Documentación del Crawler para cuba.travel

Equipo de Desarrollo

15 de junio de 2025

Índice

1. Introducción

Este documento describe el funcionamiento del crawler desarrollado para extraer información de ofertas hoteleras del sitio web `cuba.travel`. El sistema utiliza Selenium para automatizar la navegación y extracción de datos, con configuración basada en reglas de `robots.txt`.

2. Configuración del Crawler

2.1. Archivo de configuración (`crawler_config.py`)

Contiene parámetros esenciales para el funcionamiento del crawler:

```
# crawler_config.py
CRAWLER_CONFIG = {
    "sitemap": "https://www.cuba.travel/sitemapindex.xml",
    "user_agents": {
        "*": {
            "disallow": [
```

```
                "/admin/",
                "/App_Browsers/",
                # ... (lista completa de exclusiones)
            ],
            "crawl_delay": None,
        },
        # ... (configuraciones por user-agent)
    },
    "destinations": [
        "La Habana",
        "Varadero",
        # ... (lista completa de destinos)
    ],
}
```

Listing 1: Configuración principal

2.2. Configuración de Selenium

Parámetros para controlar el navegador:

```
SELENIUM_CONFIG = {
    "driver": "chrome",
    "headless": False,
    "window_size": "1920,1080",
    "user_agent": "Mozilla/5.0 (compatible; TourGuideCubaBot/1.0; ...)",
    "implicit_wait": 0,
    "page_load_timeout": 120,
    "disable_images": True,
    "disable_javascript": False,
    "disable_cookies": True,
}
```

Listing 2: Configuración de Selenium

3. Clase CubaTravelCrawler

3.1. Estructura principal

Clase que gestiona todo el proceso de crawling:

```
class CubaTravelCrawler:
    def __init__(self, base_url="https://www.cuba.travel/"):
        self.base_url = base_url
        self.config = CRAWLER_CONFIG
        self.selenium_config = SELENIUM_CONFIG
        self.driver = self._init_driver()
        self.disallow_patterns = self._compile_disallow_patterns()
        self.crawl_delay = self._get_crawl_delay()

    # ... (funciones internos)
```

Listing 3: Clase principal del crawler

3.2. Métodos principales

- `_init_driver()`: Inicializa el controlador de Chrome con las opciones configuradas
- `_compile_disallow_patterns()`: Convierte reglas de robots.txt a patrones regex
- `is_allowed(url)`: Verifica si una URL está permitida para crawling

- `_select_destination(wait, destination)`: Selecciona un destino en la interfaz
- `extract_offers()`: Extrae datos de las ofertas en la página actual
- `crawl(urls)`: Ejecuta el proceso completo de crawling
- `close()`: Cierra el navegador y limpia recursos

4. Flujo de Trabajo

El proceso de crawling sigue estos pasos:

1. Inicializar el navegador Chrome con configuración personalizada
2. Para cada URL proporcionada:
   a) Verificar si está permitida por robots.txt
   b) Cargar la página principal
   c) Para cada destino en la lista de destinos:
      1) Seleccionar el destino en el menú desplegable
      2) Hacer clic en el botón "Buscar"
      3) Recorrer todas las páginas de resultados
      4) Extraer datos de cada oferta hotelera
3. Retornar resultados estructurados por destino

5. Estructura de Salida

Los datos extraídos tienen el siguiente formato por destino:

```
{
    "La Habana": [
        {
            "name": "Hotel Nacional de Cuba",
            "stars": 5,
            "address": "Calle 21 y O, Vedado",
            "cadena": "Gran Caribe",
            "tarifa": "Todo Incluido",
            "price": "$150/noche",
            "hotel_url": "https://www.cuba.travel/.../detail"
        },
        # ... ofertas
    ],
    "Varadero": [
        # ... ofertas para Varadero
    ],
    # ... otros destinos
}
```

Listing 4: Estructura de datos de salida

6. Ejemplo de Uso

Implementación básica del crawler:

```python
from crawler import CubaTravelCrawler

# Inicializar crawler
crawler = CubaTravelCrawler()

# Ejecutar crawling en URL principal
results = crawler.crawl(["https://www.cuba.travel/"])

# Procesar resultados
for destino, ofertas in results.items():
    print(f"\nDestino: {destino} ({len(ofertas)} ofertas)")
    for hotel in ofertas:
        print(f"- {hotel['name']} ({hotel['stars']}*)")

# Liberar recursos
crawler.close()
```

Listing 5: Ejemplo de implementación

7. Optimizaciones Implementadas

- Directorio temporal para datos de usuario
- Deshabilitación de imágenes y cookies
- Patrones regex para URLs prohibidas
- Scroll automático a elementos
- Manejo robusto de paginación
- Inyección de CSS para ocultar elementos multimedia

8. Limitaciones y Mejoras Futuras

8.1. Limitaciones Actuales

- Dependencia de selectores CSS/XPATH específicos
- No manejo de CAPTCHAs o bloqueos avanzados
- Extracción de precios sensible a cambios en la estructura HTML

8.2. Mejoras Sugeridas

- Implementar sistema de proxies rotativos
- Añadir reintentos automáticos para fallos
- Integrar parámetros de búsqueda personalizables (fechas, huéspedes)
- Añadir soporte para almacenamiento en base de datos
- Implementar monitoreo de cambios en la estructura del sitio

9. Consideraciones Éticas

- Respeto estricto a robots.txt y politicas del sitio
- User-agent identificable con información de contacto
- Crawl-delay configurado para minimizar impacto
- Uso responsable de los datos extraídos