**Evaluación Parcial 2**

**“Interacción con API, Dockers, Jenkins y Gestión de Claves”**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sigla** | **Nombre Asignatura** | **Tiempo Asignado** | **% Ponderación** |
| **DRY7122** | **Programación y Redes Virtualizadas (SDN-NFV)** | **160 minutos** | **33** |

1. **Agente evaluativo**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **X** | **Heteroevaluación** |  |  | **Coevaluación** |  |  | **Autoevaluación** |

1. **Tabla de Especificaciones**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Resultado de Aprendizaje** | **Indicador de Logro (IL)** | **Indicador de Evaluación (IE)\*** | **Ponderación Indicador Logro** | **Ponderación Indicador de Evaluación** |
| Emplea soluciones de conectividad programable para mantener actualizadas las capacidades de la red según las necesidades de la organización. | 2.1 Identifica las diferentes interfaces de programación de aplicaciones (API), según los requerimientos de la organización. | 1. Obtiene token de red programable, logrando que se almace en ambiente en Postman. | 15% | **5%** |
| 1. Utiliza API que permita consultar por red física de red programable. | **5%** |
| 1. Utiliza API que permita crear usuario con contraseña requerida, permitiendo ingresar a dashboard de controlador. | **5%** |
| 2.2 Verifica las soluciones de conectividad programables según el tipo de protocolo para la infraestructura de la organización. | 1. Demuestra con script desarrollado la distancia en kilometros entre Santiago y Ovalle, además de solicitar Ciudad de Origen y Ciudad de Destino. | 25% | **5%** |
| 1. Demuestra con script desarrollado la duración el viaje en horas, minutos y segundos, además del combustible requerido en litros. | **5%** |
| 1. Demuestra con script desarrollado que todos los valores utilizan dos decimales, además de imprimir la narrativa del viaje. | **5%** |
| 1. Demuestra con script desarrollado la salida del programa con la letra q, y subir el script respectivo a repositorio público en GitHub. | **10%** |
| 2.3 Emplea técnicas para la implementación integral y continua de procesos automatizados según los requerimientos de la empresa. | 1. Ingresa al directorio solicitado y con comando respectivo poder almacenar todos los archivos en dicha ubicación. | 20% | **10%** |
| 1. Logra cambiar el puerto de la aplicación de muestra al puerto 9999 y la instación del Docker Jenkins. | **10%** |
| 2.4 Reconoce el concepto de alta disponibilidad en la implementación de aplicaciones de red en función de una arquitectura TI, con la finalidad de entregar seguridad a la organización. | 1. Crea un trabajo en Jenkins con nombre requerido, además de asociarlo al repositorio GitHub respectivo, además que el trabajo permita ejecutar y compilar script para permitir funcionamiento de página web de muestra. | 20% | **10%** |
| 1. Valida que la página web se encuentre disponible con el puerto requerido, y que los archivos se encuentren respaldados en GitHub. | **10%** |
| 2.5 Diseña arquitectura automatizadas con equipos físicos y/o virtuales para la infraestructura de la organización. | 1. Crea script con nombre requerido y código importa los paquetes requeridos para la gestión de claves. | 20% | **5%** |
| 1. Agrega código de Flask que permite al archivo crear la primera fase del contenido utilizando el puerto 5000, además de realizar la validación del sitio con el comando curl. | **5%** |
| 1. Agrega líneas de códigos dentro del script que permita almacenar nombres y contraseñas en texto plano, verificar nuevas contraseñas y en cada intento de sesión leer los parámetros de una solicitud HTTP y verificar la cuenta. | **5%** |
| 1. Valida a través del comando curl dos usuarios con sus contraseñas y luego a través de aplicación DB Browser SQLite, revisa que dichas información se encuentre almacenada. | **5%** |
| **Total** | | | **100%** | **100%** |

1. **Instrucciones para el/la estudiante**

|  |
| --- |
| Esta es una evaluación que corresponde a una evaluación práctica sin presentación y tiene un 33**%**deponderaciónsobre la nota final de la asignatura.  El **tiempo** para desarrollar esta evaluación es de **160 minutos** y se realiza de manera **individual** en **laboratorio respectivo.** Para el desarrollo de esta evaluación requiere el uso de máquina virtual DEVASC.  **Contexto:**  La empresa “DRY7122 S.A” se encuentra en proceso de evaluación tecnológica, donde buscará migrar equipos de conectividad antiguos por versiones más actualizadas. Por otro lado, se han enterado de que muchos de estos cambios involucran la automatización y la integración de la programación, por lo cual antes de hacer este cambio necesitan nivelar el conocimiento de los administradores de red en esta temática para luego proceder con el cambio respectivo.  **Instrucciones Generales:**   1. Esta evaluación puede realizarse de manera individual. 2. Utilizará máquina virtual DEVASC, los cuales deberán estar importados en VirtualBox. En caso contrario lo puede obtener del siguiente link: [bit.ly/3KqL89E](http://www.bit.ly/3KqL89E) 3. En un documento Word deben hacer capturas de pantallas, el cual debe aparecer fecha y hora, donde debe abordar el requerimiento planteado, además de una breve descripción de lo realizado ahí. 4. Es importante que guarden sus avances cada 5 minutos. 5. Una vez finalizada la evaluación deberán adjuntar el documento realizado en formato .PDF el cual debe ser adjuntado a través de la actividad AVA respectiva.   **Requerimientos:**   1. **Consumo de API en Red Programable**   Utilizando archivo Cisco Packet Tracer en este link: <http://bit.ly/3AgInnH> deberá realizar consumo de API a una controladora utilizando Postman, donde deberá lograr lo siguiente:   * Obtener token de la red programable, el cual deberá quedar almacenado en un ambiente llamado **Prueba2** * Utilizar API que permita consultar por la red física. * Utilizar API que permita crear usuario **Prueba2** y clave **SDN.2023**. Deberá demostrar el acceso al dashboard de la controladora.**z**  1. **Consumo de API Pública**   Utilizando el sitio de MapQuest y el token generado en laboratorio respectivo, deberá crear un código en VisualStudio Code de la máquina virtual DEVASC, donde el programa realice lo siguiente:   * Medir la distancia en kilómetros entre Santiago y Ovalle. * Solicitar “Ciudad de Origen” y “Ciudad de Destino” * Mostrar la duración del viaje en horas, minutos y segundos * Mostrar el combustible requerido para el viaje representado en litros * Todos los valores deben utilizar dos decimales. * Debe imprimir la narrativa del viaje. * Agregar una salida del programa con la letra “**q**” * Subir el script creado en un repositorio público en GitHub con el nombre de “**Evaluación N°2 DRY7122**” y en la descripción los nombres de los integrantes. Debe incluir un commit con el nombre de “**Consumo de API Pública”**                1. **Docker y CI/CD**   Utilizando máquina virtual DEVASC deberá realizar lo siguiente:   * En el directorio **Jenkins/sample-app** deberá utilizar comando respectivo para almacenar todos los archivos en el actual directorio. * Generar un git commit **Docker y CI/CD** * Deberá cambiar el puerto de aplicación web de muestra, el cual deberá ser el **9999** * Realizar la instalación del Docker Jenkins y configuración inicial de la aplicación respectiva. * Generar un trabajo con el nombre **Trabajo-Eva2**, el cual debe estar asociado al repositorio de GitHub. * Configurar el trabajo creado que permita ejecutar y compilar el script para la puesta en funcionamiento de la página web de muestra. * Validar que la página web de muestra esté disponible utilizando el puerto solicitado. * Revisar que los archivos del directorio actual en que trabajó con Jenkins se encuentren respaldado en repositorio de GitHub.  1. **Control de Credenciales**   Utilizando máquina virtual DEVASC deberá realizar lo siguiente:   * Crear un script con el nombre de **claves.py** * Agregar Código que importe los paquetes requeridos para la gestión de claves y el uso de la base de datos SQL. * Debe agregar el código de Flask que permita al archivo crear la primera fase del contenido web de la ruta raíz. El sitio web utilizará el puerto 5000. * Validar que el servidor Flask esté operativo utilizando comando curl. * Agregar código para almacenar nombres de usuarios y contraseñas en texto plano. * El código deberá verificar las nuevas credenciales. * El código deberá en cada intento de sesión leer los parámetros de una solicitud HTTP y verificar la cuenta. * A través del comando curl validará dos usuarios, en este caso los nombres de los integrantes de la evaluación. Las contraseñas a utilizar son a elección. * Utilizando la aplicación DB Browser para SQLite, deberá abrir el archivo que contiene los usuarios y contraseñas utilizados en el paso anterior. |

Codigo

import requests

import urllib.parse

# Configuración de la API

url\_ruta = "https://graphhopper.com/api/1/route?"

clave\_api = "API" # Reemplaza con tu clave real

def geocodificar(ubicacion, clave):

"""Convierte nombres de lugares a coordenadas geográficas"""

while ubicacion == "":

ubicacion = input("Por favor ingrese la ubicación nuevamente: ")

url\_geocodificacion = "https://graphhopper.com/api/1/geocode?"

parametros = {"q": ubicacion, "limit": "1", "key": clave}

url = url\_geocodificacion + urllib.parse.urlencode(parametros)

try:

respuesta = requests.get(url)

datos = respuesta.json()

if respuesta.status\_code == 200 and datos.get("hits"):

latitud = datos["hits"][0]["point"]["lat"]

longitud = datos["hits"][0]["point"]["lng"]

nombre = datos["hits"][0]["name"]

pais = datos["hits"][0].get("country", "")

region = datos["hits"][0].get("state", "")

if region and pais:

ubicacion\_formateada = f"{nombre}, {region}, {pais}"

elif pais:

ubicacion\_formateada = f"{nombre}, {pais}"

else:

ubicacion\_formateada = nombre

return respuesta.status\_code, latitud, longitud, ubicacion\_formateada

else:

print(f"\nError: No se encontró la ubicación '{ubicacion}'")

return respuesta.status\_code, None, None, ubicacion

except Exception as e:

print(f"\nError de conexión: {str(e)}")

return 500, None, None, ubicacion

def calcular\_combustible(distancia\_km):

"""Calcula el combustible necesario (12 km por litro)"""

return distancia\_km / 12.0

def formatear\_duracion(segundos):

"""Convierte segundos a horas, minutos y segundos"""

horas = int(segundos // 3600)

minutos = int((segundos % 3600) // 60)

segundos = int(segundos % 60)

return horas, minutos, segundos

def mostrar\_resultados(origen, destino, distancia, duracion, combustible):

"""Muestra los resultados del cálculo de ruta"""

print("\n" + "="\*60)

print("📝 RESUMEN DEL VIAJE".center(60))

print("="\*60)

print(f"🔹 Origen: {origen}")

print(f"🔹 Destino: {destino}")

print(f"📏 Distancia total: {distancia:.2f} km")

horas, minutos, segundos = formatear\_duracion(duracion)

print(f"⏱️ Duración estimada: {horas:02d}h {minutos:02d}m {segundos:02d}s")

print(f"⛽ Combustible necesario: {combustible:.2f} litros (a 12 km/l)")

print("="\*60)

def mostrar\_instrucciones(ruta):

"""Muestra las instrucciones de navegación paso a paso"""

print("\n📌 INSTRUCCIONES DE RUTA:")

for paso in ruta["paths"][0]["instructions"]:

distancia\_km = paso["distance"] / 1000

print(f"- {paso['text']} ({distancia\_km:.2f} km)")

print("="\*60)

def main():

print("\n" + "="\*60)

print("🚗 CALCULADOR DE VIAJES".center(60))

print("="\*60)

print("Este programa calcula:")

print("- Distancia entre ciudades")

print("- Duración del viaje")

print("- Combustible requerido")

print("\nInstrucciones:")

print("1. Ingrese ciudad de origen y destino")

print("2. Presione 'q' para salir en cualquier momento")

print("="\*60)

while True:

# Entrada de origen

origen\_input = input("\n🏠 Ciudad de Origen (o 'q' para salir): ").strip()

if origen\_input.lower() == 'q':

break

# Geocodificación de origen

estado\_origen, lat\_origen, lon\_origen, origen\_nombre = geocodificar(origen\_input, clave\_api)

if estado\_origen != 200:

print("No se pudo procesar el origen. Intente nuevamente.")

continue

# Entrada de destino

destino\_input = input("🏁 Ciudad de Destino (o 'q' para salir): ").strip()

if destino\_input.lower() == 'q':

break

# Geocodificación de destino

estado\_destino, lat\_destino, lon\_destino, destino\_nombre = geocodificar(destino\_input, clave\_api)

if estado\_destino != 200:

print("No se pudo procesar el destino. Intente nuevamente.")

continue

# Construcción de URL para la API de rutas

parametros\_ruta = {

"key": clave\_api,

"vehicle": "car",

"point": [f"{lat\_origen},{lon\_origen}", f"{lat\_destino},{lon\_destino}"]

}

url = url\_ruta + urllib.parse.urlencode(parametros\_ruta, doseq=True)

try:

respuesta = requests.get(url)

datos\_ruta = respuesta.json()

if respuesta.status\_code == 200:

# Cálculos principales

distancia\_km = datos\_ruta["paths"][0]["distance"] / 1000

duracion\_seg = datos\_ruta["paths"][0]["time"] / 1000

combustible = calcular\_combustible(distancia\_km)

# Mostrar resultados

mostrar\_resultados(origen\_nombre, destino\_nombre, distancia\_km, duracion\_seg, combustible)

mostrar\_instrucciones(datos\_ruta)

else:

print(f"\nError al calcular la ruta: {datos\_ruta.get('message', 'Error desconocido')}")

except Exception as e:

print(f"\nError de conexión: {str(e)}")

# Consultar por otra ruta

otra\_ruta = input("\n¿Desea calcular otra ruta? (s/n): ").lower()

if otra\_ruta != 's':

break

print("\n¡Gracias por usar el calculador de viajes! �")

if \_\_name\_\_ == "\_\_main\_\_":

try:

main()

except KeyboardInterrupt:

print("\nPrograma interrumpido por el usuario")

except Exception as e:

print(f"\nError inesperado: {str(e)}")