Documentación Técnica: Extracción de Enlaces de Servicios Geográficos en Colombia en Mapas

1. Requerimientos y Dependencias

Para ejecutar el script es necesario contar con:

- Python 3.x.
- **Selenium:** Permite la automatización del navegador.
- **Chrome WebDriver:** Se utiliza la librería webdriver_manager para gestionar la versión compatible de ChromeDriver de forma automática.
- **Dependencias adicionales de Selenium:**
  - selenium.webdriver.chrome.options.Options
  - selenium.webdriver.support.ui.WebDriverWait
  - selenium.webdriver.support.expected_conditions as EC

La instalación de estas dependencias se puede realizar, por ejemplo, mediante pip install selenium webdriver-manager.

2. Descripción General del Flujo de Trabajo

El script sigue estos pasos principales:

1. **Configuración del navegador:** Inicializa un navegador Chrome en modo headless (sin interfaz gráfica) con una resolución predefinida.
2. **Carga de la página y aceptación de términos:** Accede a la URL de Colombia en Mapas y se encarga de aceptar los términos y condiciones, además de cerrar cualquier pop-up tutorial.
3. **Selección de parámetros:**
   - **Zona geográfica:** Se selecciona el departamento o zona (por ejemplo, "Antioquia") a partir de un menú desplegable.
   - **Tópico:** Se elige un tema o categoría del mapa (por ejemplo, "Geodesia").
   - **Servicio:** Se selecciona una capa o servicio específico (por ejemplo, "Red Pasiva GNSS").
4. **Extracción y filtrado de enlaces:** Se busca en la sección de enlaces la URL correspondiente al servicio solicitado, filtrando en base al tipo de enlace (por ejemplo, “wms”).
5. **Manejo de errores:** Se incluyen bloques try/except para detectar y gestionar problemas en cada etapa del proceso (selección, carga, extracción de datos).

3. Análisis Detallado del Código

1. **Configuración del Navegador**

   El bloque inicial configura el entorno del navegador:

   - **Headless mode:** Se utiliza chromeOptions.add_argument("--headless=new") para correr el navegador sin abrir una ventana gráfica, lo cual es útil en entornos de servidores o cuando se requiere automatización sin intervención visual.
   - **ChromeDriverManager:** Permite gestionar automáticamente la versión adecuada del ChromeDriver sin tener que descargarla manualmente.

```
chromeOptions = Options()
chromeOptions.add_argument("--headless=new")
chromeOptions.add_argument("--window-size=1920,1080")
driver = webdriver.Chrome(
    service=Service(ChromeDriverManager().install()),
    options=chromeOptions
)
```

2. **Función handleTermsConditions**

   Esta función se encarga de gestionar la aceptación de los términos legales y el cierre del tutorial inicial que aparece al cargar la página. Se utiliza WebDriverWait para esperar a que los elementos sean clicables:

   - **Espera activa:** Se espera que el botón identificado por ID "checkTerminos" sea clicable, lo que indica que se puede aceptar los términos.
   - **Cierre del tutorial:** Luego se cierra el pop-up asociado al tutorial mediante un selector CSS específico.

```
def handleTermsConditions(driver):
    """Handles initial legal agreements and tutorial popups"""
    try:
        WebDriverWait(driver, 15).until(
            EC.element_to_be_clickable((By.ID, "checkTerminos"))
        ).click()
        WebDriverWait(driver, 15).until(
            EC.element_to_be_clickable((By.CSS_SELECTOR, "a.close[onclick='cancelTutorial(); return false;']"))
        ).click()
    except Exception as e:
        raise RuntimeError("Failed to accept terms and conditions") from e
```

3. **Funcion getServiceLinks**

   Esta es la función principal que realiza la extracción de enlaces. Se estructura en varias etapas:

   a) **Carga Inicial y Manejo de Términos**
      - Se accede al URL principal y se invoca **handleTermsConditions** para gestionar los pop-ups legales.

   b) **Selección de la Zona Geográfica**
      - Se localiza el selector (elemento <select>) mediante el ID searchFiltro.
      - Se recorre la lista de opciones (<option>) comparando el texto (ignorando mayúsculas y minúsculas) para encontrar la zona indicada.

- Si la zona no se encuentra, se lanza un ValueError con las opciones disponibles.

```
zoneSelector = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "searchFiltro"))
)
zoneOptions = zoneSelector.find_elements(By.TAG_NAME, "option")
# Se busca la opción que coincida con el nombre de la zona
```

c) Selección del Topico

- Se utiliza un XPath que busca elementos cuyo contenido textual (convertido a minúsculas) contenga el nombre del tópico indicado.
- Una vez localizado, se utiliza scrollIntoView para centrar el elemento en la vista antes de hacer clic.

```
topicXpath = f"""
    //div[contains(@class, 'tematica')
    and contains(translate(., 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz'),
    '{targetTopic.lower()}')]
"""
topicElement = WebDriverWait(driver, 15).until(
    EC.presence_of_element_located((By.XPATH, topicXpath))
)
driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", topicElement)
topicElement.click()
```

d) Selección del Servicio

- Se utiliza otro XPath complejo para encontrar el servicio específico (por ejemplo, "Red Pasiva GNSS"), buscando en los elementos que contengan la clase media-resultados2.
- Se asegura que el elemento esté visible y se hace scroll para centrarlo antes de ejecutar el clic.

```
serviceXpath = f"""
    //div[contains(@class, 'media-resultados2')]//*
    [contains(translate(normalize-space(.),
    'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
    'abcdefghijklmnopqrstuvwxyz'),
    '{targetService.lower()}')]
"""
serviceElement = WebDriverWait(driver, 20).until(
    EC.element_to_be_clickable((By.XPATH, serviceXpath))
)
driver.execute_script("arguments[0].click();", serviceElement)
```

e) Extracción y Filtrado de Enlaces

- Se localiza el contenedor de enlaces con el selector ul#enlacesContainer.
- Se extraen todos los elementos <a> que contengan un atributo href.
- Si se especifica un filtro (por ejemplo, 'wms'), se revisa tanto el URL como el texto del enlace para incluir únicamente aquellos que cumplan la condición.
- En caso de que no se encuentren enlaces, se lanza un error.

```
linkContainer = WebDriverWait(driver, 15).until(
    EC.presence_of_element_located((By.CSS_SELECTOR, "ul#enlacesContainer"))
)
allLinks = linkContainer.find_elements(By.CSS_SELECTOR, "a[href]")
filteredLinks = []
for link in allLinks:
    url = link.get_attribute("href")
    linkText = link.text.lower()

    if linkType:
        if linkType.lower() in url.lower() or linkType.lower() in linkText:
            filteredLinks.append(url)
    else:
        filteredLinks.append(url)
```

4. Gestion de Errores y Excepciones

El código implementa varios mecanismos para garantizar que cada paso se ejecute correctamente:

- **Bloques try/except:** Cada sección crítica (selección de zona, tópico, servicio y extracción de enlaces) está envuelta en un bloque try/except para capturar errores específicos.
- **Mensajes de error informativos:** Se incluyen mensajes que indican la causa del fallo, lo que facilita el debug y la mejora del código.
- **Re-lanzamiento de excepciones:** Se encapsulan las excepciones originales en errores más específicos (ValueError o RuntimeError) para que el usuario tenga un contexto claro de lo ocurrido.

5. Ejemplo de Uso y Configuración

El script termina con una sección de configuración y ejecución en la que se especifican:

- La **zona** (ej. "Antioquia").
- El **tópico** (ej. "geodesia").
- El **servicio** (ej. "Red Pasiva GNSS").
- El **tipo de enlace** solicitado (ej. "wms").

Posteriormente, se llama a **getServiceLinks** y se imprimen los enlaces resultantes. La gestión de errores también se realiza a este nivel para notificar al usuario en caso de problemas de configuración o fallas operativas.

```python
selectedZone = "Antioquia"
selectedTopic = "geodesia"
selectedService = "Red Pasiva GNSS"
requestedLinkType = "wms"

try:
    serviceLinks = getServiceLinks(
        zoneName=selectedZone,
        targetTopic=selectedTopic,
        targetService=selectedService,
        linkType=requestedLinkType
    )
    for link in serviceLinks:
        print(link)
except ValueError as error:
    print(f"\nCONFIGURATION ERROR: {error}")
except RuntimeError as error:
    print(f"\nOPERATIONAL ERROR: {error}")
except Exception as error:
    print(f"\nUNEXPECTED ERROR: {type(error).__name__} - {error}")
```