Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2022.

Proyecto 3

202109567 – Alvaro Norberto García Meza

Resumen

Dicho proyecto fue propuesto con la finalidad de crear una aplicación web donde los clientes podrían adquirir servicios por categorías y recursos, donde se les generaba facturas.

La aplicación brinda una aplicación web la cual permite a los usuarios crearse una cuenta, donde posteriormente podrán registrar recursos, categorías y consumos. Para luego adquirir y configurar dichas propiedades a su gusto con una limites de fecha y tiempos, dichas instancias creadas pueden ser canceladas o estar vigentes para poder ser facturadas cuando un archivo de consumos es cargado a la plataforma, el cual genera una factura con los detalles de los gatos desde que la instancia fue creada hasta la fecha de corte.

Palabras clave

Instancia, recurso, dirección, petición, datos, tiempo y configuración.

Abstract

This project was proposed with the purpose of creating a web application where clients could acquire services by categories and resources, where invoices were generated. The application provides a web application which allows users to create an account, where they can later register resources, categories, and consumption. To then acquire and configure these properties to your liking with a date and time limit, these created instances can be canceled or be in force to be billed when a consumption file is uploaded to the platform, which generates an invoice with the details of the cats since the instance was created until the cut-off date.

Keywords

instance, resource, endpoint, request, time, and configuration

Introducción

El desarrollo de la aplicación se realizó con la ayuda de diferentes paradigmas de programación de dos frameworks populares de pytthon para la construcción de una aplicación web. Los paradigmas de programación implementados fueron Programación Orientada a Objetos, programación funcional y modular para desarrollar de manera ordenada las diferentes funcionalidades las cuales en algunos casos eran dependientes de otras.

Se construyo una API con le framework Flask el cual brindaba una estructura para la construcción de un servidor que reciba peticiones con los distintos protocolos http para recibir la información, procesarla y luego enviar una respuesta. Se cargó dos archivos XML desde la parte del frontend que posteriormente fueron enviados para el servidor backend y donde se creo una base de datos. Además, desde el frontend se mandaba información o se solicitaba información al backend el cual hacia una request a nuestra simulación de base de datos que era un archivo JSON.

Posteriormente se utilizo el framework django para la construcción del frontend y donde el usuario puede interactuar.

Desarrollo del tema

La aplicación fue desarrollada con el lenguaje de programación “Python” debido a su gran facilidad de sintaxis y manejo de variables. Además, implementando librerías integradas del propio lenguaje y los frameworks flask y django.

A. Directorios y Módulos

Cada proyecto y aplicación debe contener cierta jerarquía y organización para un manejo ordenado del a información, en este caso un buen manejo de la funcionalidad y datos globales.

- **Backend:** En este directorio se encuentra dos directorios más.
  - **Components:** Contiene módulos de Python para diferentes funciones de nuestra API.
  - **src:** Este directoro contiene la base de datos que es un archivo json y el servidor en el módulo main.py

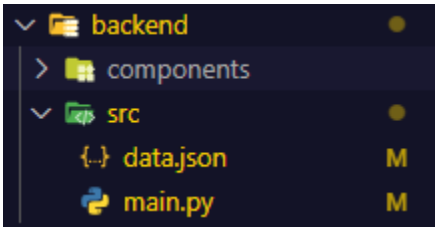

Figura 1. Directorios y Módulos de backend.

Fuente: elaboración propia, 2022.

- **Frontend:** En el frontend contamos con diferentes módulos y directorios que siguen a jerarquía, que el framework django proporciona, los distintos módulos y directorios se compaginan para poder crear una aplicación web con interfaz gráfica.

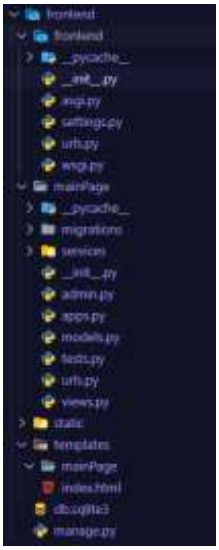

Figura 2. Directorios y Módulos de frontend.

Fuente: elaboración propia, 2022.

B. Modelo de datos relacional

Al utilizar la simulación de una base de datos en un archivo JSON, se creo un diagrama de bases para representarla.

Figura 2. Modelo de datos relacional.

Fuente: elaboración propia, 2022.

C. Endpoints

- **/consultarDatos:** Este endpoint carga todos los datos de la base de datos y los regresa en formato json.

```python
@app.route("/consultarDatos", methods=['GET'])
def consultarDatos():
    with open('data.json') as json_file:
        data = json.load(json_file)
        print(data["lista_recursos"][0]["id"])
        return jsonify(data)
```

Figura 3. consultardatos.

Fuente: elaboración propia, 2022.

- **/crearRecurso:** Recibe un json con los respectivos valores para crear un recurso, los consume y los guarda en la base de datos.

```python
@app.route("/crearRecurso", methods=['POST'])
def crearRecurso():
    data = request.get_json()
    # obtener base datos
    with open('data.json') as json_file:
        data_base = json.load(json_file)
    # validar recursos
    for recurso in data_base['lista_recursos']:
        if data['id'] == recurso['id']:
            return jsonify({"error": "El recurso ya existe"})
    # agregar recurso
    data_base['lista_recursos'].append(data)
    # guardar base datos
    with open('data.json', 'w') as outfile:
        json.dump(data_base, outfile, indent=4)

    return jsonify({"mensaje": "Recurso creado exitosamente"})
```

Figura 4. crearrecurso.

Fuente: elaboración propia, 2022.

- **/crearCategoria:** Al igual que el anterior endpoint recibe datos, los consume y los almacena en la base de datos.

```python
@app.route("/crearCategoria", methods=['POST'])
def crearCategoria():
    data = request.get_json()
    # obtener base datos
    with open('data.json') as json_file:
        data_base = json.load(json_file)
    # validar recursos
    for categoria in data_base['lista_categorias']:
        if data['id'] == categoria['id']:
            return jsonify({"error": "La categoria ya existe ya existe"})

    # agregar recurso
    data_base['lista_categorias'].append(data)
    # guardar base datos
    with open('data.json', 'w') as outfile:
        json.dump(data_base, outfile, indent=4)

    return jsonify({"mensaje": "Categoria creado exitosamente"})
```

Figura 5. crearcategoria.

Fuente: elaboración propia, 2022.

- **/crearConfiguración**
- **/crearCliente**
- **/crearInstancia**

  Estos tres endpoints tienen la misma funcionalidad, reciben una petición json, la consumen y luego la guardan en la base de datos y retornan un mensaje.

Figura 6. Métodos post.

Fuente: elaboración propia, 2022.

- **/generarFactura:** Este endpoint consume al cliente el cual se le quiere generar una factura y se le devuelve el detalle y el valor de todo lo consumido.
- **/detalleFactura:** Este endpoint solicita que factura ver y se le devolverá con más detalle lo consumido.
- **/archivoEntrada:** Este endpoint recibe el archivo XML cargado desde el frontend, para ser consumido y crear la base de datos.
- **/archivoConsumo:** Al igual que el archivo de entrada, este consumo un xml de los consumos que un cliente realizo y que servirán para la construcción de las facturas.

Conclusiones

El propósito de este proyecto es construir una aplicación web la cual se conecte a una API para consumir y obtener valores. Con ayuda del framework flask se pudo construir el backend para que luego con el con el framework django se pueda construir un frontend para comunicar estos dos servicios y el cliente pueda interactuar.

Dicho proyecto busca fortalecer los conocimientos en el desarrollo web, donde se tenga noción de los protocolos http que se pueden llegar a manejar, como construir una pagina web para que el usuario pueda interactuar y como comunicar los datos mediante la conexión a una simulación de una base de datos.

Referencias bibliográficas

xml.etree.ElementTree — The ElementTree XML API — Python 3.10.6 documentation. (s. f.). Recuperado 6 de septiembre de 2022, de https://docs.python.org/3/library/xml.etree.elementtree.html

Django. (2022). Django 4.1 documentation. Django. Recuperado el 1 de noviembre de 2022, de https://docs.djangoproject.com/es/4.1/

Flask, (2022). Flask, Desarrollo web, una gota a la vez. Flask. Recuperado el 1 de noviembre de 2022, de https://flask-es.readthedocs.io/

Anexos

Figura 9. Documento de entrada 1 en XML.

Fuente: elaboración propia, 2022

```xml
<?xml version="1.0"?>
<listadoConsumos>
    <consumo nitCliente="3000" idInstancia="31">
        <tiempo> 8 </tiempo>
        <fechaHora> 2022-06-10 13:00 Guatemala </fechaHora>
    </consumo>
    <consumo nitCliente="3000" idInstancia="31">
        <tiempo> 10.25 </tiempo>
        <fechaHora> 2022-06-12 12:20 USA</fechaHora>
    </consumo>
    <consumo nitCliente="3000" idInstancia="31">
        <tiempo> 2 </tiempo>
        <fechaHora> 2022-06-15 11:35 USA</fechaHora>
    </consumo>
    <consumo nitCliente="3002" idInstancia="321">
        <tiempo> 23.45 </tiempo>
        <fechaHora> 2022-09-15 11:35 LTS</fechaHora>
    </consumo>
    <consumo nitCliente="3002" idInstancia="321">
        <tiempo> 10.15 </tiempo>
        <fechaHora> 2022-09-23 18:12 </fechaHora>
    </consumo>
    <consumo nitCliente="3002" idInstancia="322">
        <tiempo> 23.30</tiempo>
        <fechaHora> 2022-07-15 11:31 </fechaHora>
    </consumo>
</listadoConsumos>
```

Figura 10. Documento de entrada 2 en XML.

Fuente: elaboración propia, 2022

```json
{
  "lista_recursos": [
    {
      "id": "1000",
      "nombre": "Nucleo de unidad central de procesamiento",
      "abreviatura": "CPU",
      "metrica": "Ghz ",
      "tipo": "Procesamiento",
      "valorXhora": "0.02"
    },
    {
      "id": "1001",
      "nombre": "Memoria de Acceso Aleatorio",
      "abreviatura": "RAM",
      "metrica": "GB ",
      "tipo": "Almacenamiento",
      "valorXhora": "0.03"
    },
    {
      "id": "1002",
      "nombre": "Unidad de estado solido",
      "abreviatura": "SSD",
      "metrica": "GB",
      "tipo": "Almacenamiento",
      "valorXhora": "0.03"
    },
    {
      "id": "1003",
      "nombre": "Unidad de disco duro",
      "abreviatura": "DD",
      "metrica": "GB",
      "tipo": "Almacenamiento",
      "valorXhora": "0.01"
    }
  ],
```

*Figura 11.*Base de datos JSON.

Fuente: elaboración propia, 2022

```json
"lista_categorias": [
    {
        "id": "2000",
        "nombre": "Small",
        "descripcion": "Instancias para ejecucion de proyectos en consola",
        "carga_trabajo": "Minima",
        "configuraciones": [
            {
                "id": "201",
                "nombre": "Default ",
                "description": "Configuracion por default",
                "recursos_configuracion": [
                    {
                        "id": "1000",
                        "value": " 2 "
                    },
                    {
                        "id": "1001",
                        "value": " 1 "
                    },
                    {
                        "id": "1003",
                        "value": " 5 "
                    }
                ]
            }
        ]
    },
```

```xml
<?xml version="1.0"?>
<listadoConsumos>
    <consumo nitCliente="3000" idInstancia="31">
        <tiempo> 8 </tiempo>
        <fechaHora> 2022-06-10 13:00 Guatemala </fechaHora>
    </consumo>
    <consumo nitCliente="3000" idInstancia="31">
        <tiempo> 10.25 </tiempo>
        <fechaHora> 2022-06-12 12:20 USA</fechaHora>
    </consumo>
    <consumo nitCliente="3000" idInstancia="31">
        <tiempo> 2 </tiempo>
        <fechaHora> 2022-06-15 11:35 USA</fechaHora>
    </consumo>
    <consumo nitCliente="3002" idInstancia="321">
        <tiempo> 23.45 </tiempo>
        <fechaHora> 2022-09-15 11:35 LTS</fechaHora>
    </consumo>
    <consumo nitCliente="3002" idInstancia="321">
        <tiempo> 10.15 </tiempo>
        <fechaHora> 2022-09-23 18:12 </fechaHora>
    </consumo>
    <consumo nitCliente="3002" idInstancia="322">
        <tiempo> 23.30</tiempo>
        <fechaHora> 2022-07-15 11:31 </fechaHora>
    </consumo>
</listadoConsumos>
```

*Figura 12.*Base de datos SON.

Fuente: elaboración propia, 2022