

Desarrollando en el Framework Django

Dr. Ismael Figueroa

ismael.figueroa@pucv.cl

<http://www.inf.ucv.cl/~ifigueroa>

IBC 3-23



¿Qué es un Framework Web?



framework Editar



marco de referencia

sustantivo **el marco**
framework, frame, setting, mark,
picture frame, mounting

la estructura
structure, frame, framework,
build, fabric, skeleton

el armazón
frame, framework, shell,
skeleton, carcass, armature

el sistema
system, scheme, method,
framework, machinery

el entramado
framework, truss

el esqueleto
skeleton, framework, frame

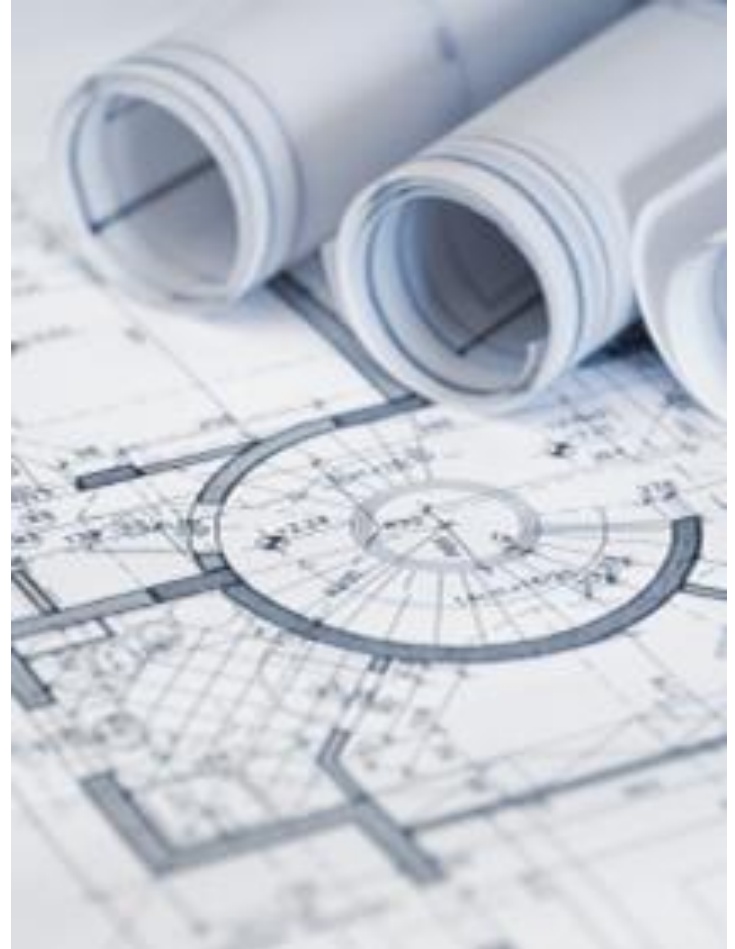
Un Framework Web es:

un conjunto de *librerías de software*
diseñadas para *trabajar en conjunto*
y *dar soporte* al desarrollo de
aplicaciones web



Un Framework Provee:

una ***arquitectura subyacente*** y mucho ***código de apoyo*** que trabaja bien sobre dicha arquitectura



Usando un Framework Web:

Reinventing the wheel.
Knowing *when* and *how*.

evitamos *reinventar la rueda*

respecto a las **tareas**

comunes que se
requieren en una aplicación
web



¿Cuáles son las Tareas Comunes en una Aplicación Web?

Routing

Todas las aplicaciones necesitan ***rutear URLs*** hacia ***acciones específicas*** que generan una ***respuesta HTTP***.

Todo request HTTP ***debe*** tener una respuesta correspondiente. Si es que la red funciona :-)



Despliegue de HTML

Todas las aplicaciones eventualmente presentan al usuario documentos en HTML.

Estos documentos HTML combinan **contenido estático o fijo**, con **contenido generado dinámicamente** en base a la petición que se está respondiendo



Persistencia

Todas las aplicaciones requieren algún tipo de ***persistencia de datos***.

Es decir, deben poder conectarse y manipular ***entidades*** en una base de datos.

Es necesario, ***crear, leer, actualizar y eliminar*** elementos.

Además es necesario ***buscar, filtrar, etc***



Autenticación y Autorización

Autenticación: el usuario efectivamente **es quién dice ser**

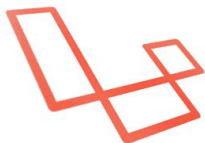
Autorización: el usuario **tiene permiso** para realizar acciones específicas



Framework de Desarrollo MVC

un conjunto de ***librerías de software*** diseñadas para ***trabajar en conjunto*** y ***dar soporte*** al desarrollo de aplicaciones web, ***en base al diseño arquitectónico indicado por el patrón de diseño MVC***

Frameworks Web



laravel



socket.io




BACKBONE.JS

El Framework Django

Documentación

<https://docs.djangoproject.com/en/2.0/>

Datos Históricos

- Desarrollado a finales del año 2003, liberado open source el 2005
 - Basado en el patrón de diseño MVC
 - ***Provee un sistema de autenticación y autorización integrado por defecto***
 - ***Provee un backend administrativo integrable por defecto***
 - Trabaja bajo la idea de un proyecto combina múltiples ***aplicaciones***
 - Bastante usado a nivel industrial/empresarial
 - Muchas otras características...
- 

Instalación de Django

- Django 2.0 requiere Python 3. Se recomienda instalar Aconda para crear un ambiente exclusivo para nuestro proyecto
- Descargar instalador de conda
- En la línea de comandos ejecutar:
 - `conda create -n todolisto python=3`
- Ahora debiera ser posible ejecutar:
 - `source activate todolisto`
 - `pip install django`



Creando un nuevo **proyecto** Django

```
> django-admin startproject todolisto  
> cd todolisto  
> python manage.py runserver
```

- El comando `django-admin startproject todolisto` crea una nueva carpeta `todolisto` con el esqueleto de un proyecto Django
- Al usar `python manage.py runserver` se levanta un servidor de pruebas en `localhost:8000`

Creando una nueva **aplicación** Django

```
> python manage.py startapp main
```

- En Django un **proyecto** puede contener una o más **aplicaciones**. La idea es que las aplicaciones pueden ser independientes entre sí, o también pueden integrarse unas con otras.
- Es dentro de las aplicaciones que crearemos nuestros modelos, vistas y controladores



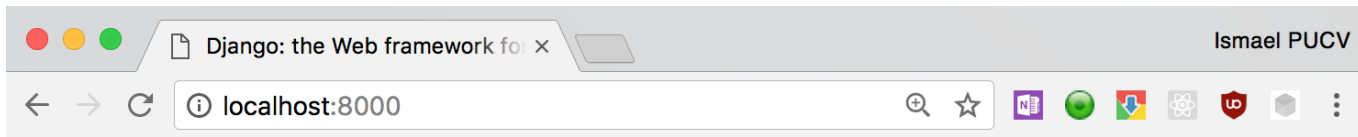
Registrando aplicación `main` en la configuración

settings.py x

```
32
33 INSTALLED_APPS = [
34     'main',
35     'django.contrib.admin',
36     'django.contrib.auth',
37     'django.contrib.contenttypes',
38     'django.contrib.sessions',
39     'django.contrib.messages',
40     'django.contrib.staticfiles',
41 ]
```

En el archivo `todo1isto/settings.py` debemos indicar que nuestra aplicación `'main'` se considera instalada...

Esto sirve después para la búsqueda de plantillas y otros elementos



django

View [release notes](#) for Django 2.0



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

Migraciones pendientes

Al ejecutar runserver la primera vez veremos un mensaje como este:

**You have 14 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.**

Para solucionarlo debemos cancelar el runserver y ejecutar

```
python manage.py migrate
```



Migraciones: entre lo ideal y lo real

En Django y otros frameworks de desarrollo web se maneja el concepto de **migraciones** para describir la estructura lógica de la base de datos que, finalmente, almacena los datos de la aplicación.

Por lo tanto, un archivo de migraciones indica operaciones a realizar, tales como:

- Crear tabla
- Agregar campo a tabla
- Borrar campo de tabla
- etc...

Migraciones en Django

En Django las migraciones se generan automáticamente a partir de los cambios en los modelos con el comando

```
python manage.py makemigrations
```



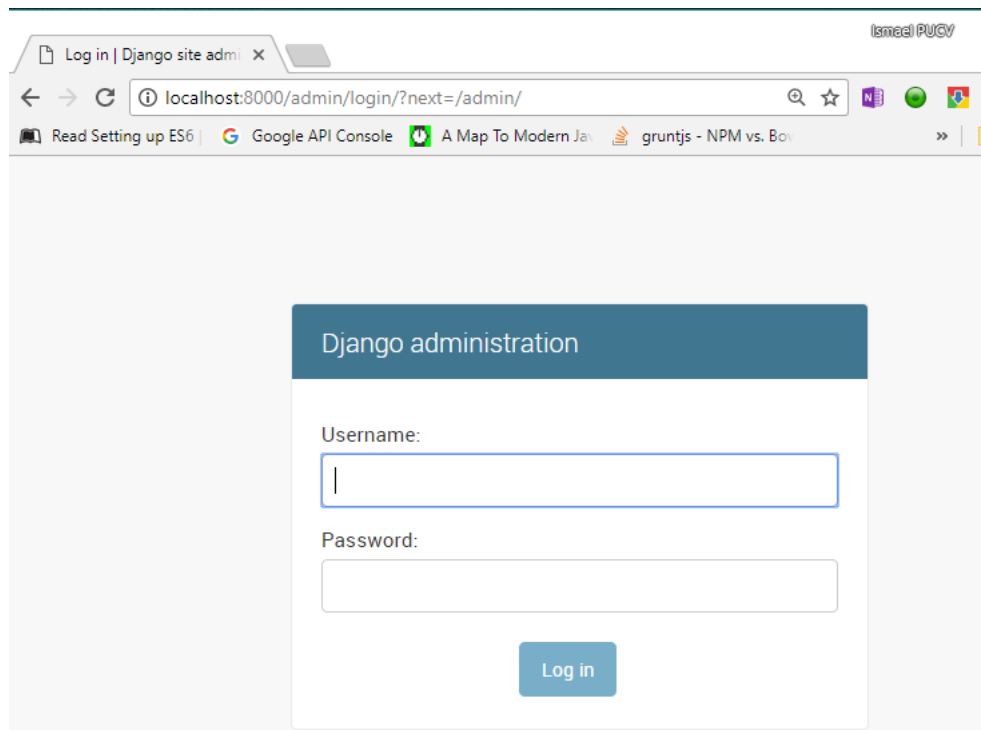
Backend Administrativo

Por defecto Django tiene un poderoso y flexible backend administrativo. Para ingresar debemos ir a

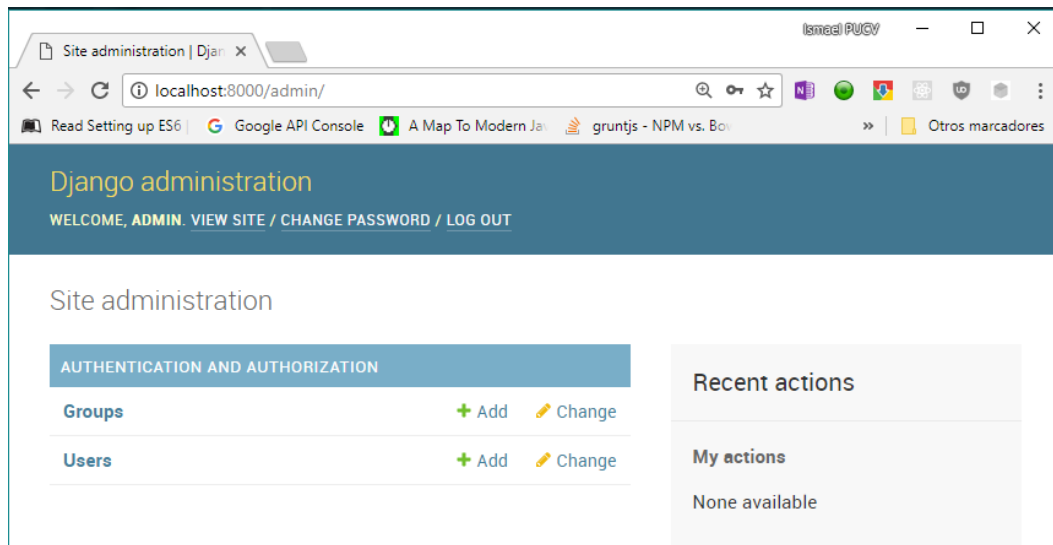
<http://localhost:8000/admin>

Necesitamos crear un superusuario para poder ingresar!

```
python manage.py createsuperuser
```



Backend Administrativo



El backend muestra los modelos de distintas aplicaciones, siempre y cuando éstas hayan sido registradas para aparecer en el administrador



Todo Listo!
Una aplicación web con MVC

Aplicación “Todo Listo!”

La aplicación “Todo Listo!” maneja una lista de tareas, donde un usuario puede desear:

- Ver el listado actual de tareas por realizar
- Visualizar un calendario con la cantidad de tareas creadas y finalizadas cada día y cada mes

Además, un administrador podría:

- Visualizar cantidad de tareas según distintos filtros, mostrando solo la cantidad, pero no el contenido de las tareas



Interfaz de Usuario: Login

A hand-drawn sketch of a web browser window. The title bar at the top is grey and contains the text "Todo Listo!". Below the title bar is a navigation bar with four icons: a left arrow, a right arrow, a close button (X), and a home button (house). To the right of these icons is a long rectangular address bar containing the text "http://". To the right of the address bar is a small circular search button with a magnifying glass icon. The main content area of the browser is white and contains the following elements: the text "Todo Listo!" in a large, bold, black font; the label "Usuario" followed by a rectangular input field; the label "Password" followed by a rectangular input field; a rectangular button labeled "Ingresar!"; and a link in blue text that reads "¿No tienes una cuenta? Créala!". The bottom of the browser window is a grey bar with a small double-slash icon in the bottom right corner.

Todo Listo!

← → X 🏠 http:// 🔍

Todo Listo!

Usuario

Password

[¿No tienes una cuenta? Créala!](#)

Interfaz de Usuario: Listado de Tareas

← → ✕ 🏠

Todo Listo! / Home Juanito

http://

Q

Mis Tareas

Calendario

[Cerrar Sesión](#)

Título de la nueva tarea

Seleccione Estado

Crear Nueva Tarea!

Título	Estado	Creada	Terminada
Preparar quizzes Ingeniería Web	Creada	2018-04-02	n/a
Ir al supermercado	Creada	2018-04-01	n/a
Comprar huevitos de chocolate	Terminada	2018-03-30	2018-03-30

Interfaz de Usuario: Editar Tarea

Todo Listo! / Home Juanito

←

→

✕

🏠

http://

🔍

Mis Tareas

Calendario

[Cerrar Sesión](#)

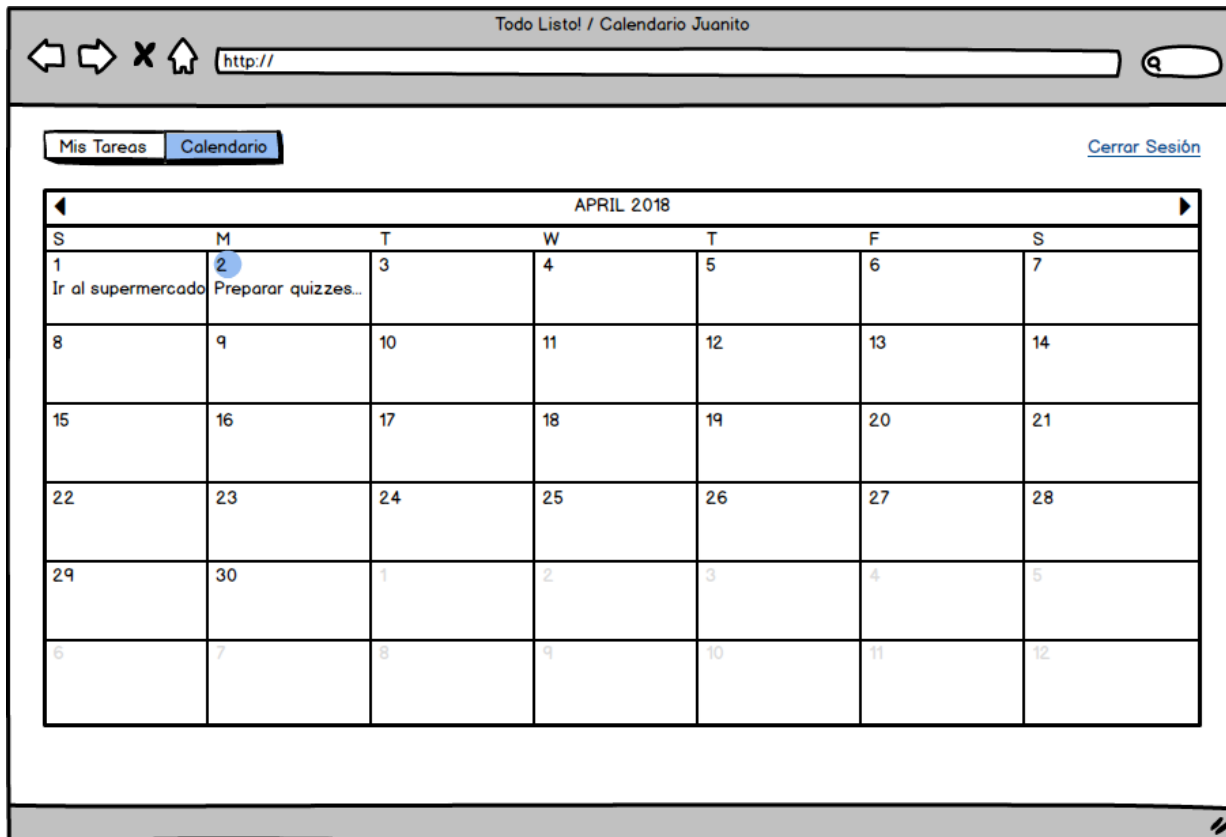
Título de la nueva tarea

Seleccione Estado

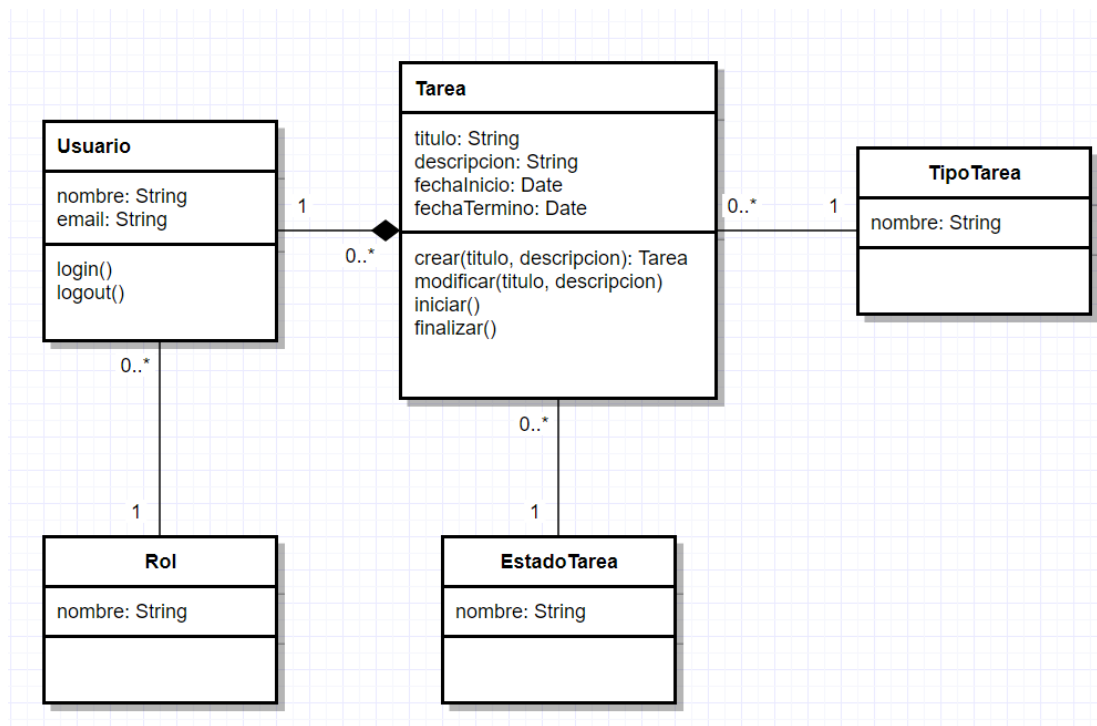
Crear Nueva Tarea!

Título	Estado	Creada	Terminada
Preparar quizzes Ingeniería Web	Creada	2018-04-02	n/a
Ir al supermercado	Creada	2018-04-01	n/a
Comprar huevitos de chocolate	Terminada	2018-03-30	2018-03-30

Interfaz de Usuario: Vista Calendario



El Modelo de la Aplicación



Configuración Base de Datos

- Ejecutar MySQL
- Crear la base de datos “todolisto”
- Crear usuario con acceso completo a la BD “todolisto” -- ***no acceso root!***
 - Usuario: todo
 - Password: listo



Creación desde consola MySQL

```
> mysql -u root -p # Ingresar como root a consola MySQL
```

```
mysql> CREATE DATABASE todolisto;
```

```
mysql> CREATE USER 'todo'@'%' IDENTIFIED BY 'listo';
```

```
mysql> GRANT ALL PRIVILEGES ON todolisto.* TO 'todo'@'%';
```

```
mysql> FLUSH PRIVILEGES;
```



Configuración Base de Datos en Django

- Por defecto Django trabaja con una base de datos SQLite.
- Necesitamos un cliente MySQL para usar la BD desde Python:
 - `pip install mysqlclient`
 - En Linux/Mac necesitamos: `apt-get install libmysqlclient-dev` / `brew install mysql` antes de instalar `mysqlclient`
- Para usar nuestra base de datos MySQL debemos editar el archivo `todo1isto/settings.py`



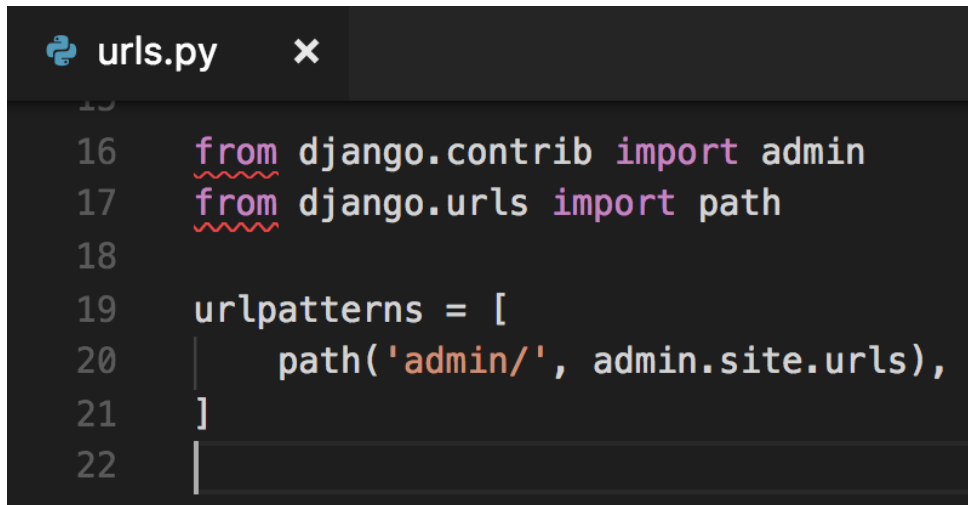
Configuración archivo settings.py

settings.py x

```
71
72
73 # Database
74 # https://docs.djangoproject.com/en/2.0/ref/settings/#databases
75
76 DATABASES = {
77     'default': {
78         'ENGINE': 'django.db.backends.mysql',
79         'NAME': 'todolisto',
80         'HOST': 'localhost',
81         'USER': 'todo',
82         'PASSWORD': 'listo'
83     }
84 }
```


Hola Mundo Django / ¿Dónde se maneja la URL?

- La pantalla de inicio estará asociada a la url <http://localhost:8000/home>
- Debemos crear esta asociación **todolisto/urls.py**
- Por defecto el único ruteo configurado es para la url 'admin' que está asociada a un conjunto de rutas definidas en la aplicación `admin.site.urls`



```
urls.py x
15
16 from django.contrib import admin
17 from django.urls import path
18
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21 ]
22
```

Hola Mundo Django / URL home

- Agreguemos la URL '**todo**listo/' y la asociaremos a las URLs manejadas por **todo**listo.main.urls

```
urls.py  x
15      """
16      from django.contrib import admin
17      from django.urls import include, path
18
19      urlpatterns = [
20          path('admin/', admin.site.urls),
21          path('todo
```

Hola Mundo Django / URL home

- Agreguemos la URL '**todo**listo/' y la asociaremos a las URLs manejadas por **todo**listo.main.urls
- Luego creamos un archivo **main/urls.py** y asociamos la url '**home**' a una vista '**index**'
- En el archivo **main/views.py** definimos el método **index**
- Es bueno ponerle nombre a las vistas, porque nos permite generar URLs dinámicamente en las plantillas

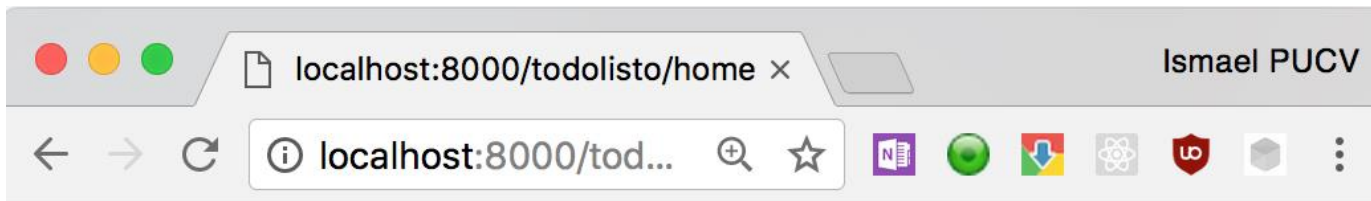
urls.py

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

views.py

```
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  # Create your views here.
5  def index(request):
6      return HttpResponse("Hola Mundo Django!")
7
```

Hola Mundo Django!!!



Hola Mundo Django!

Creando Pantalla de Login


Lo primero que haremos es implementar un login para nuestra aplicación, apoyándonos en la infraestructura que ya trae por defecto Django.

```
urls.py x
1  from django.urls import path
2  from . import views
3  from django.contrib.auth import views as auth_views
4
5  urlpatterns = [
6      path('login', auth_views.LoginView.as_view(), name='login'),
7      path('home', views.index, name="home"),
8  ]
```

<https://docs.djangoproject.com/en/2.0/topics/auth/default/>

<https://simpleisbetterthancomplex.com/tutorial/2016/06/27/how-to-use-djangos-built-in-login-system.html>

Configurando redirección post login

 settings.py x

29

30 LOGIN_URL = '/todolisto/login'

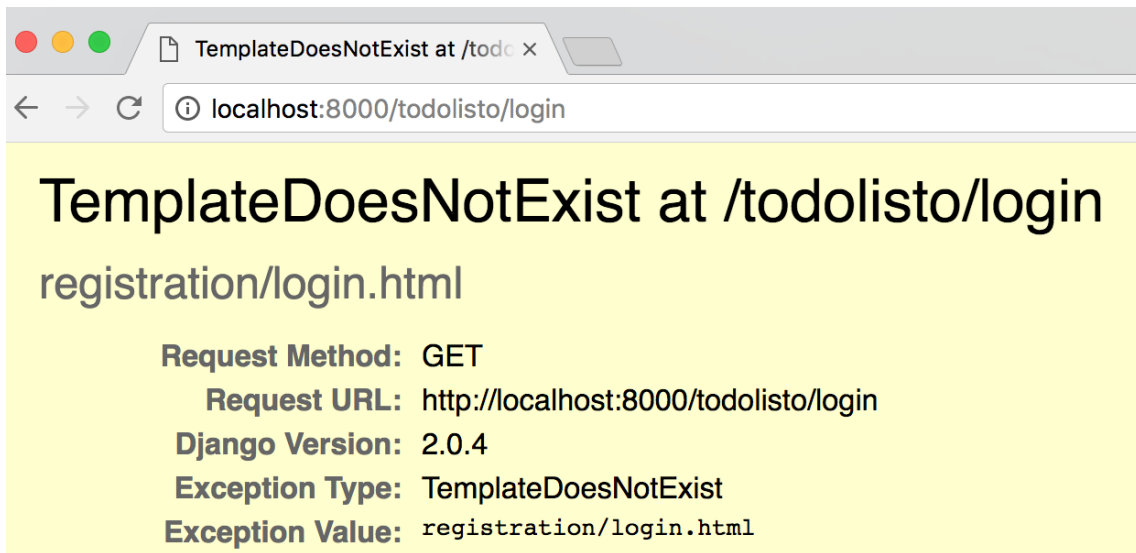
31 LOGIN_REDIRECT_URL = '/todolisto/home'

32

Debemos proveer una plantilla para login

La aplicación auth necesita una plantilla que se encuentre en `registration/login.html`

Debemos proveer esto para poder continuar



Plantilla básica login.html

El archivo está en formato DTL:
Django Template Language

Los elementos entre `{% %}` son
etiquetas...

Los elementos entre `{{ }}` son
variables que se renderizan

La vista LoginView nos provee de
la variable **form** con el formulario
de login...

```
<> login.html x
1   <h2>Login</h2>
2   <form method="post">
3       {% csrf_token %}
4       {{ form.as_p }}
5       <button type="submit">Login</button>
6   </form>
```

**¿probemos el login
con nuestro
superusuario?**

Creando enlace para cerrar sesión / logout

De manera similar, se espera una plantilla `'registration/logged_out.html'` para ser mostrada cuando el usuario cierre su sesión

```
urls.py
1  from django.urls import path
2  from . import views
3  from django.contrib.auth import views as auth_views
4
5  urlpatterns = [
6      path('login', auth_views.LoginView.as_view(), name='login'),
7      path('logout', auth_views.LogoutView.as_view(), name='logout'),
8      path('home', views.index, name="home"),
9  ]
```

De manera parecida se puede agregar el registro de usuarios, cambio de password, etc.

¿Qué pasa con la seguridad?

Lamentablemente en nuestra aplicación podemos acceder a `todo/` incluso si no hemos iniciado sesión!

Para poder prevenir el acceso a una URL a usuarios no autenticados debemos usar el decorador **`login_required`**

 `views.py` x

```
1  from django.shortcuts import render
2  from django.http import HttpResponse
3  from django.contrib.auth.decorators import login_required
4
5  @login_required()
6  def index(request):
7      return HttpResponse("Hola Mundo Django!")
```

Mejorando el layout de nuestra aplicación

Con el fin de mejorar nuestra aplicación explotaremos el mecanismo de plantillas de Django para una plantilla maestra con:

- Encabezado, pie de página y barra de navegación comunes a toda la aplicación. Con enlaces de iniciar/cerrar sesión según corresponda
- Espacio para incrustar el contenido específico de cada vista en particular
- Usaremos Bootstrap para la parte de estilos gráficos



Todo lo encerrado en rojo
corresponde al layout en
Bootstrap

```
base.html x
1  {% load staticfiles %}
2
3  <!DOCTYPE html>
4  <html>
5  <head>
6      <meta charset="utf-8">
7      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
8
9      <!-- Bootstrap CSS -->
10     <link rel="stylesheet"
11         href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
12         integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiWGPXr1jddIh0egiu1Fw05qRGvFX0dJZ4"
13         crossorigin="anonymous">
14
15     <title>Todo Listo!</title>
16 </head>
17 <body>
18     <nav class="navbar navbar-expand-lg navbar-light bg-light">
19         <a class="navbar-brand" href="#">Navbar</a>
20         <ul class="navbar-nav mr-auto">
21             <li class="nav-item active">
22                 <a class="nav-link" href="{% url 'home' %}">Home<span class="sr-only">(current)</span></a>
23             </li>
24         </ul>
25     </nav>
26
27
28     {% block content %}{% endblock %}
29
30     <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/js/bootstrap.min.js"
31         integrity="sha384-uefMccjFJAIv6A+rW+L4AHf99KvxDjWsu1z9VI8SKNVmz4sk7buKt/6v9KI65qnm"
32         crossorigin="anonymous">
33     </script>
34 </body>
35 </html>
```

<> base.html ●

```
1 {% load staticfiles %}
2
3 <!DOCTYPE html>
4 <html>
5 <head>
6 |   <title>Todo Listo!</title>
7 </head>
8 <body>
9
10 {% block content %}{% endblock %}
11
12 </body>
13 </html>
```

Etiqueta para cargar contenido estático:
imágenes, JS, CSS, etc... más adelante lo
veremos

Define un nuevo bloque llamado '**content**'
que puede ser relleno por cualquier plantilla
que extienda esta plantilla base

Aplicando el layout para nuestras vistas

<> login.html x

```
1  {% extends 'base.html' %}
2
3  {% block content %}
4  <h2>Login</h2>
5  <form method="post">
6      {% csrf_token %}
7      {{ form.as_p }}
8      <button type="submit">Login</button>
9  </form>
10  {% endblock content %}
11
```

<> logged_out.html ●

```
1  {% extends 'base.html' %}
2
3  {% block content %}
4
5  ADIOS
6
7  {% endblock content %}
8
```

Reimplementando Hola Mundo / home.html

<> home.html ●

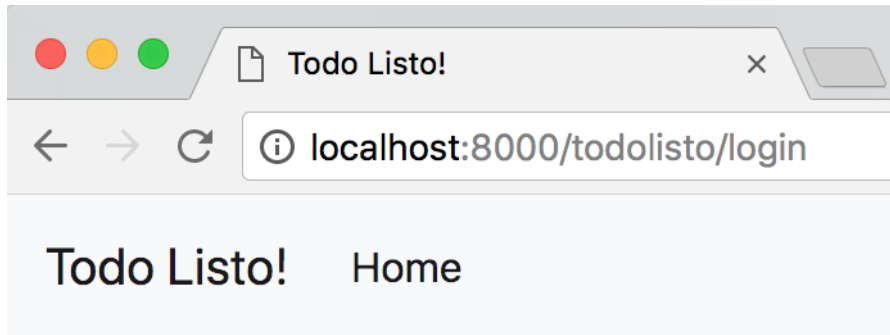
```
1 {% extends 'base.html' %}
2
3 {% block content %}
4
5     HOLA MUNDO DJANGO!
6
7 {% endblock content %}
8
```

Reimplementando Hola Mundo / views.py

views.py

```
1  from django.shortcuts import render
2  from django.http import HttpResponse
3  from django.contrib.auth.decorators import login_required
4
5  @login_required()
6  def index(request):
7      return render(request, 'home.html')
```


Ya podemos partir!



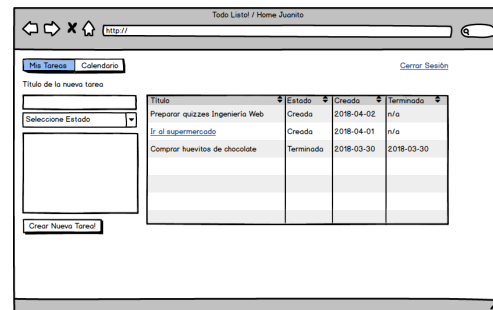
Login

Username:

Password:

Login

Implementación Listado de Tareas



- Debemos crear la vista **tareas.html**
- Debemos crear el método **tareas** en **views.py**, que debe renderizar la vista recién creada
- Debemos agregar el decorador **@login_required** para que se requiera inicio de sesión
- En **urls.py** asociar la ruta '**tareas**' al método **views.tareas**
- En el método **views.index** hacer una redirección hacia la la ruta '**tareas**'
- Codificar la vista usando HTML + Bootstrap, extendiendo el layout **base.html**
- Debemos incluir `{% csrf_field() %}` en el formulario de creación de tareas

urls.py x

```
1 from django.urls import path
2 from . import views
3 from django.contrib.auth import views as auth_views
4
5 urlpatterns = [
6     path('login', auth_views.LoginView.as_view(), name='login'),
7     path('logout', auth_views.LogoutView.as_view(), name='logout'),
8     path('home', views.index, name="home"),
9     path('tareas', views.tareas, name="tareas"),|
10 ]
```

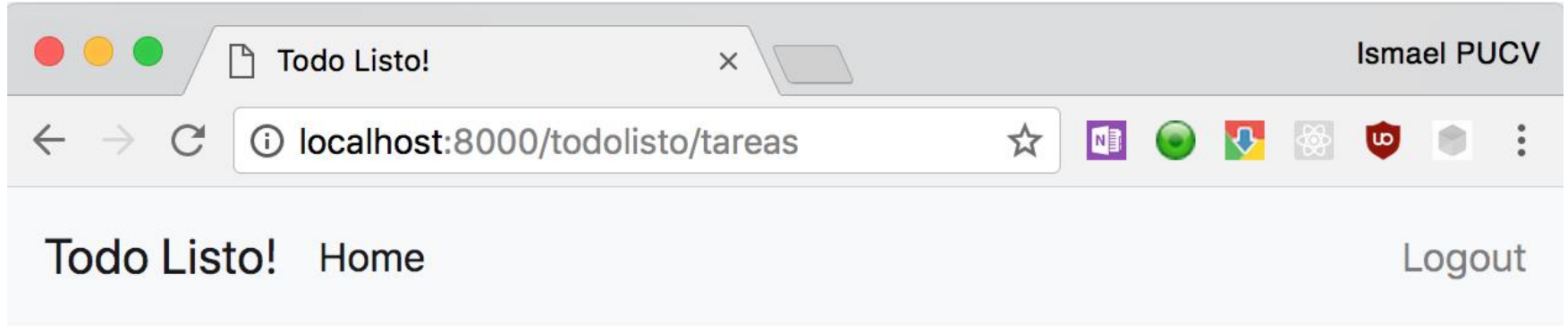
views.py x

```
1 from django.shortcuts import render, redirect|
2 from django.http import HttpResponse
3 from django.contrib.auth.decorators import login_required
4
5 @login_required()
6 def index(request):
7     return redirect('tareas')
8
9
10 @login_required()
11 def tareas(request):
12     return render(request, 'tareas.html')
13
```

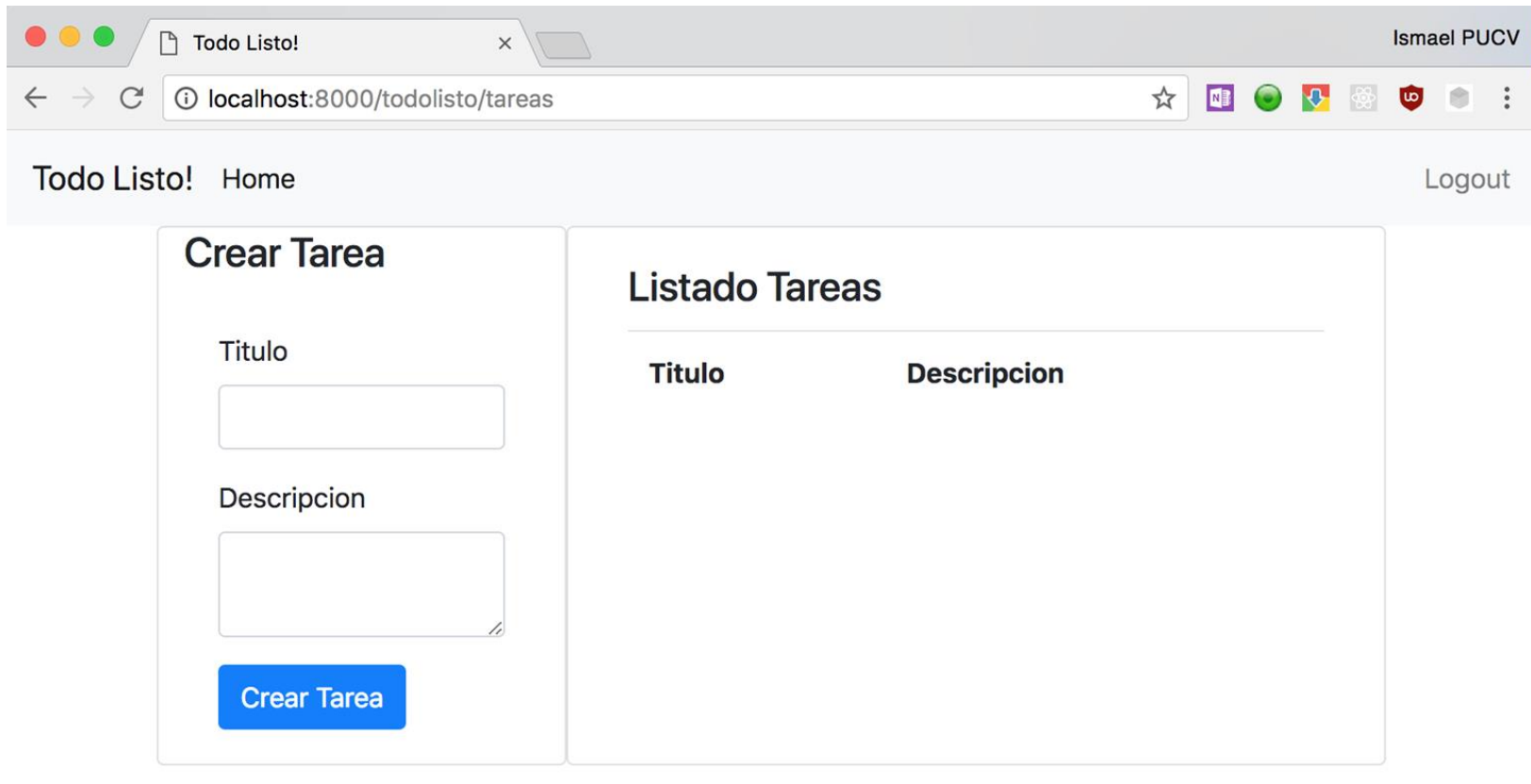
tareas.html x

```
1 {% extends 'base.html' %}
2
3 {% block content %}
4 <div class="container">
5     <div class="row">
6
7         <div class="col-4">
8             FORMULARIO
9         </div>
10
11         <div class="col-8">
12             LISTADO
13         </div>
14     </div>
15 </div>
16 {% endblock content %}
```

Interfaz de Usuario: Listado de Tareas



Interfaz de Usuario: Listado de Tareas



Ismael PUCV

Todo Listo! Home Logout

Crear Tarea

Titulo

Descripcion

Crear Tarea

Listado Tareas

Titulo	Descripcion
--------	-------------

Controlando la Creación de Tareas

- Debemos actualizar el ruteo en `urls.py` para agregar la ruta '`crear_tarea`' y mapearla a `views.crear_tarea`. **Debemos mapearla con método POST.** Pongámosle nombre '`crear_tarea`'
- En el formulario de creación de tareas debemos asignar el campo **action** con la URL `{% url 'crear_tarea' %}` recién creada, para que procese la información en esa ruta
- Debemos crear el método `views.crear_tarea`, el método retornará un redirección a la ruta '`tareas`'.



¿Ahora cómo guardo la
tarea en la base de datos?



Creando el Modelo Tarea

- Para almacenar tareas en la base de datos debemos definir un modelo en Django para representar efectivamente la información que queremos almacenar.
- Un modelo se crea por defecto en el archivo **models.py**. Un modelo es una clase Python que extiende desde **django.db.models.Model**
- La clase define los campos a incorporar en el modelo, y eventualmente la base de datos. Debemos especificar el tipo de campo de cada elemento del modelo
- Django por defecto incorpora un campo **'id'** con una llave primaria artificial autoincremental

models.py x

```
1 from django.db import models
2
3 class Tarea(models.Model):
4     titulo = models.CharField(max_length = 255)
5     descripcion = models.CharField(max_length = 255)
```


Creando y Aplicando Migraciones

Si bien ahora el modelo Tarea existe como una entidad en nuestra aplicación, todavía no tenemos su contraparte en la base de datos. Para esto debemos:

- Crear migraciones: **`python manage.py makemigrations`**
- Aplicar las migraciones: **`python manage.py migrate`**

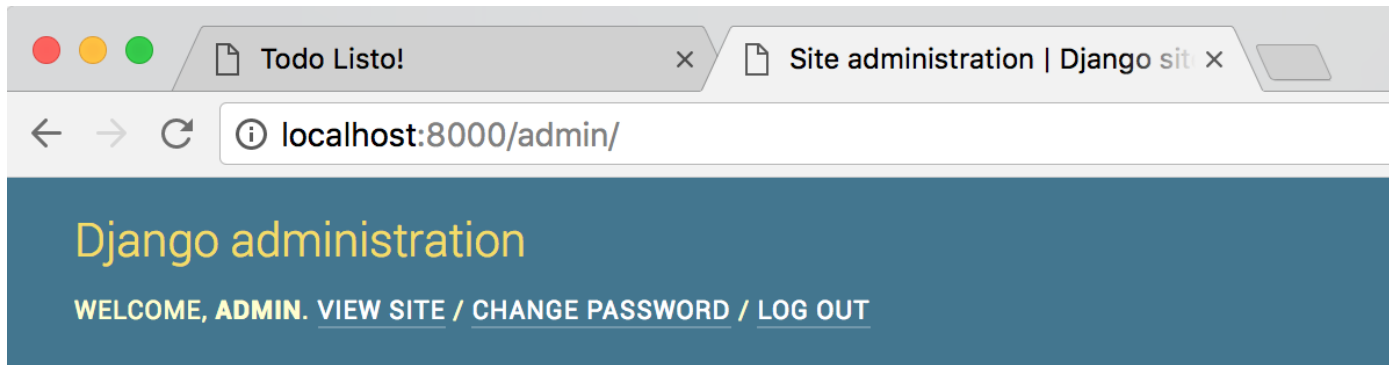
Si vemos nuestra base de datos veremos una nueva tabla `main_tarea` con los campos `id`, `titulo`, `descripcion`

Registrando nuestro modelo en admin

- Si ingresamos al backend administrativo, veremos que nuestro modelo no aparece ahí!
- En Django cada aplicación debe registrar explícitamente los modelos que se pueden manejar desde la aplicación admin.
- Para eso debemos editar el archivo **admin.py**

 admin.py x

```
1  from django.contrib import admin
2  from .models import Tarea
3
4  admin.site.register(Tarea)
```



Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

[+ Add](#) [✎ Change](#)

Users

[+ Add](#) [✎ Change](#)

MAIN

Tareas

[+ Add](#) [✎ Change](#)

¿Ahora cómo guardo la
tarea en la base de datos?



Creando una nueva Tarea / Usando el modelo

```
@login_required()
def crear_tarea(request):
    if request.method == 'POST':
        tarea = Tarea.objects.create()
        print(request.POST)
        tarea.titulo = request.POST.get('titulo_tarea')
        tarea.descripcion = request.POST.get('desc_tarea')
        tarea.save()

    return redirect('tareass')
```

- Ahora que existe el modelo **Tarea** podemos usarlo en **views.crear_tarea**
- Al crear una nueva tarea, ésta aparecerá en la base de datos, pero no en el listado de nuestra aplicación D:
- Debemos actualizar **views.tareas** para cargar los valores



Mostrando el Listado de Tareas

```
@login_required()
def tareas(request):
    tareas = Tarea.objects.all()
    return render(request, 'tareas.html', { 'tareas' : tareas })
```

- El método **views.tareas** retorna la vista donde debería aparecer el listado de tareas
- El listado debe obtenerse mediante una consulta al modelo Tarea
- Además, la vista debe tomar este listado **como un parámetro**
- La vista debe entonces renderizar el listado de tareas

Mostrando el Listado de Tareas

- El contexto pasado como parámetro define las variables disponibles en la plantilla
- La etiqueta **for** permite hacer una iteración sobre una colección

```
<table class="table">
  <tr>
    <th>Titulo</th>
    <th>Descripcion</th>
  </tr>
  {% for t in tareas %}
  <tr>
    <td>{{ t.titulo }}</td>
    <td>{{ t.descripcion }}</td>
  </tr>
  {% endfor %}
</table>
```

Todo Listo!

Ismael PUCV

localhost:8000/todolisto/tareas

Todo Listo! Home

Logout

Crear Tarea

Titulo

Descripcion

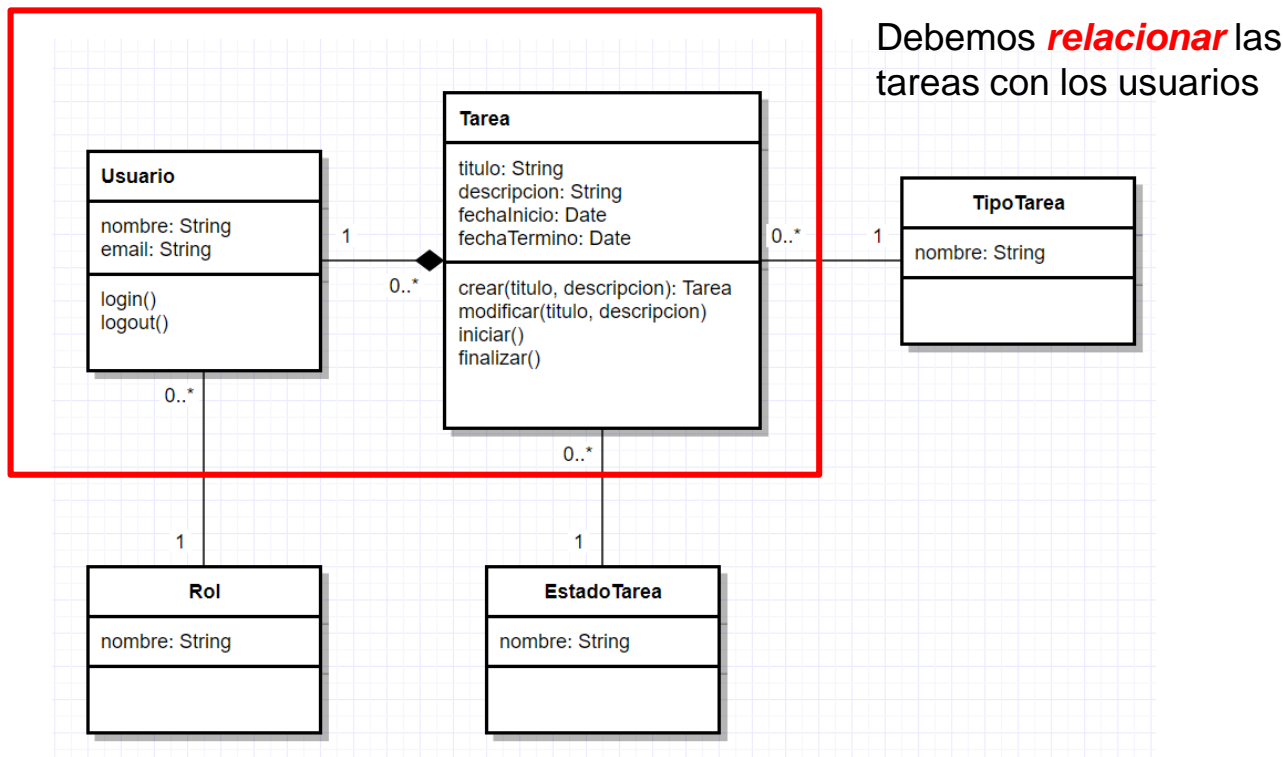
Crear Tarea

Listado Tareas

Titulo	Descripcion
A	B

El problema ahora es que estamos viendo todas las tareas de todos los usuarios!!

Revisemos el Modelo de la Aplicación



¿Creando el Modelo Usuario?

- Al usar el sistema de autenticación de Django ya existe un modelo **User**!
- Debemos entonces crear una relación entre **Tarea** y **User**
- A nivel del modelo:
 - En la clase **Tarea** debemos agregar una **ForeignKey** que apunte a User
 - El uso de ForeignKey establece una relación one-to-many

Actualizando Tarea


models.py x

```
1  from django.db import models
2  from django.contrib.auth.models import User
3
4  class Tarea(models.Model):
5      titulo = models.CharField(max_length = 255)
6      descripcion = models.CharField(max_length = 255)
7      usuario = models.ForeignKey(User, on_delete = models.CASCADE)
8
```

Ojo!

```
(todolisto) MBP-de-Ismael:todolisto ismael$ python manage.py makemigrations
You are trying to add a non-nullable field 'usuario' to tarea without a default
; we can't do that (the database needs something to populate existing rows).
Please select a fix:
  1) Provide a one-off default now (will be set on all existing rows with a null
    value for this column)
  2) Quit, and let me add a default in models.py
Select an option: █
```

**Debemos indicar qué hacer con las filas que ya existen
cuando agregamos un nuevo campo no nulo!**



Actualizando creación de Tareas

```
@login_required()
def crear_tarea(request):
    if request.method == 'POST':
        tarea = Tarea()
        tarea.titulo = request.POST.get('titulo_tarea')
        tarea.descripcion = request.POST.get('desc_tarea')
        tarea.usuario = request.user
        tarea.save()

    return redirect('tareass')
```

- **Tarea.objects.create()** crea y guarda inmediatamente en la BD. Debemos pasar los campos no nulos inmediatamente
- **Tarea()** crea un objeto en memoria, que solo se guarda en la BD cuando invocamos el método **save()**

Actualizando Listado de Tareas

views.py x

```
11
12 @login_required()
13 def tareas(request):
14     tareas = Tarea.objects.filter(usuario=request.user)
15     return render(request,
16                   'tareas.html',
17                   { 'tareas' : tareas })
```

- Para obtener las tareas del usuario actual, debemos hacer un filtro sobre los objetos del modelo **Tarea**

Ahora la primera parte funciona!

