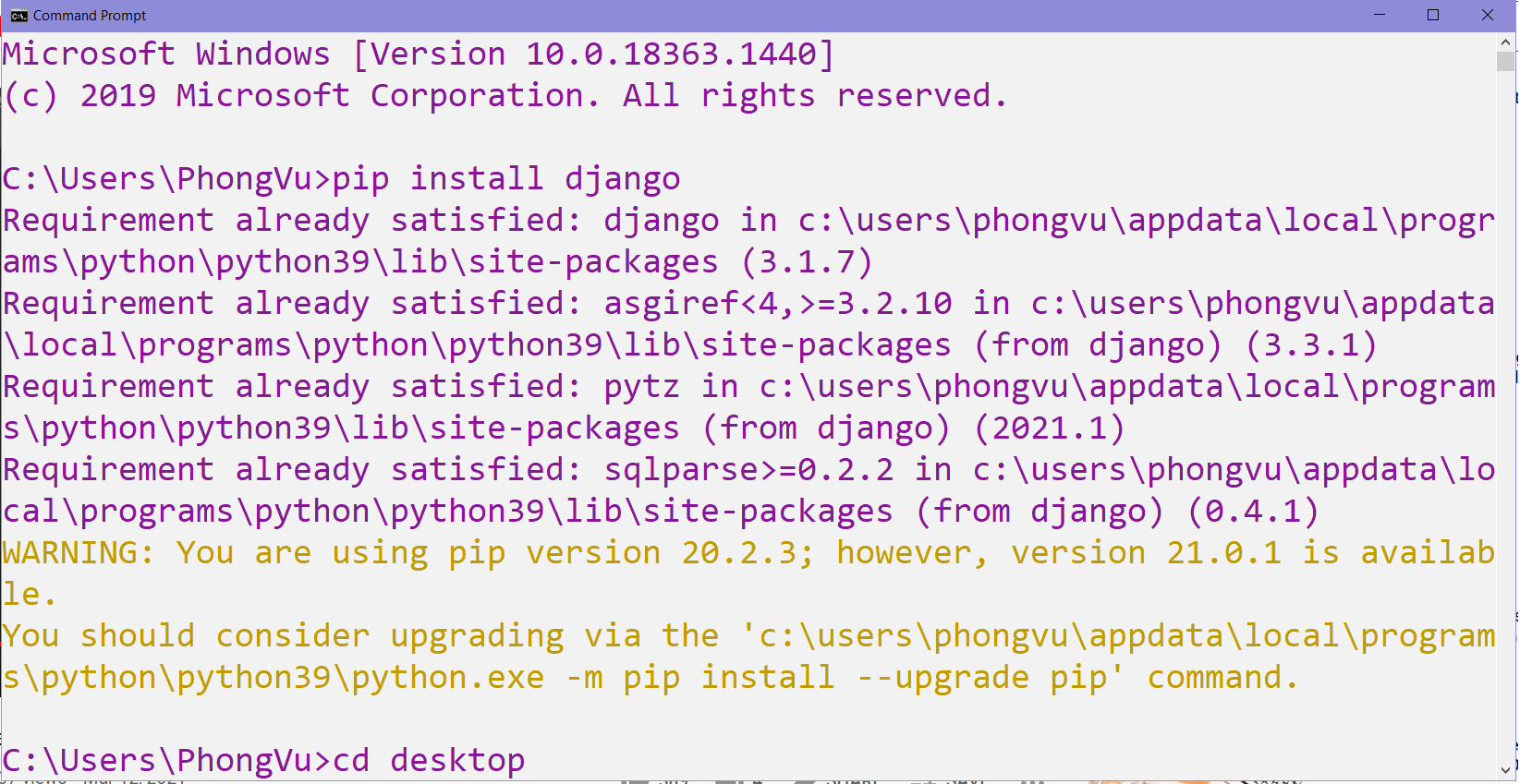
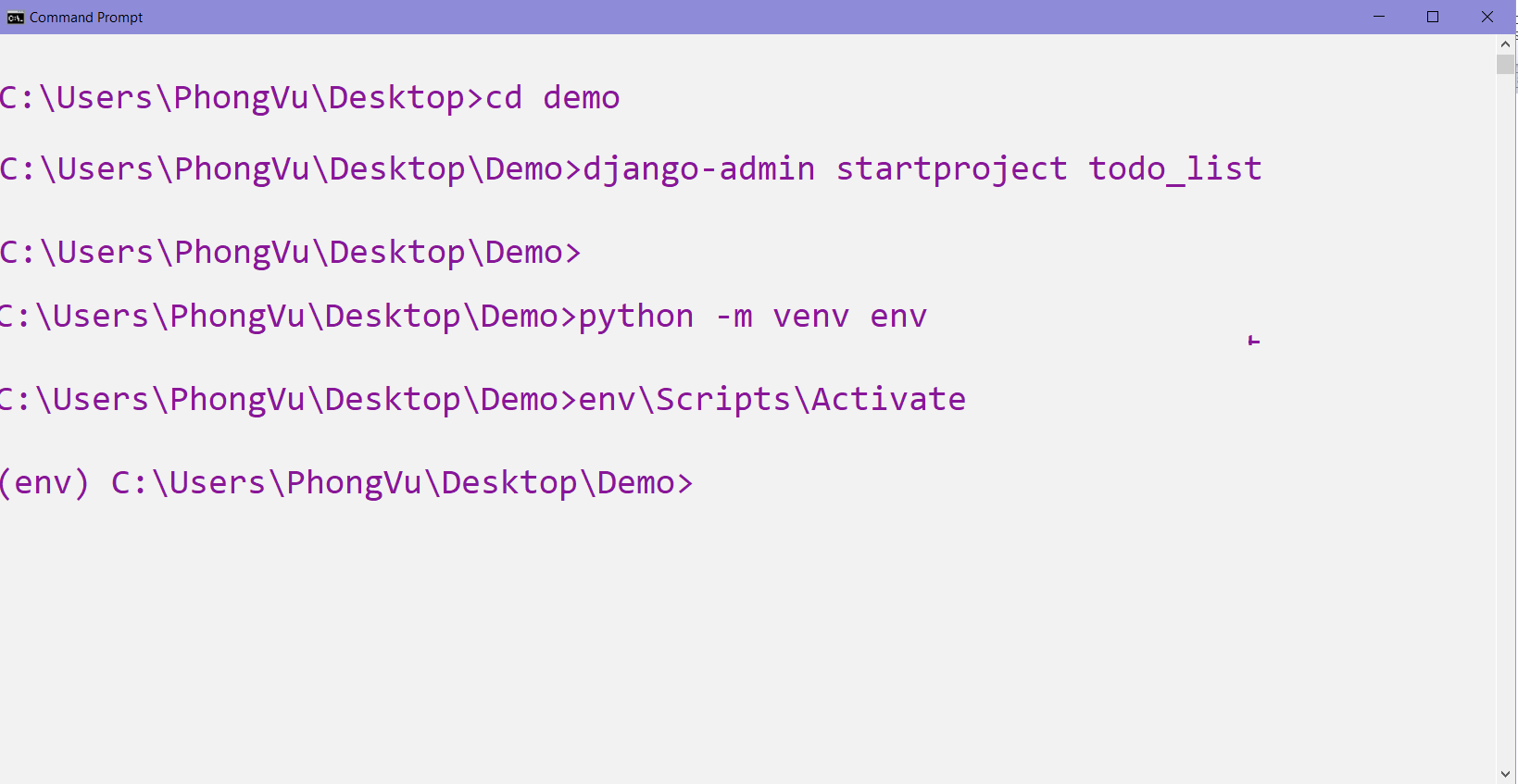
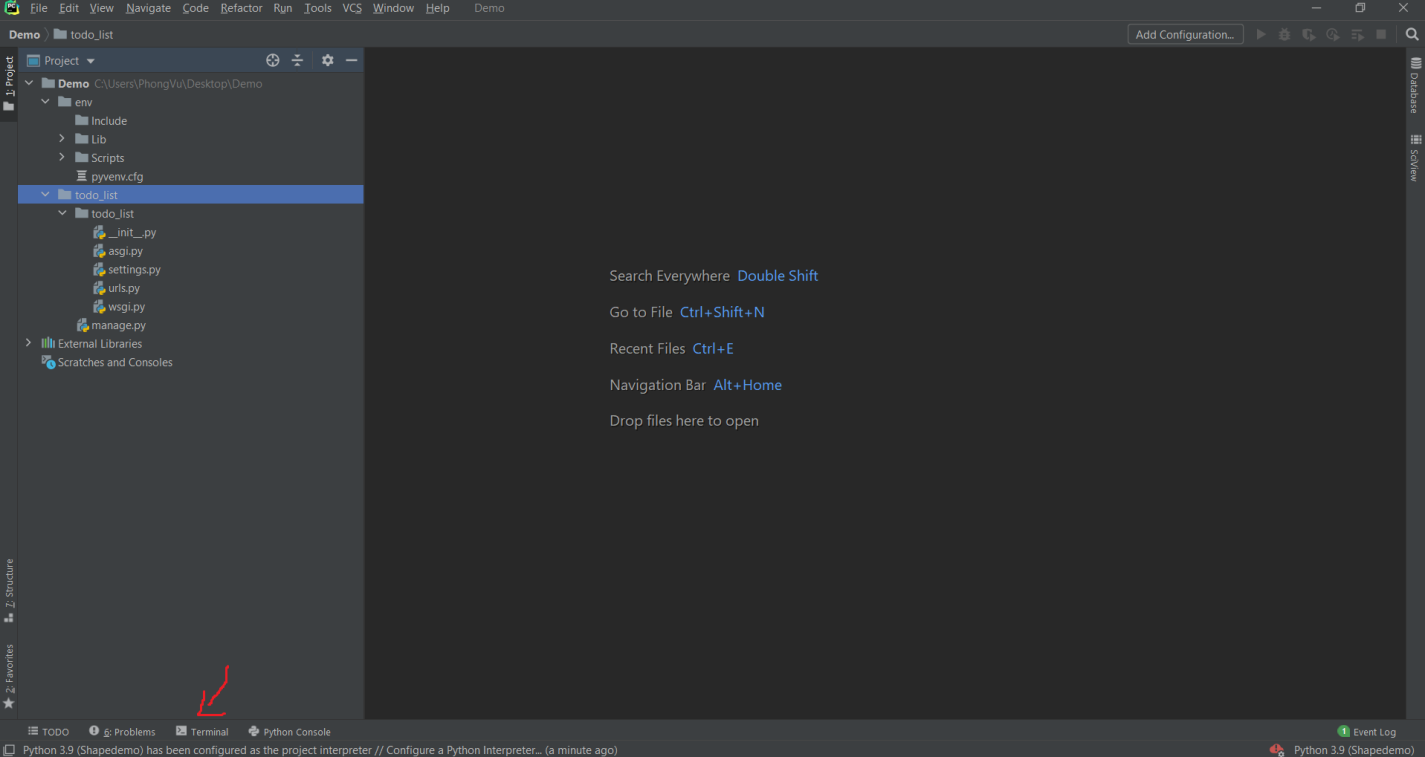
1. Setup project

Để bắt đầu vào việc, ta khởi động CMD, gõ câu lệnh pip install Django để cài đặt Django cho project.

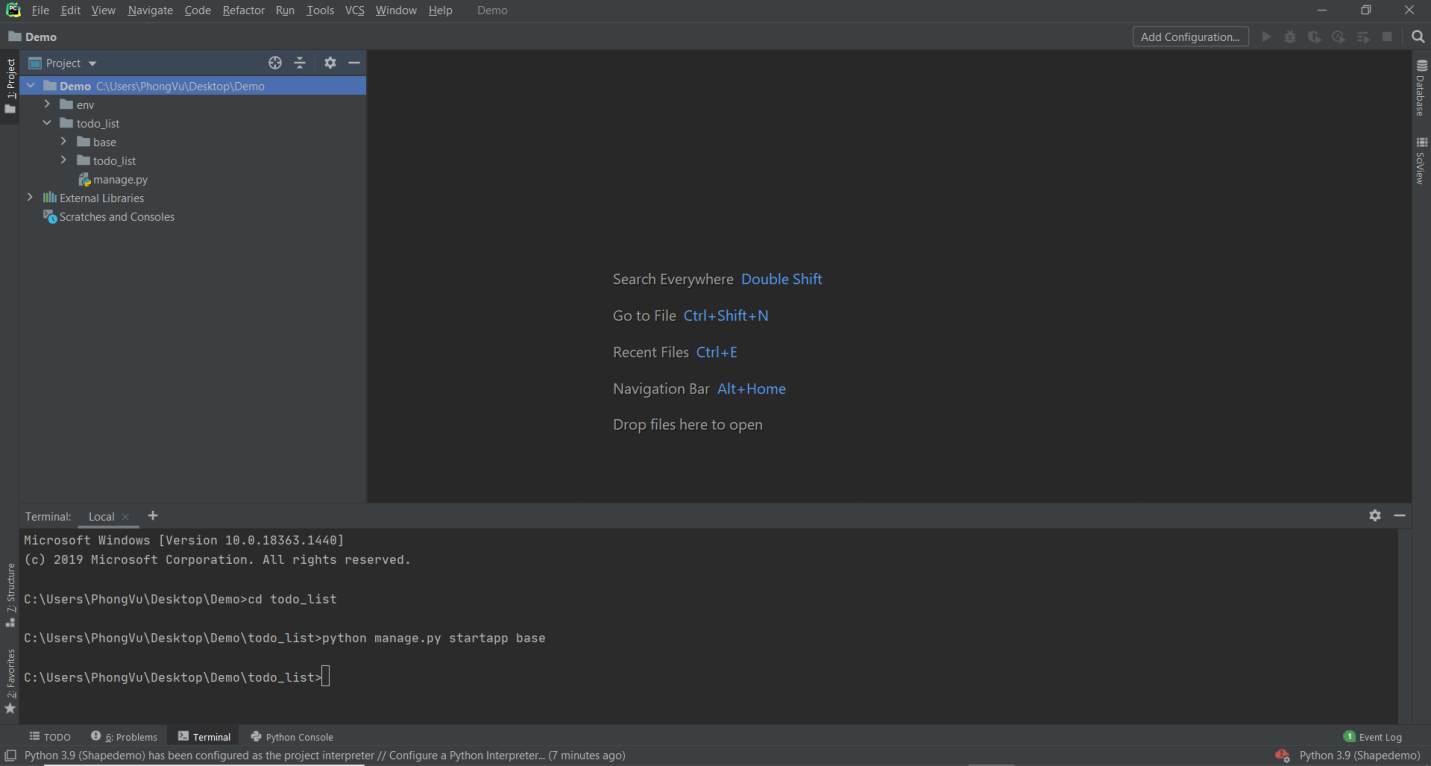


Sau đó di chuyển vào thư mục cần tạo dự án. Tạo dự án bằng câu lệnh django-admin startproject todo\_list . Sau đo di chuyển vào project mà đã tạo. Vào môi trường ảo bằng cú pháp python –m venv env . Chờ một chút xíu để máy tính thiết lập môi trường. Sau đó Activate bằng câu lệnh \env\Scripts\Activate

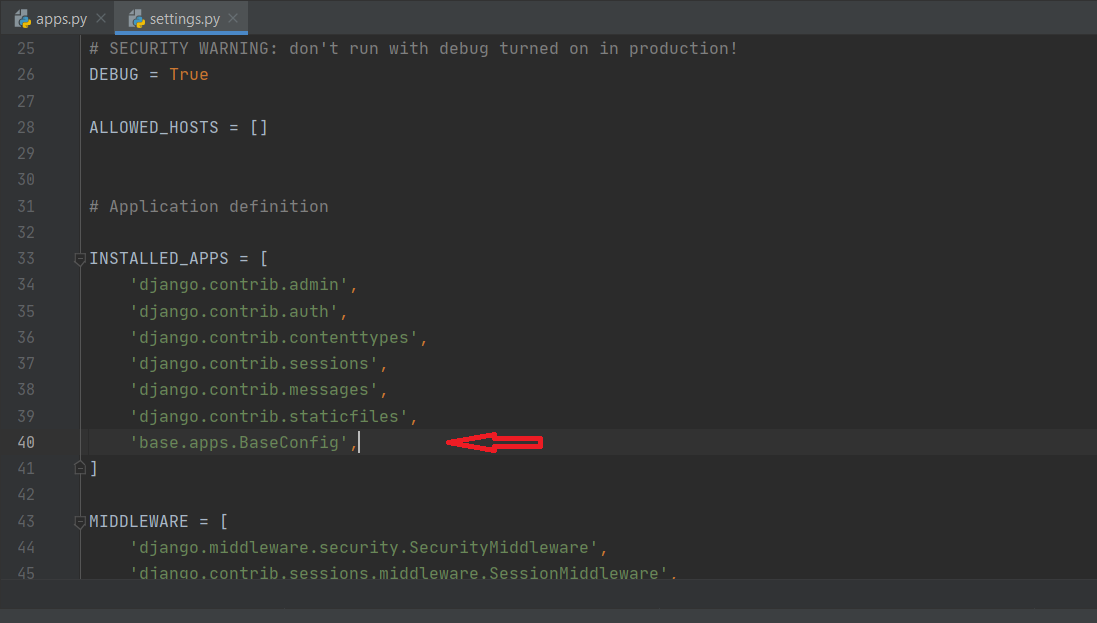


Mở trình soạn thảo code trên máy bạn, ở đây ta sử dụng PyCharm, terminal ở trong PyCharm

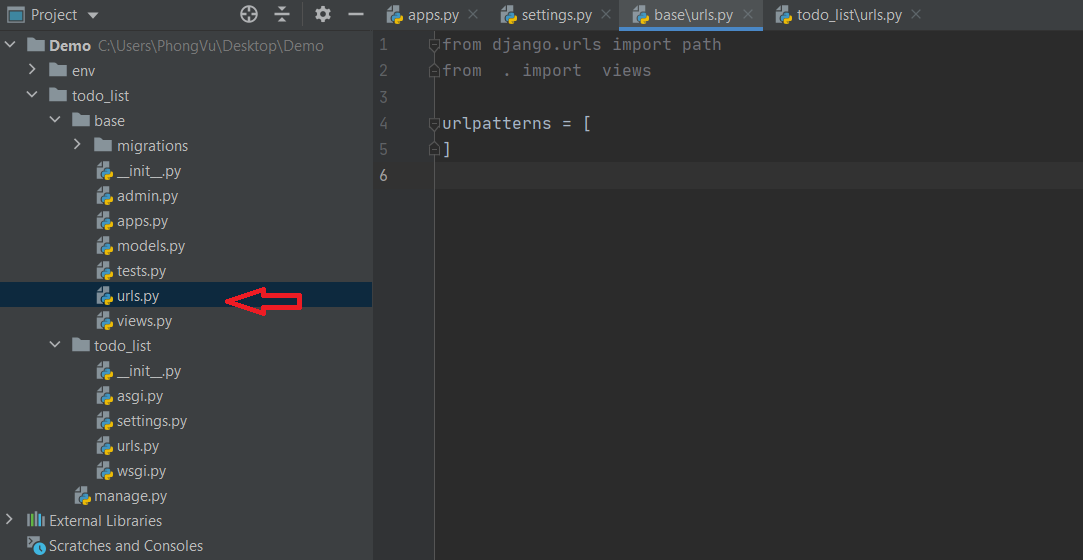
di chuyển vào thư mục todo\_list, khởi tạo app bằng câu lệnh python manage.py startapp base



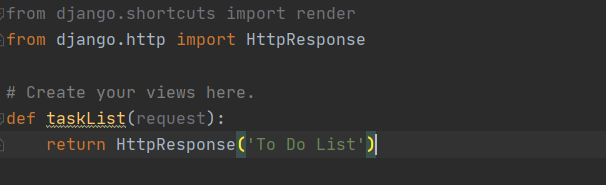
Mở file setting.py trong thư mục todo\_list để đăng kí app base vào project



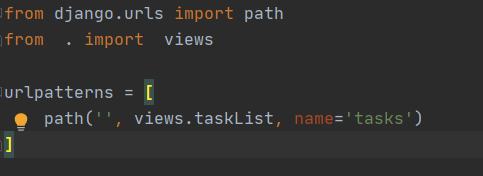
Trong thư mục base tạo thêm file urls.py và thêm các cú pháp cần thiết

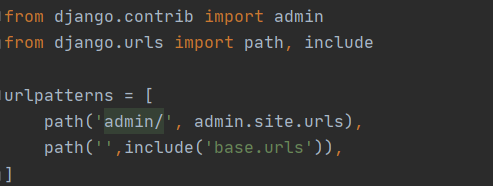


Trong base, tạo trong view.py một function taskList trả về một HttpResponse, bởi vậy bạn cần import vào HttpResponse

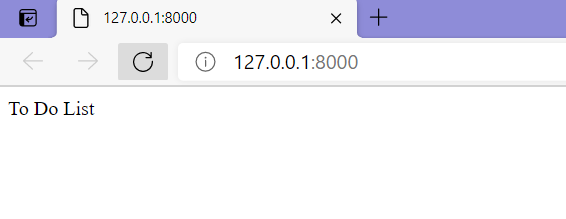


Sau đó vào base/urls.py để đăng kí function, với name là tasks



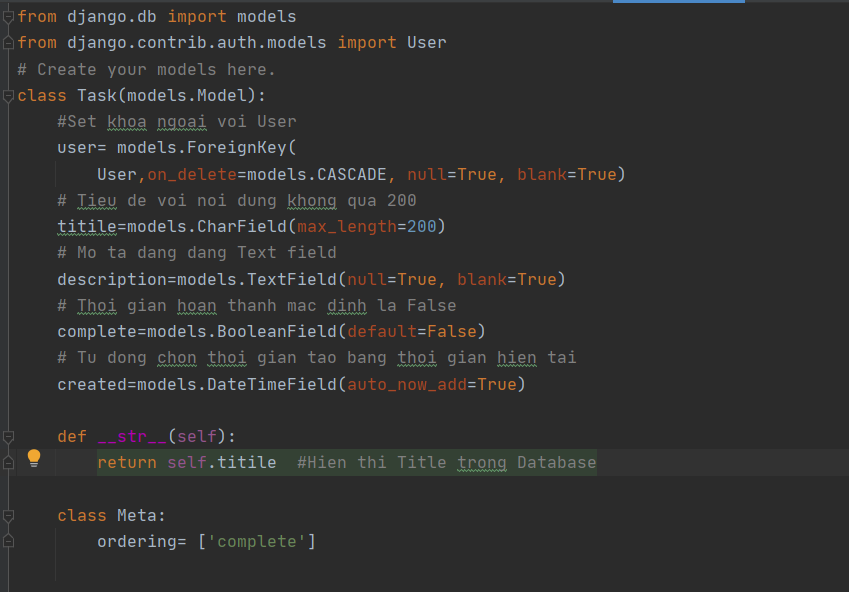
Tiếp đó vào todo\_list/urls.py để include file base.urls 

Vào câu lệnh terminal để run server với câu lênh python manage.py runserver. Sau đó truy cập địa chỉ [127.0.0.1:8000](http://127.0.0.1:8000/) để xem thành quả. Vậy là đã xong phần setup project

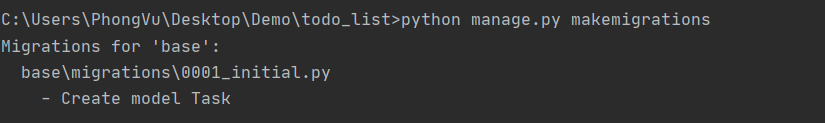


2. Setting model and database

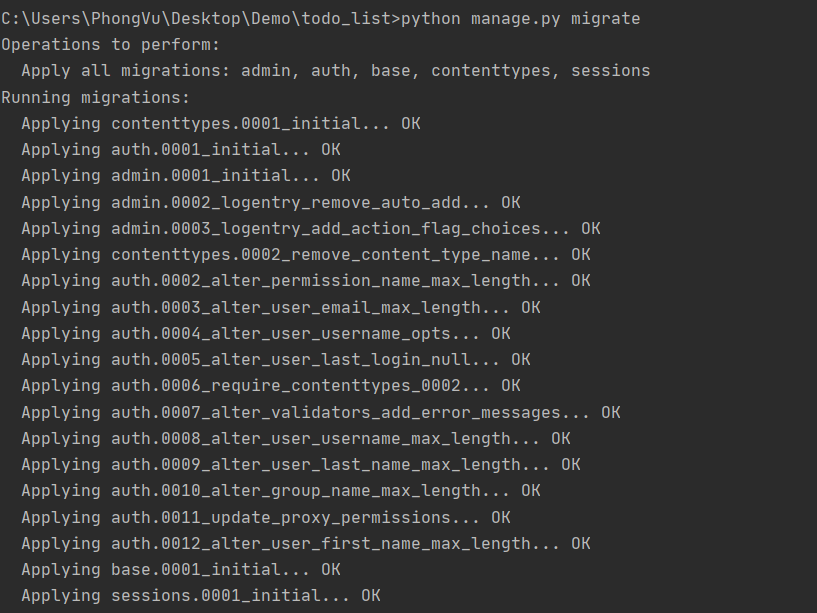
Ta thừa kế các tính chất của models trong Django và tạo các cơ sở dữ liệu như đã thiết kế, ở đây ta sử dụng database mặc định là sqlite3 của Django. Nếu bạn sử dụng cơ sở dữ liệu khác có thể cấu hình lại trong file setting.py. Để khởi tạo cơ sở dữ liệu cho app base ta vào thư mục base/models.py và soạn cơ sở dữ liệu như sau:



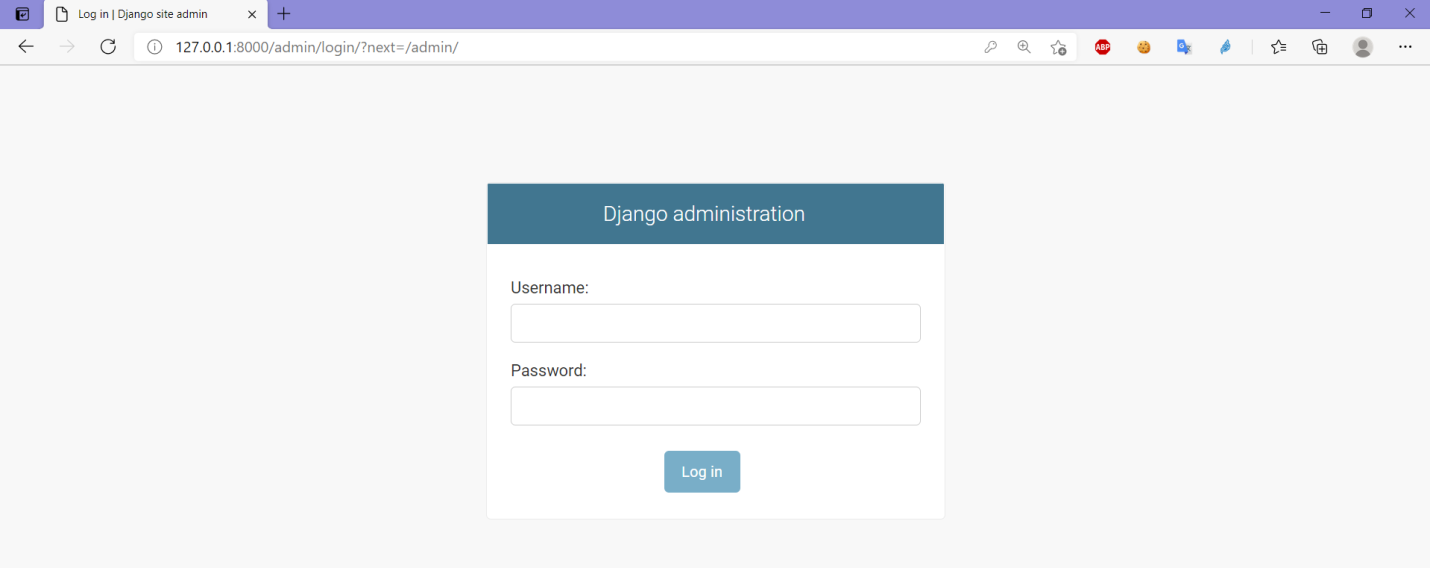
Sau đó cập nhật các thay đổi của models bằng cú pháp python manage.py makemigrations.



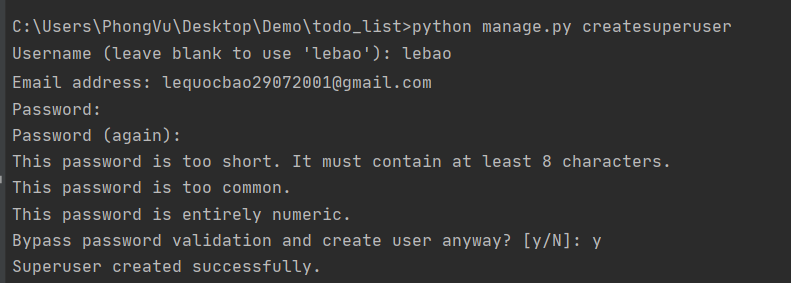
Lúc đó nó sẽ tự động tạo một file 0001\_initial.py trong base\migrations để lưu lại những thay đổi trong trong models.  Sau đó ta tiếp tục ánh xạ xuống để tạo các bảng trong cơ sở dữ liệu bằng cú pháp python manage.py migrate .



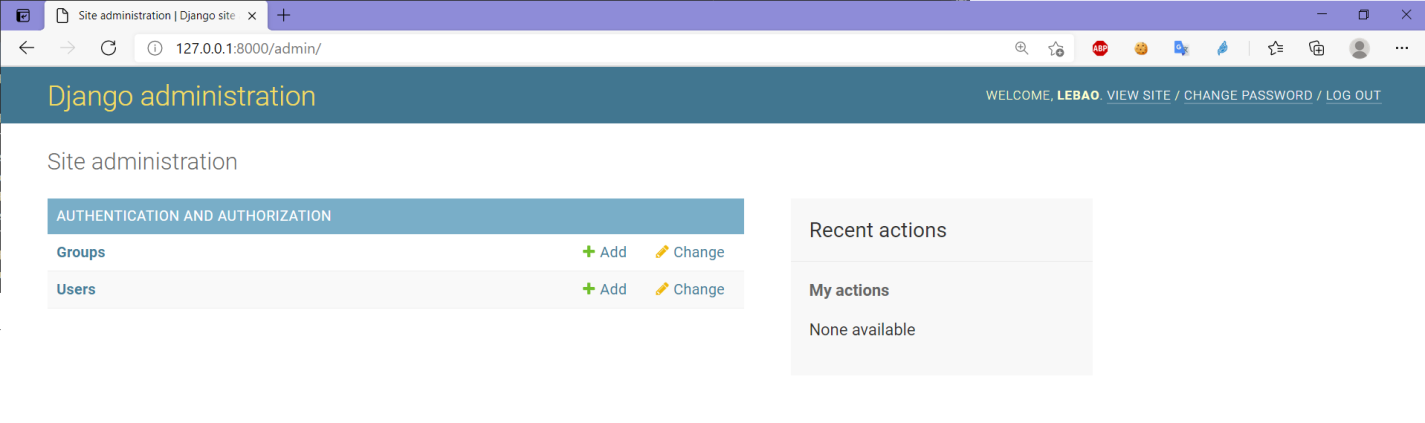
Theo mặc định thì Django sẽ tạo cho mình một trang Admin có sẵn để quản lí,tại địa chỉ <http://127.0.0.1:8000/> có giao diện như sau:



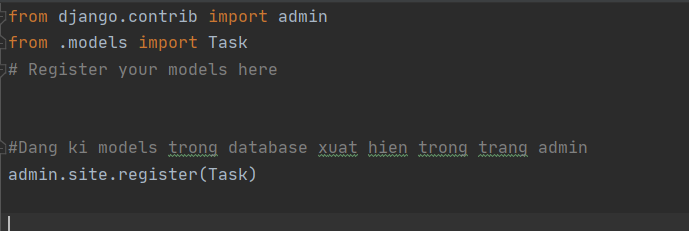
Để đăng nhập vào trang admin bạn cần có một tài khoản addmin.Chú ý để tắt server khi đang khởi chạy trong terminal bạn cần sử dụng phím Ctrl+ C. Để tạo tài khoản admin bạn cần sử dụng cú pháp python manage.py createsuperuser , tiếp đế bạn cần nhập username, email, password tài khoản mà bạn cần tạo.



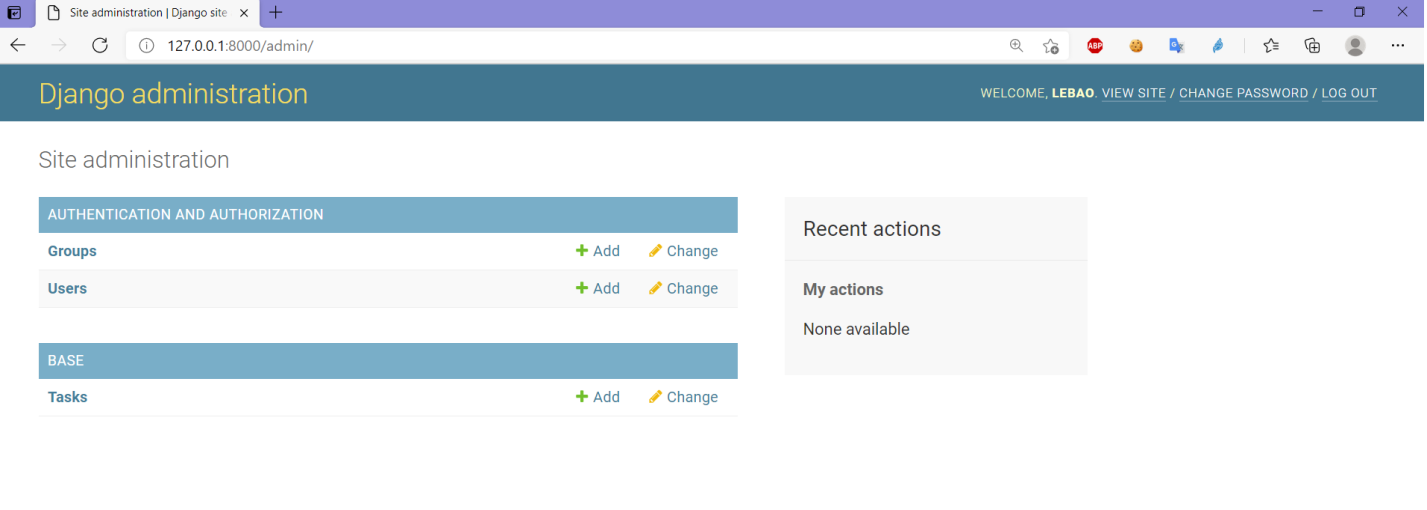
Sau khi tạo tài khoản bạn có thể sử dụng tài khoản đó để đăng nhập trang admin và sử dụng các quyền của quản trị như tạo tài khoản user, chỉnh sửa phân quyền của user.

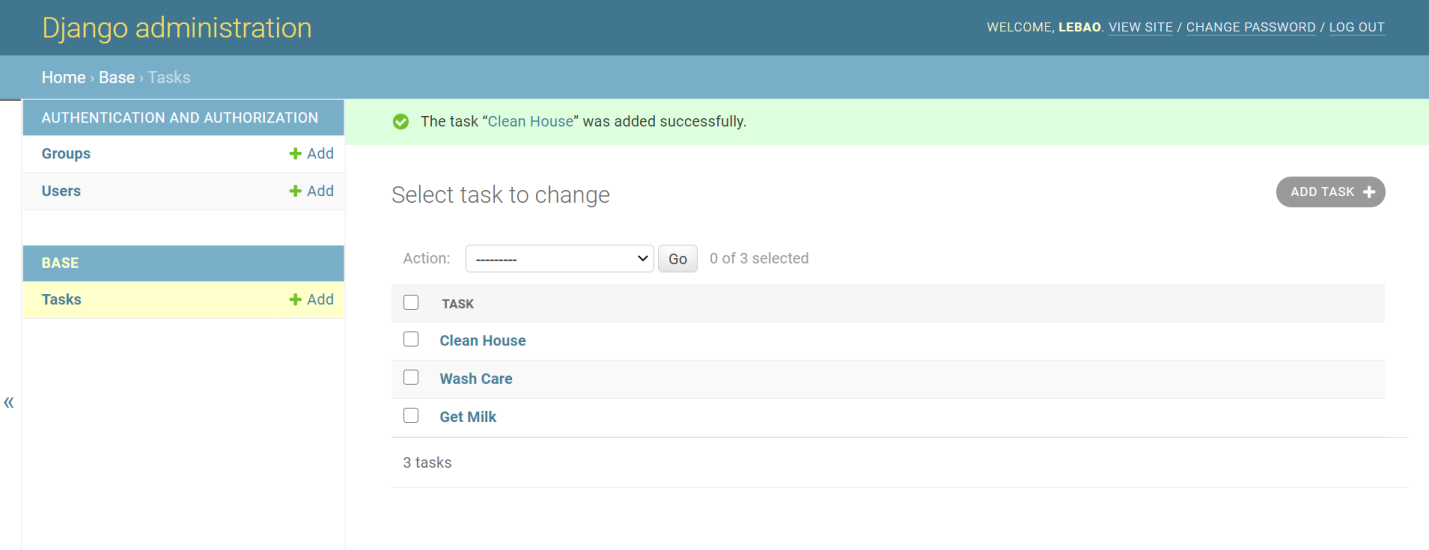


Sau đó đăng kí bảng Task để hiện trên trang Admin quản lí , khi đó ta vào base\admin.py và sử dụng các câu lệnh để đăng kí.



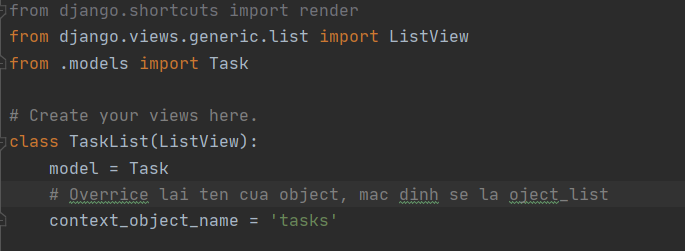
Khi đó trên trang Admin sẽ xuất hiện bảng Task để quản lí. Lúc đó bạn có thể thêm sửa xóa các Task mà bạn mong muốn .



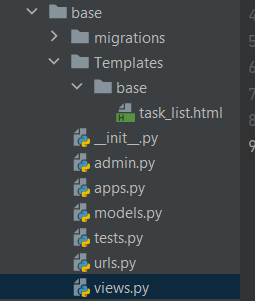


3. Trang Task List

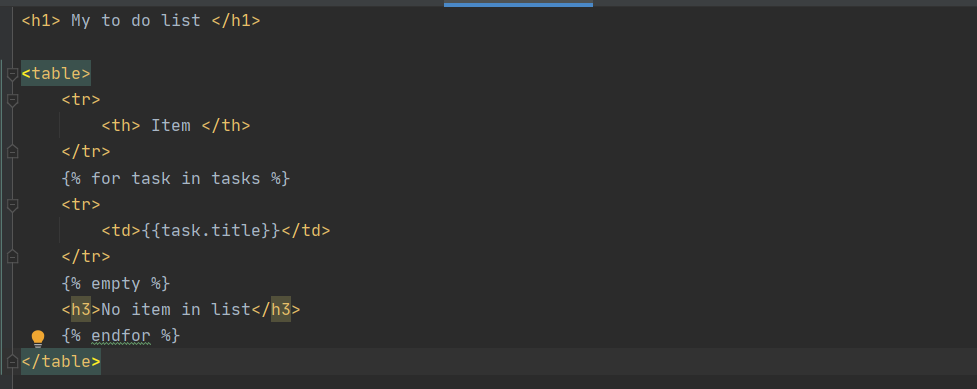
Sau khi hoàn thành cơ sở dữ liệu ta cần tạo trang để hiện thị ra các task chứa trong database. Ở phần này ta vào base\views.py để chỉnh sửa . Ở đây ta kế thừa các lớp sẵn có của lớp Views trong Django và sử dụng các Class thay vì hàm có trong views.py.



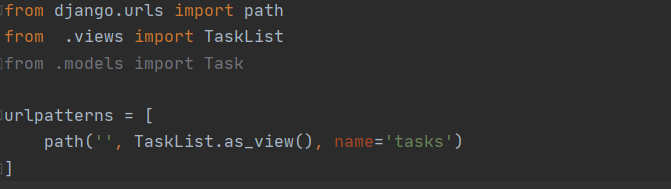
Khi sử dụng ListView sẽ sử dụng tên template mặc định của nó , ở đây sẽ xuất hiện task\_list.html . Khi đó bạn cần khởi tạo thêm thư mục mới là Templates/base, và tạo thêm file task\_list.html



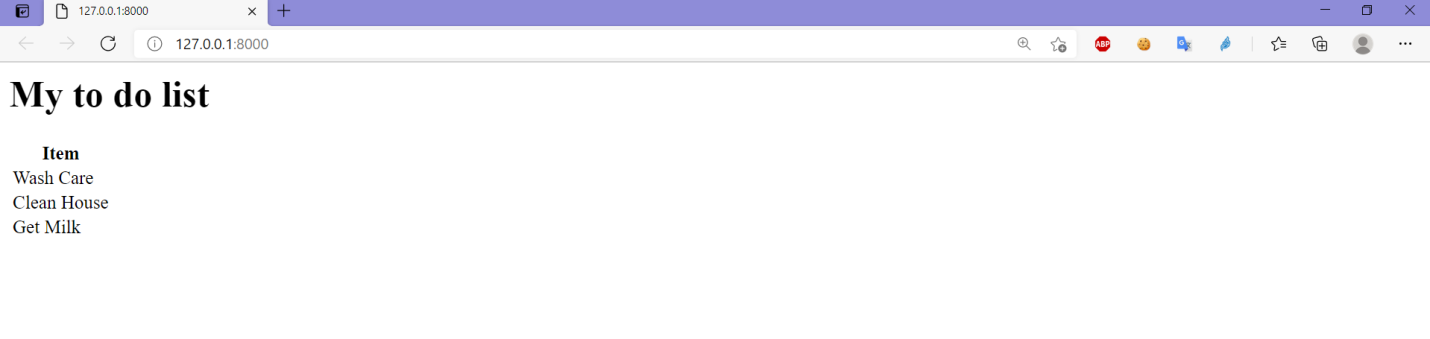
Khi tạo xong task\_list.html, bây giờ ta có thể sử dụng các cú pháp của html, kết hợp với các cú pháp jinja tron django để có thể lấy được truy suất các tên các task lúc trước mà chúng ta đã tạo .



Và bây giờ , ta chỉnh sửa đường dẫn để trang có thể truy cập. Ta vào thư mục base/urls.py. Trước tiên ta cần import thư mục views.py vào urls.py. Vì bản thân class Task được kết thừa từ lớp views khi đó cần thêm phương thức as\_Views() để có truy xuất.

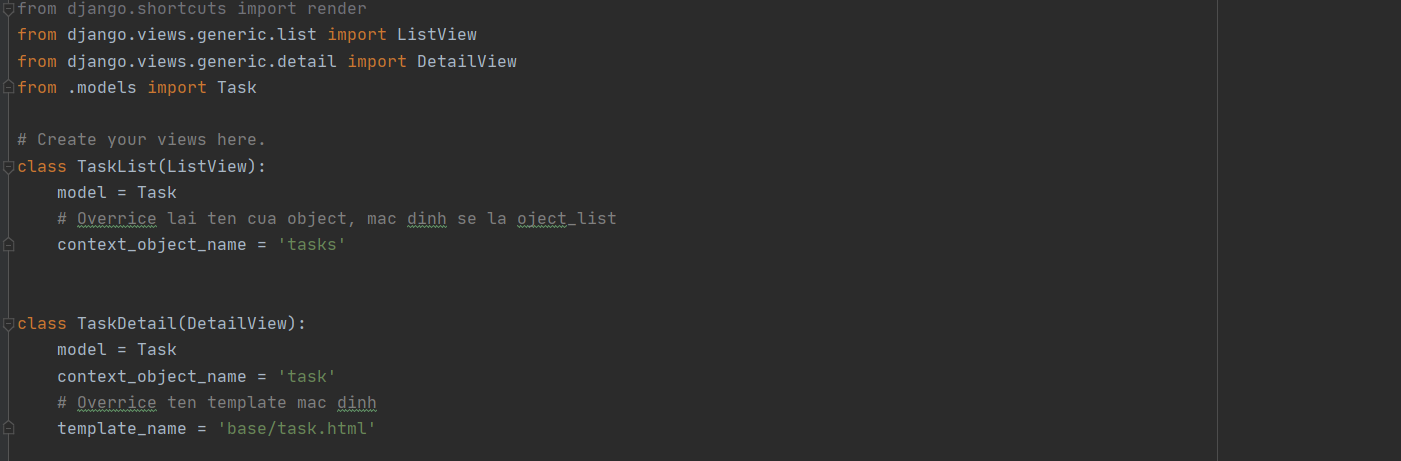


Và kết quả khi chạy server, bạn sẽ nhận được danh sách các tên các task đã tạo trong database như sau:

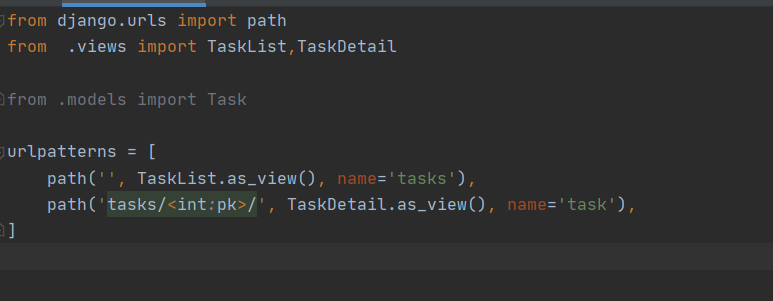


4. Detail Page

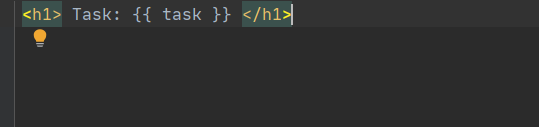
Giống như trang List Task , đầu tiên ta vào base/views.py để thiết lập view. Ở đây ta kế thừa DetailView. Trước khi sử dụng bạn cần phải import DetailView. Thay vì sử dụng tên template mặc định thì ta có thể sửa lại template bằng context\_object\_name.



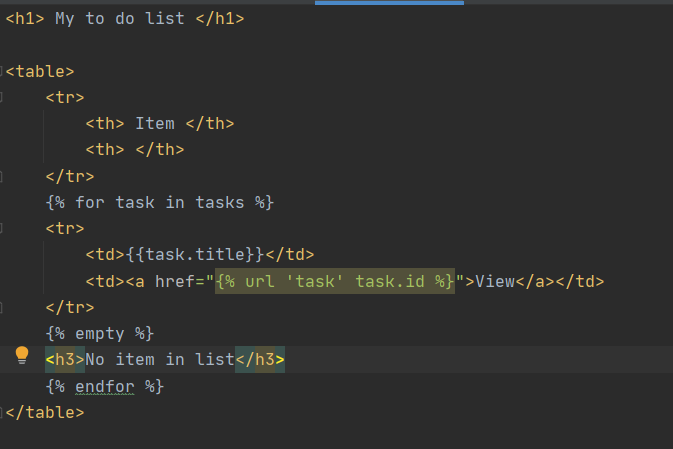
Sau đó vào base/urls.py để cấp đường dẫn cho TaskDetail. Vì TaskDetail là class nên cần sử dụng phương thức as\_views(). Ở đường dẫn mình sử dụng tham số để có thể truyền khóa vào, khi đó có thể xuất ra được chi tiết mô tả cho một task.



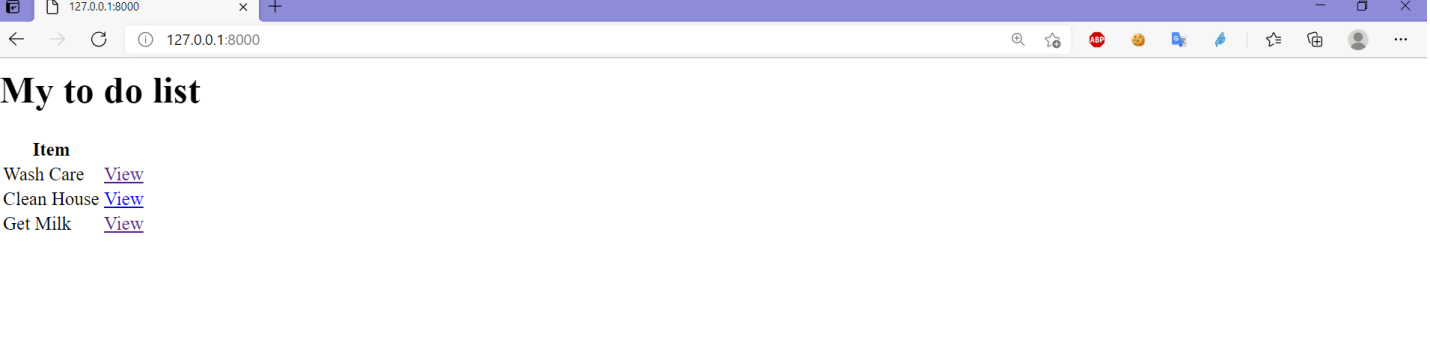
Sau cần phải tạo lại một template task.html trong thư mục template/base. Sau đó sử dụng cú pháp html để tạo trang task.html .



Sửa lại một số ở task\_list.html để tạo gallery.

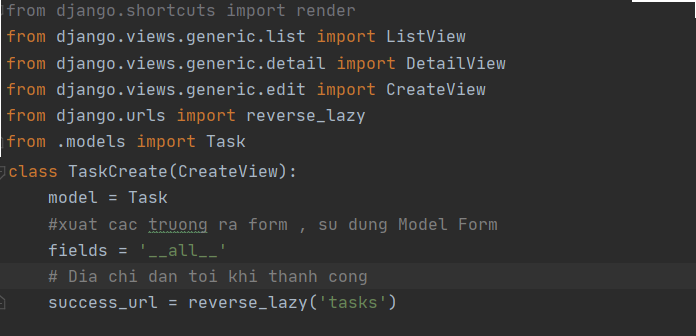


Và kết quả khi nhận được sẽ là trang task\_list sẽ chưa các task và đường dẫn view bên cạnh, khi nhấn vào view thì tham số pk được truyền vào, nó sẽ dẫn tới các trang task/<pk> tương ứng.

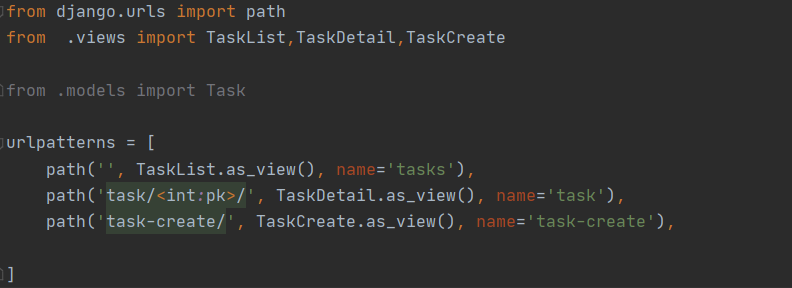


5. Create Item

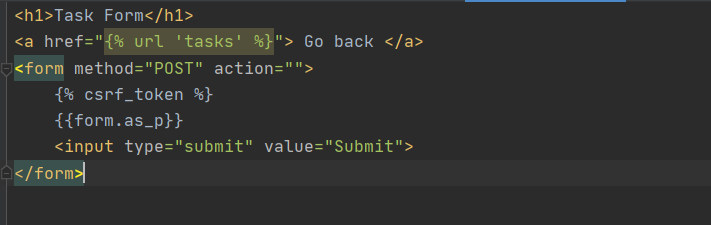
Để thuận tiện ta cần phải tạo ra một trang để tạo các Task cần thiết. Khi đó ta sẽ sử CreateView trong phần base.models. Bạn cần import CreateView. Ở đây ta có sử dụng thêm Models Form để có thể tạo thuận tiện trong việc tạo form để tạo ra các task với các thuộc tính có trong models. Sử dụng reverse\_lazy để tạo đường dẫn tới url khác khi submit form.



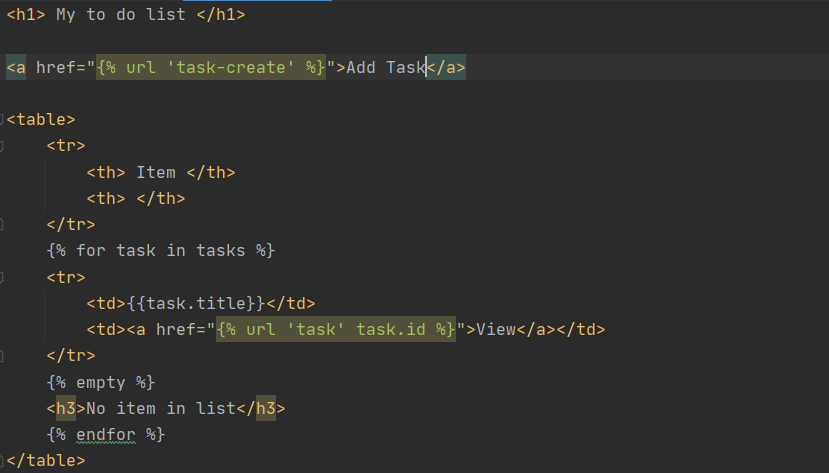
Sau khi tạo view xong ta cần tạo url cho lớp TaskCreate . Khi đó ta truy cập base/urls.py để tạo đường dẫn . Vì TaskCreate là một Class trong view nên cần sử dụng phương thức as\_view().



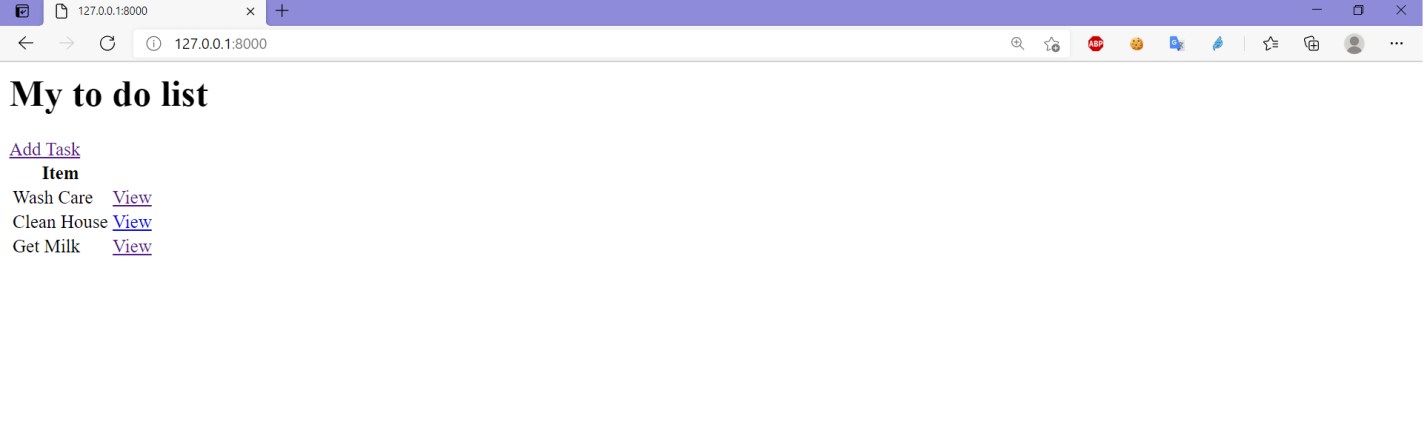
Khi sử dụng CreateView thì lớp dẫn xuất quy định cần một template, mà tên mặc định của nó là task\_form.html. Khi đó ta cần tạo một template là task\_form.html trong template/base. Trong task\_form.html ta tạo một form để nhập các thông tin, khi đó biến form được truyền xuống tạo ra các trường giống như ở trong models , giúp bạn tạo các task thuận tiện hơn. Ở đây theo quy định Django thì mỗi form truyền đi cần có một mã crsc\_token nên bạn cần để {% crsc\_token %} ở trong form của mình.

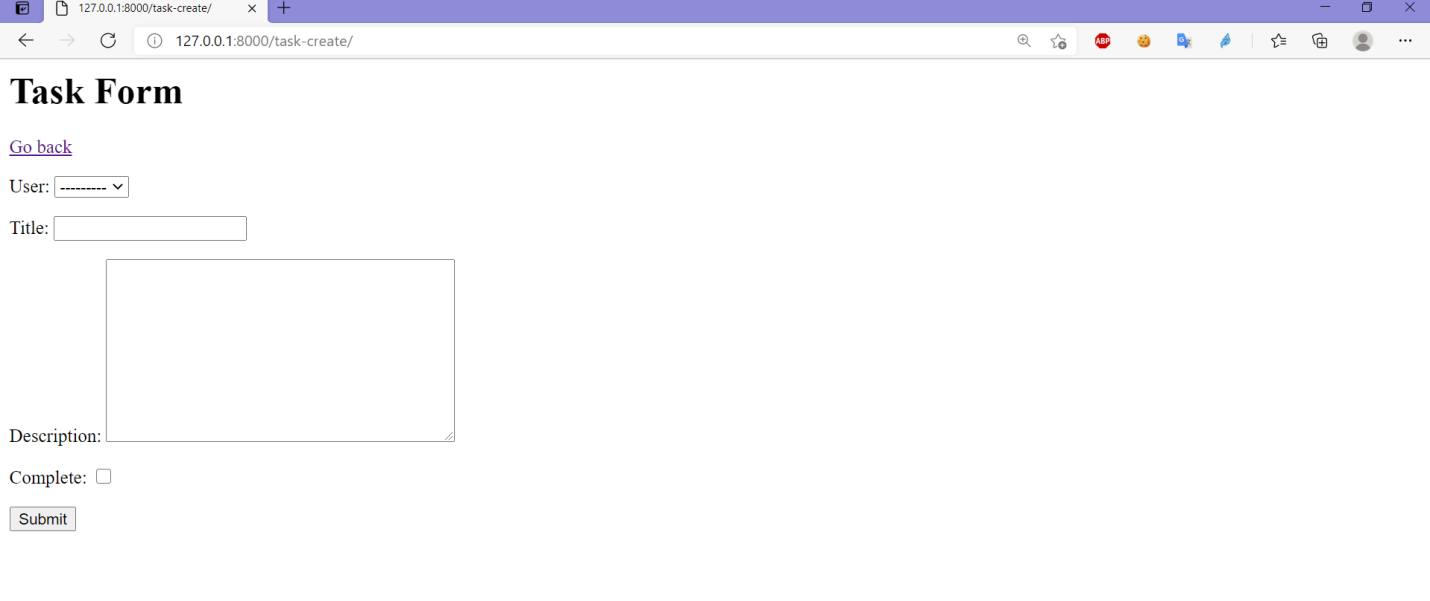


Sau đó ta thêm dòng lệnh html ở task\_list.html để dễ dàng di chuyển giữa 2 trang.



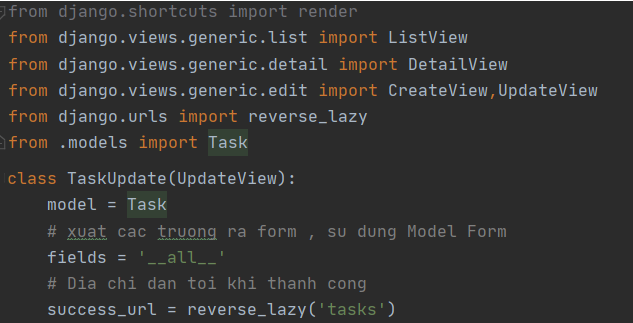
Bây giờ ta sẽ được thành quả như sau:

 Khi tạo một đối tượng trên <http://127.0.0.1:8000/task-create/> thì sau khi submit thì đối tượng đã được cập nhật.

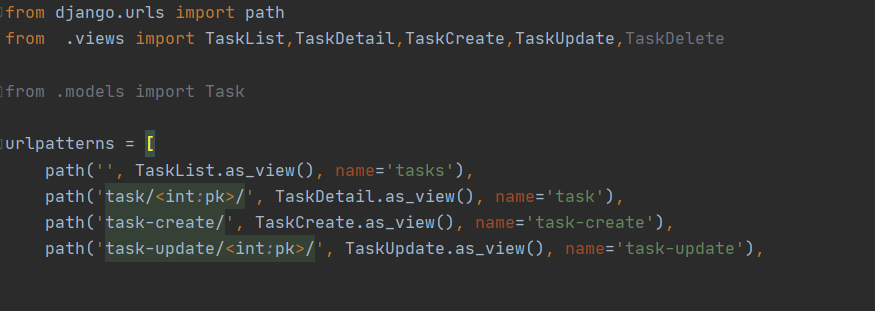


6. Update Item

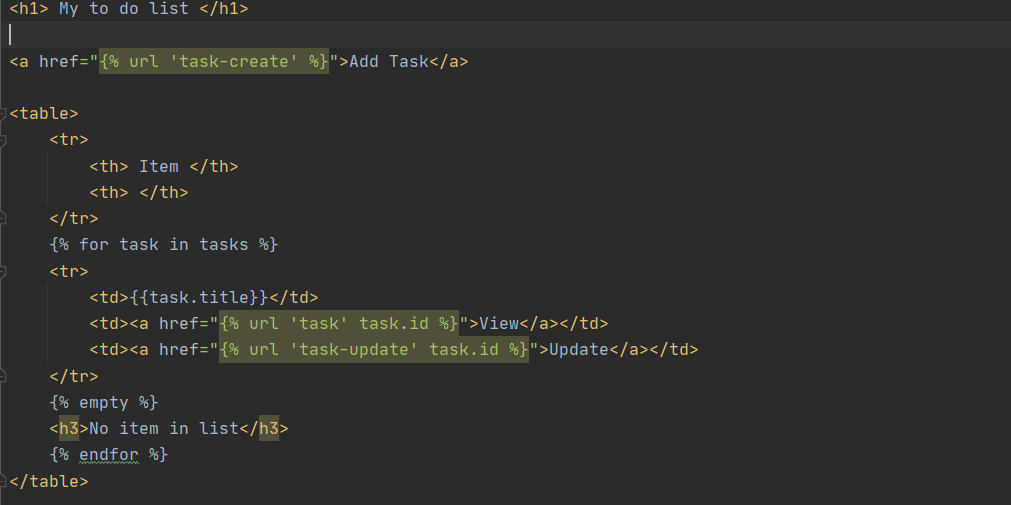
Giống như Create item thì Update sẽ thừa kế từ lớp từ Update View . Tại base/views.py ta khai báo TaskUpdate , ta phải import trước khi sử dụng . Trong đó có sử dụng thuộc tính fields của models form để tạo form update cho item. Sử dụng reverce\_lazy() để truy xuất lại trang chủ khi submit xong



CrateView và UpdateView sẽ sử dụng chung một template. Bây giờ ta sẽ tạo cho TaskUpdate một url. Ta vào vào base/urls, vì là class nên phải sử dụng phương thức as\_view(), đông thời truyền vào tham số pk.

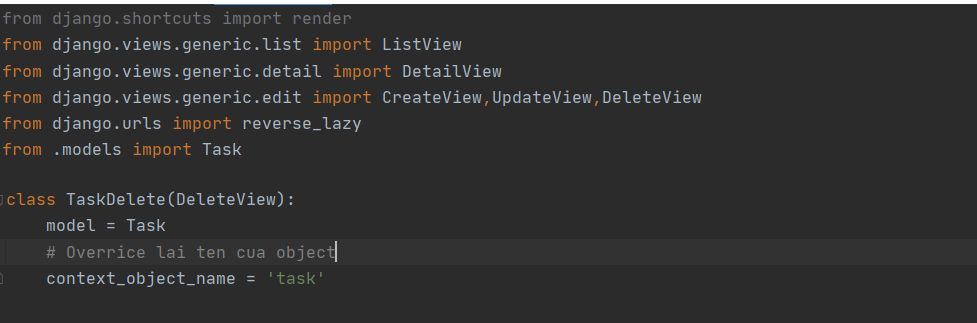


Sau đó ta tạo thêm đường dẫn tới trang update trong task\_list để thuận tiện cho việc chỉnh sửa.



7. Delete item

Giống như các chức năng ở trên, thì Delete item được tạo ra tương tự. Ban đầu ta sẽ vào base/views tạo một lớp tên là TaskDelete, lớp này kế thừa từ lớp DeleteView. Trước khi sử dụng ta phải import DeleteView.

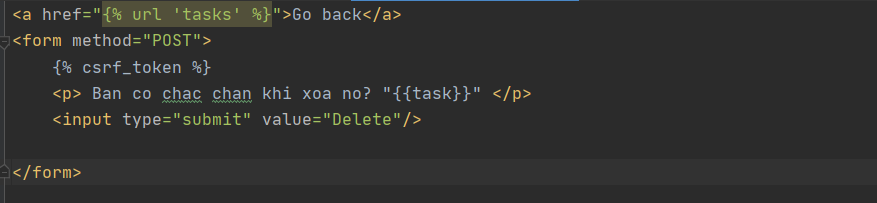


Sau đó cấp cho TaskDelete một url. Để cấp ta vào base/urls.py. Ở đây đường dẫn được truyền vào tham số pk, pk chính là id của item xóa. Khi click vào delete thì của một item, thì id sẽ tự động được truyền vào. TaskDelete được sử dụng phương thức as\_view() để có thể xuất ra views

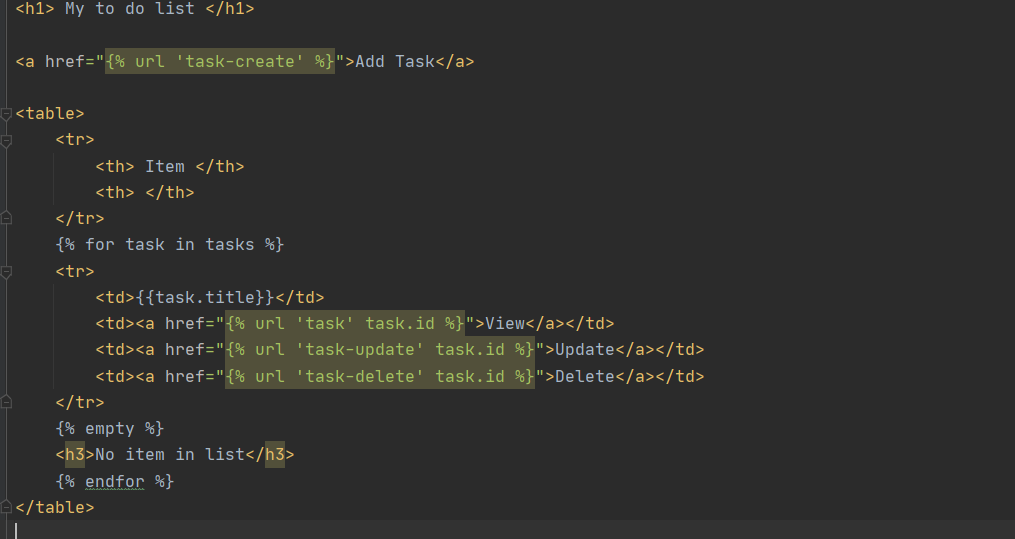


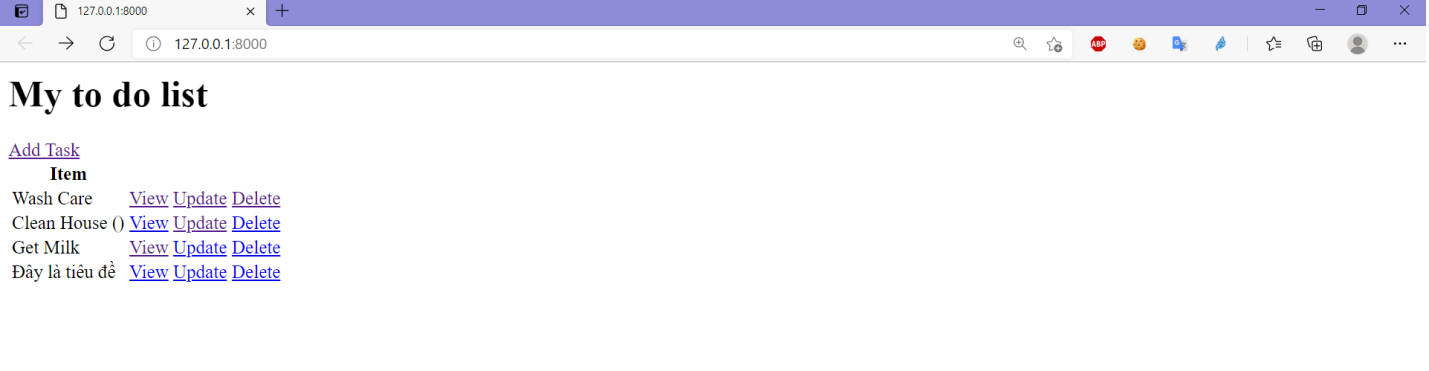
Khi sử dụng DeleteView thì mặc định sẽ sử dụng một templete . Mà ở đây cụ thể là task\_confirm\_delete. Bởi vậy ta cần tạo thêm một file task\_confirm\_delete.html ở template/base.

Sau đó khởi tạo các lệnh cụ thể trong file như sau:

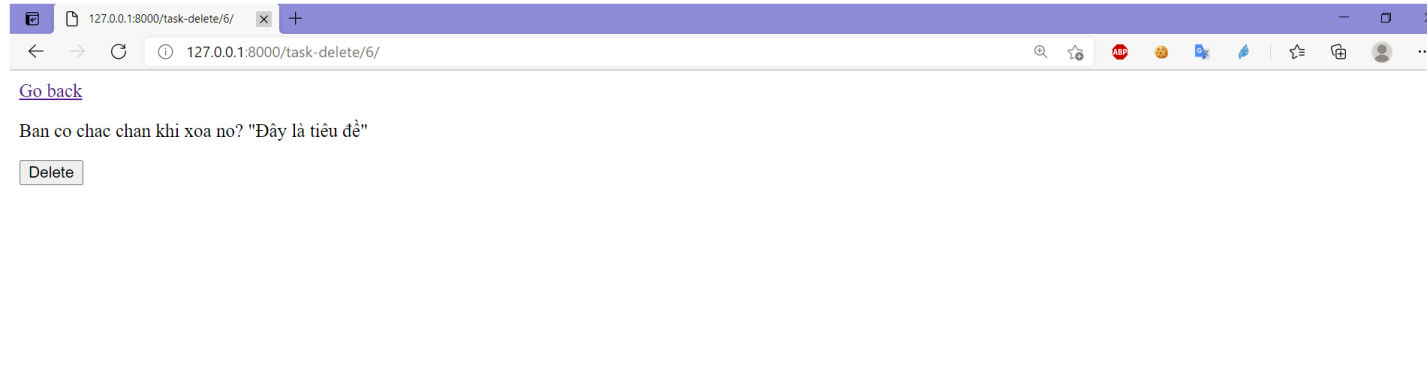


Tiếp tục ta sử dụng file task\_list để thêm garalley dẫn tới trang xóa.

Vậy ra ta đã hoàn thành cách xóa đi một item.

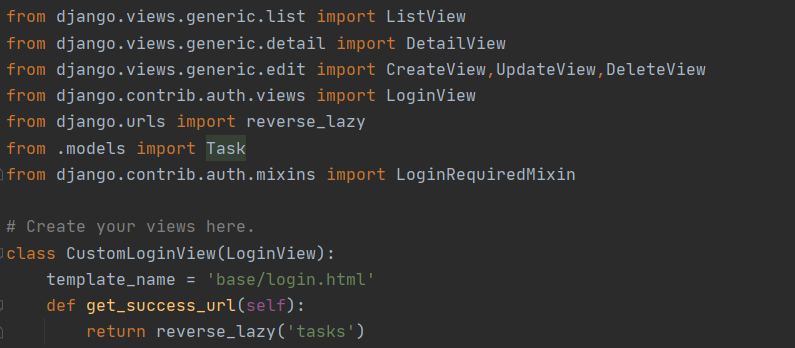


Và Khi click vào delete.

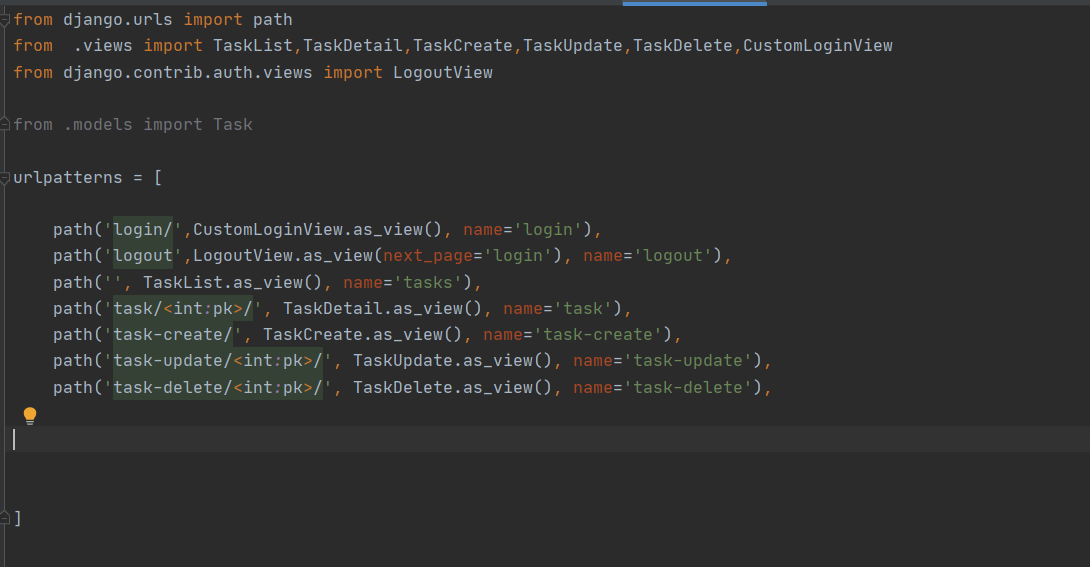


7. Login/Logout

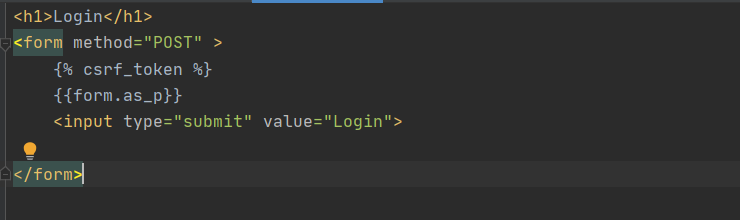
Để dễ dàng kiểm phân chia cho nhiều người sử dụng, thì tiếp đến ta sẽ sử dụng thêm trang Login/Logout cho người dùng sử dụng . Ban đầu ta tạo trong base/views.py lớp CustomLoginViews thừa kế từ LoginView , cấu hình lớp CustomLoginViews bằng sử dụng đổi tên đối tượng template và trả về khi đăng nhập thành công là tasks.



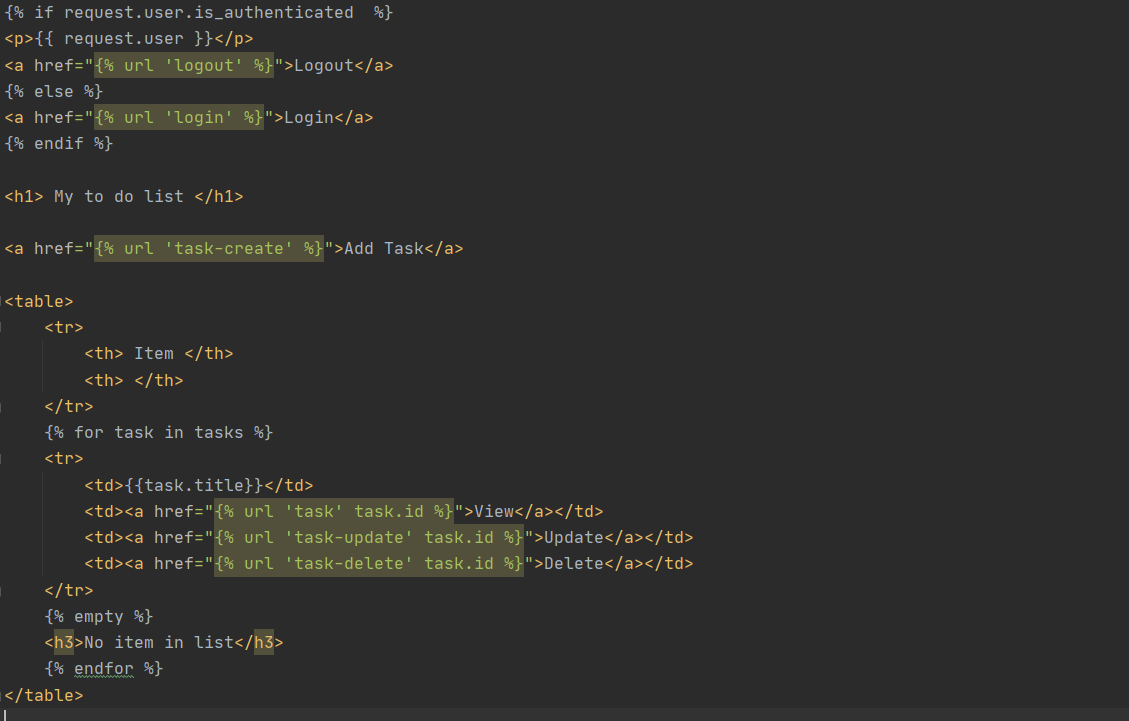
Còn trang Logout ta sẽ sử dụng luôn lớp LogoutView có sẵn của Django. Sau đó ta cấp cho 2 trang đường dẫn url tương ứng. Trước khi sử dụng ta phải import đầy đủ các lớp sử dụng. Ta cài đặt trang mặc định khi đăng xuất xong là trang login.



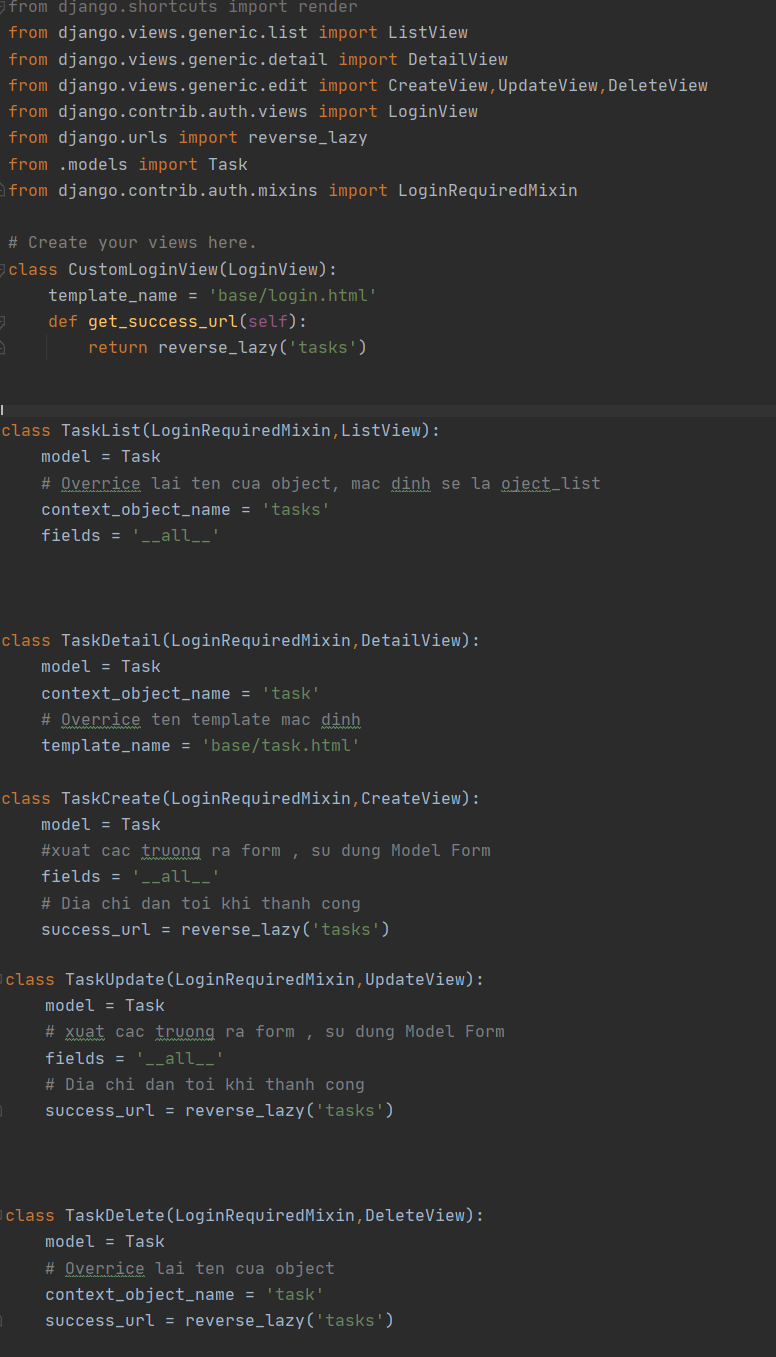
Tiếp đến ta có tạo thêm file login.html tại template/base . Đây chính làm template của lớp CustomLoginView sử dụng khi kế thừa từ lớp LoginView. Trong file ta tạo một số form đơn giản và kế thừa, sử dụng model form trong Django. Chú ý nhớ thêm mã csrf\_token trong form.



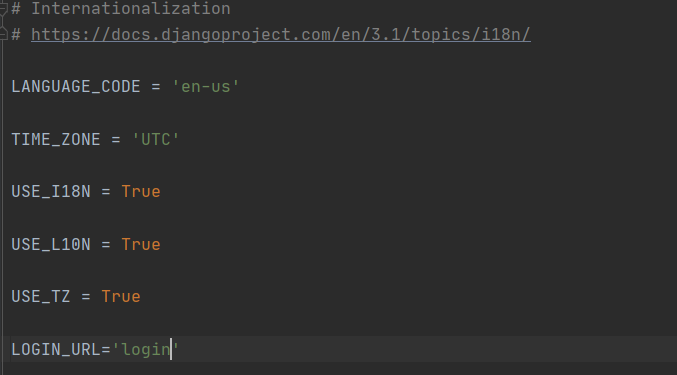
Để hoàn chỉnh hơn ta có thể sửa lại file task\_list một chút để nó có thể hoàn chỉnh hơn. Ban đầu ta sử kiểm tra trong request người dùng đã đăng nhập hay chưa. Nếu có thì hiện tên tài khoản đăng nhập và đường dẫn có thể logout. Nếu chưa thì hiện đường dẫn đến trang login.

****

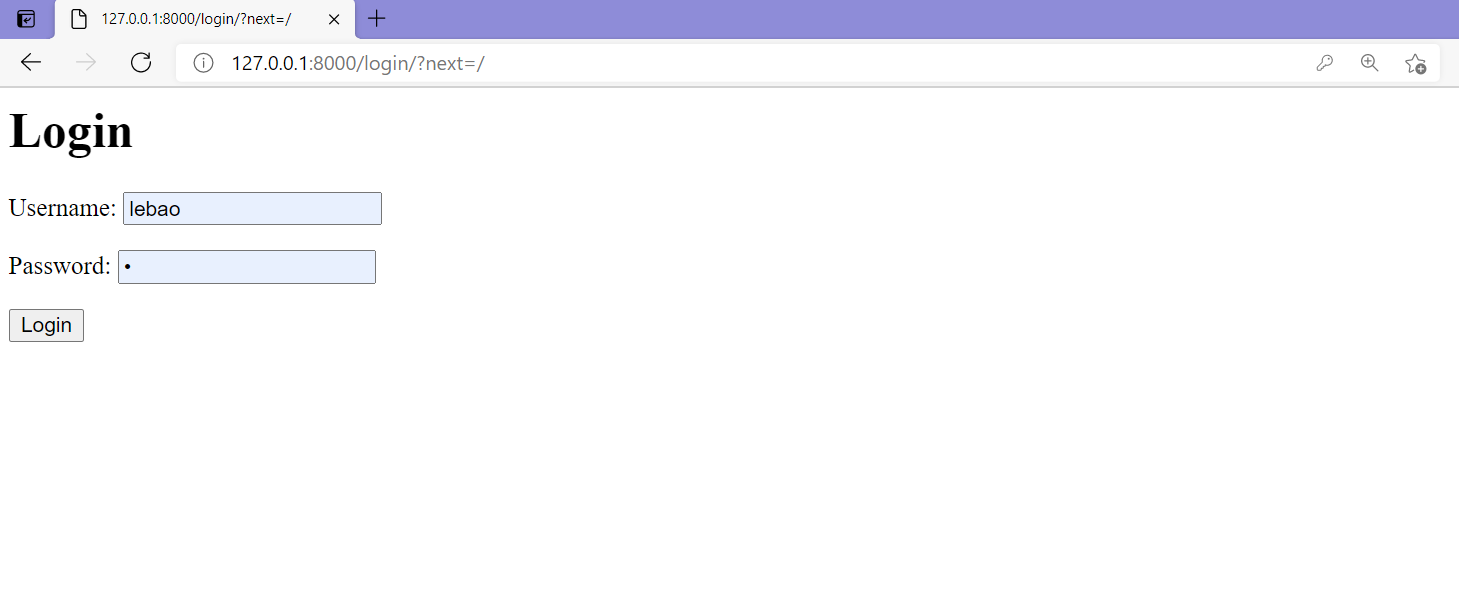
Với mỗi trang chung ta đã làm thì chúng có thể dễ đang truy cập vào mà không đăng nhập, việc đó dẫn tới dẫn tới thông tin được kiểm soát. Bởi vậy ta cần yêu cầu các trang bắt buộc đăng nhập trước khi sử dụng. Để sử dụng bắt buộc yêu cầu đăng nhập khi truy cập trang , ta sử dụng lớp LoginRequiredMixin trong Django. Ta cấu hình bắt buộc sử dụng trong lớp base/views.py. Ở đây ta sẽ bắt buộc yêu cầu đăng nhập đối với các trang tasks, task, task-update, task-delete, task-create, ta sẽ thêm LoginRequiredMixin, ở lớp view tương ứng với từng trang.



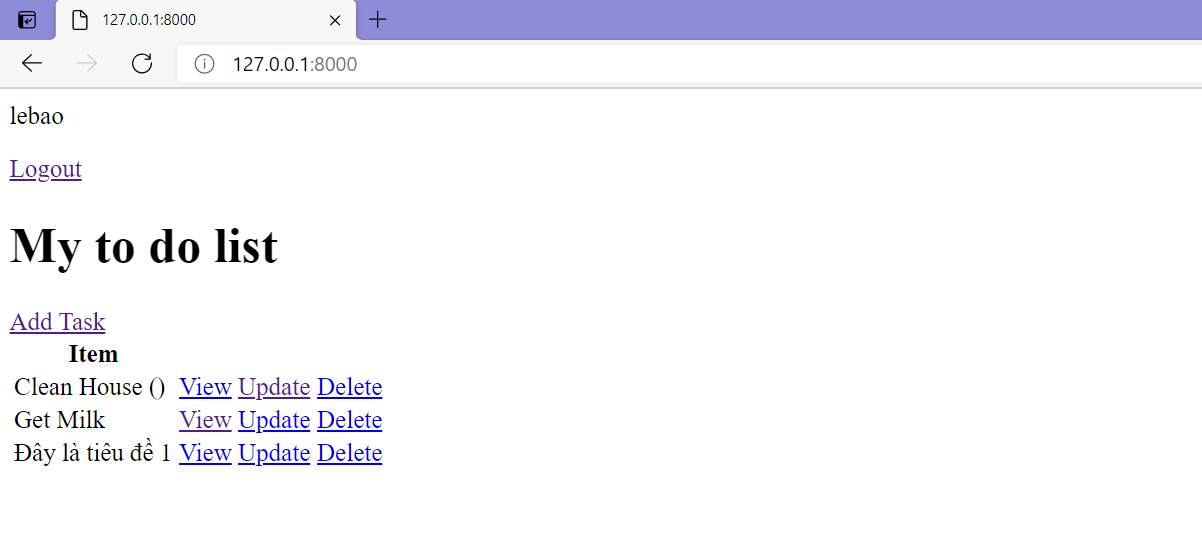
Sau khi khi cấu hình bắt buộc yêu cầu với các trang, ta phải cài đặt trang login mặc định sử dụng khi bắt buộc yêu cầu đăng nhập. Bây giờ ta vào todo\_list/settings.py thêm địa chỉ trang login . Ở đây ta sử dụng trang login cài đặt ở app base làm trang login mặc định nên cú pháp sử dụng cần thêm vào là LOGIN\_URL = ‘login’



Ta đã hoàn chỉnh trang login/ logout. Bây giờ ta có thể đăng nhập, đăng xuất cũng như cấp quyền truy cập với các trang. Ta có thể runserver để chạy thử.

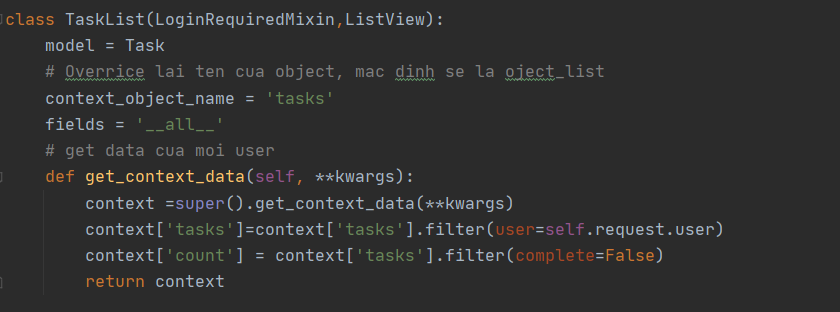


Khi bạn chưa đăng nhập nó sẽ yêu cầu đăng nhập. Tiếp đến bạn đăng nhập thì bạn có thể vào các trang khác một cách bình thường.

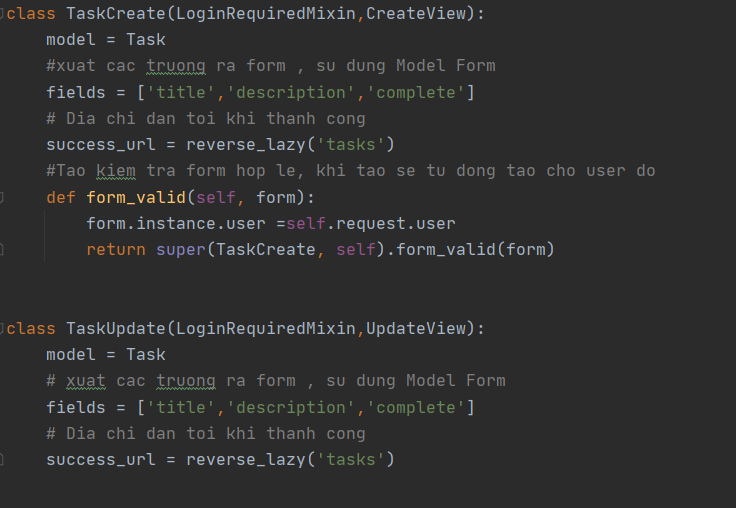


8. Truy xuất dữ liệu cho từng user.

Các bước đã sử dụng ở trên đã yêu cầu đăng nhập, cũng như xuất dữ liệu ra, tuy nhiên nó đã xuất dữ liệu của tất cả các user. Bài toán ta cần xử lí là khi đăng nhập bạn chỉ xuất ra những task mà của chính user đó. Lúc đó ta sẽ sử dụng phương thức get\_context\_data để trả về những task của user đó. Bây giờ ta vào base/views để chỉnh sửa lớp TaskList.



Sau khi xuất được hết các task của từng user. Ta sẽ phát hiện ở một số cái không tối ưu khi tạo và update các task. Bởi theo quy định, mỗi user chỉ được tạo các task cho chính user đó (ngoài trừ tài khoản admin khi vào trang admin để tạo). Tuy nhiên ở đây, nó có thể tạo các task cho các user khác. Để khắc phục , ta sẽ không cho xuất chỉ định user trong model form và cài đặt user đó chỉ cài đặt task cho chính bản thân mình. Để cài đặt ta vào base/view.py để cấu hình lại lớp TaskCreate và lớp TaskUpdate.



9. Trang đăng kí tài khoản

Đề thuận tiện trong việc tạo tài khoản user. Ta sẽ tạo một trang đăng kí tài khoản. Đầu tiên ta sử dụng các