Ejercicios POLIMENTES

Capacitación

Práctica 09082021

ORM

Arquitecturas Web

Tal cual no existe una definición absoluta para lo que es la arquitectura de software, básicamente se trata de cómo se va a conectar cada una de las partes de un sistema. Las arquitecturas web describen la relación entre los distintos elementos que componen la estructura de funcionamiento de las páginas y aplicaciones web. Es la forma que tienes de organizar la información dentro de tu página web.

Uno de las más populares era el "Patrón modelo vista controlador" ya que permitía en un mismo proyecto crear el backend como parte de controlador, la vista como el diseño (JS, CSS y HTML) y el modelado como partes de la base de datos. Entonces vemos que era construido como en capas, por lo mismo era realmente rápido, sin embargo, se terminaba con un proyecto gigantesco que no permite su fácil mantenimiento.

La arquitectura Rest es una manera de crear API´s.

Es una arquitectura que propone una serie de principios que permite crear sistemas que sea de fácil acceso y estándares para cualquier dispositivo con conexión a internet.

Las aplicaciones que se crean basada en la arquitectura REST son aplicaciones y sistemas basados en RECURSOS y no en acciones, estos elementos/recursos de información son identificador por una URI (o URL que hace referencia a "Identificador Único del Recurso).

>Principios en los que se basa la arquitectura REST

1° Las aplicaciones basadas en esta arquitectura son aplicaciones 'Cliente – Servidor'

Es decir, siempre hay un cliente que consume lo que un servidor almacena o genera

2° Las aplicaciones basadas en esta arquitectura son aplicaciones Stateless

Se trata de aplicaciones que no guardan estados, es decir no almacenan información del cliente que deba ser usada para servir la aplicación realmente, cada petición al cliente debe ser autocontenida, consiguiendo así un ahorro en variables de sesión y almacenamiento interno del servidor.

3° Todos los recursos deben exponerse a través de una interfaz uniforme

Los recursos en una app REST se exponen a través de URL´s que son identificadores de recursos universales, estas URL´s siempre hacen referencia a los recursos y NO a acciones. Dentro de las peticiones no van como tal los recursos sino una representación de ellos, lo cual nos permite recibir cierta cantidad de propiedades de un elemento en un dispositivo mientras recibimos otras de otro

4° Priorización del uso de enlaces o hipermedia

Al ser el servidor una aplicación sin estado como se menciona en el segundo principio, todo el estado de la aplicación se maneja a través de enlaces, significa que todas las representaciones que regresen del servidor hacia un cliente deben incluir enlaces hacia

objetos relacionados con él, es decir para la representación y transición de la información utiliza hipermedios únicamente que suelen ser HTML, XML o JSON.

5° Las aplicaciones son cacheables

Se refiere a que el almacenamiento debe ocurrir del lado del cliente y lo que se almacena son la peticiones que se realizan al servidor, esto con la finalidad de reducir la carga en el servidor y aumentar la velocidad de respuesta en el cliente, sin embargo, la responsabilidad de establecer si un objeto o cierta petición es cacheable recae 100% en el servidor y dependen del recurso del que se trate

6° Son aplicaciones multicapa

Es decir al cliente no le debe interesar saber si esta conectado directamente con el servidor o hay una capa intermedia que este entre él y el servidor, Esta capa intermedia puede ser cualquier cosa (un servidor de aplicaciones, proxy, un valenciador de carga, etc.) no importa que, la aplicación debe funcionar.

Ahora ¿cómo se realizan las acciones de una aplicación basada en recursos? Para poder realizar acciones en una aplicación tipo REST es necesario utilizar los verbos o peticiones HTTP, es así como las API´s se comunican entre sí y como entre front y back se comunican:

Get → para obtener y/o consultar recursos

Post → envía datos y crea nuevos recursos

Put → modificar/actualiza datos o recursos

Delete → elimina los recursos o datos

Es la opción de arquitectura para sistemas distribuidos, la anterior solo era un estilo o un patrón parea navegadores web, en cambio esta al ser para sistemas distribuidos es multiplataforma (lap, compu, teléfono, etc) en rest SOLO queda el backend y el modelo de datos, dejando la vista por fuera como proyecto independiente.

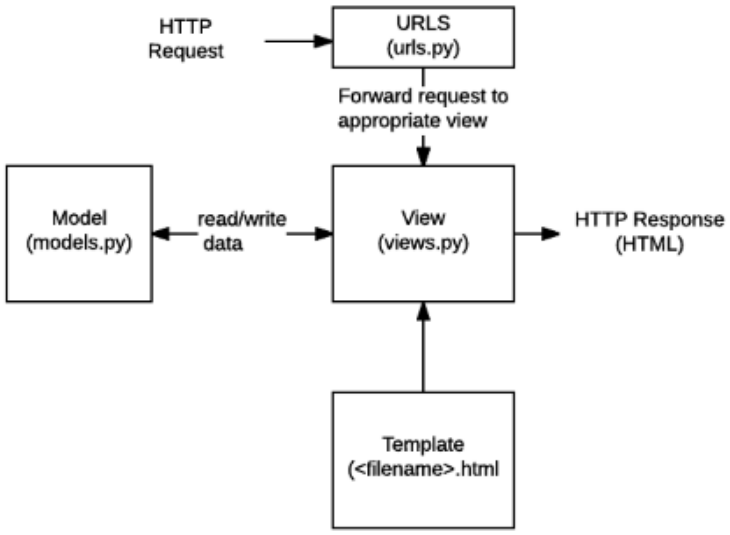
Entonces REST es un estilo arquitectónico que separa totalmente al front, así front solo se dedica a consumir servicios, es decir solo se comunica mediante API’s

DJANGO

Framework web escrito en python diseñado para realizar aplicaciones de cualquier complejidad en tiempos razonables y que permite el desarrollo de sitios web seguros y mantenibles.

En un sitio web tradicional basado en datos, una aplicación web espera peticiones HTTP del explorador web (o de otro cliente). Cuando se recibe una petición la aplicación elabora lo que se necesita basándose en la URL y posiblemente en la información incluida en los datos POST o GET. Dependiendo de qué se necesita quizás pueda entonces leer o escribir información desde una base de datos o realizar otras tareas requeridas para satisfacer la petición. La aplicación devolverá a continuación una respuesta al explorador web, con frecuencia creando dinámicamente una página HTML para que el explorador la presente insertando los datos recuperados en marcadores de posición dentro de una plantilla HTML.

Las aplicaciones web de Django normalmente agrupan el código que gestiona cada uno de estos pasos en ficheros separados:



- **URLs:** Aunque es posible procesar peticiones de cada URL individual vía una función individual, es mucho más sostenible escribir una función de visualización separada para cada recurso. Se usa un mapeador URL para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición. El mapeador URL se usa para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición. El mapeador URL puede también emparejar patrones de cadenas o dígitos específicos que aparecen en una URL y los pasan a la función de visualización como datos.
- **Vista (View):** Una vista es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP. Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de *modelos*, y delegan el formateo de la respuesta a las *plantillas* (*"templates"*).
- **Modelos (Models):** Los Modelos son objetos de Python que definen la estructura de los datos de una aplicación y proporcionan mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la base de datos.
- **Plantillas (Templates):** una plantilla (template) es un fichero de texto que define la estructura o diagrama de otro fichero (tal como una página HTML), con marcadores de posición que se utilizan para representar el contenido real. Una *vista* puede crear dinámicamente una página usando una plantilla, rellenandola con datos de un *modelo*. Una plantilla se puede usar para definir la estructura de cualquier tipo de fichero; ¡no tiene porqué ser HTML.

Django viene cargado con módulos y paquetes que nos permiten únicamente adaptarlos a las necesidades de nuestro proyecto.

Otra ventaja es que es escalable y podemos pasar de algo muy básico a una aplicación compleja perfectamente modularizada. También es seguro ya que proporciona una manera segura de administrar cuentas de usuario y contraseñas, evitando así errores comunes como colocar informaciones de sesión en cookies donde es vulnerable (en lugar de eso las cookies solo contienen una clave y los datos se almacenan en la base de datos) o se almacenan directamente las contraseñas en un hash de contraseñas. Django permite protección contra algunas vulnerabilidades de forma predeterminada, incluida la inyección SQL, scripts entre sitios, falsificación de solicitudes entre sitios y clickjacking

Django existe Django nativo y Django REST framework, REST solo recibe objetos JSON y es así como back y front se comunican, mediante objetos tipo json únicamente.

Al crear un nuevo proyecto en Django se crea una estructura de carpetas/ficheros como la siguiente:

```
NombreProyecto/

    manage.py

    NombreProyecto/

        settings.py

        urls.py
```

```
wsgi.py
```

- **Módulo manage.py** → script para ejecutar las herramientas de Django para este proyecto, es el empaquetador de 'django-admin.py'. Nos ayuda a crear migraciones cuando creamos un nuevo modelo o al hacer modificaciones en nuestros modelos y enviar estas migraciones, así como correr el servidor interno de django.
  Sirve como intermediario, a través de el es que podemos ejecutar comandos propios de django para crear aplicaciones o superusuarios y que también tiene definida las configuraciones del proyecto y al ejecutar comandos a través de esta archivo le indicamos a django que reconozca esas configuraciones para ejecutar lo que hayamos indicado.
  En su script hace referencia justo a la subcarpeta que se creó con el nombre del proyecto en la línea donde dice *NombreProyecto.settings*
- **Módulo settings.py**→ configuraciones que genera django para ESE proyecto que contiene múltiples elementos:

  `from pathlib import Path` →Importación que se hace para poder indicar en líneas posteriores la ruta en donde se encuentra nuestro proyecto.

  `SECRET_KEY = '!+qpr#cq3r(^w-%#dz*6$cf-#z543_10*edx#qn(t7yfj-7w7j'`→Llave única por cada proyecto que sirve para identificarlo

  `DEBUG = True` → Sirve para indicarnos los errores siempre que esté activado(True) sin embargo, ya que esté en producción el proyecto, el modo debug debe ser desactivado.

  `ALLOWED_HOSTS = []`→ Una vez que subimos a producción el Proyecto, debemos indicar aquí las IP o dominio válido desde los cuales pueden acceder al proyecto

  `INSTALLED_APPS = []`→ Recordemos que una aplicación es una abstracción y django toma el concepto de aplicación como cada uno de estos módulos que tiene el proyecto para separar la lógica, es decir abstrae, separa y se concentra por secciones para escribir y distribuir mejor el código, aquí es donde podremos ver los modelos que creamos, las vistas, etc. De primera instancia solo pone las aplicaciones básicas y propias de Django que necesita el framework como tal para funcionar

  `MIDDLEWARE = []` → Información que sirve como seguridad entre servidor y cliente para que se repeten las jerarquías, permisos y reglas establecidas en la comunicación entre cliente y servidor y que no sea el sistema tan vulnerable

  `ROOT_URLCONF = 'MiPrimeraAppDjango.urls'`→ Donde se encuentran las rutas principales de todo el proyecto

  `TEMPLATES [{}]`→ Estructura para las plantillas, Django utiliza un sistema de plantillas ya establecido

  `WSGI_APPLICATION = 'MiPrimeraAppDjango.wsgi.application'`→ Esta parte sirve para producción para indicarle al servidor como puede levanrtar nuestra aplicación y cómo recibir las comunicaciones haciendo referencia al documento wsgi

  `DATABASES = {}`→ Aquí es sobre las bases de datos que soporta django y en esta sección se especifica por defecto con que base de datops trabaja, que es sqlite3 y así viene configurado. Si en algún momento se quiere configurar alfo de las bases de datos nos dirigimos a esta sección

  `AUTH_PASSWORD_VALIDATORS = []`→Información sobre validaciones de passwords para los usuarios

  `LANGUAGE_CODE='es-mx'`→Correspondiente a la sección de internacionalización en la parte del idioma

  `TIME_ZONE='America/Mexico_City'`→ Correspondiente a la sección de internacionalización en la parte de la zona horaria

`USE_I18N=True`→Correspondiente a la sección de internacionalización
`USE_L10N=True`→Correspondiente a la sección de internacionalización
`USE_TZ = False`→ Correspondiente a la sección de internacionalización

`STATIC_URL = '/static/'`→Indica la ruta donde guardaremos los archivos estáticos (Java, CSS, HTML, imágenes, etc)

- **Módulo urls→** este archivo contiene las rutas que continen nuestro proyecto, en la documentación contiene las indicaciones básicas por si se desea configurar de cierta manera, lo cual se utiliza para la creación de vistas posteriormente. Contiene también un arreglo de urlpatterns.
- **Módulo wsgi→**se indica la configuración que tiene haciendo referencia al documento setting previamente descrito

Al crear una app dentro del proyecto también se crean módulos en ella:

Módulo admin → Se utiliza para el administrador de Django que renderiza la clase a maquetación HTML y CSS para facilitar la creación de registros por medio de una interfaz

Modulo apps → para que settings del proyecto sepa de su existencia (de la app correspondiente)

Módulo models→ Donde creamos los modelos de datos, las tablas con el ORM de django en código nativo

Módulo test → Originalmente pensado para hacer pruebas unitarias, sin embargo no resulta ser la mejor opción debido as que puede escalar de tamaño el proyecto y ya no resulta útil, es mejor ahí ejecutar las pruebas de querys por ejemplo a la DB

Módulo views→ donde se crean las vistas para ya crear, actualizar, eliminar y obtener datos de la app, en MVC también se ingresa lógica de programación en este módulo

NOTAS:

django-admin es la utilidad de línea de comando de django que nos apoya con los comandos para crear los proyectos, aplicaciones y superusuarios.

django-admin startproject 'NombreProyecto' se ejecuta dentro del directorio donde quiero localizar mi proyecto

django-admin startapp se ejecuta dentro del directorio que acabo de crear al crear el proyecto