

INSTITUTO TECNOLÓGICO DE COSTA RICA
ESCUELA DE INGENIERÍA EN COMPUTACIÓN
ADMINISTRACIÓN DE TECNOLOGÍAS DE INFORMACIÓN

TI-5003 APLICACIONES WEB

TWIPYMAP

Tarea programa desarrollada en Django

Profesor: Andrei Fuentes L.

Estudiante: Ariela Vargas Vargas

13/11/2012
Cartago, Costa Rica

Contenido

| | |
|--|----|
| Descripción del problema | 2 |
| Diseño del programa | 2 |
| Decisiones de diseño | 2 |
| Diagramas de clases | 3 |
| Diagrama de arquitectura | 3 |
| Uso de Django | 4 |
| Manejo de controladores (views) | 4 |
| Manejo de vistas (html) | 4 |
| Manejo de rutas (url) | 5 |
| Manejo de modelos (models) | 5 |
| Librerías externas | 6 |
| Google Maps | 6 |
| API de Twitter..... | 7 |
| Manual de usuario | 8 |
| Análisis de resultados y Conclusión personal | 10 |

Descripción del problema

Se necesita crear por medio del framework Django una aplicación que emplee el uso de los API de Google Maps y Autenticación de Twitter.

Lo que se debe desarrollar es una lista de contactos asociadas a un usuario autenticado por medio de Twitter, y en la cual permite agregar, modificar, eliminar y consultar los datos de los contactos. Así también se debe dar la dirección del contacto por medio del mapa de Google maps y realizar consultas a través de parámetros que puedan utilizarse la ubicación de la dirección.

Existen dos maneras de realizar la aplicación. Investigando sobre Django, para realizar código autogenerado se encuentra un `generate_scaffold` que al ser instalado permite crear las vistas, el modelo y el controlador. Por otro lado, se puede crear paso a paso las clases con sus respectivos, como se realizó anteriormente el laboratorio de Polls.¹

Se decide implementar la primera, por tanto se autogeneran los modelos y a partir de ello se empieza a elaborar la aplicación.

Diseño del programa

A continuación se presenta las decisiones que se tomaron para realizar la elaboración de los modelos de la aplicación y con base en ellos la creación de las funcionalidades necesarias para realizar las referencias y desplegarlas al usuario.

Decisiones de diseño

Al poseer tan poco conocimiento del framework no sé determina exactamente cómo seguir así que se empieza a investigar sobre Django para poder crear los modelos para la tarea. No obstante, al crearlos y empezar a investigar con Google Maps y Twitter se debe autogenerar nuevamente los modelos desechando las aplicaciones anteriores, debido a que Django, en comparación con Ruby on Rails, no posee migraciones.

Creados los modelos `contact` y `user`, se empieza trabajar con el API de Google Maps, para lo cual se obtiene el API Access para poder hacer uso de un token de OAuth válido la cual permita superar la conexión de las solicitudes del proyecto. Se implementan varios mapas fuera del framework para poder familiarizarse con el código y una vez que se posee un mapa el cual pueda ser representado en el html con un `maker` (elemento que permite añadir y actualizar información geográfica).

Como se mencionó, el código fue autogenerado, por lo que en decisiones de diseño respecto a esto no existieron como tal, sólo la definición de cual serían los atributos del modelo. A esto es importante añadir que conforme el avance en el proyecto existieron múltiples problemas para hacer uso del código, en su mayoría cajas negras, para poder implementar la sección de consultas de direcciones y el mapa como tal.

¹(<https://docs.djangoproject.com/en/dev/intro/tutorial01/>)

Diagramas de clases

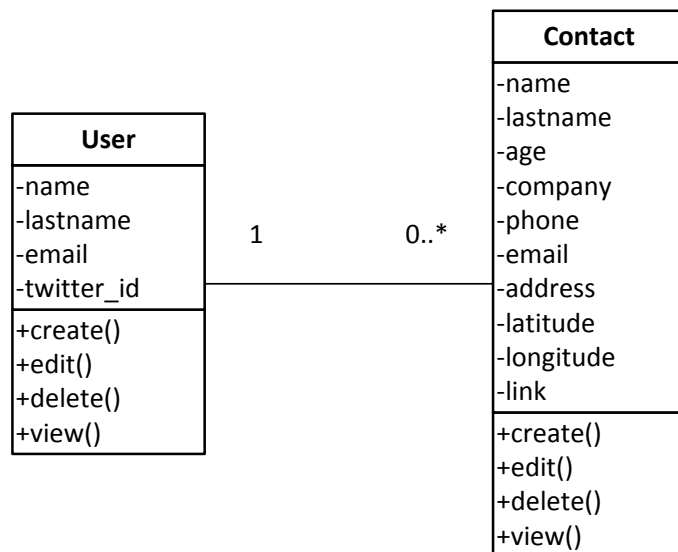
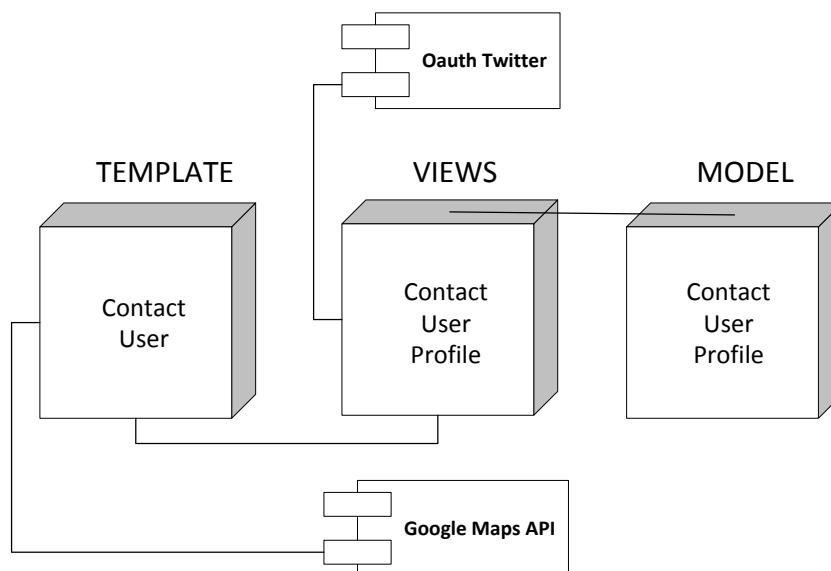


Diagrama de arquitectura



Uso de Django

A continuación se presenta el manejo de los aspectos del framework.

Manejo de controladores (views)

El manejo de los “controladores” en Django se utiliza mediante los archivos views. En el caso de la aplicación existe un `contact_views.py` y un `user_views.py`. Dentro de cada uno de ellos existe codificado las clases respectivas para cada vista, no obstante su implementación comprime la información de manera tal que resultó difícil modificar sus parámetros.

```
def get_template_names(self):
    """Nest templates within user directory."""
    tpl = super(Userview, self).get_template_names()[0]
    app = self.model._meta.app_label
    mdl = 'user'
    self.template_name = tpl.replace(app, '{0}/{1}'.format(app, mdl))
    return [self.template_name]
```

Figura 1. Ejemplo del código de `user_views.py`

Manejo de vistas (html)

El autogenerador de código crea una carpeta para cada modelo. En este caso se tiene la carpeta `contact` y `user`, las cuales contienen 13 vistas asociadas (`map.html` fue creada para tratar de insertar el mapa mediante un `include` del `html`). Cada una de ellas es asociada a una acción o, en este caso, clase del controlador.

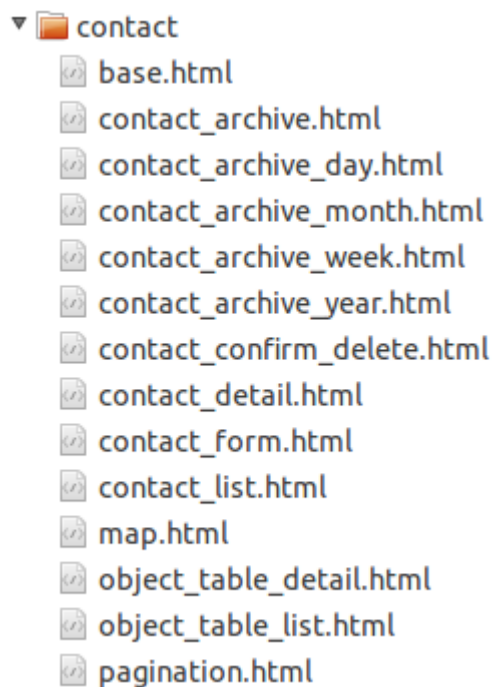


Figura 2. Vistas del modelo `contact`

Existe un html que contiene la estructura total de todas las demás vistas, este es base.html. Esencialmente este se compone de todo el css de las vistas así de cómo estructurar los datos, imágenes, tables. Esto limitó las necesidades de diseño, ya que conforme se quiso ir “acomodando” los elementos para modificar la estructura, el código “por debajo” estaba alambrado de manera tal que o los cambios no surtían efecto o no lo permitía el framework.

Manejo de rutas (url)

Dentro del archivo url.py se utilizaron las siguientes rutas para hacer uso del API de twitter y navegar dentro de la aplicación denominada twipymap2.

```
url(r'^$', include('twipymap2.urls')),
url(r'^login/?$', twitter_login),
url(r'^logout/?$', twitter_logout),
url(r'^login/authenticated/?$', twitter_authenticated),
```

Así también se hicieron los imports necesarios para el API de twitter, los defaults import y url.

```
from django.conf.urls import patterns, include, url
from django.conf.urls.defaults import *
from twipymap2.views.profile_views import twitter_login, twitter_logout, twitter_authenticated
```

Manejo de modelos (models)

Respecto a los modelos se definieron dos: Contact y User. Los siguientes son los atributos de cada uno:

```
class User(models.Model):
    name = models.CharField(max_length=200)
    lastname = models.CharField(max_length=200)
    email = models.CharField(max_length=200)
    twitter_id = models.CharField(max_length=200)

class Contact(models.Model):
    name = models.CharField(max_length=200)
    lastname = models.CharField(max_length=200)
    age = models.IntegerField()
    company = models.CharField(max_length=200)
    phone = models.CharField(max_length=200)
    email = models.CharField(max_length=200)
    address = models.CharField(max_length=200)
    link = models.CharField(max_length=200, default="vacío")
    lat = models.FloatField()
    lon = models.FloatField()
    user = models.ForeignKey(User)
```

Librerías externas

Explicación del funcionamiento y de la conexión con las librerías externas.

Google Maps

La librería de Google Maps posee muchas opciones para “configurar” un mapa según la necesidad del usuario.

En este caso, al inicio se tuvo que obtener un API Access, para hacer uso de los mapas. Con esta se ingresa en una línea del script para dar autorización al usuario que lo esta usando.

API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key :

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. A single project may contain up to 20 client IDs.

[Learn more](#)



Create an OAuth 2.0 client ID...

Simple API Access

Use API keys to identify your project when you do not need to access user data. [Learn more](#)

Key for browser apps (with referers)

| | |
|---------------|--|
| API key: | AIzaSyAu6BlnUHebSvPr2rb-ya1b0yzXAYH5sIU |
| Referers: | Any referer allowed |
| Activated on: | Nov 13, 2012 2:22 AM |
| Activated by: | arivargas17@gmail.com – you |
| Obsolete key: | AIzaSyAoGSLvYBHcYXmT4D9hv0djv5NxrsUdQfw |
| Status: | Active until Nov 14, 2012 2:22 AM |

Create new Server key...

Create new Browser key...

Figura 3. Creación del API Access de Google Maps

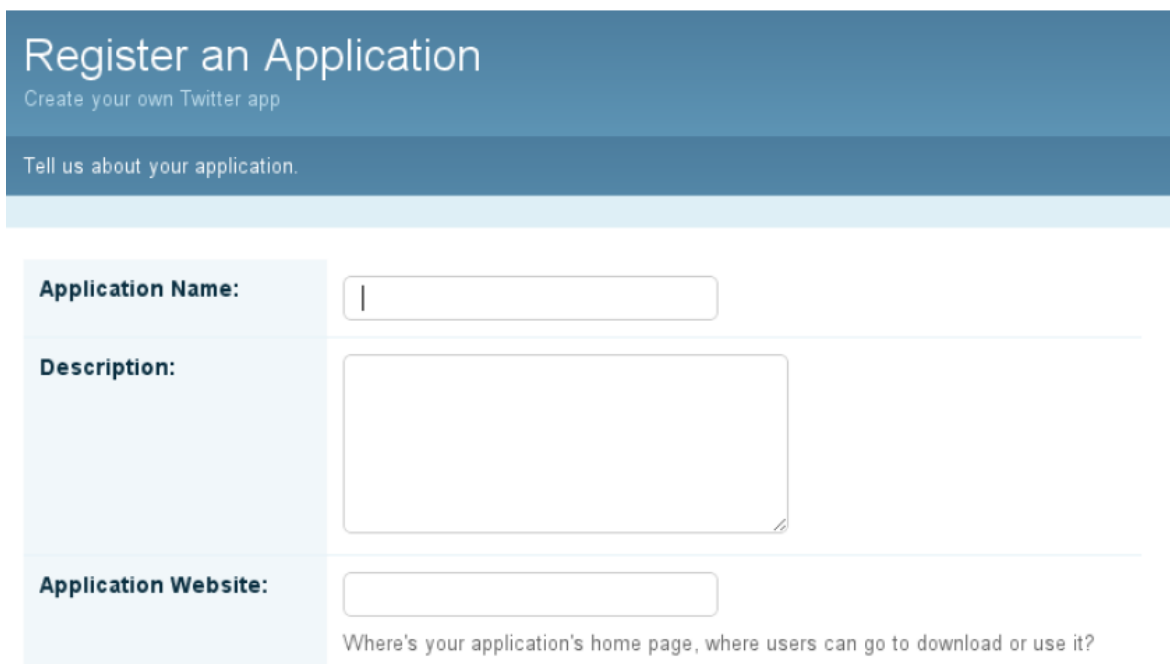
Posterior a esto, la documentación de Google Maps brinda Code Samples de donde se puede tomar ejemplos de cuales tipos de mapas se pueden realizar y así también los eventos, elementos, íconos.

La importancia de esta API para el proyecto es tomar los datos por latitud y longitud, así como por otros parámetros que son permitidos para brindar direcciones.

API de Twitter

La biblioteca con la cual se trabajó para la autenticación con Twitter fue oauth2 (Open Authentication), la cual permite mediante tokens que se emplean en la aplicación para permitir al usuario ingresar a la aplicación.

Para esta API se siguieron los pasos de la biblioteca, donde inicialmente para poder identificar la aplicación en Django se tuvo que registrar y solicitar permiso.



The image shows the 'Register an Application' page from Twitter. It has a blue header with the title 'Register an Application' and the subtitle 'Create your own Twitter app'. Below the header, there is a light blue box containing the text 'Tell us about your application.' and three input fields. The first field is labeled 'Application Name:' and is a single-line text input. The second field is labeled 'Description:' and is a multi-line text area. The third field is labeled 'Application Website:' and is a single-line text input. Below the 'Application Website:' field, there is a small text label that reads 'Where's your application's home page, where users can go to download or use it?'. The form is styled with a clean, modern design using shades of blue and white.

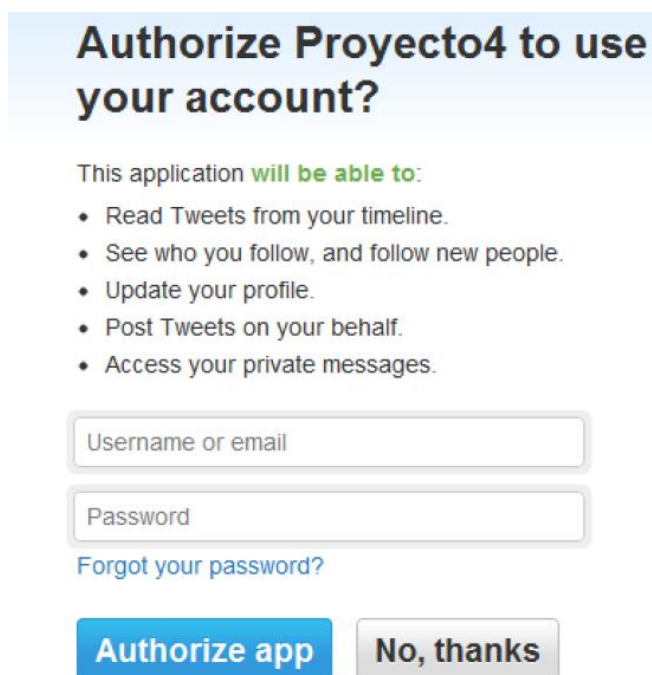
Figura 4. Registro de la aplicación en Twitter.

Al registrar esta aplicación para utilizar datos del usuario de Twitter y para poder identificar y autenticar al usuario permite un poco más de seguridad, al no compartir información privada con terceros. Una vez registrado el usuario se redirecciona a la aplicación TWIPYMAP.

Manual de usuario

Para poder hacer uso de la aplicación TWIPYMAP se deben seguir los siguientes pasos:

1. Levantar el servidor: `python manage.py runserver`
2. Ingresar a un browser y digitar la dirección: <http://127.0.0.1:8000/login>
3. Esto le direccionará al API de autenticación de Twitter, donde tendrá que ingresar su usuario y clave.



The screenshot shows a Twitter authorization interface. At the top, a blue header asks 'Authorize Proyecto4 to use your account?'. Below this, it states 'This application will be able to:' followed by a list of permissions: 'Read Tweets from your timeline.', 'See who you follow, and follow new people.', 'Update your profile.', 'Post Tweets on your behalf.', and 'Access your private messages.' There are two input fields: 'Username or email' and 'Password'. A link 'Forgot your password?' is located below the password field. At the bottom, there are two buttons: a blue 'Authorize app' button and a grey 'No, thanks' button.

4. Al ingresar los datos y estos ser validados debe esperar a que la misma aplicación lo redireccione a la página <http://127.0.0.1:8000/contact>, donde podrá crear, editar, eliminar y ver sus contactos.
5. Al ingresar a la aplicación se le desplegará una pantalla similar, sin los cuadrados de colores, estos se crearon únicamente para indicar qué acción se debe realizar.
 - a. Cuadrado anaranjado: indica el usuario que se encuentra logueado.
 - b. Cuadrado amarillo: es donde se ubica el menú de la aplicación.
 - c. Cuadrado azul: son los campos de textos que se deben llenar para ingresar el usuario.
 - d. Cuadrado verde: son las acciones sobre las cuales usted puede hacer clic para agregar, editar, consultar o borrar un contacto.

Creating Your New Contact

Please fill all the information bellow for creating your new contact.

ACTIONS

[List All Your Contacts](#)
[Create a New Contact](#)

Name:

Lastname:

Age:

Los botones de crear, modificar, eliminar se encuentran al final.

Create

- Así como se definió en el punto anterior, el html maneja la misma estructura para todas las demás acciones.
- Por último, para salir de la aplicación dar clic sobre la opción Log Out o <http://127.0.0.1:8000/logout>.

Análisis de resultados y Conclusión personal

Respecto al análisis de resultados, no se pudo realizar ni el 50% de la tarea programada y esto fue principalmente por las cajas negras que se crearon con la autogeneración del código.

Como es usual, al comenzar a entender un framework nuevo siempre existe una curva de aprendizaje, no obstante, la curva fue aún mayor (y en parte deteriorada) por asignar tiempo (más de lo estimado) a entender cómo se realizaba el código y el “alambrado” del mismo en la aplicación.

Lo único que se pudo realizar fue el mantenimiento de contactos, la autenticación de twitter y el mapa dentro de la aplicación.

Existieron problemas para posicionar el mapa y que este se viera, a lo cual se dedico mucho tiempo, y dejando de lado partes esenciales de la tarea programada.

Como conclusión personal queda de experiencia que no siempre la autogeneración de código es beneficiosa, menos cuando esta no es entendible para el programador. Así también debe investigarse primero ejemplos de cómo se implementa un API en el framework o lenguaje para no tener que realizar retrabajo de cómo se construye el código.