Contenido

Método GET: esource_path+query del url

Método POST: las variables del formulario van como contenido en el requerimiento después de todas las cabeceras.

POST /recepcion_formulario HTTP/1.1

Host: ...

resto cabeceras http

firstname = valor_variable_1

lastname = valor_variable_2

Cookies

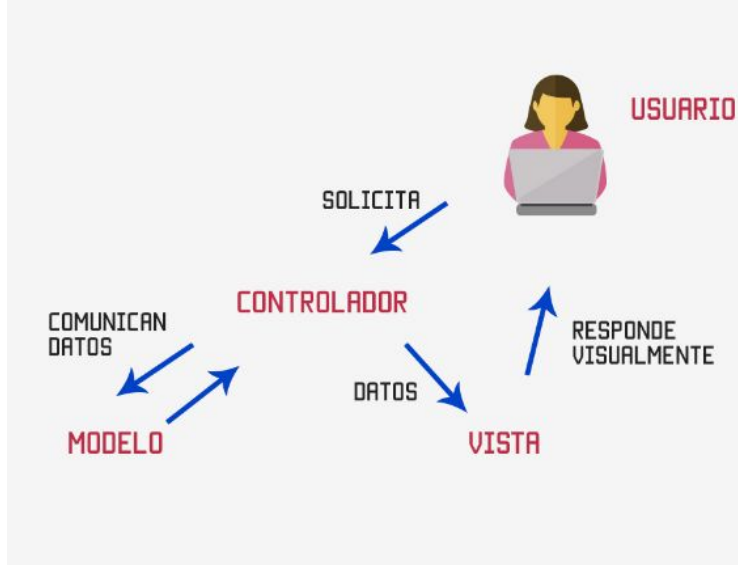
Las cookies se **almacenan en el navegador como un formato de archivo de texto**. Se almacena una cantidad límite de datos. Solo se permite **4kb [4096bytes]** . La variable $ _ COOKIE no contendrá varias cookies con el mismo nombre.

podemos acceder fácilmente a los valores de las cookies. Por lo tanto, **es menos seguro**. La función **setcookie()** debe aparecer **ANTES** de la etiqueta **<html>**

Sesiones

Las sesiones se almacenan en el lado del servidor. Se almacena una cantidad ilimitada de datos. Mantiene la variable múltiple en sesiones. no podemos acceder fácilmente a los valores de las cookies. Por lo tanto, es más seguro.

TEMA 2: MVC



- **Modelo**: se encarga de los datos, generalmente consultando la base de datos. Actualizaciones, consultas, búsquedas, etc.
- **Controlador**: recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.
- **Vistas**: representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí.

```python
from sqlalchemy import Column, Integer, String
from yourapplication.database import Base

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String(50), unique=True)
    email = Column(String(120), unique=True)

    def __init__(self, name=None, email=None):
        self.name = name
        self.email = email

    def __repr__(self):
        return '' % (self.name)
```

```
<p> My string: {{my_string}} </p>
<p> Value from the list: {{Users[2]}} </p>
<p> Loop through the list: </p>
<ul>
  {% for user in Users %}
      <li> {{user.name}} </li>
  {% endfor %}
</ul>
```

controller.py

```python
@app.route("/users")           # decorador para url (router)
def users():
        lista=[]
        # Acceso a la BD
        for user in Users.objects().order_by('-name').limit(4):
                lista.append(user)

        # rellena y envia la plantilla
        return render_template('users.html', Users=lista)
```

TEMA 3: FLASK

Es software para apoyar el desarrollo de aplicaciones web. Flask (un micro framework)

Características:

- servidor de desarrollo y depurador incorporado
- soporte integrado para pruebas unitarias
- plantillas jinja2
- soporte para cookies seguras(sesiones en el lado del cliente)
- compatible con unicode
- profusamente documentado

Manejo de URLs con Flask

```
@app.route('/')
def index():
    return 'Página principal'

@app.route('/hola')
def hola():
    return 'Hola'
```

Entrada de variables GET

/prog?par1=hola&par2=pepe

```
from flask import request

...

@app.route('/prog')
def prog():   # los parámetros están en el request
    parametro1 = request.args.get('par1')   # hola
    parametro2 = request.args.get('par2')   # pepe
    return parametro1 + ' ' + parametro2
```

Métodos HTTP (Get, post, put, ...)

```
@app.route('/login', methods=['GET', 'POST'])

def login():
    if request.method == 'POST':
        hacerLogin()
    else:
        mostrarElFormularioDeLogin()
```

Redirecciones y errores

```
from flask import abort, redirect, url_for

@app.route('/')
def index():
    return redirect(url_for('login'))

@app.route('/login')
def login():
    abort(401)
    estoNoSeEjecuta()

@app.errorhandler(404)
def page_not_found(error):
    return "Página no encontrada", 404
```

Cookies en Flask

Leer cookie

```
from flask import request

@app.route('/')
def index():            # las cookies están en el request
    username = request.cookies.get('username')
```

Escribir cookie

```
from flask import make_response

@app.route('/')
def index():
    response = make_response("logeado como the_username")
    response.set_cookie('username', 'the_username')
    return response
```

Sesiones en Flask

Las sesiones mandan un cookie de sesión con un token "aleatorio", lo asocian a un registro del servidor para guardar datos de la sesión.

```
fromfrom flask import Flask, session, redirect, url_for, escape, request

app = Flask(__name__)

# Set the secret key to some random bytes. Keep this really secret!
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'
```

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return '''
        <form method="post">
            <p><input type=text name=username>
           <p><input type=submit value=Login>
        </form>
    '''
```

```
@app.route('/')
def index():
if 'username' in session:
  return 'Logged in as %s' % escape(session['username'])
return 'You are not logged in'

...

@app.route('/logout')
def logout():
    # remove the username from the session if it's there
    session.pop('username', None)
    return redirect(url_for('index'))
```

Logs en flask, usando el módulo de logging de python:

```
@app.route('/')
def principal():
  app.logger.debug('Arranque de la aplicacion')
  return 'Ejemplo para logs'
```

Sintaxis moustache

{% ... %} Sentencias.

{{ ... }} Expresiones que serán impresas en la salida de la plantilla.

{# ... #} Comentarios (no aparecerán en la salida).

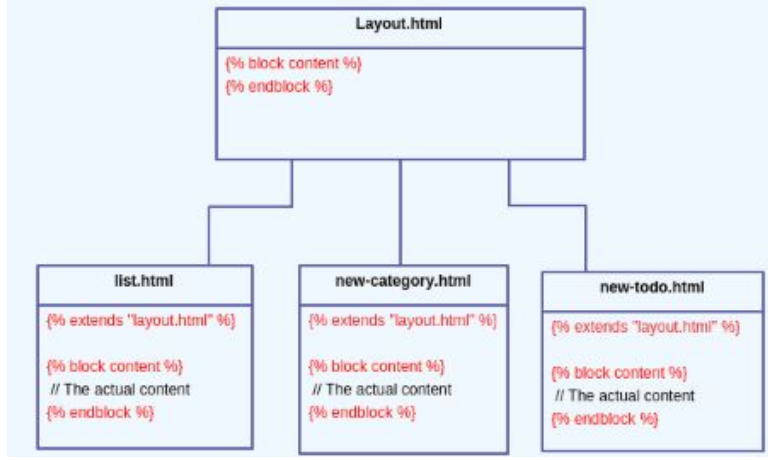
 # ... Sentencias de una línea:

HTML SEGURO

Por defecto Jinja2 “escapea” el html que le enviamos <, >, ‘. Se puede desactivar:

- usando la clase Markup en el código de Python (recomendado)
- usar el filtro “|safe” dentro de la plantilla: {{ variable|safe }}
- desactivar el sistema de “escaping” temporalmente: {% autoescape false %}

Herencia de plantillas



TEMA 5: HTML5, css frameworks

Bootstrap - ¿Qué es y Cómo funciona?

Es un framework CSS y Javascript diseñado para la creación de interfaces limpias con un diseño responsive.

Las funciones más usadas son:
- **Responsive design** (adaptación automática a móviles y tablets)
- **Grid System** (posicionar los elementos en la página)
- **Interface UI** (forms, botones, menús, etc)

- Bootstrap en líneas generales, se basa en una estructura dividida de 12 columnas que los desarrolladores pueden gestionar en función de sus necesidades y preferencias, en función de cuatro tamaños de dispositivo.

Alternativa: MaterializeCSS

Framework CSS que permite crear sitios y apps web con los principios de Material Design (Creado por Google). Puede ser usado en dos formas, Materialize y Sass, dependiendo de las preferencias y la experiencia se puede seleccionar cualquiera de las dos versiones.

Incluye:

- **Grids**: 3 tamaños de plantilla.
- **Blockquotes**: usado para dar énfasis a una frase o cita.
- flow text (responsive text), botones, check boxes, barras de navegación, estilos para tablas y cards.

TEMA 7: Persistencia: Bases de Datos

Serialización de datos

- Con el módulo pickle, guardamos un objeto en el disco
- Archivos indexados de tipo dbm
- Con dbm se accede bases de datos tipo Berkely DB como si estuvieran en memoria

Persistencia de objetos

shelve añade una capa de software a dbm para serializar los objetos

Librerías Object Relational Mapping

Una capa de software para aislar la BD SQL del resto del código

Bases de datos NO-SQL

Son base de datos para permitir 'BIG DATA', tipos de datos complejos, más índices, distintas maneras de consulta, etc

MONGODB

Es una base de datos orientada a documentos estilo JSON que incluye información del tipo de dato. Esta base de datos está formada por colecciones (equivalentes a tablas en SQL) que son un conjunto de documentos (equivalentes a filas).

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')


# base de datos
db = client['test-database']
```

MONGOENGINE

Es un ODM que nos permite mapear la estructura de nuestra base de datos a objetos en nuestras aplicaciones python que se conectan a instancias como MONGODB, esto con el fin de poder usar información de nuestra base de datos como si se tratasen de objetos. Esto ofrece ventajas, la manera CRUD de escribir nuestra aplicación y mejorar nuestro código.

TEMA 8: Servicios Web: REST

Son un conjunto de protocolos y estándares para comunicar aplicaciones y intercambiar datos en internet. Se usa http y el puerto 80 para evitar problemas con los firewall.

Este término se usa ahora para referirse a interfaces simples, que solo usan HTTP y JSON o XML en la respuesta.

Es un protocolo **sin estado**
REST-FULL: En su versión más extendida, se usan los verbos de HTTP para las llamadas del cliente al servidor, asociados a operaciones CRUD.
El servicio REST-FULL debe seguir 4 principios:

- Usar los verbos de HTTP explícitamente
- Ser completamente sin estado
- Usar URIs estilo path
- Devolver XML o JSON

Usar los verbos de HTTP explícitamente con llamadas AJAX:

- GET: para requerir un recurso (Read, SELECT)
- POST: para crear un recurso (Write, INSERT)
- PUT: para cambiar el estado de un recurso (Write, UPDATE)
- DELETE: para borrar un recurso (DELETE)

Usar URIs estilo path, Que se puedan leer por humanos (SEO) y que muestren una jerarquía, sustituyendo espacios en blanco por guiones o subrayados.

TEMA 9: XML a fondo

eXtensible Markup Language

Es un metalenguaje de propósito general para intercambio y almacenamiento de información.

¿Para que sirve XML?

Representar información estructurada en la web (todos documentos), de modo que esta información pueda ser almacenada, transmitida, procesada, visualizada e impresa, por muy diversos tipos de aplicaciones y dispositivos.

6 - Añadir los módulos y middleware en settings.py
7 - Crear los mappings para los urls en urls.py
8 - Definir las vistas en app/views.py y los templates
9 - Aplicar tests
10 - Desplegar la aplicación en el servidor web de producción

TEMA 11: Formularios

Clase Forms

Django incluye la clase forms para facilitar la entrada de datos desde formularios html. La clase incluye:

- Generación del html en las plantillas
- Validación en el servidor
- Tratamiento de los errores
- Seguridad CSRF

TEMA 12: Autentificación y Autorización en Django

Django tiene un sistema para la autentificación y la autorización de usuarios basados en un modelo para usuarios que incluye:

- Autenticación: gestión de Contraseñas
- Roles: grupos de usuarios
- Autorización: permisos sobre las vistas y el modelo

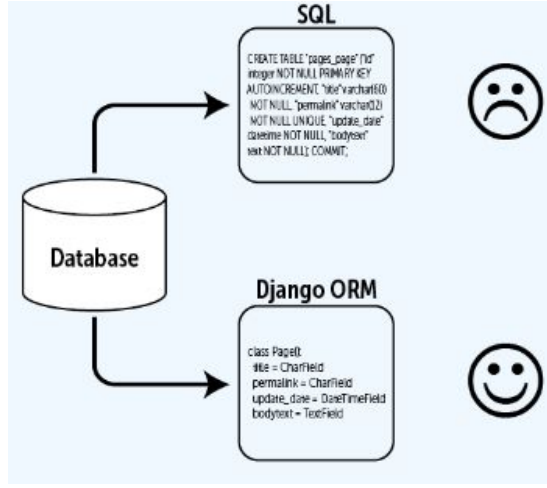
Django utiliza sesiones y midleware para gestionar automáticamente la autenticación de usuarios, con funciones para comprobar el usuario y la contraseña en la base de datos de usuarios
E incluso tiene un decorador para ponerlo en las vistas que requieran usuarios autenticados

Autenticación y registro

Django ya proporciona plantillas, formularios y vistas para la autentificación y creación de usuarios: **Authentication views**, incluyendo el cambio de contraseña, la confirmación, etc
Pero es más cómodo utilizar el plugin **django-allauth** que además tienen prevista la activación de cuentas por email, o la autenticación delegada en redes sociales.

TEMA 13: Modelos en Django

Django utiliza un ORM para Bases de Datos SQL



Tablas como Clases

```python
from django.db import models

class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

Por defecto django asigna una llave primaria automática.

id = models.AutoField(primary_key=True)

Relaciones.

Uno a muchos

- se puede cambiar de idioma en tiempo de ejecución, dependiendo de la instalación del SO (escritorio) o del navegador (web)

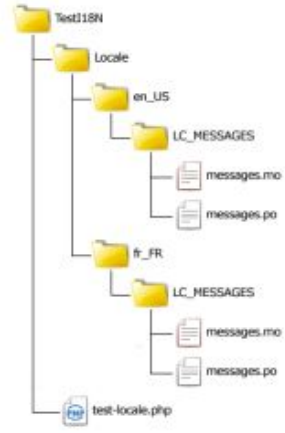
GETTEXT

Es la libreria tradicional de UNIX para I18N la que se usa en linux para los programas de escritorio, php, python, etc..

Unix: gettext, archivos .mo

los textos para cada lengua, van en archivos .mo (machine object), compilados a partir de los textos en archivos .po (portable object).

Están en un directorio a parte para cada locale (LENGUA) por defecto en /usr/share/locale.



Para cada entorno se utiliza un “locale” distinto, cada uno con su nombre:

lengua[_pais][.codificación][@modificación]

es	Español
es_AR	Español para Argentina
es_AR.UTF-8	en codificación utf-8
es_ES@euro	español para España modificación euro

LOCALE: Cambio del comportamiento de alguna de las funciones de la librería estándar del C, según el valor de las variables de entorno asignadas a cada categoría.

Para cambiar los mensajes se pone:

En lugar de: printf "Hola mundo\n";
Se pone: printf **gettext**("Hola mundo\n");
o redefiniendo la función gettext: printf _("Hola mundo\n");

gettext busca el texto correspondiente al locale actual en tiempo de ejecución, en un archivo distinto.

Archivos .po (portable object)

los archivos .mo, para cada lengua, se generan a partir de archivos de texto ".po". Por ejemplo para mi traducción al inglés, haría un archivo "mi_traduccion_EN.po" al directorio en_US que contendría:

Traducciones al inglés
msgid "Hola mundo\n"
msgstr "Hello world\n"
...

Se puede usar el programa xgetext para sacar una automáticamente una plantilla con todos los mensajes bajo gettext.

Una vez traducidos los archivos .po se compilan con msgfmt a archivos .mo y se ponen en los directorios correspondientes a los locales.

Proceso de traducción

1. Poner gettext() donde vaya a traducir en el código fuente
2. Crear la plantilla .pot pasando la utilidad xgettext al código fuente.
3. Traducir rellenando una plantilla para cada lengua, poniéndolas en un archivo .po
4. Compilar las traducciones con la utilidad msgfmt a un archivo .mo
5. Poner las traducciones en sus locales correspondientes al instalar el programa.

Formas del plural

Para estos casos se utiliza la función ngettext

printf (ngettext ("Un archivo", "%d archivos", n), n);

El primer argumento es el singular, y se usa como índice en la
búsqueda, y el segundo es el plural, y el tercero es el número

AJAX, acrónimo de Asynchronous JavaScript And XML.

AJAX por definición se usa en aplicaciones que siendo ejecutadas en cliente (en el navegador) mientras comunican con el servidor de manera asíncrona para obtener datos y solo recargar la parte de la página necesaria.

TEMA 16: Servicios Internet

- **FTP:** es un protocolo para transferencia de archivos entre ordenadores, usando servidores y clientes de FTP. Tiene la ventaja de poder transferir archivos muy grandes, con recuperación de errores si se corta la conexión. Con el módulo ftplib podemos transferir archivos usando FTP y python.
- **E-MAIL:** utiliza dos o tres protocolos para transferir mensajes entre clientes servidores de correo.
    - SMTP: para enviar correo.
    - MIME: cuando no se usa un texto simple (adjuntos, html, utf8, etc)
    - IMAP: para comunicar un cliente con el servidor de correo para recibir mensajes.
    - POP3: para comunicar un cliente con el servidor de correo para recibir mensajes (más primitivo)

Diferencias entre POP e IMAP

1.-IMAP

Este protocolo trabaja directamente sobre el servidor de correo. Para consultar el correo conecta con el servidor y muestra el contenido que hay. Cuando envían un correo es almacenado en el servidor y es accesible desde cualquier otro dispositivo que consulte el correo.

Los gestores actuales tienen opciones para que los correos también sean guardados de forma local usando este protocolo.

2.-POP3

Un cliente de correo configurado como POP3 conecta al servidor y descargar los correos al dispositivo local. Una vez hecho, elimina los mensajes descargados del servidor, estando únicamente disponibles de forma local. Con este protocolo los mensajes enviados solo están disponibles de forma local.

Los gestores actuales permiten la opción de guardar copia de los mensajes descargados en el servidor, o marcar que solo se eliminen los de una antigüedad determinada.

TEMA 17: Despliegue

Para poner en producción una aplicación web, necesitaremos hacer algún cambio:

- Deshabilitar el ambiente de depuración.
- Cambiar el servidor web, por el definitivo de producción.

En django para deshabilitar el ambiente de desarrollo nos vamos al archivo settings y se cambia:

DEBUG = True, pasa a:

DEBUG = False # y

ALLOWED_HOSTS = [] pasa a:

ALLOWED_HOSTS = ['*'] # o el que sea en producción

gunicorn

gunicorn, es un servidor de web wsgi muy sencillo de instalar, que substituye al runserver de desarrollo

# para instalar
sudo pip3 install gunicorn

para ejecutar el puerto 8000
/usr/local/bin/gunicorn mi_app.wsgi --bind: 0.0.0.0:8000

Django ya tiene el archivo .wsgi en el mismo directorio de setttings.py

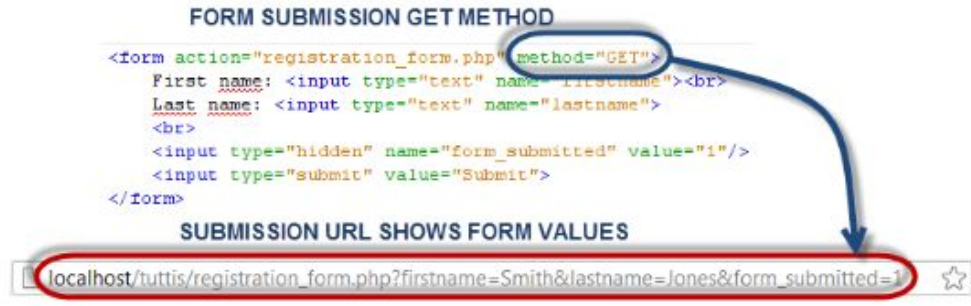
docker

De esta manera se independiza la instalación del servidor de producción, el ejecutable va ser siempre el mismo

Además la instalación va a simplificarse al utilizarse scripts de aprovisionamiento

2. ¿Cuál es la diferencia en el envío de parámetros entre una llamada GET y otra POST de HTTP?

- El concepto **GET** es obtener información del servidor. Traer datos que están almacenados en el servidor, ya sea una base de datos o archivo al cliente. Ejemplo: recibir un ID para obtener un artículo de la base de datos.



- El concepto **POST** es enviar información desde el cliente para que sea procesada y actualice o agregue información en el servidor, como sería la carga o actualización en si de un artículo.

3. ¿Como se llama en Django el archivo donde están este tipo de instrucciones? ¿Para qué sirven? Explicar con detalle qué significa cada parte de cada argumento de la llamada.

url(r'^bar/(?P<nombre_slug>[\w\-]+)/$', views.bar, name='bar')

urls.py. Sirve para poner todos los paths agrupados. Al igual que en Flask se hace con @app.route("ruta") y debajo la función, en Django se agrupan todos bajo ese fichero para tenerlos más organizados.

El primer parámetro es la ruta de la aplicación web. En este caso, es una expresión regular (lo indica que empieza con el carácter r) que empieza (^) por bar/ y obtiene palabras separadas por guiones (una o más palabras). Por ejemplo "bar/hola-adios". ?P<nombre_slug> es un grupo de captura al que se le va a llamar nombre_slug, imagino que para luego poder tratarlo en el código. El símbolo $ indica fin de la expresión regular. views.bar es la función que se va a ejecutar cuando se acceda a esa URL, y name='bar' sirve para llamar a la plantilla dentro de otra plantilla; por ejemplo, poniendo { % url 'bar' % }.

4. ¿Cuales son las funciones de una librería como Bootstrap?

Es un framework CSS y Javascript diseñado para la creación de interfaces limpias con un diseño responsive.
Las funciones más usadas son:
- Responsive design (adaptación automática a móviles y tablets)
- Grid System (posicionar los elementos en la página)
- Interface UI (forms, botones, menús, etc)

Bootstrap en líneas generales, se basa en una estructura dividida de 12 columnas que los desarrolladores pueden gestionar en función de sus necesidades y preferencias, en función de cuatro tamaños de dispositivo.

5. ¿Qué es un ORM?. En Django ¿como se llama el archivo relacionado con el ORM?

Mapeo de objeto-relacional, es un modelo programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador.

Django ORM: Muy completo y permite evitar tener que escribir SQL a mano, añadiendo la posibilidad de cambiar de motor de base de datos sin que sea traumático.

El archivo relacionado con el ORM es model.py, por ejemplo:

8. Comenta los cambios que has hecho para el despliegue de la aplicación de Django

He deshabilitado el ambiente de depuración, nos vamos al archivo settings y se cambia:

#DEBUG = True , se pasa a False

#ALLOWED_HOSTS = [], se pasa ['*'] o el que sea de producción.

Instalamos Gunicorn, que es un servidor web wsgi muy sencillo y probamos nuestra aplicación o bien la construimos en un contenedor docker y obtenemos un ejecutable para subirlo a otro servidor.

9. ¿que diferencias has encontrado entre los frameworks "Flask" y "Django"?

FLASK	DJANGO
- Microframework(Librería), 2MB - Se hace menos código para empezar, respuesta más rápida. - Prototipos, sitios sin usuarios, optimización del rendimiento.	- Framework (Librería+Scripts), 7MB - Más código hecho (usuarios, interface administración, ORM, etc) - Modelo de usuarios, autenticación, roles, moddleware, etc.

10. Para Nota: A grandes rasgos: ¿cómo crees que funciona internamente la autentificación en Django?

Sabiendo que Django utiliza una serie de funciones, plantillas, modelos y middleware, lo que hace es trabajar con la base de datos. Combinará las rutas con una serie de flags en la BD para asegurarse de qué usuario o grupos pueden o no pueden acceder a determinadas rutas.

Posibles preguntas:

1. **Cómo sería la url de una petición GET, al servidor “pepito.com” por el puerto 80, con la ruta “restaurantes/lista” y con los parámetros orden=descendente y todos=si.**

   http://pepito.com/restaurantes/lista?orden=descendente&todos=si

2. **¿Y una llamada a POST con los mismos parámetros?**

   http://pepito.com/restaurantes/lista

   Cabecera post:

   POST /restaurantes/lista HTTP/1.1
   Host: pepito.com
   resto cabeceras http
   orden = descendente
   todos = si

3. **¿Cuál es la utilidad habitual de los verbos HTTP en una app RESTFULL?**
   - **GET:** obtener información. Por ejemplo, una página o una lista de datos.
   - **PUT:** almacenar una entidad en una URL. Con PUT puedes crear una nueva o actualizar una existente. PUT es idempotente (propiedad matemática) que quiere decir que tiene la capacidad de realizar una acción varias veces y conseguir el mismo resultado que si se hiciese sólo una.
   - **POST:** pedir que el recurso en la URL haga algo determinado.
   - **DELETE:** pedir al recurso en la URL que se borre.

FLASK (MVC)

- **M**odelo: se encarga de los datos, consultando la base de datos. Altas, bajas, modificaciones y borrado. Lo podemos encontrar en el fichero principal de la aplicación (**por ejemplo: app.py**)
- **V**ista: representación visual de los datos, lo encontramos en el directorio templates (**/templates**)
- **C**ontrolador: encargado de controlar, recibir las órdenes del usuario y se encarga de solicitar los datos al modelo y comunicarles a la vista . (**por ejemplo: app.py**)

7. **Comente de manera breve algunas alternativas para guardar información en el CLIENTE de una aplicación.**
    - La manera habitual es usar cookies, que son pequeños ficheros con una fecha de caducidad asociada y que almacena el navegador.
    - Otra posibilidad es utilizar el mecanismo de persistencia local (Web Storage) disponible a partir de HTML5
8. **Comente de manera breve cuales son las diferencias de tratamiento de cadenas Unicode entre Python 3.1 y Python 2.7.**

    La diferencia fundamental es que por defecto Python 2.7 no utiliza cadenas Unicode (si quieres utilizarlas tienes que especificarlo mediante el operador u ante la cadena en cuestión) mientras que en la versión de Python 3.1 por defecto todas las cadenas son Unicode.

9. **En las prácticas de la asignatura, se ha desarrollado un sitio web usando plantillas. Describa brevemente la escritura de directorios y ficheros que permiten que dicho sitio web funcione.**