Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2021.

PROYECTO 3 IPC2

202110568 – Damián Ignacio Peña Afre

Resumen

Uno de los servicios emergentes más comunes en el ámbito de la tecnología es la nube y resulta indispensable una forma de llevar el control acerca de las instancias que se están consumiendo pues ahí reside el beneficio económico. Una API es capaz de solucionar el problema, pues permite una comunicación de algun otro servicio para la modificación, creación o eliminación de datos que de alguna forma persisten la información como lo puede ser la cantidad de recursos disponibles, el tipo o configuración de las instancias, así como los consumos que realiza cada cliente. Se plantea entonces implementar dos servicios que se comuniquen por medio del protocolo *http* de tal forma que en el primero de ellos cuente con una interfaz (*frontend*) para manipular las solicitudes que se realizarán al segundo servicio o API (*backend)*. Dentro de las tecnologías que pueden desarrollar este tipo de proyecto y teniendo en cuenta la utilización del lenguaje *Python* se encuentra *Django* y *Flask*, si bien ambos pueden cumplir la misma función se utilizaran de manera separada para simular una arquitectura de microservicios.

Palabras clave

Frontend, backend, microservicios, computación en la nube

Abstract

One of the most common emerging services in the field of technology is the cloud, and a way of keeping track of the instances that are being consumed is essential, since that is where the economic benefit lies. An API is capable of solving the problem, since it allows a communication of some other service for the modification, creation or elimination of data that in some way persists the information such as the amount of available resources, the type or configuration of the instances, as well as the consumption made by each client. It is then proposed to implement two services that communicate through the http protocol in such a way that the first one has an-interface (frontend) to handle the requests made to the second service or API (backend). Within the technologies that can develop this type of project and taking into account the use of the Python language are Django and Flask, although both can fulfill the same function, they will be used separately to simulate a microservices architecture.

Keywords

Frontend, backend, microservices, cloud computing

Introducción

Los sistemas de información web representan un activo de alto orden para las organizaciones. Tal y como señala una aplicación web es básicamente una manera de facilitar el logro de una tarea específica en la Web, mientras un sitio web estático esencialmente es una herramienta para la comunicación. Los sistemas de información web son ejemplos de aplicaciones web que han evolucionado desde páginas web dinámicas hasta sistemas de información con bases de datos.

Es un enfoque para el desarrollo de una aplicación única como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y mecanismos ligeros de comunicación, a menudo un recurso de una interfaz de programación de aplicaciones (*API*) sobre protocolo de transferencia de hipertexto (*HTTP*). Estos servicios están construidos alrededor de las capacidades del negocio y con independencia de despliegue e implementación totalmente automatizada. Existe un mínimo de gestión centralizada de estos servicios, los que pueden estar escritos en lenguajes de programación diferentes y utilizar diferentes tecnologías de almacenamiento de datos

La utilización de API's resulta común en el desarrollo moderno, particularmente en el desarrollo de aplicaciones web y más aún cuando se tiene en cuenta la arquitectura de microservicios, donde de alguna forma la responsabilidad de cada uno de los servicios es compartida haciendo mucho más fácil el entregar actualizaciones específicas.

Surge con el concepto de *API* los estándares para la comunicaciones con dichas interfaces, durante los últimos años a emergido con gran popularidad el formato *JSON* debido a su simplicidad, tamaño, fácil lectura y manipulación en distintos lenguajes. En *Python* no resulta más que un simple diccionario. De forma no muy lejana era recurrente la utilización del *XML* como estándar la comunicación.

Pasando propiamente al caso de utilización de todas estas tecnologías se plantea el caso de una empresa que brinda servicios en la nube, particularmente instancias en la nube. Debida a la gran cantidad de recursos que manejan, entre ellos: memoria RAM, núcleos utilizados por el procesador, o bien sean soluciones de software como son sistemas operativos bases de datos etc, requieren de un sistema que lleve un control atómico de la utilización de estos recursos, así como de las categorías, configuraciones y otras segmentaciones referentes a la arquitectura que implementa la empresa.

Inmediatamente surge la necesidad de almacenar datos. Para esta solución se propone la utilización de archivo *xml* con una estructura predefinida donde se registraran las distintas entidades así como las relaciones entre ellas.

Un concepto o tecnología clave a la hora de manipular datos estructurados de cierta manera son los ORM's. Un ORM *'*mapea*'* las entidades de una base de datos objetos propios del lenguaje para que

de esta manera sea mucho más sencilla y práctica su utilización. De esta manera se crea una capa de abstracción sobre los elementos que manipulan de manera directa la base de datos, en este caso los encargados de sobrescribir o leer un archivo *xml*.

Pasando a otra arquitectura que se implementa para la solución nos encontramos con la arquitectura cliente servidor. Dentro de esta arquitectura se planeta una forma de comunicación entre un *cliente* y un *servidor*, que es el que finalmente provee los recursos o requerimientos que solicita el cliente, como lo podría ser una aplicación web.

De parte del servidor es requerido procesar la petición que se realizó por medio del protocolo *http*, y de esta manera determinar qué acción realizar, ya sea renderizar alguna vista *html* o modificar de alguna forma la base de datos. A todo lo que engloba el procesamiento de lado del servidor se le conoce como *backend.*

Del lado del cliente se cuenta con el navegador, que es que el finalmente representa y consume la información o los elementos que han sido estructurados por el servidor. A esta sección se le conoce como *frontend.*

Dentro del entorno del lenguaje *Python* existen una gran cantidad de paquetes, librerías y frameworks que pueden utilizarse para correr del lado del servidor. Dentro de los mas utilizados se encuentran *Django* y *Flask*, estos, si bien realizan la misma función cuentan con un enfoque y herramientas bases diferentes.

Por parte de *Flask*, no se incluye una capa de abstracción de base de datos, validación de formularios o cualquier otra cosa donde ya existen diferentes bibliotecas que pueden manejar eso. En su lugar, *Flask* soporta extensiones para añadir dicha funcionalidad a tu aplicación como si estuviera implementada en el propio *Flask*. Numerosas extensiones proporcionan integración de bases de datos, validación de formularios, manejo de cargas, varias tecnologías de autenticación abierta, y más. Flask puede ser «micro», pero está listo para su uso en producción en una variedad de necesidades.

De manera resumida, *Flask* es conocido como un *microframework* por lo cual no restringe o da la libertad de escoger las configuraciones necesarias para su funcionamiento, así como los patrones que se necesiten implementar.

Para la solución se aprovechó esta característica y se implementó un modelo común dentro del desarrollo *backend*, el *modelo vista controlador*. En este modelo se segmenta a modo de capas las acciones que realiza cada parte de la *API*, por lo que se obtiene una estructura y código mucho mas limpia. Por poner un ejemplo, la responsabilidad de modificar o crear un registro en la base de datos no tiene por qué hacerse dentro de la declaración de la ruta de cierto endpoint, sino más bien es conveniente realizarlo en otro archivo o capa que se encargue únicamente de este tipo de acciones. Por otro lado, la definición de la estructura de la información no tiene o no es recomendable que se realice dentro del controlador, por lo que se realiza dentro de la capa de modelo.

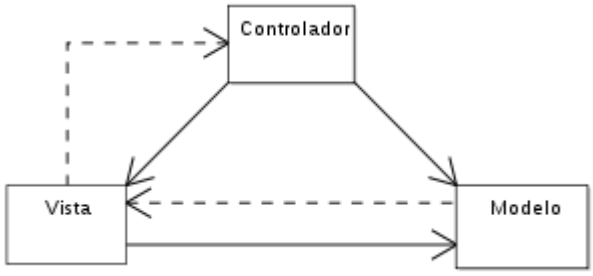

Figura 2. Modelo vista controlador.

Fuente: Wikipedia.

Pasando la otra tecnología utilizada, Django es un framework web de alto nivel que permite el desarrollo rápido de sitios web seguros y mantenibles. Desarrollado por programadores experimentados, Django se encarga de gran parte de las complicaciones del desarrollo web, por lo que puedes concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda. Por lo que a diferencia de *Flask, Django* cuenta con una estructura y capacidades iniciales básicas preconfiguradas, traduciéndose en menores tiempos de desarrollo y planificación en cuestión de organización.

Django cuenta con un motor de vistas o templates para la renderización de documentos *html,* de manera especifica utiliza un paquete ampliamente utilizado en el ambiente de *Python* que es *Jinja*, *Jinja* trae código Python a Html, por lo que es posible leer variables que hayan sido procesadas en el servicio 2, iterarlas con un ciclo y posteriormente definir una estructura.

De forma resumida la arquitectura final tendría la siguiente forma, con el servicio 1 teniendo la responsabilidad de generar las vistas y servicio 2 procesando las solicitudes que son pasadas del cliente, al servicio 1 y finalmente al propio servicio 2

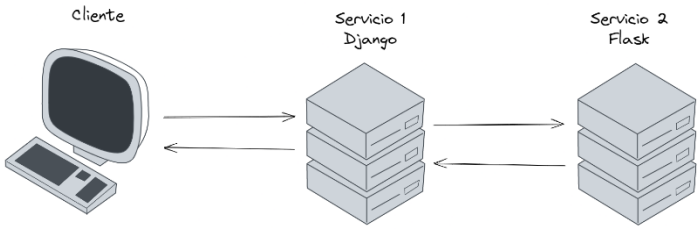

Figura 2. Arquitectura de la aplicación.

Fuente: elaboración propia.

Anexos

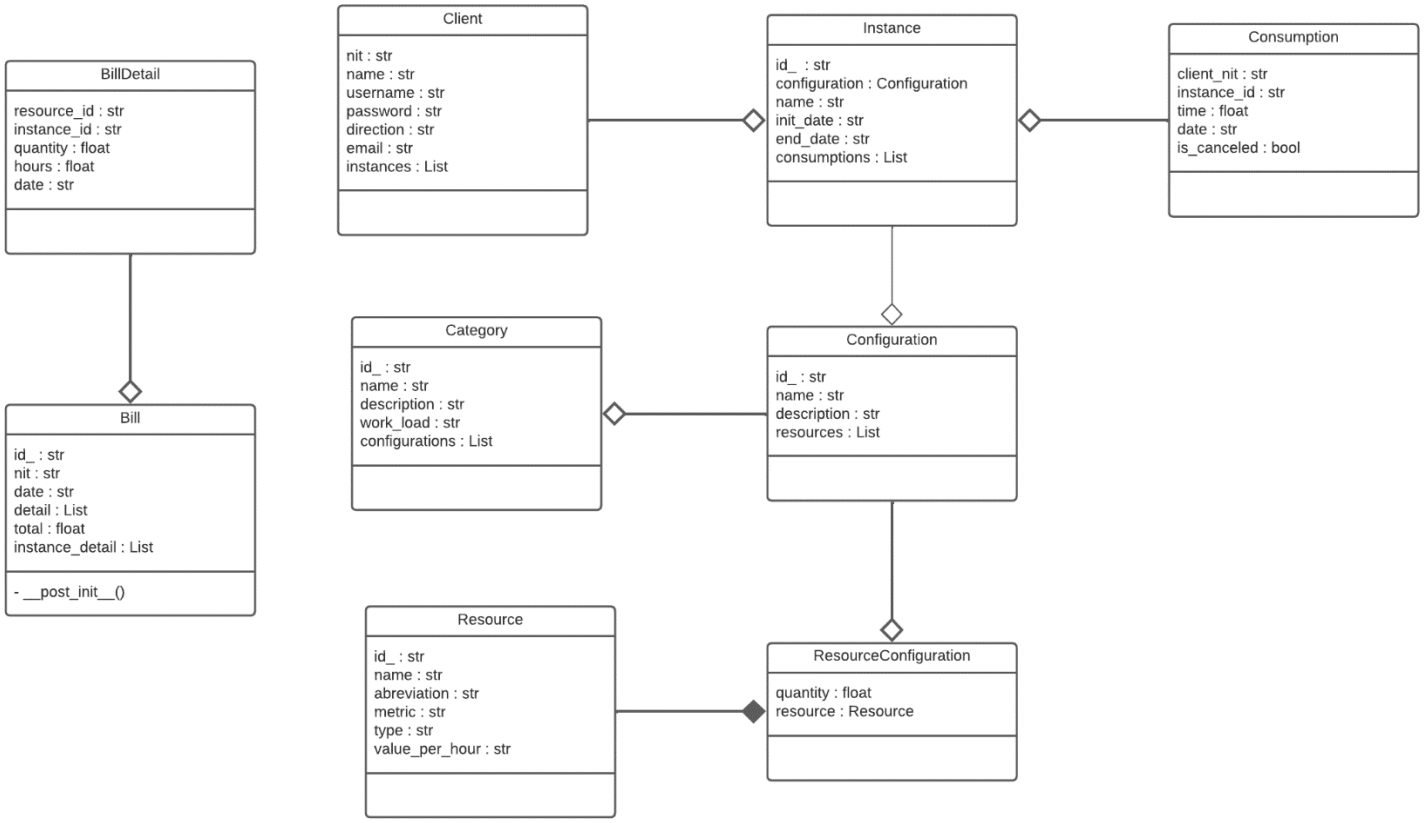

Figura 3. Diagrama de clases de la solución.

Fuente: elaboración propia.

```xml
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="db">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="listaRecursos">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="recurso">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element type="xs:string" name="nombre" />
                                        <xs:element type="xs:string" name="abreviatura" />
                                        <xs:element type="xs:string" name="metrica" />
                                        <xs:element type="xs:string" name="tipo" />
                                        <xs:element type="xs:float" name="valorXhora" />
                                    </xs:sequence>
                                    <xs:attribute type="xs:short" name="id" />
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="listaConfiguraciones">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="configuracion">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element type="xs:string" name="nombre" />
                                        <xs:element type="xs:string" name="descripcion" />
                                        <xs:element name="recursoConfiguracion">
                                            <xs:complexType>
                                                <xs:simpleContent>
                                                    <xs:extension base="xs:float">
                                                        <xs:attribute type="xs:short" name="id" />
                                                    </xs:extension>
                                                </xs:simpleContent>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                    <xs:attribute type="xs:byte" name="id" />
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="listaCategorias">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="categoria">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element type="xs:string" name="nombre" />
                                        <xs:element type="xs:string" name="descripcion" />
                                        <xs:element type="xs:string" name="cargaTrabajo" />
                                        <xs:element name="configuracionCategoria">
                                            <xs:complexType>
                                                <xs:simpleContent>
                                                    <xs:extension base="xs:string">
                                                        <xs:attribute type="xs:byte" name="id" />
                                                    </xs:extension>
                                                </xs:simpleContent>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:sequence>
                                    <xs:attribute type="xs:short" name="id" />
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="listaConsumos">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="consumo">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element type="xs:float" name="tiempo" />
                                        <xs:element type="xs:string" name="fechaHora" />
                                        <xs:element type="xs:string" name="cancelado" />
                                    </xs:sequence>
                                    <xs:attribute type="xs:long" name="nitCliente" />
                                    <xs:attribute type="xs:short" name="idInstancia" />
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="listaInstancias">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="instancia">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element type="xs:byte" name="idConfiguracion" />
                                        <xs:element type="xs:string" name="nombre" />
                                        <xs:element type="xs:string" name="fechaInicio" />
                                        <xs:element type="xs:string" name="estado" />
                                        <xs:element type="xs:byte" name="fechaFinal" />
                                    </xs:sequence>
                                    <xs:attribute type="xs:short" name="id" />
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="listaClientes">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="cliente">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element type="xs:string" name="nombre" />
                                        <xs:element type="xs:string" name="usuario" />
                                        <xs:element type="xs:byte" name="clave" />
                                        <xs:element type="xs:string" name="direccion" />
                                        <xs:element type="xs:string" name="correoElectronico" />
                                        <xs:element name="instanciaCliente">
                                            <xs:complexType>
                                                <xs:simpleContent>
                                                    <xs:extension base="xs:string">
                                                        <xs:attribute type="xs:short" name="id" />
```

Figura 4. Estructura XSD

Fuente: elaboración propia

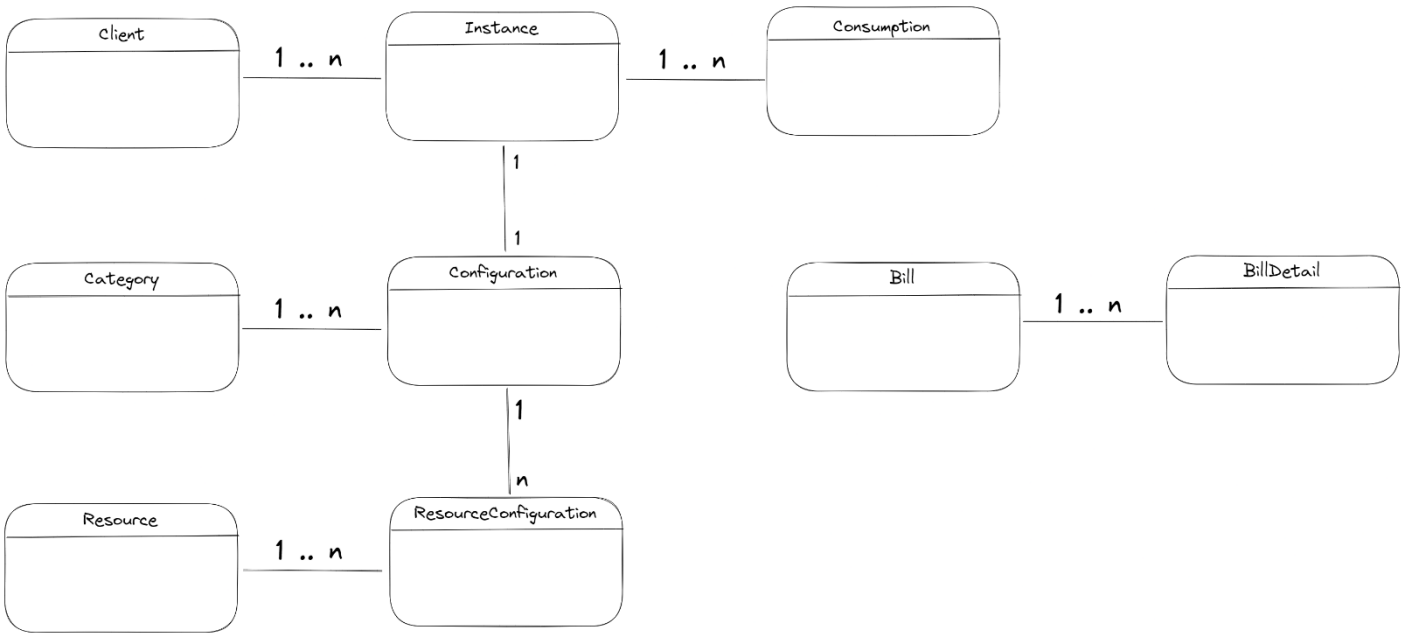

Figura 5. Diagrama entidad relación

Fuente: elaboración propia