



Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Departamento de Estadística, Informática y Matemáticas

Trabajo de fin de grado

Plataforma de adquisición de datos pluviométricos para la predicción y aviso de inundaciones

Autor:

Arkaitz Oderiz Garin

Supervisor:

Unai Pérez-Goya

Pamplona, 2023

Contenidos

Lista de Figuras	v
Lista de Códigos	vii
1 Introducción	1
1.1 Antigua intro al parecer inutil :v	6
2 Datos	9
2.1 Aemet	10
2.2 CHCantábrico	11
2.3 MeteoNavarra	12
2.4 El Agua en Navarra	15
3 Tecnologías	21
3.1 Sistema Operativo Debian	21
3.1.1 Historia	21
3.1.2 Filosofía	21
3.1.3 Por qué Debian?	22
3.2 Framework	23
3.2.1 Qué es un framework?	23
3.2.2 Librería vs Framework	24
3.2.3 Django	24
3.3 Herramienta para Web Scraping	25
3.3.1 Qué es el Web Scraping?	25
3.3.2 Procedimiento básico en Web Scraping	26
3.3.3 Scrapy	26
3.4 Lenguaje de Programación Python	28

3.4.1 Qué es Python?	28
3.4.2 Por qué Python?	28
3.5 Base de Datos	28
3.5.1 PostGreSQL	28
3.5.2 Por qué PostGreSQL?	29
4 Diseño de la Plataforma	31
4.1 Estructuras Planteadas	31
4.1.1 Problema	31
4.1.2 Solución	32
4.2 Trabajar con los Datos	32
4.2.1 Flujo de Datos	32
4.2.2 Datos obtenidos por página	33
4.2.3 Formato de Datos	33
4.2.4 Filtrado de Datos	34
4.3 Arquitectura	34
4.3.1 Elementos presentes	35
4.4 Entorno de ejecución	39
4.4.1 Preparación de entorno virtual	40
4.4.2 Instalación y configuración de PostGreSQL	40
4.4.3 Instalación y configuración de Django	42
5 Implementación	49
5.1 Creación de Spiders	49
5.1.1 Proceso de obtención de datos	51
5.1.2 Guardado de datos	53
5.1.3 Spider básica	54
5.1.4 Método start_requests()	55
5.1.5 Eliminar Log	55
5.2 Spiders usadas	56
5.2.1 Aemet	56
5.2.2 CHCantábrico	57
5.2.3 MeteoNavarra	64
5.2.4 Agua en Navarra	68
5.3 Formateo de datos	76
5.3.1 Aemet	76
5.3.2 CHCantábrico	76
5.3.3 MateoNavarra	76
5.3.4 Agua en Navarra	76
5.4 Filtrado de datos	76
6 Conclusiones y Trabajo Futuro	79

Contenidos **iii**

Referencias	81
Glosario	85
Anexos	85

Lista de Figuras

1.1 Emisiones globales de CO ₂	2
1.2 Efecto invernadero	2
1.3 Media de aumento de temperatura global entre 1901-2000	3
1.4 Precipitaciones 2022	4
1.5 Estructura de datos planteada	6
2.1 Confederaciones hidrográficas en España	9
2.2 Página Aemet de la estación en Aranguren (Navarra)	10
2.3 HTML de la tabla de datos de Aemet de la estación en Aranguren (Navarra)	11
2.4 HTML de las coordenadas de Aemet de la estación en Aranguren (Navarra)	11
2.5 Página Nivel de los ríos CHCantábrico	12
2.6 Página estaciones MeteoNavarra	13
2.7 HTML tabla estaciones MeteoNavarra	13
2.8 Apartado selección de datos MeteoNavarra	14
2.9 Página de datos de MeteoNavarra	14
2.10 Página principal de aforos de El Agua en Navarra	15
2.11 HTML mapa estaciones de El Agua en Navarra	16
2.12 Página estación de El Agua en Navarra	16
2.13 HTML estación de El Agua en Navarra	17
2.14 Gráfica de datos en estación de El Agua en Navarra	18
2.15 Error al cargar directamente la página de datos numéricos en El Agua en Navarra	18
2.16 Datos numéricos de estaciones en Agua en Navarra	19
3.1 Diagrama patrón MVT	24
3.2 Web Scraping (Krotov y Tennyson 2018)	26

4.1 Idea original de la estructura de datos planteada	31
4.2 Estructura de datos usada en el proyecto	32
4.4 Estructuración básica de una Spider	36
4.5 Tablas de la base de datos	39
4.6 Lista de tablas en PostGreSQL	44
4.3 Arquitectura de obtención y tratamiento de datos	48
5.1 Estructura del proyecto recién creado	49
5.2 Directorio de almacenamiento de las Spider	50
5.3 URL de inicio para obtener los códigos de las estaciones de Aemet	51
5.4 Inspector de webs de Chrome	52
5.5 Ruta chromedriver.exe	73

Lista de Códigos

4.1 Modelos Django	44
4.2 Modelos Django	45
4.3 Modelos Django	45
4.4 Modelos Django	46
5.1 Spider recién generada	50
5.2 Guardar datos	53
5.3 Configurar guardado en JSON	53
5.4 Spider de ejemplo (Aemet Code Spider)	54
5.5 Sobre-escritura de start_request()	55
5.6 Configurar LOG	55
5.7 Selector en parse() de Aemet Data Spider	56
5.8 Trabajar sobre los datos de Aemet Data Spider	56
5.9 Guardado de datos de Aemet Data Spider	57
5.10 Selector en parse() de CHCantábrico Code Spider	57
5.11 Trabajar sobre los datos de CHCantábrico Code Spider	57
5.12 Comprobación de límites de CHCantábrico Code Spider	58
5.13 Guardado de datos de CHCantábrico Code Spider	58
5.14 Función start_requests() CHCantábrico Nivel Spider	58
5.15 Función parse() CHCantábrico Nivel Spider	60
5.16 Guardado de datos de CHCantábrico Nivel Spider	61
5.17 Selector en parse() de CHCantábrico Coordinates Spider	61
5.18 Guardado de datos de CHCantábrico Coordinates Spider	61
5.19 Script de obtención de datos pluviométricos descartado	62
5.20 Selector en parse() de MeteoNavarra Code Spider	64
5.21 Guardado de datos de MeteoNavarra Code Spider	64
5.22 Uso de fechas en función start_requests() MeteoNavarra Data Spider	65

5.23 Selector en parse() de MeteoNavarra Data Spider	65
5.24 Trabajar sobre los datos de MeteoNavarra Data Spider	66
5.25 Comprobacion exitencia de datos y guardado de MeteoNavarra Data Spider	66
5.26 Navegacion a segunda página de datos en MeteoNavarra Data Spider . .	67
5.27 Selector en parse() de MeteoNavarra Coordenates Spider	67
5.28 Guardado de datos de MeteoNavarra Coordenates Spider	67
5.29 Función parse() Agua en Navarra Spiders	68
5.30 Función parse_area() Agua en Navarra Spiders	68
5.31 Función parse_estacion() Agua en Navarra Code Spider	69
5.32 Guardado de datos de Agua en Navarra Code Spider	69
5.33 Función parse_estacion() Agua en Navarra Data Spider	69
5.34 Selector en parse_data() de Agua en Navarra Data Spider	70
5.35 Selector en parse_data() de Agua en Navarra Data Spider	71
5.36 Agua en Navarra configuración Selenium	71
5.37 Configuración Playwright	74
5.38 Playwright basic Request	75
5.39 Agua en Navarra Playwright Request	75
5.40 Import necesarios	76
5.41 Declaración rutas JSONs y nombre de fichero	76
5.42 Declaración función openFile()	77
5.43 Declaración función saveFile()	77
5.44 Declaración función refinedData()	77
5.45 Declaración rutas JSONs	78
A1 Aement Code Spider	85
A2 Aement Data Spider	86
A3 CHCantabrico Code Spider	87
A4 CHCantabrico Nivel Spider	88
A5 CHCantabrico Pluvio Spider	89
A6 CHCantabrico Coord Spider	91
A7 MeteoNavarra Code Spider	92
A8 MeteoNavarra Data Spider	93
A9 MeteoNavarra Coord Spider	94
A10 Agua en Navarra Code Spider	95
A11 Agua en Navarra Data Spider	97

1

Introducción

Las Naciones Unidas definen el cambio climático como el conjunto de "*cambios a largo plazo de las temperaturas y los patrones climáticos*" [1]. Estos pueden ocurrir de forma natural cada cierto tiempo, ya sea debido a erupciones volcánicas, fluctuaciones de la radiación solar e, incluso, variaciones en la órbita terrestre, llegando a causar climatologías extremas como pueden ser las glaciaciones. Actualmente, a este proceso se le debe añadir como causa la actividad humana, alterando y agravando los efectos, siendo los principales causantes la contaminación, sobre-población y la deforestación.

Partiendo desde principios de la revolución industrial a mediados del siglo XVIII, el efecto de la actividad humana como causa de estos cambios ha ido en aumento, siendo desde el siglo XIX hasta llegar a pleno siglo XXI la principal causa del cambio climático. Debido al uso extendido de combustibles fósiles tales como el carbón, petróleo y el gas, formando las principales fuentes de energía durante muchos años. Imagen 1.1.

La quema de estos produce los llamados gases de efecto invernadero, principalmente dióxido de carbono y metano. Estos gases impiden la correcta liberación de la radiación irradiada por el suelo al ser calentado por el sol, absorbiendo parte de esta y liberándola nuevamente hacia la tierra, aumentando la temperatura de la superficie terrestre. Como puede verse en la imagen 1.2.

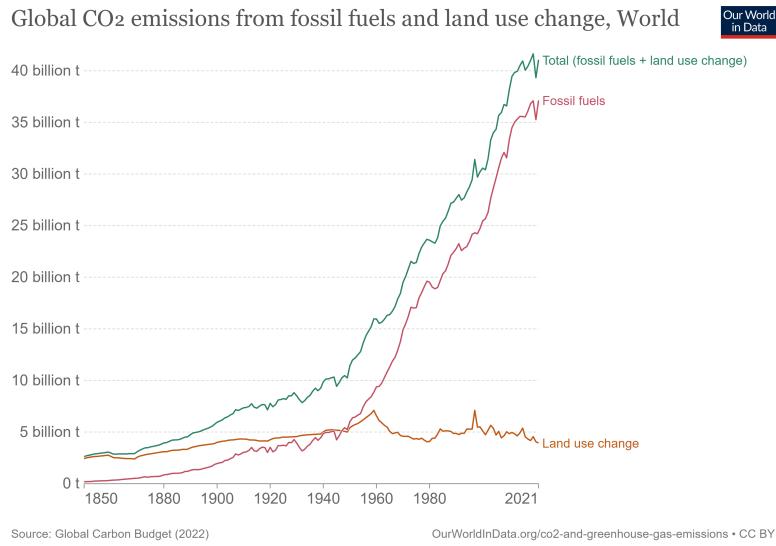


Figura 1.1: Emisiones globales de CO₂ ¹

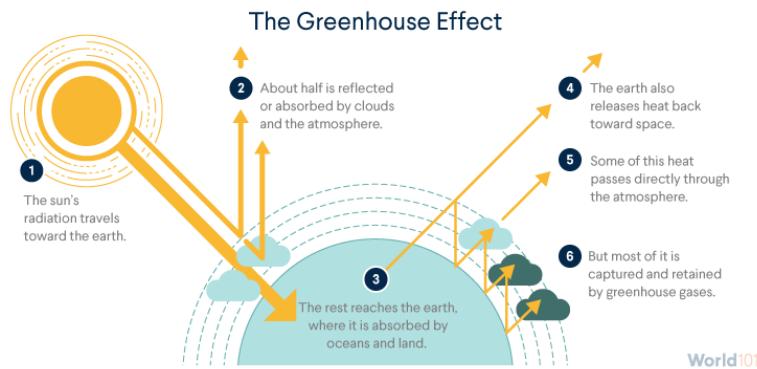


Figura 1.2: Efecto invernadero ²

El aumento de emisiones de estos gases, a fomentado una crecida sustancial en la velocidad de elevación de la temperatura terrestre. Llegando a medirse un aumento regional de, entre 0.8°C hasta 2°C en altas latitudes, estimando un aumento en la media global de 1.1°C en comparación con finales del siglo XIX [2].

Década tras década las temperaturas han ido aumentando, llegando a un punto en el que prácticamente anualmente se batén récords de temperatura máximas por todo el globo. Siendo globalmente el verano de 2023 el más caluroso registrado desde

¹ <https://ourworldindata.org/co2-emissions>

² <https://world101.cfr.org/global-era-issues/climate-change/greenhouse-effect>

1850. Imagen 1.3 [3].

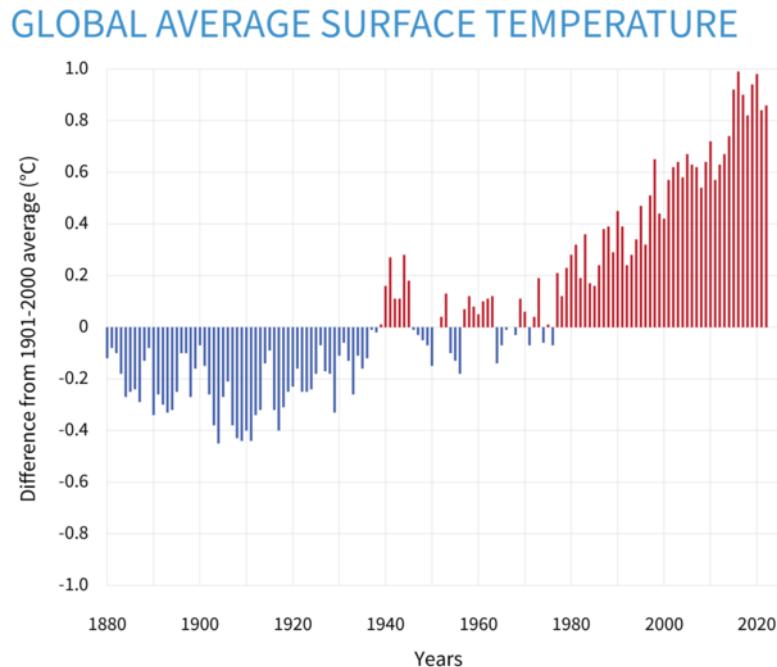


Figura 1.3: Media de aumento de temperatura global entre 1901-2000 ³

Son múltiples los efectos adversos causados por el aumento de la temperatura, entre ellos, la subida del nivel marino, reduciendo e inundando zonas costeras, la desertificación de zonas actualmente áridas y la alteración del comportamiento de especies tanto animales como vegetales, llegando a suponer una reducción sobre la población en multitud de especies [4] [5].

Es por eso que múltiples países son ya los que optan por comprometerse a alcanzar una cota de emisiones cero para 2050, tratando de reducir las emisiones globales a la mitad cara 2030, con el fin de mantener el aumento de la temperatura media por debajo de 1.5°C , estimando esta como un punto reflexivo a la hora de controlar el impacto climático, con el objetivo de mantener un clima habitable.

Pero el problema no reside únicamente en el aumento de la temperatura, como da a entender la definición proporcionada por la ONU, llegando a influenciar sobre los patrones climáticos. Empezando por las estaciones, se ha llegado a observar fluctuaciones en estas, anticipando de la llegada de la primavera, prolongando el verano, retrasando el otoño y reduciendo en la temporada invernal [6].

³<https://www.climate.gov/news-features/understanding-climate/climate-change-global-temperature>

Sumado a esto, son cada vez más los efectos visibles causados sobre los fenómenos naturales. Es tal el impacto que, ya son más de veinte las anomalías climáticas de una importancia significante remarcadas por el centro nacional de información ambiental (NCEI) en 2022 [3]. Entre ellas, sequías debido al aumento de las temperaturas, llegando a fomentar la aparición de incendios y, fuertes lluvias, marcadas por el aumento de tormentas, huracanes y tifones durante las estaciones lluviosas, causando múltiples inundaciones.

El cambio climático agrava los efectos de los fenómenos naturales, reduciendo su periodo de retorno (periodo estimado de años para la aparición de un fenómeno climático), causante de esta meteorología extrema. Zonas planetarias enteras se ven azotadas por graves sequías y tormentas, modificando la disponibilidad del agua sobre la superficie terrestre, cosa que, afecta severamente a la agricultura. La precipitación acumulada se mantiene debajo de la media globalmente, mientras que la frecuencia de aparición de precipitaciones anómalas va en aumento, imagen 1.4, fomentadas por una mayor evaporación de agua a causa del aumento de la temperatura, agravando la frecuencia de tormentas de magnitudes extremas.

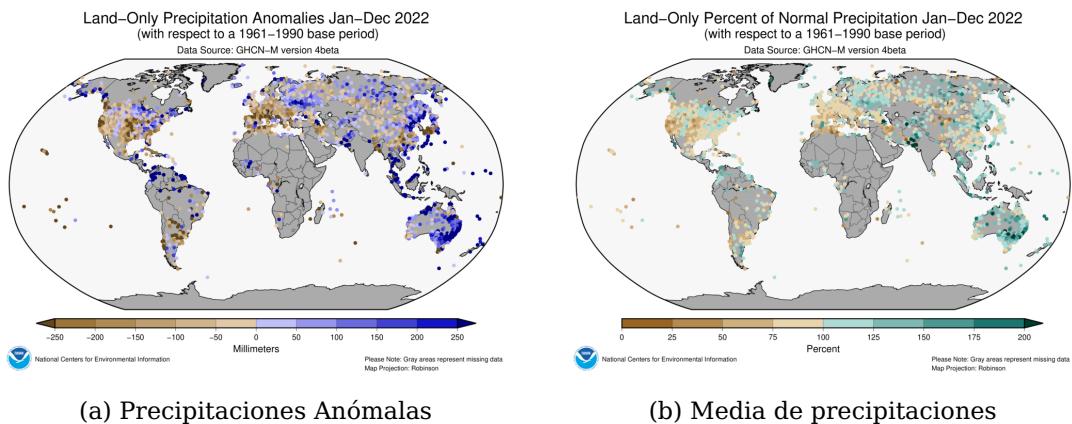


Figura 1.4: Precipitaciones 2022

Estos efectos no tienen por que darse independientemente, sin ir más lejos, en 2022, España batió el récord como tercer año más seco registrado, solo superado por los años 2005 y 2017, más tarde, ese mismo año en diciembre, fuertes lluvias causaron múltiples inundaciones provocando daños materiales sobre viviendas y carreteras [7].

Las inundaciones causadas por lluvias torrenciales son una de las principales causantes de daños materiales a nivel global [8], estimando una perdida anual de 800 millones de euros tan solo en España [9]. Es por eso que, la monitorización y predicción climática son primordiales a la hora de procurar reducir su efecto a nivel económico

como humano.

Los últimos 15-20 años la Unión Europea ha realizado un gran esfuerzo en promover políticas de datos abiertos en realización a la información generada y monitorizada por los estados miembros. El objetivo de esta política es acercar estos datos a la población para tener un mayor control territorial y medio ambiental. El gobierno de Navarra y de España contribuyen a la oferta de datos abiertos publicando mediante diferentes organismos como el geoportal (<https://geoportal.navarra.es/es/iden>) en Navarra o los datos o el sistema automático de información publicada por las diferentes confederaciones hidrográficas en España.

A fin de obtener datos tanto fluviales como pluviométricos, disponiendo de los presentes en las siguientes agencias y organismos.

La Confederación Hidrográfica del Cantábrico (<https://www.chcantabrico.es/>) es un organismo autónomo adscrito al Ministerio para la Transición Ecológica y el Reto Demográfico. Es responsable de la gestión de las cuencas hidrográficas de los ríos que vierten al mar Cantábrico, ejerciendo sobre las comunidades autónomas de: Principado de Asturias, Cantabria, Castilla y León, Galicia, País Vasco y Comunidad Foral de Navarra.

La Agencia Estatal de Meteorología (AEMET) (<https://www.aemet.es/es/portada>), también adscrita al Ministerio para la Transición Ecológica y el Reto Demográfico, tiene como objetivo el desarrollo, implantación, y prestación de los servicios meteorológicos del Estado, apoyando al ejercicio de otras políticas públicas y actividades privadas, contribuyendo a la seguridad de personas y bienes, y al bienestar y desarrollo sostenible de la sociedad.

Meteo Navarra (<http://meteo.navarra.es/>), es la agencia encargada de la monitorización de la meteorología y climatología de Navarra, trabaja junto al gobierno de Navarra, el Ministerio de Agricultura, Pesca y Alimentación (MAPA), el Instituto Navarro de Tecnologías e Infraestructuras Agroalimentarias (INTIA), la Agencia Estatal de Meteorología (AEMET) y la Universidad Pública de Navarra (UPNA).

El agua en Navarra (http://www.navarra.es/home_es/Temas/Medio+Ambiente/Agua/), otra plataforma del gobierno de Navarra, esta vez centrada en los recursos hídricos disponibles en Navarra.

Con el fin de posibilitar la creación un modelo de predicción de inundaciones sobre los ríos en Navarra, el objetivo de este trabajo es crear una plataforma habilitada en la obtención y centralizando de la información proveniente de estas diferentes fuentes,

Imagen 1.5. Mediante esta plataforma se pretende disponer de un conglomerado de datos sobre los que trabajar con el propósito de prevenir sobre inundaciones a la población navarra por medio de un sistema de notificaciones automáticas.

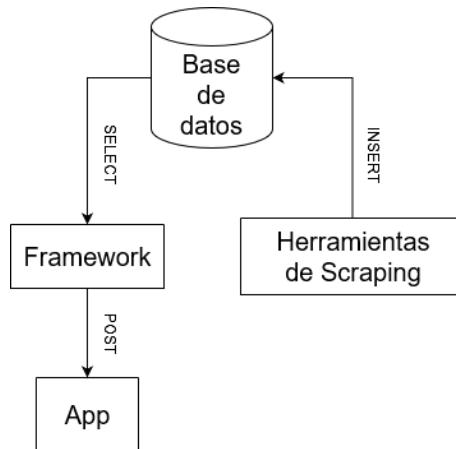


Figura 1.5: Estructura de datos planteada

Este trabajo pretende obtener la mayor cantidad de datos posibles con el fin de predecir dichas inundaciones causadas por las crecidas de los ríos, de modo que las administraciones y la población puedan actuar con tiempo y reducir el efecto devastador y económico producido por estos efectos. Como prueba de concepto trabajaremos a nivel regional en España. En concreto, la Comunidad Foral de Navarra.

1.1 Antigua intro al parecer inutil :v

Los últimos 15-20 años la Unión Europea ha realizado un gran esfuerzo en promover políticas de datos abiertos en realización a la información generada y monitorizada por los estados miembros. El objetivo de esta política es acercar estos datos a la población para tener un mayor control territorial y medio ambiental. El gobierno de Navarra y de España contribuyen a la oferta de datos abiertos publicando mediante diferentes organismos como el geoportal (<https://geoportal.navarra.es/es/idena>) en Navarra o los datos o el sistema automático de información publicada por las diferentes confederaciones hidrográficas en España.

El objetivo de este trabajo es crear una plataforma habilitada en técnicas de scraping, centralizando la información proveniente de diferentes fuentes para una posible futura predicción y aviso de inundaciones mediante los datos pluviométricos obtenidos de los ríos de Navarra.

La plataforma se dividirá en dos apartados, la obtención de los datos y el almacenamiento de estos. Para ello se harán uso de dos máquinas virtuales comunicándose entre si. Aunque hacer uso de una única máquina no solo es viable si no más sencillo, disponer de ellas, no solo aparta la plataforma de un diseño centralizado mas vulnerable, ademas, a nivel de proyecto, proporciona un mayor grado de complejidad, necesitando configurar la comunicación por red de estas.

La primera máquina sera la encargada del almacenamiento de los datos, proporcionando una base de datos en PostGreSQL. Esta recibirá los datos de la segunda máquina, encargada de la obtención de estos.

Como se ha mencionado, la plataforma busca centralizar los datos proporcionados en las distintas webs, tanto fluviales como pluviométricos, creando un punto de acceso global para su posterior uso. Es por eso que la segunda máquina dispone de una plataforma automática de obtención de datos mediante scraping web que, junto a una API en Django posibilita el envío de los datos a la base de datos.

Esta plataforma, realizada usando el Framework de Python Scrapy para el apartado de obtención de datos y, Cron para la automatización, ejecuta los scripts de forma regular en intervalos de tiempo predefinidos para cada web, con el fin de no generar más trafico web del realmente necesario.

En este proyecto se ha pretendido crear una plataforma lo más modular posible, ya sea por el uso de múltiples máquinas virtuales, como por los distintos apartados que la forman e, incluso en la forma de codificarla. Intenta ser lo más intuitiva posible a la hora de poder realizar cambios sobre esta, diferenciando claramente cada elemento.

Centrado en técnicas de scraping web, este proyecto esta limitado por los datos ofrecidos de forma abierta en la web, siendo su punto débil, aquel del cual no disponemos control alguno. A día de hoy, la plataforma dispone de cuatro fuentes distintas de las cuales obtener datos, pero esta comprometida a que se mantengan invariables en el tiempo. Es por esto, que en un proyecto así necesariamente se debe valorar la búsqueda de datos de forma activa. No solo para su mantenimiento a largo plazo sino como forma de extender su alcance.

2

Datos

Navarra dispone de dos confederaciones hidrográficas, la del cantábrico y la del ebro.

Estas son organismos encargados de la gestión de cuencas hidrográficas que discurren por múltiples comunidades autónomas. Tomando como cuenca hidrográfica a la superficie por la cual fluye un conjunto de agua, como ríos, arroyos y lagos hasta su desembocadura en el mar.

La del cantábrico ejerce sobre aquellos ríos cuya desembocadura va a parar al cantábrico, mientras, la del ebro trabaja sobre las zonas limitadas al cauce del ebro.



Figura 2.1: Confederaciones hidrográficas en España

Creadas en 1926, las confederaciones son organismos autónomos, con plena autonomía funcional, adscritas al Ministerio para la Transición Ecológica y el Reto Demográfico. El papel desempeñado por estas recae entre otras cosas en, la planificación hidrológica, gestión de recursos y aprovechamientos, protección del dominio

público hidráulico, control de calidad del agua y los bancos de datos.

Es por eso que, es de gran importancia para este proyecto la adquisición de datos relacionados a estas entidades que ejercen en Navarra.

Para este proyecto se dispone de los siguientes datos:

- Nivel (m)
- Caudal (m^3/s)
- Precipitación (mm)
- Temperatura ($^{\circ}C$)
- Humedad (%)
- Radiación (W/m^2)

A su vez, se dispone de la fecha y hora en la que se hizo la lectura de los datos, junto con los códigos y coordenadas de las estaciones sobre las cuales se obtienen los datos.

2.1 Aemet

Como se ve en la imagen 2.2, dentro de la web podemos encontrar de forma accesible múltiples datos relacionados con la meteorología tomados cada hora, de los cuales seleccionaremos únicamente temperatura, precipitación y humedad, puesto que los datos relacionados con el viento no son tan relevantes para este proyecto. A su vez, no todas las estaciones muestran estos datos, ocurriendo lo mismo con los de presión atmosférica.

Hoy y últimos días. Aranguren, Irun										(i)																		
Datos horarios		Resumen lunes 21		Resúmenes diarios anteriores																								
		Mapa de Comunidad Foral de Navarra		Gráficas		Table																						
Actualizado: lunes, 21 agosto 2023 a las 10:42 hora oficial Ind. climatológico: 9263X - Altitud (m): 572 Latitud: 42° 46' 34" N - Longitud: 1° 31' 57" O - Posición: Ver localización Municipio: Aranguren (Navarra) - Ver predicción																												
Exportar a excel Exportar a csv																												
Fecha y hora oficial Temp. (°C) V. vien. (km/h) Dir. viento Racha (km/h) Dir. racha Prec. (mm) Presión (hPa) Tend. (hPa) Humedad (%)																												
21/08/2023 10:00	22.6	0	-	4	▲	0.0	-	-	-	79.0																		
21/08/2023 09:00	20.2	3	↙	7	↘	0.0	-	-	-	90.0																		
21/08/2023 08:00	19.4	0	-	-	-	0.0	-	-	-	94.0																		
21/08/2023 07:00	19.6	0	-	6	↓	0.0	-	-	-	92.0																		

Figura 2.2: Página Datos Aemet

Los datos en HTML vienen datos dentro de un elemento tabla, imagen 2.3, lo que facilita su adquisición, pues permite tomar todas las filas (elementos tr) pertenecientes al elemento tbody de esta. Una vez obtenidas las filas, la forma de lograr los datos deseados sería elegir dentro de cada elemento tr los elementos td deseados, puesto que HTML empieza a contar elementos desde el uno (en vez de cero como suele ser común en lenguajes de programación), los td a obtener serían: uno, fecha y hora; dos, temperatura; siete, precipitación; diez, humedad.

```
<table id="table" class="width100 tabla_dinamica tabla_responsive" cellspacing="0" summary="Últimos datos meteorológicos observados por cada ciudad de la comunidad autónoma seleccionada">
  <thead>(柱)</thead>
  <tbody>
    <tr id="" class="fila_par" onmouseover="sorter.hover(0,1)" onmouseout="sorter.hover(0,0)"> (event)
      <td class="evidselected">21/08/2023 10:00</td>
      <td class=">22.6</td>
      <td class=">0</td>
      <td class=">0</td>
      <td class=">0</td>
      <td class=">0.0</td>
      <td class=">[espacio en blanco]</td>
      <td class=">[espacio en blanco]</td>
      <td class=">-79.0</td>
    </tr>
    <tr id="" class="fila_impac" onmouseover="sorter.hover(1,1)" onmouseout="sorter.hover(1,0)"> (柱)</tr> (event)
```

Figura 2.3: HTML Tabla Datos Aemet

Las coordenadas, dentro de un span, imagen 2.4, están incluidas en elementos abbr con sus respectivas clases "latitude" y "longitude", haciendo posible su obtención fácilmente mediante los elementos abbr.

```
<span class="geo">
  <span class="font_bold">Latitud</span>
  :
  <abbr class="latitude" title="-42.7761111111">42° 46' 34'' N</abbr>
  -
  <span class="font_bold">Longitud</span>
  :
  <abbr class="longitude" title="1.5325000000">1° 31' 57'' O</abbr>
  -
</span>
```

Figura 2.4: HTML de las coordenadas en Aemet

2.2 CHCantábrico

En la sección de nivel del río de la pagina de CHCantábrico, se encuentra la tabla con las estaciones, imagen 2.5a. Una peculiaridad de esta, es que aun seguir una estructuración típica de tabla mediante el uso de table, dispone de una tabla adicional en cada una de las filas. Imagen 2.5b.



Sistema	Código	Río	Estación	Valor actual (m)	Tendencia	Actualizaciones	Umbrales (m)	Gráfica
	1424	Eo	Ribera de Pajón	0.12	↓	15/08/2022 19:00 Seguimiento Prealerta Alerta	2.10 2.60 3.00	
	1426	Eo	Pontedeve	1.31	→	15/08/2022 19:45 Seguimiento Prealerta Alerta	3.20 4.00 4.70	

```
<table class="tablefixedheader niveles">
  <thead>...</thead>
  <tbody>
    <tr class="centrado">
      <td class="sistema" rowspan="5">...</td>
      <td class="codigo">1424</td>
      <td class="verde">...</td>
      <td class="verde">...</td>
      <td class="valor">...</td>
      <td class="tendencia">...</td>
      <td class="actualizacion">...</td>
    </tr>
    <td>
      <table class="umbrales_gr">
        <tbody>...</tbody>
      </table>
    </td>
    <td>...</td>
  </tr>
```

(a) Tabla estaciones

(b) HTML de la tabla de estaciones

Figura 2.5: Página Nivel de los ríos CHCantábrico

A diferencia del resto de las páginas visitadas, CHCantábrico no muestra los datos sobre la web, por el contrario, implementa un botón con el que descargarlos directamente en formato CSV. Más adelante en el documento se discutirá la repercusión de ello.

Los datos que si se pueden obtener, son aquellos presentes en la tabla secundaria de la fila, los datos de pre-alerta, alerta y seguimiento para cada una de las estaciones, siendo datos valiosos a la hora de intentar anticipar inundaciones.

Las coordenadas a su vez, tampoco vienen dados sobre la web, por lo que hace falta visitar la web del Centro de Estudios Hidrológicos (<https://ceh.cedex.es/>) para obtenerlas.

2.3 MeteoNavarra

En la página de meteorología y climatología de Navarra encontramos una gran sección de estaciones de las cuales poder obtener datos. Imagen 2.6.

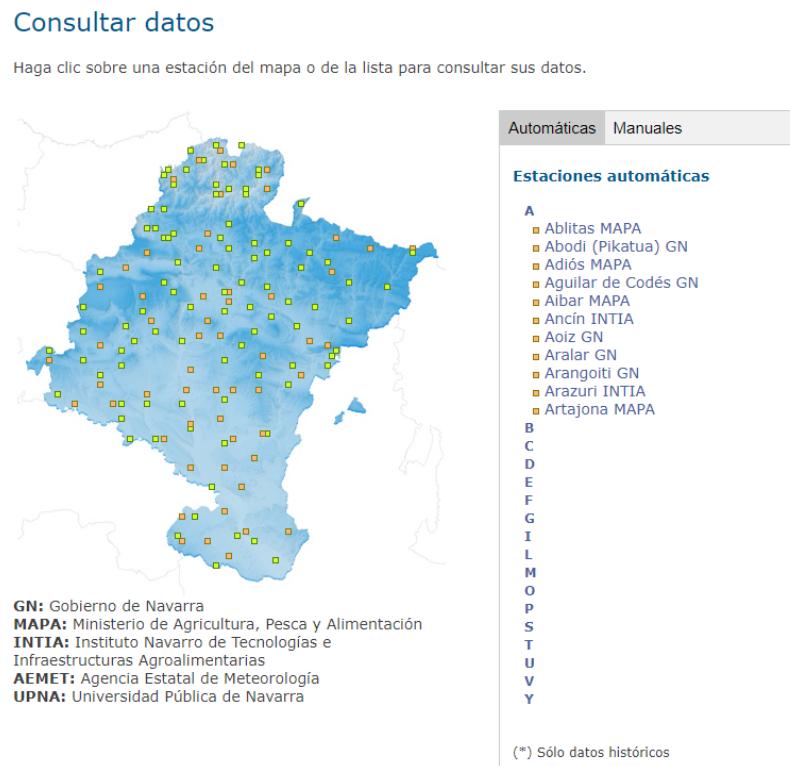


Figura 2.6: Página estaciones MeteoNavarra

De entre los dos tipos e estaciones disponibles, automática y manual, únicamente se va a trabajar con los datos de las estaciones automáticas. El motivo de esto es que las manuales solo proveen datos diarios de, la temperatura máxima, mínima y de la precipitación acumulada. Esto hace que no se disponga de ningún dato hasta que finalice el día, cosa que no es viable si lo que se propone es predecir cambios radicales en un periodo de tiempo reducido.

Debido a la estructuración HTML usada para mostrar las estaciones, imagen 2.7a, se puede ver que aun usar el elemento table, este solo dispone de una única fila y columna, haciendo de la columna un elemento div sobre el que insertar los datos, imagen 2.7b. Como es el caso de la web de Agua en Navarra.

```
<table width="260" border="0" cellspacing="0">
  <tbody>
    <tr>
      <td colspan="2" height="470" valign="top">[REDACTED]</td>
    </tr>
  </tbody>
```

(a) Tabla

(b) Filas

Figura 2.7: HTML tabla estaciones MeteoNavarra

Las estaciones automáticas proporcionan tanto datos en períodos de diez minutos como datos diarios. Tras usar el mismo razonamiento que con las estaciones manuales, no tomamos los datos diarios y, entre los actualizados cada diez minutos, se toman la temperatura, humedad relativa, radiación global y precipitación. El resto se omitirán. Imagen 2.8.

1. Parámetros

Parámetros 10 minutos <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Temperatura <input checked="" type="checkbox"/> Humedad relativa <input checked="" type="checkbox"/> Radiación global <input type="checkbox"/> Insolación <input checked="" type="checkbox"/> Precipitación <input type="checkbox"/> Velocidad viento 10 m <input type="checkbox"/> Dirección viento 10 m <input type="checkbox"/> Velocidad racha máxima 10 m <input type="checkbox"/> Dirección racha máxima 10 m <input type="checkbox"/> Velocidad viento 2 m <input type="checkbox"/> Dirección viento 2 m <input type="checkbox"/> Velocidad racha máxima 2 m <input type="checkbox"/> Dirección racha máxima 2 m <input type="checkbox"/> Desviación dirección viento 10 m <input type="checkbox"/> Desviación dirección viento 2 m <input type="checkbox"/> Desviación velocidad 10 m <input type="checkbox"/> Desviación velocidad 2 m <input type="checkbox"/> Humección hoja (resistencia) <input type="checkbox"/> Radiación Solar PAR acumulada 10min <input type="checkbox"/> Temperatura suelo 	Parámetros Diarios <ul style="list-style-type: none"> <input type="checkbox"/> Temperatura media <input type="checkbox"/> Temperatura máxima <input type="checkbox"/> Temperatura mínima <input type="checkbox"/> Humedad relativa med. <input type="checkbox"/> Humedad relativa máx. <input type="checkbox"/> Humedad relativa mín. <input type="checkbox"/> Precipitación acumulada <input type="checkbox"/> Radiación global <input type="checkbox"/> Insolación total <input type="checkbox"/> Velocidad media viento 10 m <input type="checkbox"/> Dirección viento 10 m (MODA) <input type="checkbox"/> Velocidad racha máx 10 m <input type="checkbox"/> Dirección racha máx 10 m <input type="checkbox"/> Velocidad media viento 2 m <input type="checkbox"/> Dirección viento 2 m (MODA) <input type="checkbox"/> Velocidad racha máx 2 m <input type="checkbox"/> Dirección racha máx 2 m <input type="checkbox"/> Radiación PAR diaria acumulada <input type="checkbox"/> Temperatura media suelo
---	---

Datos en **horario solar**.

2. Fechas

<input type="radio"/> hoy <input type="radio"/> ayer	Desde <input type="text"/>
	Hasta (excluido) <input type="text"/>

Figura 2.8: Datos MeteoNavarra

Los datos se muestran en formato tabla, tanto visualmente, en la imagen 2.9a como a nivel de HTML, imagen 2.9b, cosa que facilita su obtención.

Estación de Ablitas MAPA
Altitud 338 m.

Datos desde el 20/08/2023 hasta el 21/08/2023 en **horario solar**
Los datos son **PROVISIONALES**

Fecha	Temperatura	Humedad relativa	Radiación global		Precipitación
			°C	%	
20/08/2023 0:00	23.6	55	0.0	0.0	
20/08/2023 0:30	22.9	52	0.0	0.0	
20/08/2023 1:00	22.9	56	0.0	0.0	
20/08/2023 1:30	22.9	62	0.0	0.0	

(a) Tabla datos

(b) HTML de la tabla de datos

Figura 2.9: Página de datos de MeteoNavarra

2.4 El Agua en Navarra

En esta página encontraremos los datos tanto del nivel de los ríos como de su caudal en los últimos 15 días en períodos de 10 minutos, siendo una de las fuentes principales de datos.

La sección de aforos en la página, imagen 2.10, muestra el mapa de Navarra junto a varias estaciones. Aunque no todas las disponibles en la web, conformando únicamente el grupo de estaciones principales.

Con el fin de acceder a todas las estaciones disponibles, debemos centrarnos en el mapa de la esquina inferior derecha. Estructurada en 6 regiones, Norte, Arga, Ega, Ebro alto, Ebro bajo y Aragón, por medio de este mapa accedemos a cada región, mostrando el mapa de la región, dando acceso a todas las estaciones en la zona.

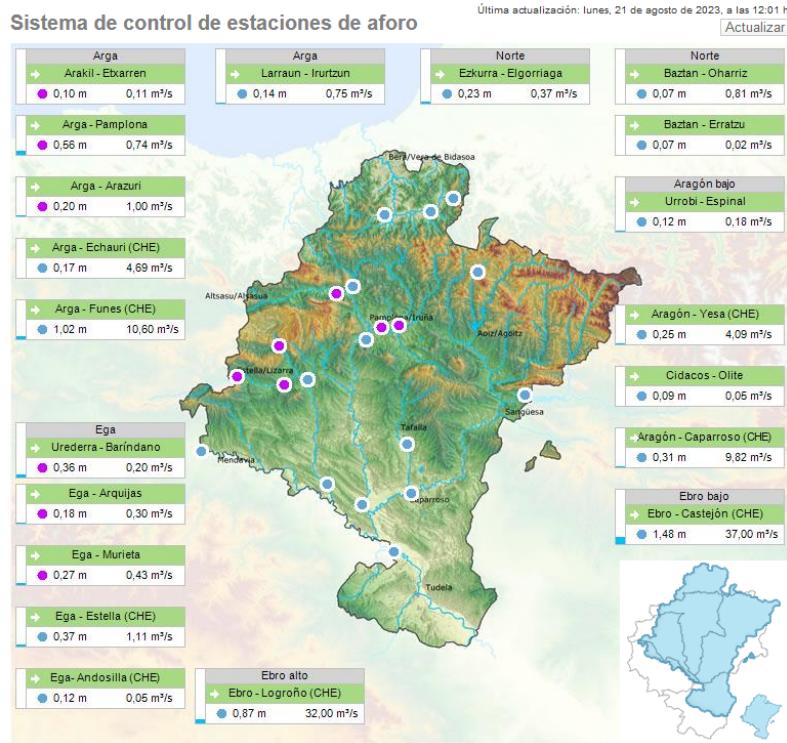


Figura 2.10: Página El Agua en Navarra

El HTML del mapa se estructura de la siguiente manera, imagen 2.11, mostrando un par de elementos area por estación, uno con shape "rect", representando las tarjetas que rodean el mapa y otro con shape "circle", siendo los puntos en el mapa. Pulsar sobre cualquiera de ellos es equivalente, aunque posteriormente hagamos uso de los

elementos "rect" dentro del código.

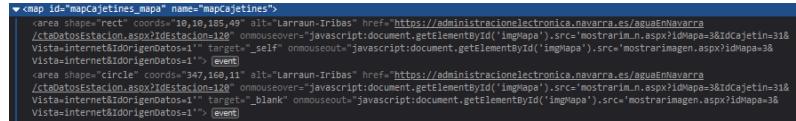


Figura 2.11: HTML mapa estaciones El Agua en Navarra

Las páginas de cada estación se muestran como en la imagen 2.12.



Figura 2.12: Página estación El Agua en Navarra

De ella obtenemos los próximos datos, descripción, municipio, río y coordenadas. Ademas de darnos acceso a los datos de nivel y caudal del río. Mostrados dentro de el div con id "blog_icons", imagen 2.13. Una vez dentro del div todos los datos siguen la misma estructuración, //div/span/span, permitiendo la adquisición de todos los elementos a la vez.

```
<div id="bloq_iconos">
  <div id="descripcion">
    Descripción
    <br>
    <span class="tit_icon">
      <span id="lblDescripcion">Río Salado en Estenoz</span>
    </span>
  </div>
  <div id="municipio">
    Municipio
    <br>
    <span class="tit_icon">
      <span id="lblMunicipio">GUESALAZ</span>
    </span>
  </div>
  <div id="rio">
    Río
    <br>
    <span class="tit_icon">
      <span id="lblRio">Salado</span>
    </span>
  </div>
  <div id="coordenadas_UTM">
    Coordenadas UTM (EPSG:25830)
    <br>
    <span class="tit_icon">
      <span id="lblUTM">X. 587899,4 | Y. 4733313 | Z. 480</span>
    </span>
  </div>
</div>
```

Figura 2.13: HTML estación El Agua en Navarra

Una vez se accede a los datos de la estación, se mostrara una gráfica como la de la imagen 2.14. A su vez, mediante el botón "Datos Numéricos", tendremos la posibilidad de observar los datos en forma numérica. Pero no sin antes haber visitado la versión gráfica. Pues da el error de la imagen 2.15.

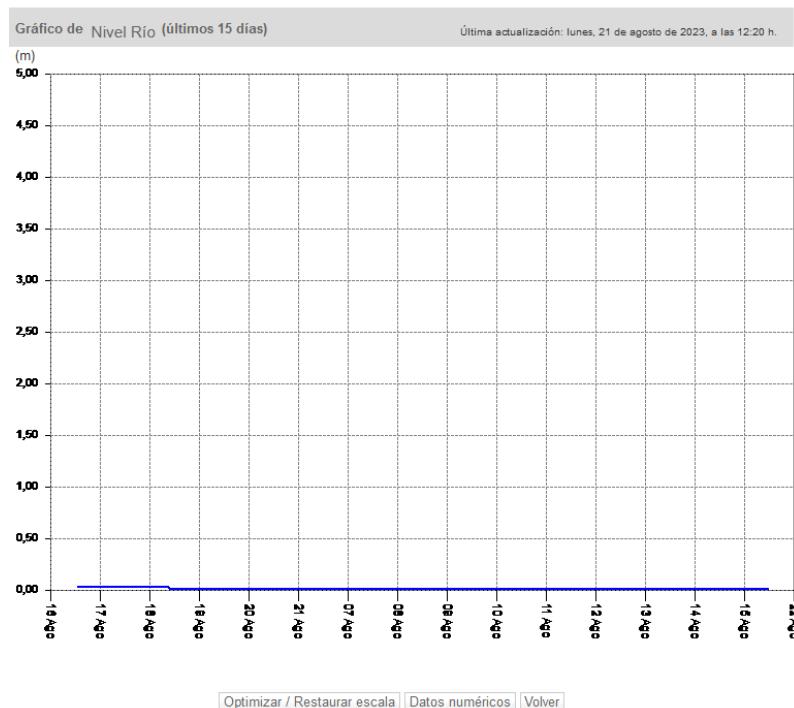


Figura 2.14: Gráfica datos estación El Agua en Navarra



Figura 2.15: Error datos numéricos en El Agua en Navarra

Los datos numéricos están presentados en formato tabla como se aprecia en la imagen 2.16a. Por el contrario, tras observar el HTML, realmente es un elemento div, que engloba los conjuntos de elementos span que representan las líneas, separados por elementos br, imagen 2.16b. El formato en el que se presentan los datos, no hace más que representar una mayor complejidad para, posteriormente, trabajar y adquirir los datos.



(a) Formato presentación de los datos

(b) HTML de los datos

Figura 2.16: Datos numéricos de estaciones en Agua en Navarra

Para este proyecto, es necesario el uso de dos servidores, un servidor de base de datos y otro para el despliegue de una API y la obtencion de datos.

Primero que todo es necesario un sistema operativo capaz de servir este propósito.

3.1 Sistema Operativo Debian

3.1.1 Historia

En 1993, tras varios intento fallidos por distintas empresas de solucionar el problema, Ian Murdock, por aquel entonces estudiante de la Universidad Purdue, encontró la solución al problema basándose en el reciente proyecto de Linus Torvalds, el kernel Linux. tras el anuncio de Ian para crear un sistema operativo de forma descentralizada en paralelo como es el caso del kernel Linux, docenas de usuarios se unieron para formar el proyecto Debian Linux con la intención de crear un sistema operativo de gran calidad y mantenimiento, publicando en enero de 1994 la primera versión de Debian 0.91. [13] [15]

3.1.2 Filosofía

Debian no intenta seguir ni competir con los líderes del sector, por el contrario, desde sus inicios el proyecto se ha basado en una filosofía centrada en la robustez y estabilidad del sistema guiada por unos estrictos estándares de calidad, actualizándose conforme las necesidades de sus usuarios a la vez que promueve el software gratuito, lo que le ha ayudado a obtener fama entre los usuarios. [16] [17]

A su vez, debido al apoyo del software libre, hace uso de múltiples licencias de software como la Licencia Pública General GNU (GPL), licencias artísticas o del tipo BSD,

lo cual ha llevado al desarrollo de las Directrices de Software Libre de Debian (DFSG) con el fin de definir la construcción del software libre. [18]

Definido por estas licencias, el software libre debe cumplir al menos las que están consideradas las cuatro libertades esenciales: [19]

- Ejecutar el programa como se deseé, con cualquier propósito.
- Estudiar cómo funciona el programa, y cambiarlo para que haga lo que se deseé.
- Redistribuir copias para ayudar a otros.
- Distribuir copias de sus versiones modificadas a terceros permitiendo ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones.

3.1.3 Por qué Debian?

El mercado está repleto de distintas posibles alternativas a Debian, ya sean de pago, Windows Server OS o Red Hat Enterprise Linux (RHEL), gratuitos, Ubuntu Server y Fedora Server, o incluso en la nube, Amazon Web Services (AWS), Google's Cloud Platform.

Descartando todo sistema de pago, aunque las posibilidades se reducen, aún hay múltiples opciones sobre las que elegir, pero son pocas las que ofrecen la misma usabilidad que Debian con sus más de 59000 paquetes en su versión estable. [23]

Aunque todos los sistemas basados en Linux disponen de las mismas características, siendo software libre y gratuito con soporte multi-usuario, multi-proceso y uso en tiempo real, Debian siendo uno de los sistemas más longevos del mercado a tomado fama entre la competencia por su seguridad y estabilidad. Siendo la base para muchas de las distribuciones más populares contra las que compite, como Ubuntu, Knoppix, PureOS o Tails.

Cabe mencionar, que parte de esta seguridad y estabilidad puede llegar a ser un limitante a la hora de elegir Debian como sistema operativo dependiendo de tus necesidades a nivel de uso pues el software compatible igual no es la versión más reciente.

Finalmente, una característica distintiva de Debian frente a la competencia gratuita es su compatibilidad con un uso 24/7, pues otras alternativas gratuitas o no dan soporte a esta característica, Fedora Server o, necesitas disponer de una licencia de pago como es el caso de Ubuntu Server mediante Ubuntu Pro.

Puesto que no afecta negativamente al proyecto no disponer de las versiones más

recientes de software y debido a la necesidad de un sistema 24/7 gratuito, parece lógica la elección de Debian como sistema operativo para usar en el servidor.

Una vez seleccionado sistema operativo, hace falta seleccionar un Framework para crear y trabajar con una API.

3.2 Framework

3.2.1 Qué es un framework?

Un framework es software que provee una infraestructura básica sobre la que desarrollar tus proyectos. Aportan las funcionalidades y estructuras básicas necesarias para este sin la necesidad de programar todo desde cero, ahorrando tiempo de desarrollo y aportando robustez al proyecto. [24]

Cada framework aportará su propia colección de módulos y paquetes específicos para ayudar en el desarrollo, es por esto que generalmente se clasifican en tres clases distintas según funcionalidades. [25]

Tipos de frameworks

Full Stack

Un framework full-stack es apto tanto para desarrollo back-end como front-end, aportando todas las herramientas posibles que ayuden con el desarrollo gráfico de la interfaz de usuario (UI), gestión de bases de datos, protocolos de seguridad y lógica de negocio entre tantos. Siendo Django un ejemplo de framework full-stack.

Micro

Los framework micro son ligeros por definición, siendo en cierta medida lo contrario de un framework full-stack, pues aunque los componentes que aportan como puede ser la gestión de bases de datos son los mismos, estos no vienen incluidos de forma nativa. Esto se debe a que buscan aportar flexibilidad y libertad a los desarrolladores para que incluyan únicamente aquellas herramientas que necesiten.

Como se explica en la documentación de Flask, uno de los framework tipo micro más relevante, el 'micro' de microframework significa que el núcleo del framework es simple pero extensible.

Asíncrono

Estos framework están dirigidos por eventos. en vez de hacer un manejo operacional

linea a linea de las funciones en la que se van ejecutando una detrás de otra, el código asíncrono no es bloqueante por lo que no se espera que un evento termine para ejecutar el siguiente, ejecutándolo de forma simultanea. Debido a esto un framework asíncrono puede llegar a conseguir un gran rendimiento si se usa en un servidor que lo permita.

3.2.2 Librería vs Framework

Aunque ambas ofrecen funcionalidades operacionales, su mayor diferencia radica en la especificidad y complejidad de estas.

Las librerías están compuestas por múltiples métodos para un uso específico sin aportar mucha complejidad, realizando una tarea por función.

Por el contrario, los framework tienen en cuenta las posibles necesidades de tu proyecto, pudiéndose permitir ser aún más específicos, ofreciendo la arquitectura y comportamiento básico de la aplicación, sin comprometer la flexibilidad de desarrollar las funcionalidades necesarias para su funcionamiento, aportando herramientas sobre las que trabajar. [26]

3.2.3 Django

Qué es Django?

Django es un framework web de tipo full-stack gratuito y de código abierto para Python. Sigue el principio DRY "Don't Repeat Yourself" por lo que se enfoca en el menor uso de código, el desarrollo rápido y la reutilización de componentes. [27] [28]

Hace uso de su auto denominado patrón Modelo-Vista-Template (MVT) 3.1 una variante del conocido Modelo-Vista-Controlador (MVC). [29]

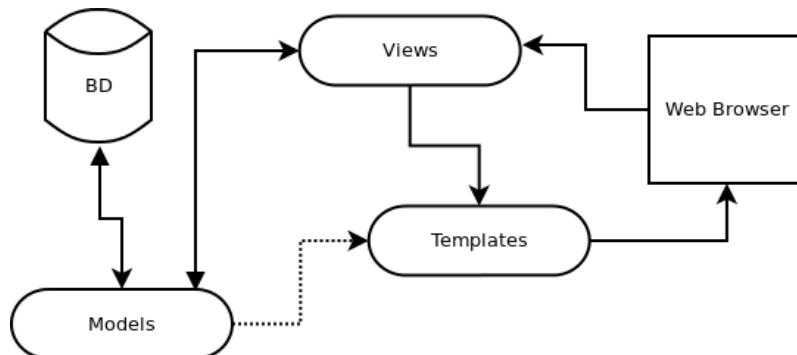


Figura 3.1: Diagrama patrón MVT ¹

Para poder trabajar con bases de datos relacionales tales como, Oracle, MySQL y PostgreSQL, Django usa un Mapeador Relacional de Objetos (ORM) que permite interactuar con ellas mediante SQL.

Por qué Django?

Django es uno de los frameworks más reputados en Python, teniendo en cuenta su naturaleza gratuita y de código abierto. Siendo usado tanto por la comunidad como por empresas tales como Instagram, Mozilla y Facebook. Aunque no por ello significa que no disponga de poca competencia, siendo TurboGears, web2py, Bottle y Flask de las más notables.

El que sea un framework full-stack no va a aportar la flexibilidad, libertad y ligereza que da el uso de un microframework, sobre todo a la hora de agregar únicamente aquellas herramientas que considere necesarias, pero si que atenuará la carga de trabajo que puede suponer un microframework si no se dispone de la experiencia necesaria para usarlo.

Otra característica por la que decantarse por Django, aunque no sea única de él, es su compatibilidad con bases de datos relacionales de forma nativa, cosa que facilitará el uso de estas en el proyecto.

A su vez, con el fin de obtener los datos, se necesita de una herramienta centrada en el scraping de datos.

3.3 Herramienta para Web Scraping

3.3.1 Qué es el Web Scraping?

Web scraping, también conocido como web extraction o web harvesting, es una técnica de extracción de datos desestructurados de la World Wide web (WWW) y guardarlos de forma estructurada en una base de datos o en un sistema de ficheros en formato XML, JSON o CSV para su posterior recuperación o análisis. Generalmente, los datos web son adquiridos mediante el uso de Hyper-text Transfer Protocol (HTTP) o a través de un navegador web, ya sea de forma manual o automática mediante web crawlers, herramientas diseñadas con este propósito, siendo capaces de convertir páginas web enteras en información bien estructurada. [30] [31]

Debido a la gran cantidad de datos que son generados constantemente en la WWW, web scraping es considerado una forma eficiente y poderosa de amasar big data.

¹<https://docs.hektorprofe.net/django/web-personal/patron-mvt-modelo-vista-template/>

Web scraping puede ser usado en una gran variedad de entornos, como la recolección de comentarios en redes sociales, listado de la propiedad inmobiliaria o en este caso el monitoreo y comparación de los niveles de los ríos y datos pluviométricos.

3.3.2 Procedimiento básico en Web Scraping

El proceso de recolección de datos de Internet se puede dividir en tres procesos secuenciales, el estudio de la página sobre la que trabajar, adquirir los recursos web y, luego, organizar la información deseada. [3.2](#)

El primer paso consiste en mirar la estructuración de la web para seleccionar aquellos datos que queramos extraer, comprobando los recursos HTML, CSS y viendo si hace uso de JavaScript. Para realizar el segundo paso de adquisición de los recursos, el proceso empieza con los programas de web scraping enviando un request, ya sea mediante GET o POST, a la página web deseada. Una vez el request sea recibido y procesado por la web, esta enviará los recursos solicitados al programa. Después, pasariamos al tercer paso, en el que analizaríamos los recursos obtenidos y los filtraríamos de tal forma que nos quedásemos únicamente con la información que nos sea necesaria. Finalmente almacenaríamos la información obtenida para su posterior análisis.

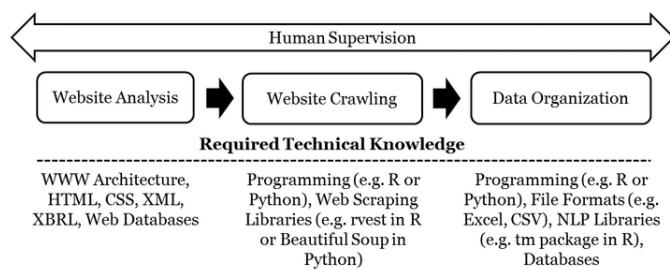


Figura 3.2: Web Scraping [2](#)

3.3.3 Scrapy

Qué es Scrapy?

Scrapy es un framework asíncrono para web crawling y web scraping gratuito y de código abierto. Por defecto proporciona todas las herramientas necesarias para realizar la tarea de extracción, procesado y estructurado de los datos adquiridos, ade-

²https://www.researchgate.net/figure/Web-Scraping-Adapted-from-Krotov-and-Tennyson-2018_fig1_324907302

mas de dar la opción de extender funcionalidades en caso necesario, haciéndolo extremadamente versátil. [32]

Pensado para navegar entre webs y extraer información de forma estructurada de ellas, Scrapy se usa para múltiples ámbitos, por ejemplo, el minado de datos o la monitorización y análisis de datos.

Por qué Scrapy?

A la hora de buscar herramientas para web scraping en Python, fueron tres las alternativas principales encontradas, BeautifulSoup, Selenium y Scrapy.

La primera es una librería de parseo de HTML y XML, que, aunque podría haber servido para cumplir el propósito del proyecto inicialmente, a la larga su sencillez hubiera sido más un problema que una ayuda.

La segunda por el contrario, no es una herramienta de web scraping como tal y, se centra en la automatización de la navegación web como entorno de pruebas. Debido a esto, no es una herramienta que haya usado para el web scraping, pero si que ha sido usada junto con Scrapy para navegar entre webs y sacar los datos de estas. Proporcionando la posibilidad de hacer uso de JavaScript sobre las páginas, pues Scrapy no dispone de renderizado JavaScript de forma nativa.

Scrapy fue elegida gracias a la flexibilidad y versatilidad que proporciona a la hora de trabajar, pudiendo crear proyectos sencillos en cuestión de minutos con las herramientas base proporcionadas o investigar como trabajar con estas herramientas y crear proyectos complejos que satisfagan tus necesidades de la manera que deseas. Esto implica que la curva de aprendizaje de Scrapy sea mayor que la que puede tener BeautifulSoup, sobre todo al inicio, llegando a parecer abrumador. A su vez, la naturaleza de Scrapy le hace tener el mejor rendimiento de entre las tres. [26]

Finalmente, cabe mencionar que Scrapy no dispone de rotación de IP ni geolocalización como pueden tener las alternativas de pago, cosa que no es un impedimento para llevar a cabo el proyecto.

El hecho de seleccionar estos Framework, impone la necesidad de usar como lenguaje de programación Python.

3.4 Lenguaje de Programación Python

3.4.1 Qué es Python?

Siendo su primera aparición en el año 1991 por manos de Guido van Rossum, Python es un lenguaje de programación de alto nivel interpretado que prima la legibilidad del código, siendo este a veces nominado como "seudocódigo ejecutable". [10]

Python a su vez es un lenguaje multiplataforma, fuertemente tipado, dinámico y multiparadigma, pues soporta programación orientada a objetos, imperativa y funcional. [11] [12]

3.4.2 Por qué Python?

La elección de Python sobre otros lenguajes se basa en su sintaxis sencilla y clara que facilita la programación, junto a la gran cantidad de librerías y frameworks potentes de los que dispone para satisfacer las necesidades que presenta este proyecto, como pueden ser la extracción de datos meteorológicos mediante web scraping haciendo uso de Scrapy y la creación de una API con Django.

Para satisfacer el segundo servicio necesario se requerirá de una base de datos.

3.5 Base de Datos

Debido a la naturaleza de los JSON obtenidos, se pueden trasladar fácilmente a tablas relacionales, por lo que se hará uso de una base de datos relacional.

3.5.1 PostGreSQL

PostGreSQL, es un sistema de gestión de bases de datos relacional de código abierto. PostgreSQL destaca por su sistema de gestión de bases de datos, su soporte para consultas complejas, transacciones ACID, integridad referencial y escalabilidad.

Además, ofrece una amplia gama de tipos de datos, incluyendo geoespaciales y JSONB, almacenando JSONs de forma binaria (de ahí la B) para su fácil acceso.

La comunidad activa detrás de PostgreSQL contribuye continuamente con mejoras y extensiones, lo que lo convierte en una solución versátil y confiable. Lo que la hace popular tanto para aplicaciones empresariales como para proyectos web.

3.5.2 Por qué PostGreSQL?

Entre las bases de datos con soporte oficial en Django se encuentran, PostgreSQL, MariaDB, MySQL, Oracle y SQLite. A excepción de Oracle, la única base de datos comercial, el resto son de código abierto, aunque puedan disponer de versiones de pago.

A nivel de funcionalidades, todas ofrecen implementadas de una forma u otra una misma gama de estas. Es por eso que, la elección se hizo en base a la flexibilidad y robustez de estas. Lo que hace a PostgreSQL una opción interesante en proyectos que lleguen a requerir de datos geoespaciales y JSON.

4

Diseño de la Plataforma

4.1 Estructuras Planteadas

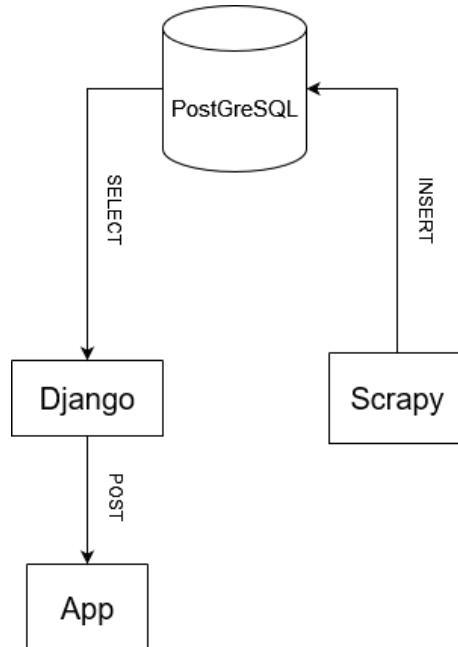


Figura 4.1: Estructura descartada

4.1.1 Problema

Aunque sea la estructura más simple y fácil de implementar, pues scrapy mismo puede insertar los datos scrapeados de las web de forma directa en PostGreSQL, Djando hace uso de un sistema de gestión de versiones de los cambios realizados en la BBDD con el fin de reducir la carga de peticiones a la BBDD, esto se aplica desde el lado

de Django, lo que supondría un problema a la hora de insertar datos directamente de Scrapy a PostGreSQL, pues los nuevos datos podrían no llegar a ser detectados por Django, generando la situación de que una vez se vayan a pedir los nuevos datos Django no devuelva nada pues desde su punto de vista los datos que ya dispongo son la versión mas reciente, luego no es necesario realizar llamada alguna a la BBDD.

4.1.2 Solución

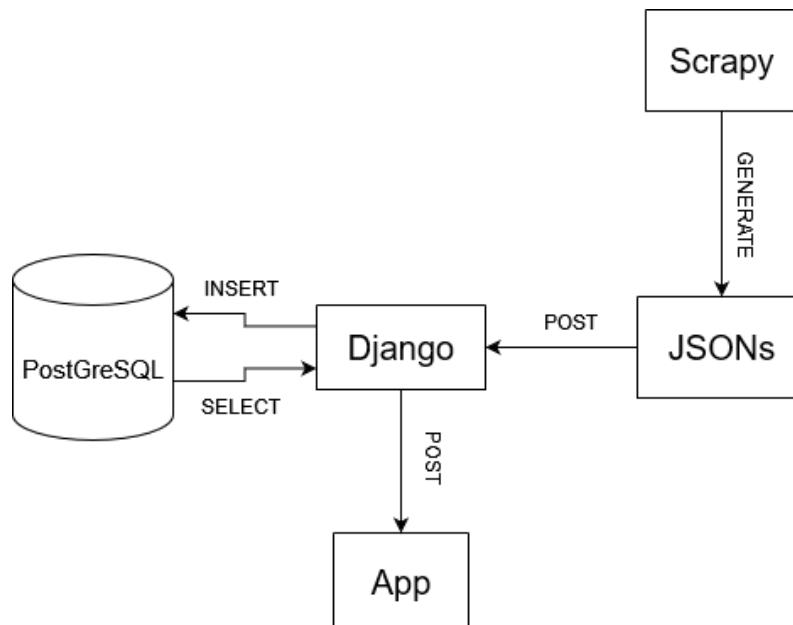


Figura 4.2: Estructura usada

Pasa solventar el problema se ha incluido un paso intermedio en el que inicialmente almacena los datos obtenidos de las distintas webs en formato JSON, para posteriormente enviárselos a Django, el cual se encargara de subir los datos a la BBDD, mientras, el resto de la arquitectura seria idéntico a la versión original.

4.2 Trabajar con los Datos

4.2.1 Flujo de Datos

- 1.- Pedir datos (JSON1)
- 2.- Formatear datos en un nuevo JSON (JSON1 -> JSON2)
- 3.- Mandar datos a la BBDD (JSON2)
- 4.- Marcar JSON como old (JSON2 -> JSON3)

5.- Borrar JSON original (JSON1)

While(true)

1.- Pedir datos (JSON1)

2.- Formatear datos en un nuevo JSON (JSON1 -> JSON2)

6.- Crear JSON con las nuevas fechas (JSON2 != JSON3 y fecha de hoy -> JSON4)

7.- Mandar datos a la BBDD (JSON4)

4.- Marcar JSON como old (JSON2 -> JSON3)

5.- Borrar JSON original (JSON1)

8.- Borrar JSON viejo (JSON3)

9.- Borrar JSON diferencias (JSON4)

Puesto que los datos obtenidos no son los mismos para cada web, antes de ser enviados a Django deben ser parseados para compartir una estructura heterogénea, una vez obtenido el JSON parseado, en caso de ser la primera iteración se mandaran directamente los datos a la BBDD, de no ser la primera iteración, se tomaran únicamente los nuevos datos obtenidos como datos a enviar, una vez enviados los datos el JSON parseado sera marcado como old (viejo) para ser el punto de comparación respecto a los datos mas recientes que obtendremos de las webs.

4.2.2 Datos obtenidos por página

Aemet:

temperatura, humedad, precipitación

meteoNavarra:

temperatura, humedad, precipitación, radiación

aguaEnNavarra:

nivel, caudal

chcantabrico:

nivel, precipitación, seguimiento, alerta, pre-alerta

En todas las webs se proporciona las coordenadas junto con la fecha y hora en la que se ha hecho la medida.

4.2.3 Formato de Datos

No todas las webs presentan sus datos de la misma manera, es por eso que nos encontramos con que los datos que hemos obtenido pueden llegar a estar repartidos

en distintas páginas, haciendo necesario el uso de múltiples Spiders, resultando en multiples JSON.

Debido a ello, para cada web se ha creado una función para parsear (formatear) los datos obtenidos, de esta manera se dispone de una estructura única para los datos recibidos, haciendo su uso posterior más fácil, ya sea a la hora de tratarlos como para almacenarlos en la base de datos.

Esquema obtenido:

```

1 [
2 {
3     "coordenadas": "X. 598270,3 | Y. 4659333 | Z. 37928",
4     "estacion": "64",
5     "datos": [
6         {
7             "fecha y hora": "01/06/2023 11:20:00",
8             "temperatura (C)": null,
9             "humedad (%)": null,
10            "precipitacion (mm)": null,
11            "nivel (m)": "0,05",
12            "caudal (m^3/s)": null,
13            "radiacion (W/m^2)": null
14        }
15    ]
16 }
17 ]
```

4.2.4 Filtrado de Datos

Una vez formateados los datos, con el fin de reducir la carga a la base de datos, estos son filtrados mediante la comparación con los ficheros anteriormente marcados como old, de esta forma, nos aseguramos de mandar a la base de datos únicamente las instancias nuevas de los datos recogidos, pues no disponemos de ninguna manera de filtrar los datos a la hora de obtenerlos. Estos datos serán guardados en un tercer JSON.

4.3 Arquitectura

Para poder realizar el trabajo se ha diseñado la siguiente arquitectura.[4.3](#)

4.3.1 Elementos presentes

Entornos virtuales

Con el fin de que la plataforma sea lo mas fácilmente ampliable se ha decidido que cada Spider disponga de su propio entorno virtual, esto permite añadir dependencias de tal forma que no afecten a el resto de los Scripts presentes, ayudando en la encapsulación de dependencias.

El mayor inconveniente de una plataforma así es la redundancia de código, al ser múltiples entornos independientes es necesario que mucho código sea repetido en cada uno de ellos, cosa que si fuera un único entorno en el que se ejecutara todo, con una clase sobre la que heredar y o una interfaz, seria mucho el código que nos ahorraríamos.

Actualmente la plataforma dispone de cuatro entornos virtuales para cada Spider y un entorno virtual sobre el que ejecutar el servidor de Django.

Spiders

Cada Spider representa una web, de tal forma que cada una de ellas obtiene los datos de la web sobre la que se ha diseñado exclusivamente, para poder realizar esta tarea por cada web han sido necesarias varias Spider, aunque de forma simplificada se pueden agrupar por aquellas que obtienen las estaciones junto con sus códigos y, las de obtención de datos.

De esta forma se dividen las tareas dando la posibilidad de ejecutar aquella que mejor venga en cada momento.

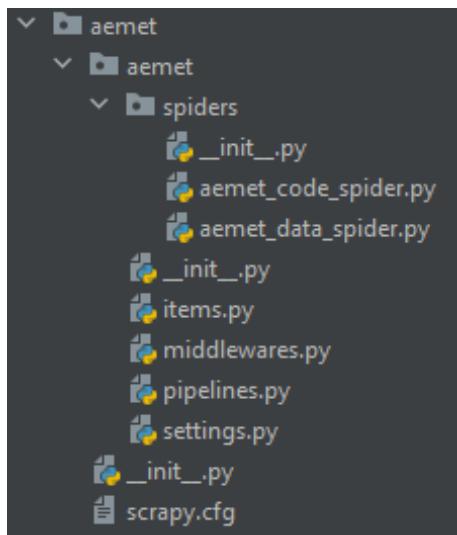


Figura 4.4: Estructuración básica de una Spider

Runners

Runner es la forma en la que han sido nombrados los Scripts cuya función es permitir la ejecución en nuestro caso asíncrona de una o múltiples Spiders mediante un único comando.

Es un Script simple en el que una vez dispones de la estructura básica en caso de necesitar añadir o eliminar una Spider solo tienes que agregar o eliminar la Spider en cuestión y ya estaría listo.

A su vez, al añadir un intermediario el comando de ejecución pasa de ser, `scrapy crawl nombreSpider` por cada Spider que se desea ejecutar, a, `python nombreRunner.py` facilitando la automatización de ejecución de las Spider.

Executers

Aumentado un poco mas la abstracción nos encontramos con los Executers, Scripts en Bash encargados de activar el entorno virtual de la Spider deseada y acto seguido ejecutar su respectivo Runner.

Estos existen (en mayor medida que los Runners) con el fin de ayudar con el mantenimiento de la arquitectura, creando un nuevo intermediario en la cadena de ejecución.

Están diseñados de tal forma que funcionen pasándoles un único argumento representando la web de la que quieras obtener los datos, haciendo que la agregación

de nuevas Spiders junto con sus entornos sea tan sencillo como respetar las rutas y nombres predefinidos.

JSONs

Este apartado es un conjunto de directorios en los que se van almacenando los JSON obtenidos tras los distintos procesos a los que son sometidos.

Los directorios en cuestión, siguiendo orden de creación para los JSON de datos son los siguientes:

1. RawData
2. ParsedData
3. RefinedData
- 4.OldData

Y los de códigos (estación):

1. RawCode
2. ParsedCode
3. RefinedCode (TODO)
4. OldCode (TODO)

El primer nivel es aquel que se obtiene de la llamada con la Spider a la web, devolviendo todos los datos de esta.

El segundo, es obtenido tras ejecutar ya sea `formatear_data_JSONs.py` en caso de querer parsear los datos o `formatear_code_JSONs.py` para los códigos.

El tercero se obtiene tras eliminar la duplicidad de datos en comparación con los ya almacenados en la base de datos mediante la ejecución de `filtrar_data_JSONs.py`.

Finalmente el cuarto es el subproducto obtenido tras la comparación, tomando el fichero original ya formateado y cambiándole de directorio y el nombre de tal forma que se distinga del resto de ficheros.

Posts

Estos Scripts son los encargado de enviar los datos mediante un Post Request al servidor Django.

Diseñados bajo el mismo principio de fomentar la ampliabilidad del proyecto que los Executer, reciben el nombre de la web de la cual deseas enviar los datos a la hora de ejecutarlo junto al comando en forma de argumento.

Cron

Cron es un administrador regular de procesos en segundo plano presente en los sistemas basados en Unix. Con el es posible programar la automatización de ejecución de procesos en intervalos de tiempo, pudiendo indicar el minuto, hora, día, mes e incluso día de la semana.

La especificación de los procesos se realiza en el archivo crontab y, su estructuración es la siguiente:

```
----- minuto (0-59)
| .---- hora (0-23)
| | .---- día del mes (1-31)
| | | .---- mes (1-12) o meses en inglés
| | | | .-- día de la semana (0-6) (domingo=0 o 7) o días en inglés
| | | |
* * * * * comando a ejecutar
```

En nuestro caso queremos el equivalente a dos instancias de Cron, una que se encargue de obtener los datos ya sea cada quince minutos, media hora y una hora manteniendo la base de datos actualizada para poder realizar las posteriores predicciones y otra que mensualmente compruebe la existencia de nuevas estaciones o el cese del uso de alguna de las ya disponibles.

Así pues, Cron es el encargado de ejecutar cada proceso necesario dentro de la plataforma, ya sea, ejecutar las Spiders, los Scripts de formateo como de filtrado y, el envío de los datos a Django para su inserción en PostGreSQL.

De esta forma dispondríamos de un sistema cerrado automático, el cual nos permitiría trabajar en otros apartados como puede ser la integración de mas webs en la plataforma o la mejora del sistema de predicción.

Django

La instancia del servidor de Django es la encargada de tanto recibir los datos obtenidos como de enviarlos a la base de datos. Finalmente, una vez se dispusiera de una aplicación de predicción, se encargaría de pedir los datos almacenados y enviarlos mediante POST a la aplicación.

PostGreSQL

La base de datos, dispone de las siguientes dos tablas de la imagen 4.5.

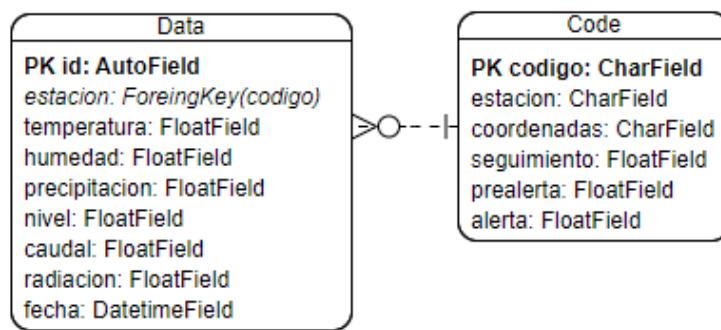


Figura 4.5: Tablas de la base de datos

Tienen una relación uno a muchos, siendo el campo estacion de la tabla Data la ForeignKey usada.

Para la tabla Data, al almacenar miles de datos, se ha optado por usar una PrimaryKey auto incremental. A su vez, a excepción del campo fecha, cualquiera de los campos pueden ser nulos, pues cada web proporciona únicamente un conjunto de los datos.

En la tabla Code, exceptuando la clave primaria y el nombre de la estación, el resto de campos tienen la posibilidad de ser nulos.

4.4 Entorno de ejecución

El proyecto se dividirá en dos ordenadores (maquinas virtuales). Uno de ellos será el responsable de la base de datos en PostGreSQL, mientras que, el otro, se encargará

de la ejecución del código de obtención de datos, tratarlos y enviarlos a la base de datos.

4.4.1 Preparación de entorno virtual

Primero instalaremos las dependencias para crear y activar un entorno virtual sobre el que trabajar, para ello ejecutaremos los siguientes comandos:

```
#Instalamos las dependencias
user@host:~$ sudo apt install python3-venv python3-dev

#Creamos un nuevo directorio sobre el que trabajar
user@host:~$ mkdir ~/dirdemiproyecto
user@host:~$ cd ~/dirdemiproyecto

#Creamos el entorno virtual
user@host:~/dirdemiproyecto$ python3 -m venv envdemiproyecto

#Activamos el entorno virtual
user@host:~/dirdemiproyecto$ source envdemiproyecto/bin/activate

#Una vez seguidos los pasos nuestra terminal debe mostrarse así
(envdemiproyecto) user@host:~/dirdemiproyecto$
```

Cada Spider dispondrá de su propio entorno virtual, nombrado tras la pagina web a la que representa, de esta forma, por ejemplo, el entorno virtual para la web de chcantabrico también se llamará chcantabrico. Otro entorno virtual sera usado para el servidor Django y los Scripts encargados del envío de datos.

4.4.2 Instalación y configuración de PostGreSQL

Comenzamos con los comandos necesarios para la instalación inicial de PostGreSQL y con la creación de un usuario y una base de datos.

```
#Instalamos PostGreSQL y las dependencias
user@host:~$ sudo apt install libpq-dev postgresql postgresql-contrib

#Iniciamos sesion usando el role postgres y accedemos a PostGreSQL
user@host:~$ sudo -u postgres psql

#Si todo esta bien la terminal deberia mostrarse asi
postgres=#
```

```
#Creamos un nuevo usuario
postgres=# CREATE USER miusuario WITH PASSWORD 'micontraseña';

#Configuramos varios parametros del usuario para una mejor integracion con Django
postgres=# ALTER ROLE miusuario SET client_encoding TO 'utf8';
postgres=# ALTER ROLE miusuario SET default_transaction_isolation TO 'read committed';
postgres=# ALTER ROLE miusuario SET timezone TO 'Europe/Madrid';

#Creamos la Base de Datos
postgres=# CREATE DATABASE bbddmiproyecto;

#Otorgamos permisos de administrador a nuestro usuario en la base de datos
postgres=# GRANT ALL PRIVILEGES ON DATABASE bbddmiproyecto TO miusuario;

#Una vez finalizado
postgres=# \q
```

En este punto ya se dispondría de una base de datos (aun sin tablas en ella) y de un usuario con el que conectarse a esta desde Django.

Ahora quedaría configurar el permiso de comunicación entre dispositivos, más concretamente la conexión remota de Django con PostGreSQL.

```
#Abrimos el archivo postgresql.conf con un editor
user@host:~$ sudo nano /etc/postgresql/11/main/postgresql.conf

#Buscamos la linea "#listen_addresses = 'localhost'", borramos la almohadilla y
sustituimos localhost por *, permitiendo la escucha de cualquier direccion IP
listen_addresses = '*'

#Guardamos y cerramos el fichero

#Abrimos el archivo pg_hba.conf con un editor
user@host:~$ sudo nano /etc/postgresql/11/main/pg_hba.conf

#Por defecto solo permite conexiones desde localhost
# IPv4 local connections:
host      all            all            127.0.0.1/32          md5

#Configurararemos el permiso de conexion remota desde cualquier IP añadiendo
la siguiente linea debajo de la anterior
```

```

host    all          all          0.0.0.0/0          md5
#0 podemos configurar el permiso unicamente para la IP de la otra maquina virtual
host    all          all          127.18.83.198/19      md5
#Guardamos y cerramos el fichero
#Finalmente permitimos el trafico mediante el puerto 5432 (puesto por defecto)
user@host:~$ sudo ufw allow 5432/tcp

```

Una vez realizados los siguientes pasos ya está la instancia de PostGreSQL configurada.

4.4.3 Instalación y configuración de Django

Antes de instalar Django, es necesario generar un entorno virtual sobre el que trabajar.

```

#Instalamos Django y las dependencias
(envdemiproyecto) user@host:~/dirdemiproyecto$ pip install django psycopg2

#Creamos un nuevo proyecto de Django
(envdemiproyecto) user@host:~/dirdemiproyecto$ django-admin startproject djangoAPI
(envdemiproyecto) user@host:~/dirdemiproyecto$ cd djangoAPI

#Creamos una nueva app Django
(envdemiproyecto) user@host:~/dirdemiproyecto/djangoAPI$ python manage.py startapp appAPI

```

Una vez realizado lo siguiente, quedaría configurar la API.

```

#Abrimos el archivo settings.py con un editor
(envdemiproyecto) user@host:~/dirdemiproyecto/djangoAPI$ nano djangoAPI/settings.py

#Añadimos nuestra app en la lista de INSTALLED_APPS
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'appAPI.apps.AppapiConfig',
]

```

```
#Configuramos la base de datos sustituyendo
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

por

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'bbddmiproyecto',
        'USER': 'miusuario',
        'PASSWORD': 'micontraseña',
        'HOST': '172.18.83.197',
        'PORT': '5432',
    }
}
```

```
#Configuramos las IP admitidas
ALLOWED_HOSTS = ['172.18.83.197', 'localhost']
```

```
#Configuramos la zona horaria
TIME_ZONE = 'Europe/Madrid'
```

```
#Configuramos el tamaño maximo de datos a None, con el fin de poder
enviar cualquier cantidad de datos
DATA_UPLOAD_MAX_MEMORY_SIZE = None
```

```
#Guardamos y cerramos el fichero
```

```
#Finalmente permitimos el trafico mediante el puerto 8000 (puesto por defecto)
user@host:~$ sudo ufw allow 8000
```

Tras configurar la API, toca crear los modelos de la base de datos.

```
#Abrimos el archivo models.py dentro de la carpeta appAPI con un editor
(en demiproyecto) user@host:~/dirdemiproyecto/djangoAPI$ nano appAPI/models.py
```

Dentro definimos los modelos Data y Code.

```

1   from django.db import models
2
3   class Data(models.Model):
4       temperatura = models.FloatField(null=True)
5       humedad = models.FloatField(null=True)
6       precipitacion = models.FloatField(null=True)
7       nivel = models.FloatField(null=True)
8       caudal = models.FloatField(null=True)
9       radiacion = models.FloatField(null=True)
10      fecha = models.DateTimeField()
11      estacion = models.ForeignKey("Code", on_delete=models.CASCADE)
12
13  class Code(models.Model):
14      estacion = models.CharField(max_length=50)
15      codigo = models.CharField(max_length=20, primary_key=True)
16      coodenadas = models.CharField(max_length=50)
17      seguimiento = models.FloatField(null=True)
18      prealerta = models.FloatField(null=True)
19      alerta = models.FloatField(null=True)

```

Código 4.1: Modelos Django

Para importar los modelos a la base de datos, con el fin de crear las respectivas tablas.

```
#Actualizamos los datos realizados en models.py
(envdemiproyecto) user@host:~/dirdemiproyecto$ python3 manage.py makemigrations
```

```
#Migramos los datos
(envdemiproyecto) user@host:~/dirdemiproyecto$ python3 manage.py migrate
```

Si se quiere confirmar la correcta creación de las tablas, desde la terminal de PostGre.

```
#Nos conectamos a la base de datos
postgres=# \c bbddmiproyecto
```

```
#Mostramos las tablas
postgres=# \d
```

Si todo a ido bien, se deberían mostrar como en la imagen 4.6.

```

scrapedata=# \d
                                         Listado de relaciones
  Esquema |          Nombre          |  Tipo   | Dueño
-----+-----+-----+-----+
  public | appAPI code           | tabla  | oderiz
  public | appAPI data           | tabla  | oderiz
  public | appAPI data id seq  | secuencia | oderiz

```

Figura 4.6: Lista de tablas en PostGreSQL

Tras configurar la API y crear las tablas de la base de datos, se creará la API como tal.

```
#Creamos el archivo urls.py con un editor dentro de la carpeta appAPI  
(envdemiproyecto) user@host:~/dirdemiproyecto/djangoAPI$ nano appAPI/urls.py
```

Dentro se incluye el siguiente código.

```
1  from django.urls import path  
2  from django.views.decorators.csrf import csrf_exempt  
3  
4  from . import views  
5  
6  urlpatterns = [  
7      path("storeData", csrf_exempt(views.storeData), name="storeData"),  
8      path("storeCode", csrf_exempt(views.storeCode), name="storeCode"),  
9  ]
```

Código 4.2: Modelos Django

Aquí se definen las rutas para realizar una llamada a la API, en este caso storeData y storeCode.

Si se quiere que el proyecto tenga constancia de ellas.

```
#Abrimos el archivo urls.py dentro de la carpeta djangoAPI con un editor  
(envdemiproyecto) user@host:~/dirdemiproyecto/djangoAPI$ nano djangoAPI/urls.py
```

Dentro se incluye el siguiente código.

```
1  from django.contrib import admin  
2  from django.urls import path, include  
3  
4  urlpatterns = [  
5      path('admin/', admin.site.urls),  
6      path('appapi/', include('appapi.urls'))  
7  ]
```

Código 4.3: Modelos Django

Para finalizar, se crean las vistas a las que dirigen las rutas.

```
#Abrimos el archivo views.py dentro de la carpeta appAPI con un editor  
(envdemiproyecto) user@host:~/dirdemiproyecto/djangoAPI$ nano appAPI/views.py
```

Dentro se incluye el siguiente código.

```
1 import json
2
3 from django.http import JsonResponse
4 from .models import Data, Code
5
6 # Create your views here.
7 def storeData(request):
8     data = json.loads(request.body.decode("utf-8"))
9     for estacion in data:
10         for datos in estacion['datos']:
11             dato = Data(
12                 temperatura=datos['temperatura ( C )'],
13                 humedad=datos['humedad (%)'],
14                 precipitacion=datos['precipitacion (mm)'],
15                 nivel=datos['nivel (m)'],
16                 caudal=datos['caudal (m^3/s)'],
17                 radiacion=datos['radiacion (W/m^2)'],
18                 fecha=datos['fecha y hora'],
19                 estacion=estacion['estacion']
20             )
21             dato.save()
22     return JsonResponse(data, safe=False)
23
24 def storeCode(request):
25     data = json.loads(request.body.decode("utf-8"))
26     for estacion in data:
27         dato = Code(
28             estacion=estacion['estacion'],
29             codigo=estacion['codigo'],
30             coodenadas=estacion['coodenadas'],
31             seguimiento=estacion['seguimiento'],
32             prealerta=estacion['prealerta'],
33             alerta=estacion['alerta'],
34         )
35         dato.save()
36     return JsonResponse(data, safe=False)
```

Código 4.4: Modelos Django

Estas funciones se encargan de recibir los datos enviados, iterar por ellos y subirlos a la base de datos con el método save().

Con esto concluiría la creación y configuración de la API.

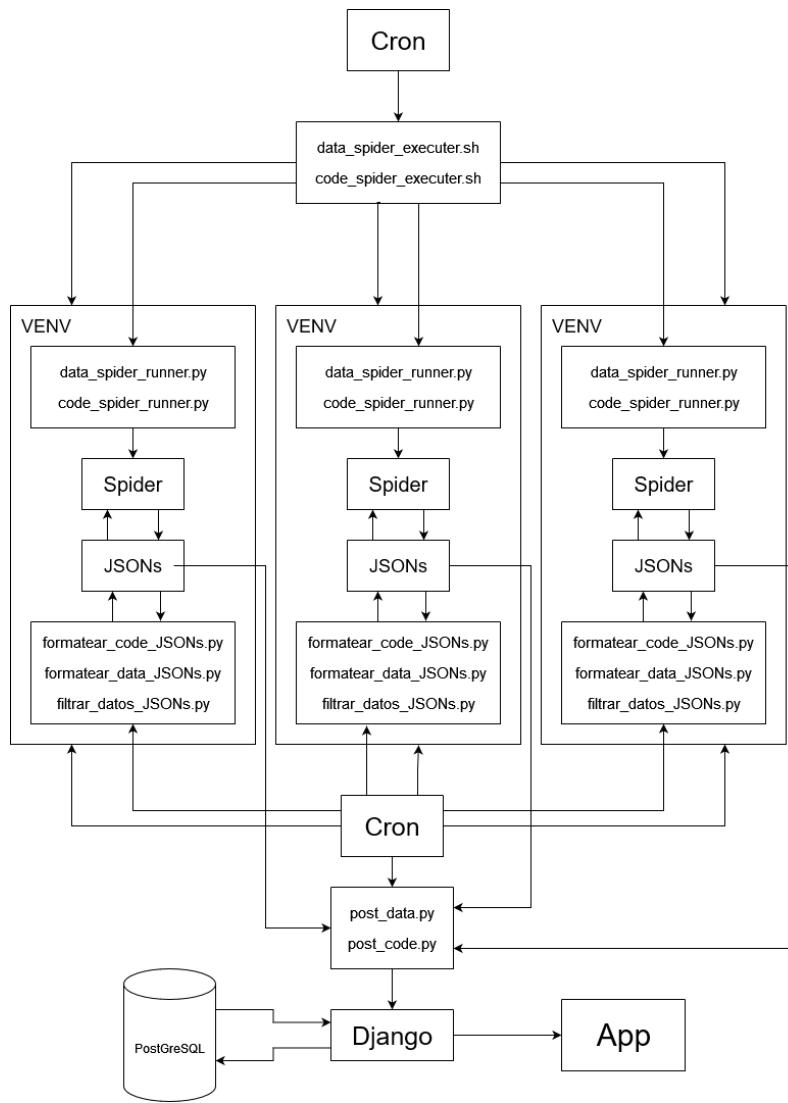


Figura 4.3: Arquitectura de obtención y tratamiento de datos

Implementación

5.1 Creación de Spiders

Una vez instalado Scrapy, en el directorio escogido escribimos el siguiente comando para generar un nuevo proyecto de Scrapy.

```
scrapy startproject miproyecto
```

Nos creara un nuevo directorio con el siguiente contenido.

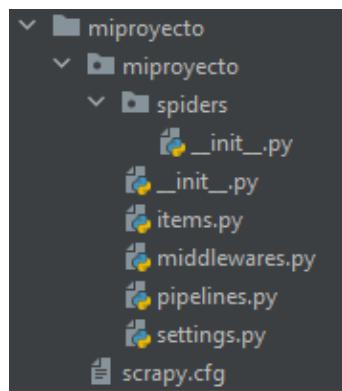


Figura 5.1: Estructura del proyecto recién creado

Primero entraremos en el directorio recientemente creado y luego ejecutaremos el comando encargado de crear la Spider.

```
cd miproyecto
scrapy genspider mispider webausar.com
```

En caso de no especificar el protocolo usado por la web Scrapy asumirá que usa HTTPS.

Tras ejecutar el comando la Spider habrá sido generada dentro de la carpeta spiders.

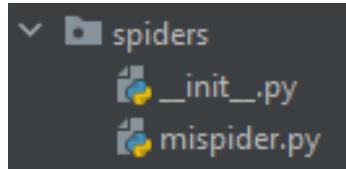


Figura 5.2: Directorio de almacenamiento de las Spider

Una vez abierto el archivo vemos que dispone del siguiente código.

```

1 import scrapy
2
3
4 class MispiderSpider(scrapy.Spider):
5     name = "mispider"
6     allowed_domains = ["webausar.com"]
7     start_urls = ["https://webausar.com"]
8
9     def parse(self, response):
10        pass

```

Código 5.1: Spider recién generada

Scrapy usa una programación orientada a objetos, siendo cada Spider una clase representada dentro del proyecto.

Analizando las variables definidas vemos las siguientes, name, nombre por el que debemos referenciar la Spider a la hora de ejecutarla; allowed_domains, indica que dominios podemos visitar, negando la entrada a cualquier dominio que no este definido en ella, es importante no especificar protocolo, de esta manera funcionara para cualquier web ya sea HTTP como HTTPS que pertenezca a ese dominio, de lo contrario se limitara al protocolo indicado; start_urls, URL inicial sobre la que se hará la request de petición de datos.

El método parse es aquel al que se envía la respuesta obtenida de la web, para realizar el filtrado de la información, quedándose únicamente con la deseada. Este método es invocado automáticamente por la Spider, sin necesidad real de hacerlo tú manualmente. Puede ser un método recursivo en caso de así quererlo o incluso se pueden definir nuevas funciones parse (usando un nombre distinto) en caso de necesitarlas.

5.1.1 Proceso de obtención de datos

Para poder realizar es la extracción de los datos, primero debemos ir a la web deseada e inspeccionar su estructuración. Para ello como ejemplo vamos a usar la web de aemet.

The screenshot shows the AEMET website interface. At the top, there's a header with the Spanish Government logo, the AEMET logo, and a search bar. Below the header, a navigation menu includes 'EL TIEMPO', 'SERVICIOS CLIMÁTICOS', 'CONOCENOS', 'H-D-H', 'CONOCER MÁS', 'EMPLEO PÚBLICO Y BECAS', 'DATOS ABIERTOS', and 'SEDE ELECTRÓNICA'. Below the menu, there are weather icons for visibility, temperature, and pressure. The main content area is titled 'Hoy y últimos días. Comunidad Foral de Navarra'. It features tabs for 'Datos horarios', 'Resumen viernes 04', 'Resúmenes diarios anteriores', 'Mapa de España', 'Mapa', and 'Tabla'. The 'Tabla' tab is selected. Below the tabs, there's a note about the last update ('Actualizado: viernes, 04 agosto 2023 a las 11:42 hora oficial') and the current time ('Fecha y hora: viernes, 04 agosto 2023 a las 11:00 hora oficial'). At the top right of the table area are 'Exportar a excel' and 'Exportar a csv' buttons. The table itself has columns for Estación, Provincia, Temp. (°C), V. vien. (km/h), Dir. viento, Racha (km/h), Dir. racha, Prec. (mm), Presión (hPa), Tend. (hPa), and Humedad (%). The data rows list various stations in Navarra with their respective values.

Estación	Provincia	Temp. °C	V. vien. km/h	Dir. viento	Racha km/h	Dir. racha	Prec. mm	Presión hPa	Tend. hPa	Humedad %
Aranguren, Ildain	Navarra	16.1	10	↙	26	↙	0.0			78.0
Areso	Navarra									
Bardenas Reales, Base Aérea	Navarra	19.0	37	↖	54	↖	0.0	986.2	0.9	56.0
Baztan, Irurita	Navarra	15.6	5	↑	8	←	1.8			97.0
Bera	Navarra	17.4	3	↗	14	↖	1.0			94.0
Cáseda	Navarra	18.3	17	↓	36	↘	0.0			65.0
Esteribar, Embalse de Eugi Irurtzun	Navarra	13.2				1.4				99.0
Isaba/Izaba	Navarra	12.4	9	↖	24	↓	0.6			95.0
Larreaga	Navarra	18.4	23	↙	49	↘	0.0			62.0
Los Arcos	Navarra									
Monreal/Elo	Navarra	14.1	25	↖	37	↘	0.2			86.0
Monteagudo	Navarra									
Navascués/Nabaskoze	Navarra	15.6					0.0			73.0
Olite/Erriberri	Navarra	17.8	18	↖	37	↘	0.0			63.0
Oroz-Betelu/Orotz-Betelu	Navarra									
Pamplona, Aeropuerto	Navarra	18.5	17	↓	30	↘	0.0	969.2	0.9	66.0
Roncesvalles/Orreaga	Navarra	10.5	6	↗	21	↖	2.0			99.0
Tudela	Navarra	19.7	14	↖	40	↘	0.0			56.0
Valcarlos/Luziaide	Navarra									

Figura 5.3: Web de Aemet para obtención de datos

Una vez encontrada la web deseada, accediendo mediante el F12 a la herramienta de inspección, buscamos el elemento representativo del dato deseado. En nuestro caso queremos obtener tanto el nombre como el código de la estación. Ambos se encuentran en el mismo elemento que forma la primera columna de la tabla.



Figura 5.4: Inspector de webs

Como en este caso es posible filtrar fácilmente los datos, los obtendremos todos directamente, aunque lo más común sería obtener las filas primero para luego iterar por cada una de ellas. Para obtenerlos podemos hacerlo mediante el selector de XPath como con el de CSS.

Ambos se pueden obtener fácilmente en la herramienta de inspección, una vez seleccionado el elemento deseado, hacemos click derecho sobre él, vamos al apartado copiar y en el nos mostrará la posibilidad de copiar ambos selectores. Es probable que el selector proporcionado no sea del todo lo que busquemos o se pueda simplificar, por lo que es recomendable comprobarlo manualmente.

```
rows = response.xpath('//div[@id="contenedor_tabla"]/table/tbody/tr/td/a')
ó
rows = response.css("div#contenedor_tabla tbody tr a")
```

Esto devuelve una lista de objetos tipo Selector, cosa que permite conforme se itera por cada elemento volver a usar un selector para filtrar únicamente los datos deseados. En este caso,

```
path = rows[i].xpath("@href").get()
name = rows[i].xpath("./text()").get()
ó
path = rows[i].css("*::attr(href)").get()
name = rows[i].css("*::text").get()
```

De esta forma, mediante el uso de la función get(), pasamos de tener un objeto Selector a un String. El uso de get() sobre una lista devuelve el primer elemento, en caso de querer transformar toda la lista el método a usar es getall().

Finalmente, como de la URL obtenida,

```
'/es/eltiempo/observacion/ultimosdatos?k=nav&l=9263X&w=0&datos=det&f=precipitacion'
```

solo es de interés el código de la estación (parámetro l de la query), se filtra mediante splits.

```
code = path.split('&')[1].split('=')[1]
```

5.1.2 Guardado de datos

Scrapy almacena todos los datos en forma de múltiples diccionarios, tantos como webs usadas. Para acceder a esta información Scrapy nos proporciona dos alternativas, el uso de Items junto a ItemLoaders, siendo clases específicas de Scrapy o, mediante la palabra reservada yield de Python, que tiene una funcionalidad parecida a return, siendo esta la opción elegida debido a su fácil implementación.

De esta forma escribiremos.

```
1  yield {
2      'estacion': name,
3      'codigo': path.split('&')[1].split('=')[1],
4  }
```

Código 5.2: Guardar datos

Actualmente si se ejecuta la Spider nos imprimiría los datos obtenidos por pantalla, aunque pueden ser almacenados en un fichero tanto CSV como JSON, a la hora de ejecutar la Spider añadiendo en el comando "-o nombre.csv ó -o nombre.json".

Para un uso ligero de forma manual esa alternativa es más que suficiente, pero en nuestro caso, al querer ejecutarlas de forma automática mediante el uso de Runners, debemos implementar una variable llamada custom_settings para cada una de las Spider. Esta permite, sin la necesidad de modificar el archivo settings.py, añadir configuraciones o dependencias independientes en las Spiders.

```
1  custom_settings = {
2      'FEEDS': {
3          'JSONs/RawCode/codigos_aemet.json': {
4              'format': 'json',
5              'encoding': 'utf-8',
6              'overwrite': True,
7          }
8      }
9  }
```

Código 5.3: Configurar guardado en JSON

Con esto indicamos que, en la ruta especificada, nos almacene un fichero JSON utf-8 y, que cada vez que se llame a esta Spider sobre-escriba el fichero anterior.

5.1.3 Spider básica

Una vez obtenemos los datos y los podemos almacenar, ya estaría nuestra Spider básica terminada.

```

1 import scrapy
2
3
4 class AemetCodeSpider(scrapy.Spider):
5     name = "aemet_code"
6     allowed_domains = ["www.aemet.es"]
7     start_urls = ["https://www.aemet.es/es/eltiempo/observacion/"
8                   "ultimosdatos?k=nav&w=0&datos=det&x=h24&f=precipitacion"]
9     custom_settings = {
10         'FEEDS': {
11             'JSONs/RawCode/codigos_aemet.json': {
12                 'format': 'json',
13                 'encoding': 'utf-8',
14                 'overwrite': True,
15             }
16         }
17     }
18
19     def parse(self, response):
20         rows = response.css("div#contenedor_tabla tbody tr a")
21
22         for row in rows:
23             path = row.xpath("@href").get()
24             name = row.xpath("./text()").get()
25             code = path.split('&')[1].split('=')[1]
26
27             yield {
28                 'estacion': name,
29                 'codigo': code,
30             }
```

Código 5.4: Spider de ejemplo (Aemet Code Spider)

5.1.4 Método start_requests()

El método start_requests() es llamado de forma automática al iniciar la Spider, siendo el encargado de hacer la llamada a la web indicada en start_urls y, una vez obtenidos los datos llamar a la función parse, todo mediante un objeto Request de Scrapy, el cual devolverá un objeto tipo HTMLResponse. En caso de querer alterar el funcionamiento de la Spider este es el método a sobre-escribir.

Como en nuestro caso queremos obtener los datos de todas las estaciones dentro de un mismo dominio, reescribiremos la función para que recorra el JSON con los códigos de estas y, hacer una llamada por estación con Request.

El código quedaría de la siguiente manera.

```
1 import json
2
3
4 def start_requests(self):
5     with open("JSONs/RawCode/codigos_aemet.json", encoding="utf-8") as
6         f:
7             data = json.load(f)
8             for estacion in data:
9                 url = f'https://www.aemet.es/es/eltiempo/observacion'
10                '/ultimosdatos?k=nav&l={estacion["codigo"]}&w=0&'
11                'datos=det&x=&f=temperatura'
12                 yield scrapy.Request(url, self.parse)
```

Código 5.5: Sobre-escritura de start_request()

Al definir la función de esta manera no es necesario declarar la variable start_urls, por lo que siempre que necesitemos sobre-escribir la función, no usaremos la variable.

5.1.5 Eliminar Log

Cuando se verifique el correcto funcionamiento de la Spider es recomendable quitar el maximo numero de Log por pantalla posible, es por eso que, en el fichero settings.py escribiremos las siguientes lineas.

```
1 LOG_LEVEL = 'WARNING'
2 LOG_ENABLED = False
```

Código 5.6: Configurar LOG

5.2 Spiders usadas

Para este proyecto se han creado cuatro proyectos de Scrapy, uno por cada web.

5.2.1 Aemet

Siendo la Spider de obtención de códigos la usada como ejemplo, no hay mucho más que comentar al respecto de ella, obtiene los nombres y códigos de cada estación.

Data Spider

La Spider de obtención de datos es un poco más compleja, necesitando sobre-escribir la función de start_request() como se muestra en el ejemplo del apartado anterior. Primero lee el fichero JSON con los códigos de las estaciones, itera por ellos y, crea una llamada Request por cada estación.

Dentro de la función parse():

```

1 latitud = response.css('abbr.latitude::text').get()
2 longitud = response.css('abbr.longitude::text').get()
3 estacion = response.css("a.separador_pestanhas").get()
4 rows = response.css('tbody tr')
```

Código 5.7: Selector en parse() de Aemet Data Spider

Obtiene las coordenadas, latitud, longitud; una URL con el código de la estación, variable estacion y, todas las filas presentes en el cuerpo de la tabla de datos, rows.

```

1 datos = []
2 for row in rows:
3     dato = {
4         'fecha y hora': row.xpath('./td[1]/text()').get() + ':00',
5         'temperatura (C)': row.xpath('./td[2]/text()').get(),
6         'humedad (%)': row.xpath('./td[10]/text()').get(),
7         'precipitacion (mm)': row.xpath('./td[7]/text()').get(),
8     }
9
10    if dato['precipitacion (mm)'] != " ":
11        datos.append(dato)
```

Código 5.8: Trabajar sobre los datos de Aemet Data Spider

Se crea una lista datos vacía para almacenar los datos. Posteriormente, se recorren las filas obtenidas, creando un objeto JSON llamado dato, que almacena, obteniendo mediante selectores, la fecha y hora (añadiendo :00 al final para tener el mismo formato dd/mm/aa hh:mm:ss que en el resto de webs), la temperatura, la humedad y la

precipitación.

En caso de que el apartado precipitación no esté vacío, pues puede darse el caso en el que la web aun no disponga de el a cierta hora, pero si muestre esta franja horaria pues dispone de otros datos como pueden ser aquellos relacionados con el viento, el objeto dato sera almacenado en la lista datos.

```

1   yield {
2     'coordenadas': latitud + ' | ' + longitud,
3     'estacion': estacion.split('=')[3].split('&')[0],
4     'datos': datos,
5   }

```

Código 5.9: Guardado de datos de Aemet Data Spider

Finalmente se almacenan las coordenadas con un formato global para todas las plataformas y, se filtra mediante splits el código de la estación.

5.2.2 CHCantábrico

Code Spider

La Spider para la adquisición de códigos de CHCantábrico no necesita de sobreescritura del método start_requests(), por lo que el código a analizar sera el presente en la función parse().

```

1   rows = response.xpath('//table[@class="tablefixedheader
niveles"]/tbody/tr')

```

Código 5.10: Selector en parse() de CHCantábrico Code Spider

Puesto que la estructuración HTML de la web no proporciona una manera clara de lograr los datos mediante un selector CSS, los datos se logran filtrar por su selector XPath. De este, se obtendrán las filas que representan cada estación.

```

1   for row in rows:
2     codigoBusqueda = row.css('td.codigo::text').get()
3     limites = row.css('table.umbrales_gr td.datos::text').getall()
4     path = row.xpath('./td/a/@href').getall()[-1]
5     estacion = row.xpath('./td/a/text()').getall()[-3]

```

Código 5.11: Trabajar sobre los datos de CHCantábrico Code Spider

Una vez obtenidas las filas, se itera sobre ellas para obtener, dos códigos, el primero, codigoBusqueda como código representativo de la estación cara a obtener las coordenadas y, el código necesario para acceder a los datos, inicialmente almacenado en

una URL en la variable path; los límites marcados de pre-alerta, alerta y seguimiento en la variable limites, siendo estos una buena base para empezar con las predicciones de inundación y, el nombre de la estación.

Inicialmente, para las variables path y estacion, se obtienen todo los elementos que corresponden con el selector asignado, para posteriormente quedarse con el que realmente interesa. Se hace así al no poder filtrar mediante HTML los datos concretos.

```

1  for i in range(len(limites)):
2      if limites[i] == 'No definido':
3          limites[i] = None

```

Código 5.12: Comprobar límites de CHCantábrico Code Spider

Aun dentro del bucle, se comprueba si los límites están definidos, marcando como None (null) aquellos que no lo estén.

```

1  yield {
2      'estacion': estacion,
3      'codigo': path.split("=")[-1],
4      'codigoSecundario': codigoBusqueda,
5      'seguimiento': limites[0],
6      'prealerta': limites[1],
7      'alerta': limites[2],
8  }

```

Código 5.13: Guardado de datos de CHCantábrico Code Spider

Por ultimo, se filtra el código de la estación (aquel usado para acceder los datos) de la URL y, se guardan los datos.

Data Spiders

CHCantábrico muestra datos tanto del nivel del río como de la precipitación, aunque lo hace en dos direcciones distintas, haciendo necesario el uso de dos Spiders.

Puesto que ambas comparten la misma estructuración de código, solo se explicara una de ellas. La de nivel del río.

```

1  def start_requests(self):
2      with open("JSONs/RawCode/codigos_chcantabrico.json",
3                  encoding="utf-8") as f:
4          data = json.load(f)
5          for estacion in data:

```

```
5     params_nivel = {
6         'p_p_id': 'GraficaEstacion_INSTANCE_wH0LL6jTUysu',
7         'p_p_lifecycle': '2',
8         'p_p_state': 'normal',
9         'p_p_mode': 'view',
10        'p_p_resource_id': 'downloadCsv',
11        'p_p_cacheability': 'cacheLevelPage',
12        '_GraficaEstacion_INSTANCE_wH0LL6jTUysu_cod_estacion':
13            f'{estacion["codigo"]}',
14        '_GraficaEstacion_INSTANCE_wH0LL6jTUysu_tipodato': 'nivel',
15    }
16
17
18    url = 'https://www.chcantabrico.es/evolucion-de-niveles'
19
20    yield scrapy.FormRequest(url=url,
21        method='GET',
22        formdata=params_nivel,
23        callback=self.parse,
24        cb_kwargs={'estacion': estacion['codigo']})
```

Código 5.14: Función start_requests() CHCantábrico Nivel Spider

Como se necesitan los datos de todas las estaciones, se reescribirá la función start_requests().

A su vez, como CHCantábrico no muestra los datos por pantalla, incluyendo un botón sobre el que pulsar para obtenerlos descargando un fichero CSV, en vez de hacer una request básica mediante la clase Request, se hace uso de FormRequest para hacer una llamada GET, que simula la llamada a un formulario y obtiene los datos que este devuelve.

De esta forma, pasandole los parámetros necesarios en el argumento formdata a la URL indicada, definidos como un diccionario en la variable params_nivel, se obtendrán los datos sin la necesidad de ningún CSV simulando en cierto modo una llamada mediante cURL.

Cabe mencionar que, Request devuelve un HTMLResponse y que, la respuesta de estas llamadas no es código HTML, por lo que, aun en caso de que llegue a ser posible usar Request, es más correcto el uso de FormRequest devolviendo un FormResponse para este tipo de casos.

El uso del argumento cb_kwargs sirve para enviar un mayor numero de argumentos

a la función parse() de los que normalmente recibe.

```

1 def parse(self, response, estacion):
2     if not response.text.startswith('-'):
3         urlData = response.text
4         rawData = pd.read_csv(io.StringIO(urlData), delimiter=';',
5                               encoding='utf-8', header=1)
6         rawData.columns = ['fecha y hora', 'nivel (m)']
    parsedData = rawData.to_json(orient="records")

```

Código 5.15: Función parse() CHCantábrico Nivel Spider

Debido al argumento cb_kwargs, la función parse() recibe un tercer argumento, estación. En este caso, para poder enviar a cada conjunto de datos recibidos el código de la estación a la que pertenecen, pues dentro de la respuesta obtenida solo se proporciona el nombre de esta.

Dentro de la función parse() lo primero que se hace es comprobar que realmente se ha recibido una respuesta correcta, pues, aunque todas las estaciones disponen de datos del nivel del río, no todas disponen los de precipitación. El problema viene cuando a estas estaciones se les piden los datos, ya que en vez de enviar un error 404 como sería esperado.

-
FECHA;VALOR(mm)

Una alternativa para deshacerse de esta comprobación sería eliminar aquellas estaciones que no proporcionen datos o filtrandolas para no hacer la llamada directamente, aunque esto no solo resultaría más complejo, si no que crearía el problema de que cada cierto tiempo habría que comprobar si alguna estación ha empezado a proporcionar datos para incluirla nuevamente en la lista de estaciones a las cuales hacer llamada.

Una vez hecha la comprobación, en caso de que no sea una respuesta vacía, el texto viene proporcionado con el siguiente formato.

Ribera de Piquín
 FECHA;VALOR(m)
 03/08/2023 11:30:00;0.153
 03/08/2023 11:45:00;0.153
 03/08/2023 12:00:00;0.153
 ...

Al tener formato CSV se lee mediante la función `read_csv()` incluida en la librería pandas, se indica el delimitador, la codificación y la linea que representa la cabecera, empezando de la 0, en este caso la 1, pues no nos interesa el nombre de la estación. Finalmente, como la función espera que se le pase una ruta a un fichero, lo que hacemos mediante `io.StringIO()` es crear un objeto con el que simular un fichero en memoria, pasando de disponer texto plano a un DataFrame de pandas.

Como últimos pasos, se cambian los nombres de las cabeceras a aquellos definidos de forma global para todas las webs y, se convierte el DataFrame en un JSON con la función `to_json()` indicando que el formato sea "records", esto implica que cada linea del DataFrame va a representar un objeto JSON.

```
1  yield {
2      'estacion': estacion,
3      'datos': json.loads(parsedData)
4 }
```

Código 5.16: Guardado de datos de CHCantábrico Nivel Spider

Con el uso de la función `loads()` de la librería json nos aseguramos el correcto formato del JSON.

Coordenates Spider

Aunque en la web misma de CHCantábrico se proporciona un mapa indicando la localización de cada estación, las coordenadas de esta no están disponibles para adquirir dentro de la web, es por eso que es necesario el uso de la web del centro de estudios hidrológicos para poder obtener las coordenadas.

En esta página se pueden encontrar mediante el "codigoSecundario" anteriormente obtenido, desgraciadamente, no todas las estaciones incluidas en CHCantábrico están listadas en esta página, siendo el mayor inconveniente para el correcto funcionamiento del apartado de predicción.

```
1 longitud = response.css('p::text')[6].get().strip()
2 latitud = response.css('p::text')[7].get().strip()
3 estacion = response.css('font::text')[14].get().strip()
```

Código 5.17: Selector en parse() de CHCantábrico Coordinates Spider

```
1 yield {
2     'coordenadas': f'Lat: {latitud} | Lon: {longitud}',
3     'estacion': estacion,
```

```
4 }
```

Código 5.18: Guardado de datos de CHCantábrico Coordinates Spider

Exceptuando el uso del "codigoSecundario" para referenciar estaciones, esta es una Spider muy simple la cual no dispone de nada que no haya sido anteriormente explicado.

Código descartado

Siendo CHCantábrico la primera web de la que se obtuvo los datos, sin gran conocimiento de Scrapy y sobre todo, sin saber realmente como sería la plataforma, el planteamiento de la obtención de datos se realizó de forma ajena a las Spider de Scrapy.

Viendo que los datos no están presentes en la web y, que Scrapy no dispone de interacción JavaScript por defecto, la única alternativa viable conocida en esos momentos fue probar a realizar una llamada cURL por terminal, al ver que efectivamente mediante cURL era posible obtener los datos deseados, se escribió un script para los datos de nivel y otro para los de precipitación.

```
1 import pandas as pd
2 import io
3 import requests
4 import json
5
6 with open('códigos_estaciones_chcantabrido.json', 'r',
7     encoding='utf-8') as f:
8     data = json.load(f)
9
10    datos = []
11    for item in data:
12        params_pluvio = {
13            'p_p_id': 'GraficaEstacion_INSTANCE_ND81Xo17PIZ7',
14            'p_p_lifecycle': '2',
15            'p_p_state': 'normal',
16            'p_p_mode': 'view',
17            'p_p_resource_id': 'downloadCsvPluvio',
18            'p_p_cacheability': 'cacheLevelPage',
19            '_GraficaEstacion_INSTANCE_ND81Xo17PIZ7_cod_estacion':
20                f'{item["codigo"]}',
21            '_GraficaEstacion_INSTANCE_ND81Xo17PIZ7_tipodato': 'pluvio',
22        }
23 }
```

```
22     response_pluvio =
23         requests.get('https://www.chcantabrico.es/precipitacion-acumulada',
24             params=params_pluvio)
25     if response_pluvio.status_code == 200:
26         if not response_pluvio.text.startswith('-'):
27             urlData = response_pluvio.text
28             rawData = pd.read_csv(io.StringIO(urlData), delimiter=';',
29                 encoding='utf-8', header=1)
30             rawData.columns = ['fecha y hora', 'precipitacion (mm)']
31             parsedData = rawData.to_json(orient="records")
32
33             estacion = {
34                 'estacion': item["codigo"],
35                 'datos': json.loads(parsedData)
36             }
37             datos.append(estacion)
38         else:
39             print(f'{item["estacion"]} Error retrieving data: 404')
40             print("-----")
41     else:
42         print(f'{item["estacion"]} Error retrieving data:
43             {response_pluvio.status_code}')
44         print("-----")
45
46     with open('.../JSONs/RawData/datos_pluvio_chcantabrico.json', 'w',
47         encoding='utf-8') as outfile:
48         json.dump(datos, outfile)
```

Código 5.19: Script de obtención de datos pluviometricos descartado

Todo el apartado de tratamiento de los datos es prácticamente idéntico al realizado con la Spider, solo que en vez de usar FormRequest, se hace uso de la librería requests para realizar la llamada get(), tras realizarla, se comprueba que haya sido exitosa (esta comprobación la realiza Scrapy automáticamente) y, en caso de serlo se realiza todo el tratamiento, guardando los datos en la lista datos. Una vez realizadas todas las llamadas, guardamos los datos obtenidos usando la función json.dump().

Aunque en esta versión se almacenan todos los datos en un mismo JSON, originalmente los datos eran guardados en un CSV por cada estación, de tal manera que el nombre del CSV era el mismo que el de la estación perteneciente. Más adelante al consolidar más la plataforma, sobre todo el uso de Django para la creación de una API, se vio que era más útil guardar los datos no solo en formato JSON si no disponer

de un único fichero por estación, de esta forma solo seria necesario realizar una única llamada por estación a la API para cargar los datos. Llegando a esta versión del script.

Posteriormente, llegado el momento de la automatización quedo claro que, aun siendo posible automatizar el proceso con el script anterior, iba a suponer un problema para la modularidad del proyecto. Pues tener múltiples scripts de diferentes fuentes, solo aumentaba la complejidad a la hora de crear un script ya sea en Python o Bash encargado de ejecutar cada parte individual de cada web. Siendo la alternativa proporcionada por Scrapy de ejecutar múltiples Spider mediante un simple script Python la mejor opción.

Es por eso que se investigo la posibilidad de obtener los datos mediante Scrapy, estudiando los diferentes objetos Request proporcionados, hasta llegar a la versión actual con el uso de FormRequest.

5.2.3 MeteoNavarra

Code Spider

La estructuración HTML de la web y la falta de experiencia son el motivo principal asociado a la complejidad de esta Spider, la mayor parte del tiempo a la hora de crearla ha sido invertido explorando el código HTML buscando la forma más optima de lograr los datos necesarios. Son múltiples las iteraciones sufridas hasta llegar a la versión actual.

```
1 rows = response.css('div#tabAUTO script::text').getall()
```

Código 5.20: Selector en parse() de MeteoNavarra Code Spider

Del selector se obtiene el código JavaScript perteneciente a cada fila de la tabla, siendo la forma mas sencilla de lograr tanto el nombre como el código de la estación.

```
1 for row in rows:
2     yield {
3         'estacion': row.split(',')[3],
4         'codigo': row.split(',')[0].split('(')[1],
5     }
```

Código 5.21: Guardado de datos de MeteoNavarra Code Spider

A la hora de guardar los datos, se itera por filas y se filtra del código aquellos deseados.

Data Spider

La necesidad de recorrer múltiples estaciones, obliga como en el resto de Spiders de obtención de datos, a sobre escribir la función start_requests().

```
1 import datetime
2
3     current_date = datetime.date.today()
4     delta = datetime.timedelta(days=1)
5     tomorrow_date = current_date + delta
6     yesterday_date = current_date - delta
7     tomorrow_date_format = tomorrow_date.strftime("%d%2F
8         %m%2F%Y").replace(' 0', '')
9     yesterday_date_format = yesterday_date.strftime("%d%2F
10        %m%2F%Y").replace(' 0', '')
```

Código 5.22: Uso de fechas en función start_requests() MeteoNavarra Data Spider

La curiosidad en esta radica en que, a diferencia del resto de páginas, que definen automáticamente una franja de fechas a mostrar, meteoNavarra fuerza la necesidad de que el usuario elija las fechas que desee ver, es por eso que hace uso de la librería datetime, pues facilita el trabajo con fechas.

Con la intención de que funcione proporcionando fechas distintas cada día se realiza el siguiente proceso. Se obtiene la fecha actual con la función today(), con el método timedelta(days=1) se crea un objeto timedelta que representa una duración de un día, gracias a él, se calcula el día de mañana y el de ayer, sumando y restando esa duración al día de hoy. Para terminar, puesto que la fecha debe ir incluida en una URL, mediante el método strftime(), formateamos los datetime calculados para que correspondan con el codificado URL.

```
1 rows = response.css('table.border tr:not([bgcolor*="#FFFFFF"])')
2 estacion = response.css('table a::attr(href)')[7].get()
```

Código 5.23: Selector en parse() de MeteoNavarra Data Spider

De los siguientes selectores cabe mencionar el primero. Aunque en la web se hace uso del elemento HTML table, este no indica que elementos tr pertenecen a la cabecera o al cuerpo de la tabla con el uso de thead y tbody, por el contrario, las filas correspondientes a la cabecera se muestran con el color de fondo representado hexadecimalmente #FFFFFF. Es por eso que se indica explícitamente no obtener esas filas con el uso de tr:not([bgcolor*="#FFFFFF"]).

```

1  datos = []
2  for row in rows:
3      dato = {
4          'fecha y hora':
5              row.xpath('./td[1]/text()').get().strip().replace(' ', ' ') +
6                  ':00',
7          'temperatura (C)': row.xpath('./td[2]/font/text()').get(),
8          'humedad relativa (%)': row.xpath('./td[3]/font/text()').get(),
9          'radiacion global (W/m^2)':
10             row.xpath('./td[4]/font/text()').get(),
11          'precipitacion (l/mm^2)': row.xpath('./td[5]/font/text()').get(),
12      }
13
14      if dato['radiacion global (W/m^2)'] == '- -':
15          dato['radiacion global (W/m^2)'] = None
16
17      if dato['precipitacion (l/mm^2)'] != '- -':
18          datos.append(dato)

```

Código 5.24: Trabajar sobre los datos de MeteoNavarra Data Spider

En la función parse(), como en otras Spider, se crea una lista datos vacía que es llenada con los objetos JSON obtenidos al recorrer cada fila de las anteriormente obtenidas y, filtrar mediante selectores aquellos datos de utilidad.

Luego, se comprueba la disponibilidad de un valor numérico de radiación global, pues en caso de no existir, la web proporciona '- -'. Posteriormente, puesto que la web muestra todas las franjas horarias dentro del que se corresponde con el día actual, se eliminan todas esas horas sin datos con la comprobación respecto a la precipitación.

```

1  if datos:
2      yield {
3          'estacion': estacion.split('idestacion=')[1].split('&')[0],
4          'datos': datos,
5      }

```

Código 5.25: Comprobacion exitencia de datos y guardado de MeteoNavarra Data Spider

A continuación, en caso de que existan datos a guardar, pues algunas de las estaciones puedes llegar a no disponer de datos, se realiza el yield.

```

1 next_page = response.css("table a::attr(href)").getall()
2 page_number = response.xpath("//b/text()").getall()
3 if not page_number[0] == page_number[1]:
4     next_page = response.urljoin(next_page[7])
5     yield scrapy.Request(next_page, callback=self.parse)

```

Código 5.26: Navegacion a segunda página de datos en MeteoNavarra Data Spider

Finalmente, aun dentro de parse(), como los datos de ciertas estaciones están repartidos en dos páginas, adquiere la posible URL a la segunda página y, el numero de la página actual y siguiente, en caso de que los números sean distintos, navega a esa segunda página para realizar el mismo proceso descrito anteriormente.

Esto es posible gracias a que yield, aunque a groso modo funcione como un return, no fuerza el final de la ejecución, por lo que todo código por debajo de este sera ejecutado, llegando a haber múltiples yields en una misma función.

Coordenates Spider

Por último, necesitamos de una Spider para la obtención de las coordenadas pues no están disponibles para obtener dentro de las páginas anteriores, aunque sí dentro del mismo dominio.

Mediante start_requests(), realiza tantas llamadas como estaciones.

```

1 coordenadas = response.css('td::text')[19].get()
2 estacion = response.css('input::attr(value)').get()

```

Código 5.27: Selector en parse() de MeteoNavarra Coordenates Spider

Una vez en la función parse(), toma las coordenadas mostradas en una tabla y, obtiene el código de la estación del atributo value de un input.

```

1 yield {
2     'coordenadas': coordenadas.strip().replace('\r\n\t\t', ' ')
3         .replace('(*)', ''),
4     'estacion': estacion,
5 }

```

Código 5.28: Guardado de datos de MeteoNavarra Coordenates Spider

Finalmente, las coordenadas, con el fin de verse bien en la web, en vez de estar estilizadas con CSS, vienen estilizadas con elementos textuales, los cuales son eliminados.

5.2.4 Agua en Navarra

Las Spiders de Agua en Navarra son las más complejas entre todas, usando una filosofía ligeramente distinta al resto de Spiders y, por el uso de Selenium para poder interactuar con la web mediante JavaScript.

Ambas Spider, Code y Data, parten de las misma start_urls y van recorriendo las páginas presentes hasta llegar a aquella que muestre los datos deseados. Esto se debe hasta cierto punto a la estructuración de la página, haciendo más sencillo navegar por ella que crear una Spider por cada página a visitar. La navegación por la web se realiza usando múltiples funciones de la misma naturaleza que parse().

Para Data Spider esto implica no necesitar de un fichero JSON del cual leer los códigos de las estaciones.

```

1 def parse(self, response):
2     for link in response.css('dl#navarramap a::attr(href)'):
3         if link.get() != 'ctaMapa.aspx?IdMapa=1&IDOrigenDatos=1':
4             yield response.follow(link.get(), callback=self.parse_area)

```

Código 5.29: Función parse() Agua en Navarra Spiders

La función parse() es compartida por ambas Spider. Extrae los link representantes de cada área definida sobre el mapa y, navega por todos ellos a excepción del link de origen, aquel en el que se encuentra.

El uso de response.follow() es equivalente a realizar un Request, pero no necesita pararle una URL completa, solo con la ruta es capaz de generar la llamada, en caso de necesitar explícitamente el uso de Request esta sería la manera.

```
yield Request(url=response.urljoin(link.get()), callback=self.parse_area)
```

De ambas formas, en el argumento callback se indica a qué función debe realizar la llamada, pasando de parse() a parse_area().

```

1 def parse_area(self, response):
2     for link in response.css('area[shape="rect"]::attr(href)'):
3         yield response.follow(link.get(), callback=self.parse_estacion)

```

Código 5.30: Función parse_area() Agua en Navarra Spiders

Siendo la última función implementada de la misma manera en las Spider. Se realiza el mismo proceso, obtiene todos los link a las webs, ya sí, de la estación y, llama a parse_estacion().

Code Spider

```

1 def parse_estacion(self, response):
2     urls = response.xpath('//span/a/@href').getall()
3     estacion = response.css('div#bloq_iconos span span::text').getall()
4     codigoEstacion =
5         response.css("form#frmDatosEstacion::attr(action)").get()
6
7     codigos = []
8     for url in urls:
9         codigos.append(url.split('=')[1])

```

Código 5.31: Función parse_estacion() Agua en Navarra Code Spider

Una vez en la función parse_estacion() de Code Spider, como ocurre con CHCantábrico, la estación dispone de múltiples códigos, aquel que referencia la estación, uno para el nivel y aunque no en todas, otro para el caudal del río.

Tanto el código de nivel y caudal del río originalmente se almacenan en la variable urls, almacenando las direcciones en las cuales están presentes. Posteriormente, se toma de la URL y se guardan en la lista codigos.

La variable estación es una lista que almacena, la descripción de la estación, el municipio al que pertenece, el río y, las coordenadas.

```

1 yield {
2     'descripcion': estacion[0],
3     'municipio': estacion[1],
4     'rio': estacion[2],
5     'coordenadas': estacion[3],
6     'estacion': codigoEstacion.split('=')[-1],
7     'codigos': codigos,
8 }

```

Código 5.32: Guardado de datos de Agua en Navarra Code Spider

Al final estos datos son almacenados.

Data Spider

```

1 from scrapy_selenium import SeleniumRequest
2 from selenium.webdriver.common.by import By
3 from selenium.webdriver.support import expected_conditions as EC

```

```

4
5     def parse_estacion(self, response):
6         for link in response.css('div#divResultadosAforo a::attr(href)'):
7             yield SeleniumRequest(
8                 url=response.urljoin(link.get()),
9                 wait_time=2,
10                wait_until=EC.element_to_be_clickable((By.ID,
11                                           'btnDatosNumericos')),
12                callback=self.parse_data,
13                script='document.querySelector("#btnDatosNumericos").click()',
14            )

```

Código 5.33: Función parse_estacion() Agua en Navarra Data Spider

Lo primero que llama la atención de esta Spider respecto al resto, es el uso de SeleniumRequest, este tipo de objeto Request pertenece a la librería scrapy_selenium, siendo una alternativa sencilla de unificar las funcionalidades de Selenium en Scrapy, de esta forma, se lida con la falta de integración de interacción con JavaScript en Scrapy.

Tras realizar el mismo proceso que antes, la Spider llega a la función parse_estacion(), esta vez, en vez de obtener los datos de la estación, accedemos a los links que permiten mostrar los datos mediante SeleniumRequest y llama a la función parse_data().

Como el link proporcionado dirige a una página que muestra los datos en forma de gráfica y, para acceder a los datos numéricos es necesario pulsar un botón, SeleniumRequest se configura de tal forma que, espera a que este esté correctamente cargado con el argumento wait_until, indicando que elemento deseas esperar a ser cargado; en caso de que no se cargue, es recomendable usar el argumento wait_time, que usado junto al anterior, establece el tiempo antes de dar la llamada por errónea; si el botón se carga correctamente, en el argumento script indicamos que se debe pulsar sobre este, llevándonos finalmente a los datos numéricos.

```

1     tipo = response.css('span#lblSenal::text').get()
2     estacion = response.css('li#cabeecera_nombreEstacion
3                               a::attr(href)').get()
4     fechas = response.css('span.cont_fecha_gra::text').getall()
5     valores = response.css('span.cont_valor_gra::text').getall()

```

Código 5.34: Selector en parse_data() de Agua en Navarra Data Spider

En la función parse_data(), se obtienen los datos por columna en vez de fila, pues la web no hace un correcto uso formateo por tabla, pues no existe esta tabla, insertando los datos sobre un elemento div y formateandolos posteriormente con CSS. Esta forma de mostrar los datos fuerza tener que obtener las fechas por un lado y los valores por otro, resultando en dos listas que debemos unir. La variable tipo (Nivel o caudal) es necesaria para posteriormente guardar los datos de forma correcta.

```
1  datos = []
2  if tipo == "Nivel Rio":
3      for i, fecha in enumerate(fechas):
4          dato = {
5              'fecha y hora': fecha.strip() + ':00',
6              'nivel (m)': valores[i].strip(),
7          }
8          datos.append(dato)
9  else:
10     for i, fecha in enumerate(fechas):
11         dato = {
12             'fecha y hora': fecha.strip() + ':00',
13             'caudal (m^3/s)': valores[i].strip(),
14         }
15         datos.append(dato)
16
17     yield {
18         'estacion': estacion.split('=')[-1],
19         'datos': datos,
20     }
```

Código 5.35: Selector en parse_data() de Agua en Navarra Data Spider

Al hacer uso de una Spider para obtener todos los datos, estando presentes en distintas páginas, hace necesaria la comprobación inicial con la variable tipo de que datos estamos recibiendo, para poder almacenarlos correctamente. Una vez realizada, se itera la lista de fechas y se unen con su respectivo valor (el orden de los valores corresponde con el de las fechas) y se insertan el la lista datos. Finalmente son guardados.

Configuración Selenium

Para que Selenium funcione, es necesario incluir las siguientes líneas de código en el archivo settings.py generado por Scrapy.

```
1  from shutil import which
2
```

```

3  SELENIUM_DRIVER_NAME = 'chrome'
4  SELENIUM_DRIVER_EXECUTABLE_PATH = which('chromedriver')
5  SELENIUM_DRIVER_ARGUMENTS = [ '--headless',
6    '--disable-logging',
7    '--disable-in-process-stack-traces',
8    '--log-level=1',
9    '--disable-extensions'
10 ]
11
12 DOWNLOADER_MIDDLEWARES = {
13     'scrapy_selenium.SeleniumMiddleware': 800
14 }

```

Código 5.36: Agua en Navarra configuración Selenium

En este caso se indica que el navegador deseado para realizar el trabajo es Chrome y mediante el método which selecciona la ruta del ejecutable del driver de chrome.

Los argumentos usados indican, –headless significa que las instancias de Chrome creadas sean sin interfaz, de esta forma no tendremos cientos de ventanas de Chrome abriéndose y cerrándose cada vez que se ejecute la Spider; –disable-logging como su nombre indica elimina el Log; –disable-in-process-stack-traces desactiva el stack-trace, esto se realiza únicamente con el fin de intentar mejorar el rendimiento; –log-level indica el tipo de log que se desea mostrar, siendo INFO = 0, ALERTA = 1, ERROR = 2 y FATAL = 3, aunque el Log este desactivado esta bien indicar que clase de Log deseas que aparezca en caso de que se vuelva a activar; –disable-extensions desactiva todas las extensiones que podamos tener instaladas, nuevamente con el fin de mejorar el rendimiento.

Una vez incluido el middleware, ya solo quedaría descargar el driver de chrome.

Chromedriver.exe puede ser obtenido en la web <https://chromedriver.chromium.org/downloads>, descargando aquel que corresponda a la versión de Chrome instalada. Una vez descomprimido debe ser incluido dentro de la carpeta de proyecto de Scrapy.

En caso de estar en Windows con esto ya es suficiente, aunque al usar Debian son necesarios unos paso más para configurar chromedriver.

```

#Otorgamos permiso de ejecucion a chromedriver
sudo chmod +x chromedriver

#Movemos el .exe a directorio /usr/local/share/
sudo mv chromedriver /usr/local/share/chromedriver

```

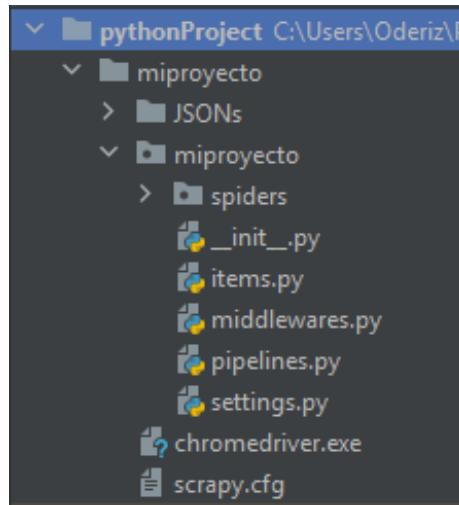


Figura 5.5: Ruta chromedriver.exe

```
#Creamos la relacion entre el .exe y el directorio
sudo ln -s /usr/local/share/chromedriver /usr/bin/chromedriver

#Ejecutamos
chromedriver --version

#En caso de haber realizado correctamente el proceso deberia aparecer
una linea parecida a esta
ChromeDriver 115.0.5790.110
(5e87dfef0c85687ea835e444d33466745cc0725f-refs/branch-heads/5790_90@{#20})
```

Hecho esto ya estaría configurado chromedriver.

Alternativas probadas

Antes de saber que es necesario dirigirse a la pagina donde se muestra la gráfica para poder acceder a la página con los datos numéricos, se probó con una Spider que accedía directamente a esta. Visto que el resultado obtenido estaba vacío y, que en apariencia la Spider era correcta, obteniendo teóricamente aquellos datos deseados, se comprobó la web manualmente, resultando en la obtención del error mostrado en la imagen [2.15](#).

Es por eso que, partiendo de un uso de Scrapy sin uso de extensiones de terceros, se probó una solución al problema que seguía la metodología de recorrer las webs,

de esta manera, se visita inicialmente la web con la gráfica para posteriormente redirigirse a los datos numéricos. Resultando en un nuevo fracaso.

Esta vez si que se obtenían datos, aunque no de forma correcta, muchas estaciones aparecían repetidas múltiples veces, de tres a siete veces en el peor de los casos visto. Siendo el resultado esperado la aparición de una misma estación un total de dos veces, una con los datos del caudal y otra por los datos del nivel. Tras volver a la comprobación manual, resultó en el mismo comportamiento, por alguna razón, una vez dentro de la web de los datos numéricos, al cambiar el código de la estación y recargar la página, lo mismo se actualizaban los datos para mostrar los de la nueva estación como no lo hacían, mostrando los datos de la anterior.

Esto causó la incertidumbre de si era posible obtener datos la web de forma fiable. Finalmente, tras barajar la posibilidad de pulsar el botón para acceder los datos, se dio con la herramienta usada en la versión final, Selenium y, tiempo después con una alternativa más moderna a esta, Playwright.

Por suerte, Selenium, al ser usado mediante la extensión `scrapy_selenium`, resultó funcionar a la primera, aunque a un costo temporal relativamente alto, en comparación con las demás Spider, con una media de dos minutos y medio de ejecución.

Al ser código destinado a ejecutarse en intervalos de quince minutos, un tiempo de ejecución así no debería acarrear ningún problema pero, aun y todo, se probó una cuarta alternativa mediante Playwright, al prometer mejoras en los tiempos de ejecución y una mejor optimización.

Playwright

Antes que nada, cabe mencionar que Playwright no es compatible con Windows y, tampoco lo es con Debian 11 en su versión actual, resultando ser Debian 11 la versión usada para este proyecto, por lo que no se ha podido llegar a comprobar el funcionamiento del siguiente código, aunque debería ser correcto.

```
#Instalamos scrapy-playwright
pip install scrapy-playwright

#Instalamos los navegadores compatibles
playwright install
```

Para hacer uso de Playwright con Scrapy, primero que todo es configurar las dependencias de Playwright, implementándolas mediante `custom_settings` en una Spider o,

si se desea, también pueden ser configuradas en settings.py.

```
1 custom_settings = {
2     "TWISTED_REACTOR":
3         "twisted.internet.asyncioreactor.AsyncioSelectorReactor",
4     "DOWNLOAD_HANDLERS": {
5         "https":
6             "scrapy_playwright.handler.ScrapyPlaywrightDownloadHandler",
7         "http":
8             "scrapy_playwright.handler.ScrapyPlaywrightDownloadHandler",
9     }
10 }
```

Código 5.37: Configuración Playwright

Finalmente, para hacer uso de la herramienta, a diferencia de con Selenium, no dispone de un nuevo objeto Request, si no que hace uso del Request implementado en Scrapy indicándole por argumento que debe usar Playwright.

```
1 yield Request(
2     url=url,
3     callback=self.parse,
4     meta=dict(
5         playwright=True,
6     ),
7 )
```

Código 5.38: Playwright basic Request

Para obtener el comportamiento previsto, la Request configurada es la siguiente.

```
1 from scrapy_playwright.page import PageMethod
2
3
4 yield Request(
5     url=url,
6     callback=self.parse,
7     meta=dict(
8         playwright=True,
9         playwright_page_methods=[
10             PageMethod("wait_for_selector", selector="div.botoneraGrafico",
11                         state="visible"),
12             PageMethod("click", selector="input#btnDatosNumericos"),
13             PageMethod("waitForEvent", event="click"),
14         ]
15     )
16 )
```

```

13     ],
14   ),
15 )

```

Código 5.39: Agua en Navarra Playwright Request

Los PageMethods indican las acciones a realizar una vez se hace la request, la alternativa optada por Playwright al uso de JavaScript. Primero, espera a que el botón esté cargado dentro de la página, pulsa sobre el y, espera a que la acción de pulsar sea realizada.

Según la documentación y los ejemplos estudiados, esta alternativa debería funcionar, pero puesto que no se ha podido ejecutar no es seguro su funcionamiento.

5.3 Formateo de datos

Para formatear los datos de forma global, cada proyecto individual dispone de dos scripts encargados de hacerlo, uno para los códigos y otro para los datos per se.

5.3.1 Aemet

5.3.2 CHCantábrico

5.3.3 MateoNavarra

5.3.4 Agua en Navarra

5.4 Filtrado de datos

Tras formatear los ficheros JSON, estos se pasan por el script `filtras_datos_JSONs.py`.

En el se realiza lo siguiente.

```

1 import json
2 from datetime import date, datetime
3 from pathlib import Path

```

Código 5.40: Import necesarios

Se importan las dependencias.

```

1 OldDir = "JSONs/OldData/"
2 ParsedDir = "JSONs/ParsedData/"

```

```

3   RefinedDir = "JSONs/RefinedData/"
4   DataJSON = "datos_aemet.json"

```

Código 5.41: Declaración rutas JSONs y nombre de fichero

Se declaran las variables representativas de las rutas y el nombre del fichero JSON deseado.

```

1 def openFile(fileDir):
2     try:
3         with open(fileDir + DataJSON, "r", encoding="utf-8") as f:
4             file = json.loads(f.read())
5     except FileNotFoundError:
6         file = None
7     return file

```

Código 5.42: Declaración función openFile()

La función prueba a abrir el fichero anteriormente declarado (datos_aemet.json) sobre la ruta indicada (fileDir), en caso de no existir, se marca el fichero como None, finalmente devuelve el fichero. El tratado de la excepción FileNotFoundError se realiza pensada en la primera vez que se haga uso de la plataforma, ya que esta no dispondría de datos posteriores con los que hacer una comparación.

```

1 def saveData(jsonDir, DataFile):
2     Path(jsonDir).mkdir(parents=True, exist_ok=True)
3     with open(jsonDir + DataJSON, 'w', encoding='utf-8') as outfile:
4         json.dump(DataFile, outfile)

```

Código 5.43: Declaración función saveFile()

Recibiendo como argumento el directorio sobre el que se desea guardar el fichero (jsonDir) y un diccionario JSON con los datos a guardar (DataFile), esta función, comproueba la existencia del directorio y en caso de no hacerlo lo crea. Tras ello, guarda los datos en el directorio con el uso de json.dump().

```

1 def refineData(newFile, oldFile):
2     refinedFile = []
3
4     for i, item in enumerate(newFile):
5         newData = []
6         for data in item["datos"]:
7             if data["fecha y hora"] not in [x["fecha y hora"] for x in
8                 oldFile[i]["datos"]]:
9                 newData.append(data)

```

```

9   if newData:
10    refinedFile.append(
11      {
12        "coordenadas": item["coordenadas"],
13        "estacion": item["estacion"],
14        "dato": newData
15      }
16    )
17
18  saveData(RefinedDir, refinedFile)

```

Código 5.44: Declaración función refinedData()

Esta es la encargada de la comparación de los ficheros, recibidos en los argumentos newFile y oldFile.

Empezando con la explicación de arriba a abajo, se define la lista vacía refinedFile, encargada de almacenar los datos nuevos de todas las estaciones; se itera por cada estación almacenada en newFile; se define una segunda lista newData encargada de almacenar los nuevos datos por estación; se itera por todos los datos de los presentes en la estación; se comprueba que no exista previamente en los datos de esa misma estación en oldFile; en caso de que el dato no este presente, se almacena en newData; en caso de que exista algún dato nuevo (newData no esta vacía), se crea un objeto JSON siguiendo la estructuración definida para los ficheros de datos y, se almacena en refinedData; finalmente los datos son guardados como fichero en la ruta especificada RefinedDir.

```

1 def main():
2   newFile = openFile(ParsedDir)
3   oldFile = openFile(OldDir)
4
5   if oldFile is None:
6     saveData(RefinedDir, newFile)
7     return
8
9   refineData(newFile, oldFile)

```

Código 5.45: Declaración rutas JSONs

La función main(), abre los ficheros indicados en las rutas ParsedDir y OldDir, en caso de que oldFile sea None, guarda los nuevos datos directamente en el directorio refinedDir y fuerda el fin del programa.

6

Conclusiones y Trabajo Futuro

Referencias

- [1] "United Nations what is climate change." <https://www.un.org/es/climatechange/what-is-climate-change>. Accessed: 2023-09-19.
- [2] W. F. Ruddiman, "The anthropogenic greenhouse era began thousands of years ago," *Climatic change*, vol. 61, no. 3, pp. 261–293, 2003.
- [3] "National Centers for Environmental Information annual 2022 global climate report." <https://www.ncei.noaa.gov/access/monitoring/monthly-report/global/202213>. Accessed: 2023-09-19.
- [4] N. W. Arnell, J. A. Lowe, A. J. Challinor, and T. J. Osborn, "Global and regional impacts of climate change at different levels of global temperature increase," *Climatic Change*, vol. 155, pp. 377–391, 2019.
- [5] M. New, D. Liverman, H. Schroder, and K. Anderson, "Four degrees and beyond: the potential for a global temperature increase of four degrees and its implications," 2011.
- [6] T. H. Sparks and A. Menzel, "Observed changes in seasons: an overview," *International Journal of Climatology: A Journal of the Royal Meteorological Society*, vol. 22, no. 14, pp. 1715–1725, 2002.
- [7] "National Centers for Environmental Information annual 2022 global climate report precipitation." <https://www.ncei.noaa.gov/access/monitoring/monthly-report/global/202213#precip>. Accessed: 2023-09-19.
- [8] C. Wasko, S. Westra, R. Nathan, H. G. Orr, G. Villarini, R. Villalobos Herrera, and H. J. Fowler, "Incorporating climate change in flood estimation guidance," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2195, p. 20190548, 2021.

- [9] "Ministerio para la transformación ecológica y el reto demográfico gestión de los riesgos de inundación." <https://www.miteco.gob.es/es/agua/temas/gestion-de-los-riesgos-de-inundacion.html>. Accessed: 2023-09-20.
- [10] C. Dierbach, "Python as a first programming language," *Journal of Computing Sciences in Colleges*, vol. 29, no. 3, pp. 73–73, 2014.
- [11] "Python 3.11.3 Documentation general python faq." <https://docs.python.org/3/faq/general.html>. Accessed: 2023-04-07.
- [12] L. E. Borges, *Python para desenvolvedores: aborda Python 3.3*. Novatec Editora, 2014.
- [13] M. F. Krafft, *The Debian system: concepts and techniques*. No Starch Press, 2005.
- [14] R. Hertzog and R. Mas, *The Debian Administrator's Handbook: Debian Jessie From Discovery To Mastery*. Freexian, 2015.
- [15] "Debian Project Documentation a brief history of debian." <https://www.debian.org/doc/manuals/project-history/index.en.html>. Accessed: 2023-04-12.
- [16] "Debian Project Documentation our philosophy: Why we do it and how we do it." <https://www.debian.org/intro/philosophy.en.html>. Accessed: 2023-04-12.
- [17] R. P. Pollei, *Debian 7: System Administration Best Practices*. Packt Publishing, 2013.
- [18] "Debian Project Documentation what does free mean?." <https://www.debian.org/intro/free.en.html>. Accessed: 2023-04-12.
- [19] "GNU About Free Software Documentation what is free software?." <https://www.gnu.org/philosophy/free-sw.en.html>. Accessed: 2023-04-12.
- [20] "Debian Project Documentation debian social contract version 1.2 ratified on october 1st, 2022.." https://www.debian.org/social_contract.en.html. Accessed: 2023-04-12.
- [21] "Debian Project Documentation what is the partners program?." <https://www.debian.org/partners/index.en.html>. Accessed: 2023-04-12.
- [22] "Debian Project Documentation how to donate to the debian project." <https://www.debian.org/donations.en.html>. Accessed: 2023-04-12.
- [23] "Debian Project Documentation reasons to use debian." https://www.debian.org/intro/why_debian.en.html. Accessed: 2023-04-12.

- [24] D. Ghimire, "Comparative study on python web frameworks: Flask and django," 2020.
- [25] "Python Documentation web frameworks for python." <https://wiki.python.org/moin/WebFrameworks>. Accessed: 2023-04-12.
- [26] D. Glez-Peña, A. Lourenço, H. López-Fernández, M. Reboiro-Jato, and F. Fdez-Riverola, "Web scraping technologies in an api world," *Briefings in bioinformatics*, vol. 15, no. 5, pp. 788–797, 2014.
- [27] "Django Documentation design philosophies." <https://docs.djangoproject.com/en/3.0/misc/design-philosophies/>. Accessed: 2023-04-12.
- [28] M. Alchin, *Pro Django*. Apress, 2013.
- [29] A. Ravindran, *Django Design Patterns and Best Practices*. Packt Publishing Ltd, 2015.
- [30] B. Zhao, "Web scraping," *Encyclopedia of big data*, pp. 1–3, 2017.
- [31] V. Krotov and L. Silva, "Legality and ethics of web scraping," 2018.
- [32] H. Yang, "Design and implementation of data acquisition system based on scrapy technology," in *2019 2nd International Conference on Safety Produce Informatization (IICSPI)*, pp. 417–420, IEEE, 2019.

Anexos

```
1 import scrapy
2
3
4 class AemetCodeSpider(scrapy.Spider):
5     name = "aemet_code_spider"
6     allowed_domains = ["www.aemet.es"]
7     start_urls =
8         ["https://www.aemet.es/es/eltiempo/observacion/ultimosdatos?k=nav&w=0&datos=d"]
9     custom_settings = {
10         'FEEDS': {
11             'JSONs/RawCode/codigos_aemet.json': {
12                 'format': 'json',
13                 'encoding': 'utf-8',
14                 'overwrite': True,
15             }
16         }
17     }
18
19     def parse(self, response):
20         rows = response.css("div#contenedor_tabla tbody tr a")
21
22         for row in rows:
23             path = row.xpath("@href").get()
24             name = row.xpath("./text()").get()
25             yield {
'estacion': name,
```

```
26         'codigo': path.split('&')[1].split('=')[1],  
27     }
```

Código A1: Aement Code Spider

```
1 import json  
2  
3 import scrapy  
4  
5  
6 class AemetDataSpider(scrapy.Spider):  
7     name = "aemet_data_spider"  
8     allowed_domains = ["www.aemet.es"]  
9     custom_settings = {  
10         'FEEDS': {  
11             'JSONs/RawData/datos_aemet.json': {  
12                 'format': 'json',  
13                 'encoding': 'utf-8',  
14                 'overwrite': True,  
15             }  
16         }  
17     }  
18  
19     def start_requests(self):  
20         with open("JSONs/RawCode/codigos_aemet.json", encoding="utf-8")  
21             as f:  
22                 data = json.load(f)  
23                 for estacion in data:  
24                     url =  
25                         f'https://www.aemet.es/es/eltiempo/observacion/ultimosdatos?k=nav&l={estacion}'  
26                     yield scrapy.Request(url, self.parse)  
27  
28     def parse(self, response):  
29         latitud = response.css('abbr.latitude::text').get()  
30         longitud = response.css('abbr.longitude::text').get()  
31         estacion = response.css("a.separador_pestanas").get()  
32         rows = response.css('tbody tr')  
33  
34         datos = []  
35         for row in rows:  
36             dato = {  
37                 'fecha y hora': row.xpath('./td[1]/text()').get() +
```

```
36         ':00',
37     'temperatura ( C )': row.xpath('.//td[2]/text()').get(),
38     'humedad (%)': row.xpath('.//td[10]/text()').get(),
39     'precipitacion (mm)': row.xpath('.//td[7]/text()').get(),
40 }
41
42     if dato['precipitacion (mm)'] != " ":
43         datos.append(dato)
44
45     yield {
46         'coordenadas': latitud + ' | ' + longitud,
47         'estacion': estacion.split('=')[3].split('&')[0],
48         'datos': datos,
49     }
```

Código A2: Aement Data Spider

```
1 import scrapy
2
3
4 class ChcantabricoCodeSpider(scrapy.Spider):
5     name = "chcantabrico_code_spider"
6     allowed_domains = ["www.chcantabrico.es"]
7     start_urls = ["https://www.chcantabrico.es/nivel-de-los-rios"]
8     custom_settings = {
9         'FEEDS': {
10             'JSONs/RawCode/codigos_chcantabrico.json': {
11                 'format': 'json',
12                 'encoding': 'utf-8',
13                 'overwrite': True,
14             }
15         }
16     }
17
18     def parse(self, response):
19         rows = response.xpath('//table[@class="tablefixedheader
20         niveles"]/tbody/tr')
21
22         for row in rows:
23             codigoBusqueda = row.css('td.codigo::text').get()
24             limites = row.css('table.umbrales_gr
25                 td.datos::text').getall()
```

```

24     paths = row.xpath('.//td/a/@href').getall()
25     estaciones = row.xpath('.//td/a/text()').getall()
26
27     for i in range(len(lmites)):
28         if lmites[i] == 'No definido':
29             lmites[i] = None
30
31     yield {
32         'estacion': estaciones[-3],
33         'codigo': paths[-1].split("=".split("-1"),
34         'codigoSecundario': codigoBusqueda,
35         'seguimiento': lmites[0],
36         'prealerta': lmites[1],
37         'alerta': lmites[2],
38     }

```

Código A3: CHCantabrico Code Spider

```

1 import io
2 import json
3
4 import pandas as pd
5 import scrapy
6
7
8 class ChcantabricoNivelSpider(scrapy.Spider):
9     name = "chcantabrico_nivel_spider"
10    allowed_domains = ["www.chcantabrico.es"]
11    custom_settings = {
12        'FEEDS': {
13            'JSONs/RawData/datos_nivel_chcantabrico.json': {
14                'format': 'json',
15                'encoding': 'utf-8',
16                'overwrite': True,
17            }
18        }
19    }
20
21    def start_requests(self):
22        with open("JSONs/RawCode/codigos_chcantabrico.json",
23                  encoding="utf-8") as f:
24            data = json.load(f)

```

```
24     for estacion in data:
25         params_nivel = {
26             'p_p_id': 'GraficaEstacion_INSTANCE_wH0LL6jTUysu',
27             'p_p_lifecycle': '2',
28             'p_p_state': 'normal',
29             'p_p_mode': 'view',
30             'p_p_resource_id': 'downloadCsv',
31             'p_p_cacheability': 'cacheLevelPage',
32             '_GraficaEstacion_INSTANCE_wH0LL6jTUysu_cod_estacion':
33                 f'{estacion["codigo"]}',
34             '_GraficaEstacion_INSTANCE_wH0LL6jTUysu_tipodato':
35                 'nivel',
36         }
37         url = 'https://www.chcantabrico.es/evolucion-de-niveles'
38         yield scrapy.FormRequest(url=url,
39             method='GET',
40             formdata=params_nivel,
41             callback=self.parse,
42             cb_kwargs={'estacion':
43                 estacion['codigo']})
44
45
46
47
48
49
50
51
52
53
```

Código A4: CHCantabrico Nivel Spider

```
1 import io
2 import json
3
4 import pandas as pd
5 import scrapy
```



```
42
43     def parse(self, response, estacion):
44         if not response.text.startswith('-'):
45             urlData = response.text
46             rawData = pd.read_csv(io.StringIO(urlData), delimiter=';',
47                                   encoding='utf-8', header=1)
48             rawData.columns = ['fecha y hora', 'nivel (m)']
49             parsedData = rawData.to_json(orient="records")
50
51             yield {
52                 'estacion': estacion,
53                 'datos': json.loads(parsedData)
54             }
```

Código A5: CHCantabrico Pluvio Spider

```
1 import json
2
3 import scrapy
4
5
6 class ChcantabricoCoordSpider(scrapy.Spider):
7     name = "chcantabrico_coord_spider"
8     allowed_domains = ["ceh.cedex.es"]
9     custom_settings = {
10         'FEEDS': {
11             'JSONs/RawData/coordenadas_chcantabrico.json': {
12                 'format': 'json',
13                 'encoding': 'utf-8',
14                 'overwrite': True,
15             }
16         }
17     }
18
19     def start_requests(self):
20         with open("JSONs/RawCode/codigos_chcantabrico.json",
21                   encoding="utf-8") as f:
22             data = json.load(f)
23             for estacion in data:
24                 url =
25                     f'https://ceh.cedex.es/anuarioaforos/af0/estaf-datos.asp?indroeae
26
27                 yield scrapy.Request(url, self.parse)
```

```

25
26     def parse(self, response):
27         longitud = response.css('p::text')[6].get().strip()
28         latitud = response.css('p::text')[7].get().strip()
29         estacion = response.css('font::text')[14].get().strip()
30
31         yield {
32             'coordenadas': f'Lat: {latitud} | Lon: {longitud}',
33             'estacion': estacion,
34         }

```

Código A6: CHCantabrico Coord Spider

```

1 import scrapy
2
3
4 class MeteonavarraCodeSpider(scrapy.Spider):
5     name = "meteoNavarra_code_spider"
6     allowed_domains = ["meteo.navarra.es"]
7     start_urls =
8         ["http://meteo.navarra.es/estaciones/mapadeestaciones.cfm#"]
9     custom_settings = {
10         'FEEDS': {
11             'JSONs/RawCode/codigos_meteoNavarra.json': {
12                 'format': 'json',
13                 'encoding': 'utf-8',
14                 'overwrite': True,
15             }
16         }
17     }
18
19     def parse(self, response):
20         rows = response.css('div#tabAUTO script::text').getall()
21
22         for row in rows:
23             yield {
24                 'estacion': row.split(',')[-1],
25                 'codigo': row.split(',')[0].split('(')[1],
26             }

```

Código A7: MeteoNavarra Code Spider

```
1 import datetime
2 import json
3
4 import scrapy
5
6
7 class MeteonavarraDataSpider(scrapy.Spider):
8     name = "meteoNavarra_data_spider"
9     allowed_domains = ["meteo.navarra.es"]
10    custom_settings = {
11        'FEEDS': {
12            'JSONs/RawData/datos_meteoNavarra.json': {
13                'format': 'json',
14                'encoding': 'utf-8',
15                'overwrite': True,
16            }
17        }
18    }
19
20    def start_requests(self):
21
22        current_date = datetime.date.today()
23        delta = datetime.timedelta(days=1)
24        tomorrow_date = current_date + delta
25        yesterday_date = current_date - delta
26        tomorrow_date_format = tomorrow_date.strftime("%d%/%F
27            %m%/%F%Y").replace(' 0', '')
28        yesterday_date_format = yesterday_date.strftime("%d%/%F
29            %m%/%F%Y").replace(' 0', '')
30
31        with open("JSONs/RawCode/codigos_meteoNavarra.json",
32                  encoding="utf-8") as f:
33            data = json.load(f)
34            for estacion in data:
35                url =
36                    f'http://meteo.navarra.es/estaciones/estacion_datos_m.cfm?IDEstac
37
38                    \f'=1&p_10=2&p_10=3&p_10=11&fecha_desde={yesterday_date_format}'
39                yield scrapy.Request(url, self.parse)
```

```

36     def parse(self, response):
37         rows = response.css('table.border tr:not([bgcolor="#FFFFFF"])')
38         estacion = response.css('table a::attr(href)')[7].get()
39
40         datos = []
41         for row in rows:
42             dato = {
43                 'fecha y hora':
44                     row.xpath('./td[1]/text()').get().strip().replace(' ', '',
45                                         1) + ':00',
46                 'temperatura ( C )':
47                     row.xpath('./td[2]/font/text()').get(),
48                 'humedad relativa (%)':
49                     row.xpath('./td[3]/font/text()').get(),
50                 'radiacion global (W/m^2)' :
51                     row.xpath('./td[4]/font/text()').get(),
52                 'precipitacion (l/mm^2)' :
53                     row.xpath('./td[5]/font/text()').get(),
54             }
55
56             if dato['radiacion global (W/m^2)'] == '- -':
57                 dato['radiacion global (W/m^2)'] = None
58
59             if dato['precipitacion (l/mm^2)'] != '- -':
60                 datos.append(dato)
61
62             if datos:
63                 yield {
64                     'estacion':
65                         estacion.split('idestacion=')[1].split('&')[0],
66                     'datos': datos,
67                 }
68
69             next_page = response.css("table a::attr(href)").getall()
70             page_number = response.xpath("//b/text()").getall()
71             if not page_number[0] == page_number[1]:
72                 next_page = response.urljoin(next_page[7])
73                 yield scrapy.Request(next_page, callback=self.parse)

```

Código A8: MeteoNavarra Data Spider

```
1 import json
```

```
2 import scrapy
3
4
5
6 class MeteonavarraCoordSpider(scrapy.Spider):
7     name = "meteoNavarra_coord_spider"
8     allowed_domains = ["meteo.navarra.es"]
9     custom_settings = {
10         'FEEDS': {
11             'JSONs/RawData/coordenadas_meteoNavarra.json': {
12                 'format': 'json',
13                 'encoding': 'utf-8',
14                 'overwrite': True,
15             }
16         }
17     }
18
19     def start_requests(self):
20         with open("JSONs/RawCode/codigos_meteoNavarra.json",
21                    encoding="utf-8") as f:
22             data = json.load(f)
23             for estacion in data:
24                 url =
25                     f'http://meteo.navarra.es/estaciones/estacion.cfm?IDestacion={estacion["id"]}'
26                 yield scrapy.Request(url, self.parse)
27
28     def parse(self, response):
29         coordenadas = response.css('td::text')[19].get()
30         estacion = response.css('input::attr(value)').get()
31
32         yield {
33             'coordenadas': coordenadas.strip().replace('\r\n\t\t', ' '
34                                         ).replace('(*)', ''),
35             'estacion': estacion,
36         }
```

Código A9: MeteoNavarra Coord Spider

```
1 import scrapy
2
3
4 class AguaEnNavarraCodeSpider(scrapy.Spider):
```

```
5     name = "aguaEnNavarra_code_spider"
6     allowed_domains = ["administracionelectronica.navarra.es"]
7     start_urls =
8         ["https://administracionelectronica.navarra.es/aguaEnNavarra/ctaMapa.aspx?IDOrigen=1"]
9     custom_settings = {
10         'FEEDS': {
11             'JSONs/RawCode/codigos_aguaEnNavarra.json': {
12                 'format': 'json',
13                 'encoding': 'utf-8',
14                 'overwrite': True,
15             }
16         }
17     }
18
19     def parse(self, response):
20         for link in response.css('dl#navarramap a::attr(href)'):
21             if link.get() != 'ctaMapa.aspx?IdMapa=1&IDOrigenDatos=1':
22                 yield response.follow(link.get(),
23                                       callback=self.parse_area)
24
25     def parse_area(self, response):
26         for link in response.css('area[shape="rect"]::attr(href)'):
27             yield response.follow(link.get(),
28                                   callback=self.parse_estacion)
29
30     def parse_estacion(self, response):
31         urls = response.xpath('//span/a/@href').getall()
32         estacion = response.css('div#bloq_iconos span
33             span::text').getall()
34         codigoEstacion =
35             response.css("form#frmDatosEstacion::attr(action)").get()
36         codigos = []
37         for url in urls:
38             codigos.append(url.split('=')[1])
39
40             yield {
41                 'descripcion': estacion[0],
42                 'municipio': estacion[1],
43                 'rio': estacion[2],
44                 'coordenadas': estacion[3],
45                 'estacion': codigoEstacion.split('=')[-1],
```

```
41     'codigos': codigos,  
42 }
```

Código A10: Agua en Navarra Code Spider

```
1 import scrapy  
2 from scrapy_selenium import SeleniumRequest  
3 from selenium.webdriver.common.by import By  
4 from selenium.webdriver.support import expected_conditions as EC  
5  
6  
7 class AguaEnNavarraDataSpider(scrapy.Spider):  
8     name = "aguaEnNavarra_data_spider"  
9     allowed_domains = ["administracionelectronica.navarra.es"]  
10    start_urls =  
11        ["https://administracionelectronica.navarra.es/aguaEnNavarra/ctaMapa.aspx?ID0rigenDatos=1"]  
12    custom_settings = {  
13        'FEEDS': {  
14            'JSONs/RawData/datos_aguaEnNavarra.json': {  
15                'format': 'json',  
16                'encoding': 'utf-8',  
17                'overwrite': True,  
18            }  
19        }  
20    }  
21  
22    def parse(self, response):  
23        for link in response.css('dl#navarramap a::attr(href)'):  
24            if link.get() != 'ctaMapa.aspx?IdMapa=1&ID0rigenDatos=1':  
25                yield response.follow(link.get(),  
26                                      callback=self.parse_area)  
27  
28    def parse_area(self, response):  
29        for link in response.css('area[shape="rect"]::attr(href)'):  
30            yield response.follow(link.get(),  
31                                      callback=self.parse_estacion)  
32  
33    def parse_estacion(self, response):  
34        for link in response.css('div#divResultadosAforo  
35                                     a::attr(href)'):  
36            yield SeleniumRequest(  
37                url=response.urljoin(link.get()),
```

```
34         wait_time=2,
35         wait_until=EC.element_to_be_clickable((By.ID,
36             'btnDatosNumericos'))),
37         callback=self.parse_data,
38         script='document.querySelector("#btnDatosNumericos").click()',
39     )
40
41     def parse_data(self, response):
42         tipo = response.css('span#lblSenal::text').get()
43         estacion = response.css('li#cabecera_nombreEstacion
44             a::attr(href)').get()
45         fechas = response.css('span.cont_fecha_gra::text').getall()
46         valores = response.css('span.cont_valor_gra::text').getall()
47
48         datos = []
49         if tipo == "Nivel R o":
50             for i, fecha in enumerate(fechas):
51                 dato = {
52                     'fecha y hora': fecha.strip() + ':00',
53                     'nivel (m)': valores[i].strip(),
54                 }
55                 datos.append(dato)
56         else:
57             for i, fecha in enumerate(fechas):
58                 dato = {
59                     'fecha y hora': fecha.strip() + ':00',
60                     'caudal (m^3/s)': valores[i].strip(),
61                 }
62                 datos.append(dato)
63
64         yield {
65             'estacion': estacion.split('=')[-1],
66             'datos': datos,
67         }
```

Código A11: Agua en Navarra Data Spider