Informe de la práctica de Laboratorio Nº2: Una aplicación demo en Rails y Django.

Autor: Adán Rafael López Lecuona.

La principal diferencia de la creación de aplicaciones en Rails y en Django es que Rails es un marco de Ruby y Django un marco de Python.
Al principio de esta práctica tuve problemas con el repositorio de Github ya que había creado archivos de forma remota y no de forma local y me daba problemas al realizar push y commit.
Al tenerlo así, tuve que clonar el repositorio remoto a local para poder trabajar localmente y enviar los archivos trabajados localmente al repositorio remoto.

Aplicación demo en Rails

Ahora nos centramos en crear primero una aplicación demo en Rails siguiendo una serie de pasos.

```
$ cd ~/RAILS_proyectos
$ rails new prac1_app
$ cd prac1_app
```

Una vez generado una serie de archivos con rails new, editamos uno de ellos llamado Gemfile y rellenamos lo que no contenga el que viene por defecto con el siguiente completo ya.

```
source 'https://rubygems.org'
gem 'rails', '3.2.12'

group :development do
  gem 'sqlite3', '1.3.5'
end
# Gems used only for assets and not required
# in production environments by default.
group :assets do
  gem 'sass-rails',   '3.2.5'
  gem 'coffee-rails', '3.2.2'
  gem 'uglifier', '1.2.3'
end
gem 'jquery-rails', '2.0.2'
group :production do
  gem 'pg', '0.12.2'
end
```

Actualizamos el bundle y lo instalamos sin producción:

```
$ bundle update
$ bundle install --without production
```

Generamos lo siguiente:

```
$ rails generate scaffold User name:string email:string
```

Debemos migrar la base de datos usada:

```
$ bundle exec rake db:migrate
```

La aplicación la podemos ver en:
http://localhost:3000/.

Mediante las extensiones siguientes se nos verán en vistas diferentes:

URL	Action	Purpose
/users	index	page to list all users
/users/1	show	page to show user with id 1
/users/new	new	page to make a new user
/users/1/edit	edit	page to edit user with id 1

En el archivo routes.rb debemos poner:

```
DemoApp::Application.routes.draw do
 resources :users
 .
 .
 .
End
```

En el controlador concretamente en el archivo users_controller.rb ponemos lo siguiente:

```
class UsersController < ApplicationController
.
.
.

 def index
  @users = User.all

  .
  .
 end

 def show
  .
  .
  .
 end

 def new
  .
  .
```

```
  .
 end

 def create
  .
  .
  .
 end

 def edit
  .
  .
  .
 end

 def update
  .
  .
  .
 end

 def destroy
  .
  .
  .
 end
end
```

En el archivo user.rb dentro de modelos tenemos la clase User:

```
class User < ActiveRecord::Base
end
```

Para la vista de index debemos crear archivos que permitan visualizar nuestra app:

```
<h1>Listing users</h1>

<table>
 <tr>
  <th>Name</th>
  <th>Email</th>
  <th></th>
  <th></th>
  <th></th>
 </tr>

<% @users.each do |user| %>
 <tr>
  <td><%= user.name %></td>
  <td><%= user.email %></td>
```

```
    <td><%= link_to 'Show', user %></td>
    <td><%= link_to 'Edit', edit_user_path(user) %></td>
    <td><%= link_to 'Destroy', user, method: :delete,
                            data: { confirm: 'Are you sure?' } %></td>
  </tr>
<% end %>
</table>

<br />

<%= link_to 'New User', new_user_path %>
```

Realizamos lo mismos pasos para crear los microposts.

Ahora queremos inicializar nuestro repositorio para poder realizer nuestro primer commit.

$ git init
$ git add .
$ git commit -m "Initial commit"

Para cada práctica creamos un repositorio remoto diferente. En este caso creé el repositorio prac3.git.
Después de esto ya sabemos dónde tenemos que enviar los cambios que hagamos de manera local.
$ git remote add origin git@github.com:<alu3954>/prac3.git
$ git push -u origin master

Para generar nuestra primera aplicación automáticamente realizamos este comando que nos crea una serie de archivos automáticamente mediante el comando:

$ rails generate scaffold User name:string email:string

Para actualizar la base de datos introducimos el comando:

$ bundle exec rake db:migrate

Ahora podemos crear muchos usuarios y micropost , editarlos, elegirlos, monstrarlos..

Una vez realizado tus cambios oportunos lo podemos enviar a Github mediante los siguientes comandos:

$ git add .
$ git commit -m "Finish demo app"
$ git push

Aplicación Demo en Django

Para crear una aplicación en Django tenemos que crear un directorio para guardar el código generado por nuestro framework Django, generamos nuestra estructura de app con el comando para crear la aplicación:

Django-admin.py startproject mysite

El startproject creado tiene la siguiente estructura:

Mysite/

__init__.py

Manage.py

Settings.oy

Urls.py

Para ejecutar nuestra debemos estar donde se encuentre el archivo manage.py, utilizando el servidor por defecto de Django se introduce:

$python manage.py runserver 8080.

Para sincronizar la base de datos de la aplicación se introduce:

$python manage.py syncdb

Para la creación de modelos de nuestra app tenemos los archivos models.py, dentro de éste creamos nuestras clases.

```
class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')


class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
```

Para activar los modelos que implantemos debemos modificar en el archivo settings.py e introducir nuestro nombre(en este caso es polls).

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'polls'
)
```

Una vez incluido, podemos sincronizar nuestra base de datos para pueda crear las tablas:

```
python manage.py sql polls
```

En sql:

```
BEGIN;
CREATE TABLE "polls_poll" (
    "id" serial NOT NULL PRIMARY KEY,
    "question" varchar(200) NOT NULL,
    "pub_date" timestamp with time zone NOT NULL
);
CREATE TABLE "polls_choice" (
    "id" serial NOT NULL PRIMARY KEY,
    "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),
    "choice" varchar(200) NOT NULL,
    "votes" integer NOT NULL
);
COMMIT;
```

Incluimos en nuestras clases lo siguiente:

```
class Poll(models.Model):
    # ...
```

```
    def __unicode__(self):
        return self.question

class Choice(models.Model):
    # ...
    def __unicode__(self):
        return self.choice
```

Rails y Django son unos buenos framework y te facilitan mucho trabajo a la hora de crear aplicaciones, creando por defecto una estructura para la app.