WebServers con python-django

Esteban Lanzarotti - Leandro Radusky

November 26, 2014

Requerimientos

Paquetes

`python`
`python-django`

En linux y mac está disponible en los repositorios y su uso es similar. En windows se descarga un instalable y los comandos se ejecutan desde el cmd.

Nosotros les enseñamos esto

Para lograr esto hay que practicar!

Dónde escribir el código

Editores de texto

`gedit`
`geany`
`sublime`
`notepad++`

Entornos de desarrollo

`eclipse+PyDev`
`PyCharm`

Etcétera, etcétera...

Creando un proyecto

Escribimos en el terminal/cmd:

```
django-admin startproject DjangoE2B2C
```

Django-admin es el comado que django nos provee para administrar proyectos. Startproject es solo un subcomando de los muchos disponibles.

Creando un proyecto

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

manage.py

Permite administrar el proyecto mediante subcomandos que vamos a ver a continuacion.

settings.py

Contiene las definiciones de las propiedades del proyecto.

urls.py

Contiene las definiciones de cómo el proyecto procesará los pedidos del cliente.

Un vistazo a las propiedades

Las propiedades a definir son las siguientes:

- `INSTALLED_APPS`: indica qué aplicaciones están instaladas en el proyecto.

Projects vs. apps

What's the difference between a project and an app? An app is a Web application that does something – e.g., a Weblog system, a database of public records or a simple poll app. A project is a collection of configuration and apps for a particular Web site. A project can contain multiple apps. An app can be in multiple projects.

Philosophy

Django apps are "pluggable": You can use an app in multiple projects, and you can distribute apps, because they don't have to be tied to a given Django installation.

Un vistazo a las propiedades

- `DATABASES`: Qué bases de datos y con qué motores va a manejarse el proyecto.
- `ROOT_URLCONF`: Ubicación del archivo con las urls.
- `STATIC_URL`: Carpeta con contenido estático para el proyecto.

Ejecutando el proyecto

python manage.py migrate

Migra todo lo necesario a base de datos y prepara todo para poder ejecutar el proyecto con las aplicaciones instaladas.

python manage.py runserver

Ejecuta el proyecto levantando un webserver.

Listo! Ya tenemos un webserver minimal que no sirve para nada corriendo.

Creando una aplicacion

Hasta el momento tenemos un proyecto con aplicaciones instaladas que son básicas, ofrecidas por django. Ahora queremos hacer algo nosotros.

En el terminal:

```
python manage.py startapp Estructuras
```

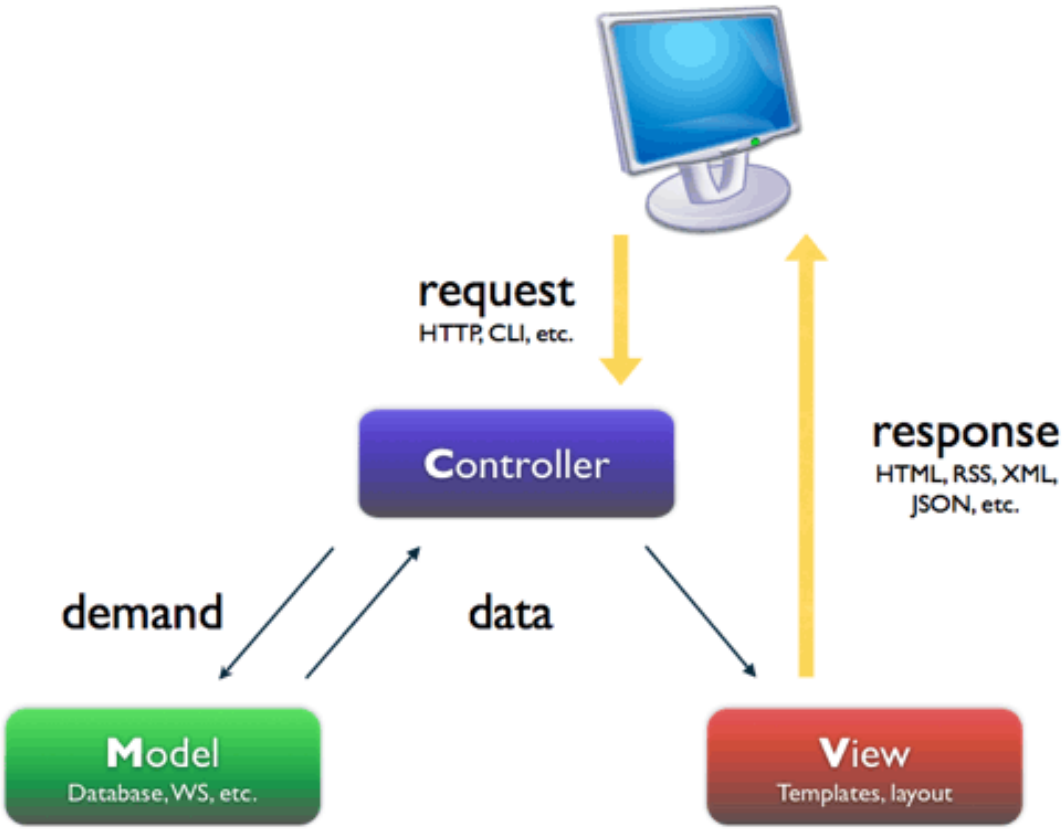
Creando una aplicacion

Esto crea varios archivos de interés:

- `admin.py` Archivo donde se controlará el funcionamiento de la aplicación.
- `models.py` Archivo donde se definirá el modelo.
- `views.py` Archivo donde se definirán las vistas.

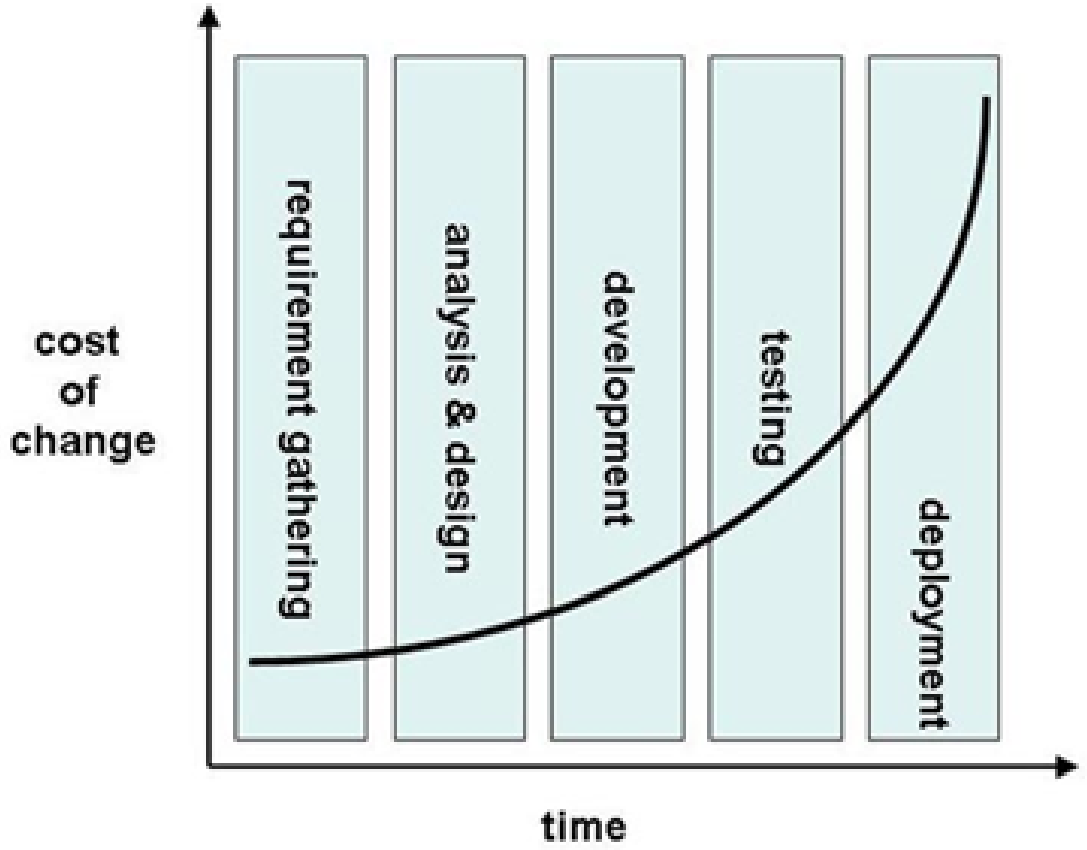
Cada uno de los archivos representa un componente del patrón de diseño MVC.

Patrón de diseño MVC



Definiendo el modelo

Debemos pensar qué clases son las que representan el modelo de datos de nuestro problema. Las clases tienen atributos de distintos tipos y pueden estar relacionadas entre sí.



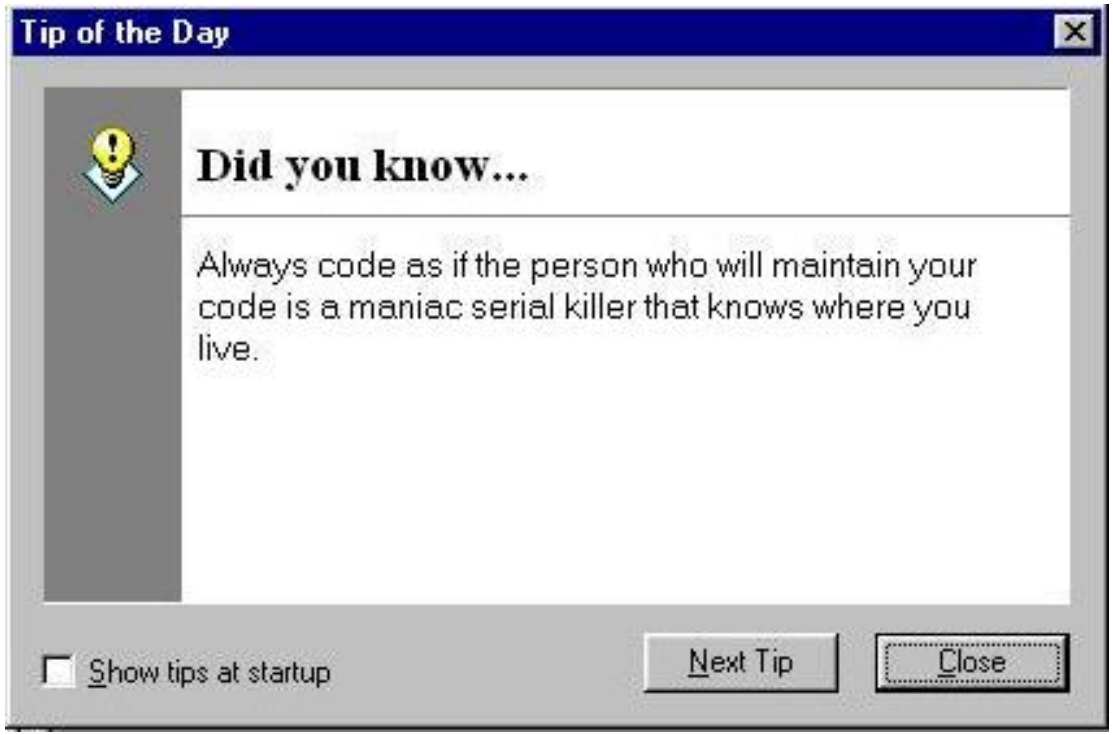
Definiendo el modelo

En nuestro problema de ejemplo vamos a guardar estructuras de proteínas.

- `class Estructura:` Tiene un código, una descripción y una resolución.
- `class Cadena:` Tiene un código y una estructura a la que está relacionada.
- `class Residuo:` Tiene un código, una posición y una cadena a la que está relacionada.

Debemos escribirlo en model.py.

Siempre tener en cuenta a la hora de codificar



Instalando la aplicación

Queremos impactar los cambios, por lo que tenemos que 'instalar' la aplicación en nuestro proyecto. Luego de instalarla, queremos impactar los cambios:

```
python manage.py makemigrations Estructura
```

Creará un archivo python en la carpeta migraciones en el cual se basará para manejar el comportamiento.

Instalando la aplicación

Queremos impactar los cambios, por lo que tenemos que 'instalar' la aplicación en nuestro proyecto. Para esto hay que agregarla en settings.py. Luego de instalarla, queremos impactar los cambios:

- `python manage.py sqlmigrate Estructura 0001:` Impactará los cambios en la base de datos. El 0001 el número de versión arrojado por el makemigrations, si queremos volver a una versión anterior tenemos todo lo necesario para hacerlo.

Instalando la aplicación

```
python manage.py migrate
```

Impacta estas migraciones efectivamente en el proyecto.

Antes de la web

```
python manage.py shell
```

Nos permite acceder por línea de comando al proyecto.

Creando un usuario para nuestra aplicación

```
python manage.py createsuperuser
```

Con este super usuario podremos acceder al sitio de administración del proyecto, el cual nos servirá para realizar las operaciones básicas de alta, baja y modificación de registros.

Si corremos el proyecto y accedemos al sitio de administración tendremos acceso con este superusuario. Podremos crear otros usuarios y grupos de usuarios (que comparten privilegios).

Haciendo accesibles los datos de nuestra aplicación

Agregamos a admin.py

```
from models import estructura, cadena
admin.site.register(estructura)
admin.site.register(cadena)
```

Esto le dice a nuestro proyecto que desde el sitio de admin podemos acceder a los registros de estas clases.

Mostrando las clases de una manera amigable

Redefinimos la función __str__ y decimos qué nos interesa que se muestre en la tabla con todos los objetos.

Mostrando las clases de una manera amigable

Redefinimos la función __str__ y decimos qué nos interesa que se muestre en la tabla con todos los objetos.

Cargando los datos de una manera más amigable

Podemos, al cargar estructuras, en el momento pedir las cadenas que la componen:

En el archivo admin.py

```
class CadenaInline(admin.StackedInline):
    model = cadena
    extra = 1
```

Cargando los datos de una manera más amigable

- Podemos, al cargar estructuras, en el momento pedir las cadenas que la componen de manera **inline**.
- Podemos también definir **fieldsets** que agrupan los campos de manera visual.
- Podemos, en vez de mostrar lo que nos devuelve la función str, mostrar una tabla con los campos de lo que pongamos en la variable **list_display**.

Cargando los datos de una manera más amigable

Definimos esas variables en una nueva clase de administracion

En el archivo admin.py

Definimos la clase:

```
class EstructuraAdmin(admin.ModelAdmin)
```

y al registrarla en el site la definimos como clase para display de la clase estructura:

```
admin.site.register(estructura, EstructuraAdmin)
```

Creando vistas personalizadas

Para el alta, baja y modificación de registros de la base de datos podemos usar lo que el sitio de administración nos proporciona.
Pero cuando necesitamos hacer cosas más sofisticadas, podemos crear nuestras propias vistas.
Vamos a crear una página de bienvenida a nuestra aplicación.

Proyecto/urls.py

Agregamos la línea

```
url(r'^ structs/', include('structs.urls')),
```

Creando vistas personalizadas

Ahora podemos definir las vistas específicas de la aplicación en el archivo structs/urls.py.

structs/urls.py

Agregamos la línea:

```
url(r'^ $', views.index, name='index'),
```

Cuando el usuario quiera acceder a la url de la aplicación, le mostraremos una pantalla de bienvenida. El manejo de los request de usuario es con estructuras regulares (todo un tema aparte).

Creando vistas personalizadas

Tenemos que definir qué hacer para resolver lo que el cliente pide a través de las urls.

structs/views.py

Debemos agregar la funcion index para responder al pedido que acabamos de definir.

Cuando el usuario quiera acceder a la url de la aplicación, le mostraremos una pantalla de bienvenida.

Creando vistas personalizadas

Podemos directamente enviar código o definir un template.
Veamos un ejemplo al indagar por el detalle de una estructura.

Creando vistas personalizadas

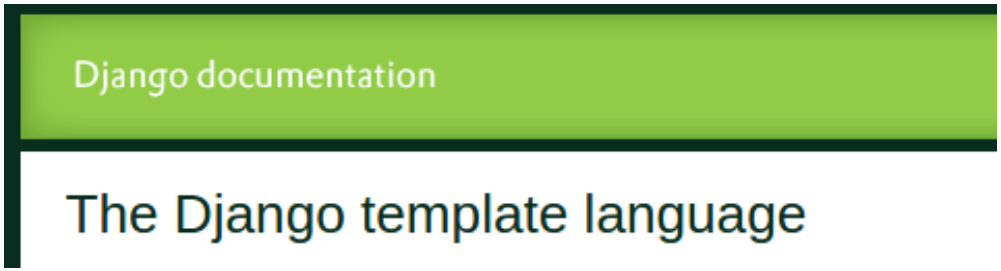
Los templados nos permiten:

- Mantener estructuras de control (if, for).
- Hacer inclusiones de porciones de código que se mantendrán estáticas en todas las páginas.
- Acceder a la dirección de archivos estáticos.
- etc, etc.

Veamos un ejemplo al indagar por el detalle de una estructura.

Cosas con las que hay que seguir

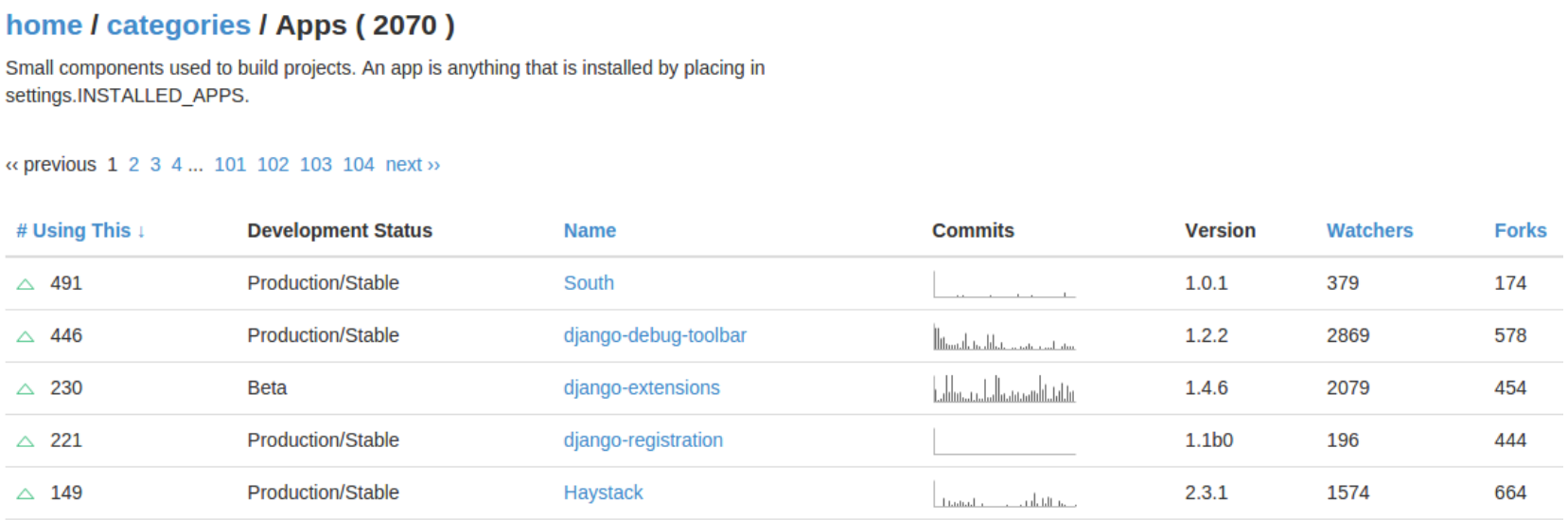
- Aprender el lenguaje de templates



Hay una variedad de cosas que me solucionan la vida, como la herencia de templates.

Cosas con las que hay que seguir

- Usar aplicaciones que otros ya hicieron



home / categories / Apps (2070)

Small components used to build projects. An app is anything that is installed by placing in settings.INSTALLED_APPS.

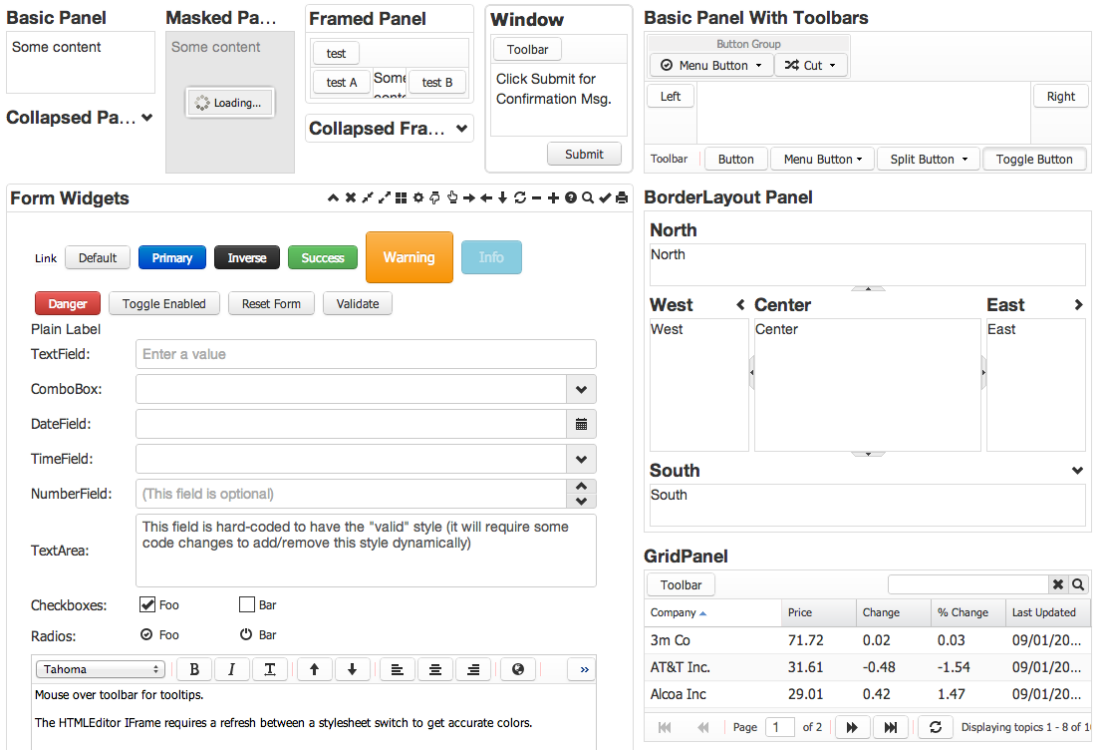
‹‹ previous 1 2 3 4 ... 101 102 103 104 next ››

# Using This ↓	Development Status	Name	Commits	Version	Watchers	Forks
△ 491	Production/Stable	South		1.0.1	379	174
△ 446	Production/Stable	django-debug-toolbar		1.2.2	2869	578
△ 230	Beta	django-extensions		1.4.6	2079	454
△ 221	Production/Stable	django-registration		1.1b0	196	444
△ 149	Production/Stable	Haystack		2.3.1	1574	664

Manejo amigable de formularios, logeo a las redes sociales, carrito de compras, internacionalizacion etc, etc.

Cosas con las que hay que seguir

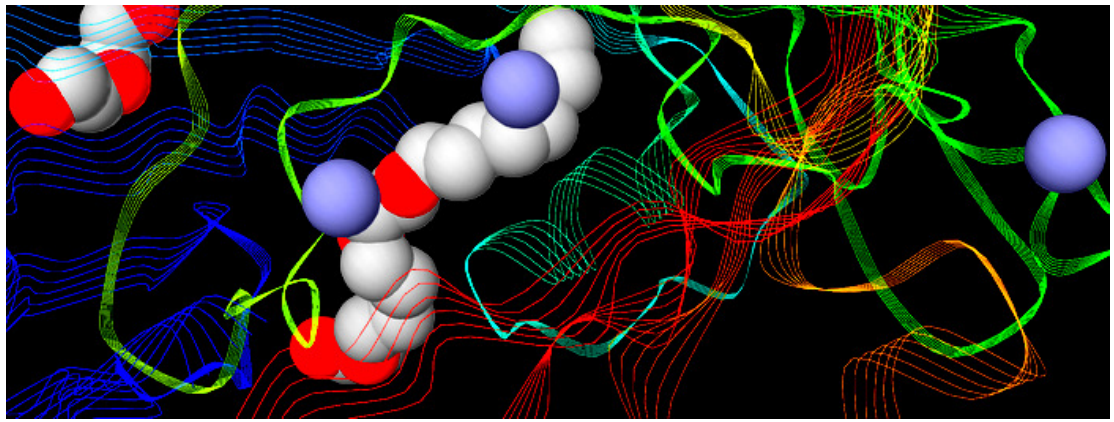
- Usar frameworks como bootstrap



Todo se muestra de una manera amigable, se acomoda en base al tamaño, se puede ver en celulares, etc.

Cosas con las que hay que seguir

- Mostrar moleculas con GLMol



Es una librería de javascript que permite hacerlo, hay otras como JMol, también hay librerías para mostrar gráficos como D3.js, etc.