Trabajo práctico N.º 3

Laboratorio de Algoritmos y Estructuras de Datos

Preentrega Django

Lucila Fernández Achille

4to AO

Instituto Industrial Luis A. Huergo

Laboratorio de Algoritmos

Prof. Ignacio García

1. **¿Qué es Django y por qué lo usaríamos?**

Django es un *framework*, es decir, un grupo de herramientas ya estandarizadas que permite resolver problemas comunes de forma más fácil. En este caso, se especializa en el desarrollo de páginas web. Algunos sitios muy conocidos utilizan Django, como Instagram o Pinterest. Usar django tiene muchas ventajas:

- Reduce notablemente los tiempos de trabajo al ya tener hechos bloques de código con funciones que se suelen necesitar para una página web.
- Incluye numerosas funciones que se pueden llegar a necesitar.
- Es muy seguro.
- Sirve tanto para sitios web pequeños como sitios de alcance a nivel mundial.
- Es muy versátil, permite mucha libertad a la hora de elegir qué hacer, las herramientas no te limitan.
- Es gratuito y de código abierto, haciéndolo rentable.

2. **¿Qué es el patrón MTV (*Model-Template-View*) en Django? (simplificado de MVC). Compará MTV con MVC.**

El patrón MTV consiste en la división de una aplicación web en tres partes: modelo, plantilla y vista. Esto permite que sea fácil de mantener, escalar y que sea flexible. El modelo se centra en los datos, su estructura y operaciones. La plantilla consiste en cómo se muestran los datos al usuario, sea, la presentación. Por último, la vista maneja las solicitudes y respuestas de los usuarios. El MTV es una forma simplificada de MVC. En este último,se divide en modelo, vista y controlador. El modelo se encarga de las consultas o actualizaciones de los datos. La vista trata más de qué datos se muestran, no tanto cómo se muestran. El controlador funciona como intermediario entre el modelo y la vista, le comunica lo que el usuario necesita, pide los

datos al modelo y luego se los comunica a la vista. Es una arquitectura similar pero más compleja que la MTV.

3. ¿Qué entendemos por *app* en Django?

Una app en django es una aplicación web, o sea, un *software* que se ejecuta en el navegador web, que está diseñado para una tarea específica (generalmente con menor complejidad) y que luego se emplea en otros proyectos. Las apps son *reutilizables*, permitiendo que el desarrollo de otras aplicaciones más complejas cuente con un código modular y mantenible. Cada proyecto puede estar formado por varias apps con las tareas más específicas, como la autenticación de un usuario o comentarios, entre otras funciones particulares. La *modularidad* de estas aplicaciones es un factor importante para su reutilización en distintos proyectos,esto implica que su desarrollo es independiente y es fácil de implementar en distintas aplicaciones más complejas. La implementación de las apps de django impacta en el código, facilitando su *organización* y *mantenimiento*. En django existen apps pre-instaladas las cuales podemos utilizar en nuestros proyectos, como por ejemplo 'django.contrib.admin' o 'django.contrib.auth', entre otras. Además de estas, también es posible crear nuevas aplicaciones, para esto, hay que seguir una serie de pasos que están especificados en la documentación oficial.

4. ¿Qué es el flujo *request-response* en Django?

El flujo request-response en django se usa para manejar las interacciones web, haciendo uso de los *request-response objects*. Las solicitudes y respuestas HTTP (Protocolo de Transferencia de Hipertexto) permiten el paso de datos a través del sistema además de dejar herramientas a disposición para lograr una mayor eficiencia del flujo. Cuando el usuario hace una *request*, el *framework* crea automáticamente una *HttpRequest* que contiene los datos que

sirven para identificar a otro dato, o sea metadatos. Una vez creada, Django selecciona la vista más óptima para procesar los datos y devolver una *HttpResponse*. Cada objeto tiene ciertos atributos:

- ***Request*:**

  – *.scheme*: puede ser https (Protocolo de Transferencia de Hipertexto Seguro) o http, indica si la conexión es segura o no.

  – *.body*: si se usan APIs *(Application Programming Interface)* o maneja imágenes, permite acceso al código crudo de la *request.*

  – *.path*: contiene una parte de la *URL (Uniform Resource Locator)*, se puede usar para buscar o filtrar contenido.

  – *.path_info*: contiene la otra parte de la *URL* después del dominio, es útil para trabajar con las configuraciones de un servidor web.

  – *.method*: es un *string* que indica el método.

  – *.encoding*: define la forma de codificación de caracteres.

  – *.content_type*: define el MIME *(Multipurpose Internet Mail Extensions)*, o sea, la extensión que indica qué tipo de datos contiene un archivo.

  – *.content_params*: es un diccionario que guarda los parámetros del contenido.

- ***Response*:**

  – *.content*: es el contenido de la respuesta.

  – *.status_code*: el predeterminado es 200 (significa que está bien), pero hay otros como el 404 (significa *page not found*).

  – *.headers*: permite agregar o modificar *headers* si es necesario.

Cuando hablamos de WSGI *(Web Server Gateway Interface)*, nos referimos a una especificación que cumple la función de describir la forma en la que Django interactúa con el servidor web. Se trata de una interfaz estandarizada, lo cual permite asegurar la compatibilidad entre los servidores y las aplicaciones de Python. Se necesita de WSGI como "intermediario" ya que los servidores web no pueden leer directamente las aplicaciones de Django, que son código de Python. Cuando el servidor web recibe una *httpRequest*, la reenvía al servidor WSGI para que cree una función en un archivo generalmente llamado *wsgi.py* dentro de la carpeta del proyecto de Django. A partir de ahí, la aplicación de django es la que procesa y formula una respuesta, y el servidor WSGI se encarga de enviarla al servidor web que se la hace llegar al cliente.

5. **¿Qué es el concepto de ORM (*Object-Relational Mapping*)?**

El ORM es una herramienta que sirve para permitir la comunicación con la base de datos sin tener que utilizar SQL, en cambio usando código de Python. Su implementación logra una mayor productividad y funcionamiento de las aplicaciones web. Hay varios conceptos que permiten el funcionamiento del ORM. Para empezar, ORM usa *modelos*, son clases para representar las tablas de la base de datos, mientras que cada atributo de la clase equivale a las columnas de la tabla, a lo que se le llama *campo*. Un *QuerySet* es un conjunto de consultas de una base de datos que recuperan, modifican o filtran datos. Estos *QuerySets* no se ejecutan hasta que se necesiten los datos. La interfaz a través de la cual las consultas se pasan a los modelos de Django se denomina *managers*. Las operaciones CRUD (create, read, update, delete) en Django se manejan de forma directa.

6. **¿Qué son los *templates* en Django?**

Los templates en Django representan una forma conveniente de generar código HTML de forma dinámica. Es un archivo HTML que contiene las partes estáticas de la página que se quiere lograr y además cuenta con descripciones de cómo se van a implementar las partes dinámicas. No es necesario utilizar un template a la hora de hacer un proyecto, pero es flexible y también se permite el uso de más de uno. Existe lo que se llama DTL (*Django template language*), el cual ya tiene un backend incorporado para su propio template.

7. **Leé con detenimiento este *django at a glance*:**

   https://docs.djangoproject.com/en/5.2/intro/overview/

8. **¿Cómo se lo instala?** https://docs.djangoproject.com/en/5.2/intro/install/

Para instalar Django primero debemos asegurarnos de tener Python instalado, lo cual se puede chequear fácilmente desde la terminal, ejecutando *python*. Si tu proyecto va a requerir de una base de datos considerable, entonces se debe instalar una, pero de caso contrario, no es necesario. La versión recomendada para instalar es un lanzamiento oficial, aunque no es la única opción. El último paso es la verificación para asegurarse de que Django sea visible para Python, para esto hay que ejecutar los siguientes comandos en la terminal:

```
>>> import django
>>> print(django.get_version())
```

Ahora que saben un poco más sobre django, los invito a hacer el siguiente tutorial, hasta la parte 8 inclusive:

- https://docs.djangoproject.com/en/5.2/intro/tutorial01/

Referencias

Chirilov, A. (2023). *Apps in Django – Concept & free samples*. Codementor. https://www.codementor.io/@chirilovadrian360/apps-in-django-concept-free-samples-294vudyim5

Django Software Foundation. (2024a). *Django at a glance*. https://docs.djangoproject.com/en/5.2/intro/overview/

Django Software Foundation. (2024b). *How to install Django*. https://docs.djangoproject.com/en/5.2/intro/install/

Django Software Foundation. (2024c). *Templates*. https://docs.djangoproject.com/en/5.2/topics/templates/

Django Software Foundation. (2024d). *Django*. https://www.djangoproject.com/

EspiFreelancer. (s.f.). *MTV en Django*. https://espifreelancer.com/mtv-django.html

Geeks for Geeks. (s.f.). *How to create an app in Django*. https://www.geeksforgeeks.org/python/how-to-create-an-app-in-django/

Pandey, A. (2023). *WSGI in Django*. Medium. https://medium.com/@ashishpandey2062/wsgi-in-django-252d48a36a8c

Roomee. (2023). *Mastering Django ORM: Core concepts and advanced techniques*. Medium. https://medium.com/@roomee/mastering-django-orm-core-concepts-and-advanced-techniques-068967c73da1

Towards Dev. (2023). *Understanding request and response objects in Django*. https://towardsdev.com/understanding-request-and-response-objects-in-django-488652b91cd3

Python Plain English. (2022). *Creating an MTV (Model Template View) architecture in Django*. https://python.plainenglish.io/creating-an-mtv-model-template-view-architecture-in-django-59c6502e15c8