

Idearium Consultores

Informe de entrega 1

DESARROLLO DE UN CATÁLOGO Y UN API DE CONSULTA A LA INFORMACIÓN DE EIDOS MEDIANTE CKAN

ÍNDICE

| | |
|--|----|
| Introducción..... | 3 |
| Detalle de tareas realizadas..... | 3 |
| 1. Análisis preoperacional y toma de requisitos..... | 3 |
| 2. Instalación del entorno y extensiones..... | 4 |
| 3. Creación de modelos de datos..... | 5 |
| 4. Carga masiva de datos..... | 9 |
| 5. Desarrollo de perfiles RDF..... | 12 |

INTRODUCCIÓN

En este documento se recogen los resultados de los trabajos incluidos en la entrega parcial 1 del presente proyecto. El documento se desglosa en las diferentes fases establecidas en la planificación inicial.

A modo de resumen cabe destacar que el objetivo principal del proyecto es el desarrollo de un catálogo y un API de consulta a la información disponible en EIDOS (base de datos del Ministerio de Transición Ecológica sobre especies y biodiversidad).

DETALLE DE TAREAS REALIZADAS

1. Análisis preoperacional y toma de requisitos

En esta fase se ha realizado como tarea principal la realización de un análisis de la situación actual y de evolución prevista a medio plazo del proyecto, requisitos tecnológicos y otros condicionantes. Además, se ha realizado un cronograma actualizado, ajustado a la fecha efectiva de inicio de proyecto, así como una actualización del calendario de entregas, el cual difiere respecto al previsto inicialmente, estableciéndose dos entregas (una, la actual, parcial de las primeras 5 fases, y otra posterior con el resto de fases para el año 2022).

Como consecuencia de este análisis preoperacional, se ha realizado una valoración de los posibles **riesgos y amenazas** del proyecto, estimando que **no son significativos** para comprometer la correcta ejecución del proyecto.

Las principales dependencias y riesgos que se han contemplado pueden resumirse en la siguiente tabla:

| Riesgo | Medida correctora | Impacto |
|--|--|--|
| Cambio en el modelo de datos EIDOS (prevista finalización en enero-2022) | Retraso de fases involucradas | Moderado, siempre que se cumplan los plazos |
| Dependencia de plataforma tecnológica | Posible implantación en servidor local | Impacto bajo |
| Integración con portal de datos | Utilización de API | Bajo, siempre que puedan resolverse las necesidades con el API actual o con modificaciones simples |
| Cambio en la fuente de datos (previsto Postgis) | Proceso de carga independiente de | Impacto moderado, por posibles implicaciones tecnológicas |

| | | |
|--|------------|--|
| | tecnología | |
|--|------------|--|

Tabla 1: Análisis de riesgos y amenazas

En esta fase también se ha realizado un estudio detallado de las extensiones CKAN a incluir, y que permitirán un desarrollo más sencillo al poder reutilizarse gran parte de su código y funcionalidad:

- <https://github.com/ckan/ckanext-dcat>
- <https://github.com/ckan/ckanext-scheming>

Estas extensiones tienen la principal ventaja de ser ampliamente conocidas y estar muy actualizadas y testadas, por lo que su fiabilidad es alta. Es por ello que ha permitido utilizar la **última versión estable** del producto CKAN.

2. Instalación del entorno y extensiones

En esta fase se ha realizado una instalación de todos los productos software necesarios para el desarrollo del proyecto, en un servidor propio, documentando detalladamente todas las acciones realizadas para la futura instalación en los servidores finales del cliente.

Concretamente se ha realizado una instalación de CKAN según código fuente (no por paquete) conforme a la última versión estable 2.9, la cual a su vez incluye la instalación de:

- PostgreSQL (puede ser reemplazado en el futuro por otro servidor ya existente).
- Python3
- Solr sobre jetty
- OpenJDK
- Redis
- Pip
- virtualenv
- supervisor

Además, se han instalado las extensiones mencionadas anteriormente (ckanext-dcat y ckanext-scheming).

También se ha creado una extensión propia que incluye todos los modelos de datos necesarios, adecuaciones a generación RDF, lógica funcional, gestión de rutas (esquema de URIs), etc.

Para aquellas tareas automáticas se ha realizado, en su mayoría, en Python haciendo uso del propio API de CKAN. Las principales tareas automáticas desarrolladas han sido para la carga de datos desde la fuente original (conforme al modelo de datos propio); al margen de estas tareas específicas se ha revisado otras tareas asíncronas (*jobs*) incluidos en CKAN para garantizar que siguen estando operativos y su resultado es correcto.

3. Creación de modelos de datos

Se han desarrollado los modelos de datos establecidos en el proyecto, haciendo uso de los mecanismos establecidos en ckanext-scheming, es decir, en YAML y la sintaxis requerida y los campos obligatorios. Un trozo de ejemplo del código de la definición del modelo de datos del concepto InvasivenessType en YAML puede verse en la siguiente figura:

```

1  scheming_version: 2
2  dataset_type: invasiveness_type
3  about: Modelo de invasiveness_type - EIDOS
4  about_url: "http://github.com/ckan/ckanext-scheming"
5  dataset_fields:
6    - field_name: title
7      label: Title
8      preset: title
9      form_placeholder: eg. A descriptive title
10   - field_name: name
11     prefix: invasiveness_type-
12     required: true
13     label: ID
14     help_text: Referencia inequívoca al recurso dentro de un contexto dado
15     preset: dataset_slug
16     form_placeholder: eg. my-dataset
17     validators: if_empty_same_as(title) package_name_validator
18   - field_name: uri
19     display_snippet: link.html
20     label: uri
21   - field_name: Plinian_IDInvasiveness
22     preset: multiple_text_single
23     label: IDInvasiveness
24   - field_name: Plinian_InvasivenessUnstructured
25     preset: multiple_text_single
26     label: InvasivenessUnstructured
27   - field_name: Plinian_Route
28     preset: multiple_text_single
29     label: Route
30   - field_name: Plinian_origin
31     preset: multiple_text_single
32     label: origin

```

Figura 1: Muestra de código YAML para modelo de datos

Además de los 16 previstos inicialmente (y que se detallan en la tabla siguiente con el número de instancias aproximado previsto para cada uno), **se ha añadido uno más**, correspondiente al concepto de conjunto de datos (**dataset**), el cual se ha modelado conforme a DCAT-AP.

| Concepto | Número de instancias |
|----------------------------|----------------------|
| Collection | 37 |
| DirectThreatsType | 1999 |
| DistributionType | 27149 |
| HabitatsType | 3204 |
| InvasivenessType | 441 |
| LegislationType | 28 |
| Location | 16132 |
| ManagementConservationType | 6257 |
| MeasurementOrFact | 24405 |

| | |
|-----------------------|-------|
| PopulationBiologyType | 1668 |
| RecordMetadataType | 36311 |
| ReferenceType | 98 |
| Taxon | 27274 |
| TaxonRecord | 27376 |
| ThreatStatusType | 8584 |
| VernacularName | 34567 |

Tabla 2: Conceptos definidos en EIDOS y número de instancias

Los modelos de datos desarrollados se han generado conforme a la versión actual, si bien está previsto que podría haber cambios en el futuro y hubiera que realizar algún ajuste futuro. En todo caso, se han desarrollado de forma flexible y adaptable para que esta circunstancia no suponga una limitación posterior.

Para cada uno de los conceptos anteriores, se ha desarrollado un editor de contenidos sencillo (basado en formulario) y una visualización en formato ficha que muestra de manera simple la información almacenada. Si bien no está previsto que se haga un uso intensivo de este editor puesto que la gran parte de los contenidos se hará mediante procesos masivos de carga. Un ejemplo de este editor de contenidos para el concepto dataset puede verse en la siguiente figura:

Edit metadata
Resources
View dataset

*** Dataset - title:**

Prueba de conjunto de datos

ⓘ This property contains a name given to the Dataset. This property can be repeated for parallel language versions of the name.

*** URL:** 35.195.31.179/conjunto_datos/Prueba+de+conjunto+de+datos Edit

ⓘ Referencia inequívoca al recurso dentro de un contexto dado

*** Dataset - description:**

1. Prueba descripción Remove

This property contains a free-text account of the Dataset. This property can be repeated for parallel language versions of the description.

[+ Add](#)

Dataset - contact point:

1. Remove

This property contains contact information that can be used for sending comments about the Dataset.

[+ Add](#)

Dataset - keyword/ tag:

1. Remove

This property contains a keyword or tag describing the Dataset.

[+ Add](#)

Dataset - publisher:

Figura 2: Muestra editor de contenidos para el concepto *dataset*

Mientras que, por otro lado, la ficha informativa de un contenido quedaría de la siguiente forma (por ejemplo un Taxon Record):

Ammodaucus nanocarpus (Beltran-Tej.) P.Pérez & A.Velasco

Data and Resources

This dataset has no data, why not add some?

Additional Info

| Field | Value |
|-----------------------------|---|
| uri | https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/TaxonRecord/18431 |
| CodeCITES | |
| CodeEUNIS | 22379 |
| CodeEURING | |
| CodePhoto | |
| CodeRedNatura | |
| FullDescriptionUnstructured | |
| TaxonRecordID | 18431 |
| grupoTaxonomico | Ammodaucus nanocarpus (Beltran-Tej.) P.Pérez & A.Velasco |
| hasCollection | https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/Collection/15 |
| hasDirectThreats | |
| hasDistribution | <ul style="list-style-type: none"> https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/DistributionType/19607 https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/DistributionType/17025 https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/DistributionType/17025 |

Figura 3: Muestra de una ficha informativa de Taxon Record

4. Carga masiva de datos

En esta fase se ha desarrollado un proceso de carga masivo Python para realizar la importación de todos los contenidos en CKAN.

Se ha partido de la fuente de datos actual, sin perjuicio de que en el futuro se modifique para obtener los datos de una base de datos Postgis o de donde se decida.

El proceso sigue un flujo de trabajo en el que se realizan dos tareas principales:

1. Troceado de fichero dump RDF en múltiples ficheros individuales.
2. Proceso iterativo de carga de un fichero individual.

El hecho de realizar la **primera** tarea de división en ficheros más pequeños ha sido la forma que se ha considerado óptima de gestionar el actual fichero dump que ocupa unos 240MB. De esta manera, además, se permite parar y reanudar el proceso, puesto que es iterativo e incremental, procesando cada vez un pequeño fichero y eliminándolo en caso de ejecución con éxito, de tal forma que cada vez se va reduciendo el número de ficheros a procesar y se puede ir viendo avances paulatinos, conforme se va progresando en la ejecución.

El amplio conjunto de ficheros generados tienen la característica de que siguen una política de nombrado:

```
14K Dec 13 18:33 Darwin_Location5437.ttl
1.6K Dec 13 18:33 Darwin_Location5489.ttl
1.2K Dec 13 18:33 Darwin_Location5504.ttl
6.6K Dec 13 18:33 Darwin_Location581.ttl
8.9K Dec 13 18:33 Darwin_Location612.ttl
12K Dec 13 18:33 Darwin_Location651.ttl
1.2K Dec 13 18:33 Darwin_Location663.ttl
13K Dec 13 18:33 Darwin_Location742.ttl
7.3K Dec 13 18:33 Darwin_Location767.ttl
11K Dec 13 18:33 Darwin_Location768.ttl
14K Dec 13 18:33 Darwin_Location775.ttl
12K Dec 13 18:33 Darwin_Location800.ttl
9.3K Dec 13 18:33 Darwin_Location803.ttl
9.8K Dec 13 18:33 Darwin_Location805.ttl
9.7K Dec 13 18:33 Darwin_Location826.ttl
8.3K Dec 13 18:33 Darwin_Location832.ttl
11K Dec 13 18:33 Darwin_Location850.ttl
1.1K Dec 13 18:33 Darwin_Location858.ttl
11K Dec 13 18:33 Darwin_Location878.ttl
8.2K Dec 13 18:33 Darwin_Location883.ttl
11K Dec 13 18:33 Darwin_Location925.ttl
1.3K Dec 13 18:33 Darwin_Location973.ttl
1.3K Dec 13 18:33 Darwin_MeasurementOrFact100.ttl
1.1K Dec 13 18:33 Darwin_MeasurementOrFact1004.ttl
1.4K Dec 13 18:33 Darwin_MeasurementOrFact1018.ttl
1.4K Dec 13 18:33 Darwin_MeasurementOrFact1025.ttl
1.3K Dec 13 18:33 Darwin_MeasurementOrFact1032.ttl
1.3K Dec 13 18:33 Darwin_MeasurementOrFact1046.ttl
1.4K Dec 13 18:33 Darwin_MeasurementOrFact1050.ttl
1.1K Dec 13 18:33 Darwin_MeasurementOrFact1070.ttl
1.3K Dec 13 18:33 Darwin_MeasurementOrFact1071.ttl
1.4K Dec 13 18:33 Darwin_MeasurementOrFact1074.ttl
1.3K Dec 13 18:33 Darwin_MeasurementOrFact1095.ttl
1.3K Dec 13 18:33 Darwin_MeasurementOrFact1118.ttl
```

Figura 4: Muestra parcial de ficheros individuales RDF

El código Python de esta primera tarea puede verse en la siguiente figura:

```
1  #!/usr/bin/python3
2
3  # Requisito: Descargar desde datos.iepnb.es y ubicar en una ruta conocida
4
5  path_orig = '/usr/lib/ckan/default/src/loadProcess/eidos.ttl.completo'
6  path_tmp = '/usr/lib/ckan/default/src/loadProcess/tmp/'
7  headers = ''
8  file_orig = open(path_orig, 'r')
9  lines = file_orig.readlines()
10
11  had_headers = False
12  new_item = ''
13  count = 0
14
15  for line in lines:
16      if not had_headers:
17          if line.startswith('@prefix'):
18              headers += line
19          else:
20              had_headers = True
21      else:
22          new_item += line
23          if line.strip().startswith('a '):
24              new_type = line.strip()[2:].strip()[:-1].strip().replace(':', '_')
25
26          if not line.strip():
27              count = count + 1
28              file_tmp = open(path_tmp + '{}{}.ttl'.format(new_type, count), 'w')
29              file_tmp.write(headers)
30              file_tmp.write(new_item)
31              file_tmp.close()
32
33          new_item = ''
```

Figura 5: Código de splitFilesTTL.py

Para la **segunda** de las tareas, la de carga a partir del RDF de cada contenido, se realiza un procesamiento mediante el listado de los ficheros anteriores y su procesamiento en secuencia de los tipos de conceptos previstos. En este caso, el código Python es bastante más extenso, si bien se muestra a continuación el trozo principal:

```

1 def insert_plinian_vernacularname(g, filename):
2     fields = ['Plinian_Language', 'Plinian_Name', 'Plinian_VernacularNameID']
3
4     insert_generic(g, filename, fields, 'vernacular_name',
5                   'Plinian:VernacularName', 'Plinian_VernacularNameID', 'Plinian_Name')
6
7 def insert_generic(g, filename, fields, _type, rdf_type, name_field,
8                   title_field):
9
10    dataset_dict = {}
11    dataset_dict['owner_org'] = owner_org
12    dataset_dict['type'] = _type
13
14    for fld in fields:
15        sparql_query = sparql_header + """
16            SELECT ?uri ?""" + fld + """
17            WHERE {
18                ?uri rdf:type """ + rdf_type + "."
19
20        sparql_query += '    OPTIONAL { '
21        sparql_query += '?uri {} ?{}'.format(fld.replace('_', ':'), fld)
22        sparql_query += '}. \n'
23
24        sparql_query += """
25            }
26        """
27        results = g.query(sparql_query)
28
29        aux = []
30        for row in results:
31            _valor = eval('row["' + fld + '"]')
32            if _valor:
33                # dataset_dict[fld] = str(_valor)
34                aux.append(str(_valor))
35
36            if fld == title_field:
37                dataset_dict['title'] = str(row[title_field])
38                if fld == name_field:
39                    # si no está este campo, que dé fallo
40                    dataset_dict['name'] = _type + '_' + str(row[name_field])
41                    # aprovechamos para rellenar este campo tambien
42                    dataset_dict['uri'] = str(row.uri)
43
44    dataset_dict[fld] = aux

```

Figura 6: Muestra principal del código de loadFiles.py

5. Desarrollo de perfiles RDF

Gracias a la reutilización de la extensión ckanext-dcat es posible realizar la tarea de generación de RDF de una forma sencilla, mediante la definición de un perfil (*profile*) RDF.

Una muestra del código de este profile puede verse en la siguiente figura:

```
def add_rdf_type(self, g, dataset_ref, _type):
    if _type == 'collection':
        g.add((dataset_ref, RDF.type, PLINIAN.Collection))
    elif _type == 'direct_threats_type':
        g.add((dataset_ref, RDF.type, PLINIAN.Direct_Threats_Type))
    elif _type == 'distribution_type':
        g.add((dataset_ref, RDF.type, PLINIAN.Distribution_Type))
    elif _type == 'habitats_type':
        g.add((dataset_ref, RDF.type, PLINIAN.Habitats_Type))
    elif _type == 'invasiveness_type':
        g.add((dataset_ref, RDF.type, PLINIAN.Invasiveness_Type))
    (...)

def graph_from_dataset(self, dataset_dict, dataset_ref):
    g = self.g

    for prefix, namespace in namespaces.items():
        g.bind(prefix, namespace)

    self.add_rdf_type(g, dataset_ref, dataset_dict['type'])

    for item in dataset_dict:
        if (dataset_dict[item]):
            item_short = None
            item_ns = None

            if item.startswith("Plinian_"):
                item_short = item.replace("Plinian_", "")
                item_ns = "PLINIAN"
            elif item.startswith("Darwin_"):
                item_short = item.replace("Darwin_", "")
                item_ns = "DARWIN"
            elif item.startswith("DC_"):
                item_short = item.replace("DC_", "")
                item_ns = "DCT"

            for val in dataset_dict[item]:
                if item_short:
                    if val.startswith("http"):
                        g.add((dataset_ref, eval(item_ns + '.' + item_short), URIRef(val)))
                    else:
                        g.add((dataset_ref, eval(item_ns + '.' + item_short), Literal(val)))
```

Figura 7: Muestra principal del código del *profile* RDF desarrollado

Además, se ha realizado la negociación de contenido de tal forma que se puede elegir si se desea obtener la información de la URI solicitada en HTML o en RDF/XML, entre otras. Si por ejemplo se solicita en RDF/XML del Taxon Record anteriormente mostrado, se obtiene el siguiente resultado:

```
<rdf:RDF>
  <Plinian:Taxon_Record rdf:about="https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/TaxonRecord/18431">
    <Plinian:TaxonRecordID>18431</Plinian:TaxonRecordID>
    <Plinian:hasCollection rdf:resource="https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/Collection/15"/>
    <Plinian:hasRecordMetadata rdf:resource="https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/RecordMetadataType/18431_OTHREF16"/>
    <Plinian:hasDistribution rdf:resource="https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/DistributionType/19607"/>
    <Plinian:hasDistribution rdf:resource="https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/DistributionType/6122"/>
    <Plinian:hasDistribution rdf:resource="https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/DistributionType/17025"/>
    <Plinian:CodeEUNIS>22379</Plinian:CodeEUNIS>
  </Plinian:grupoTaxonomico>
    Ammodaucus nanocarpus (Beltran-Tej.) P.Pérez & A.Velasco
  </Plinian:grupoTaxonomico>
  <Plinian:hasHierarchy rdf:resource="https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/Taxon/18431"/>
  <Plinian:hasDistribution rdf:resource="https://datos.iepnb.es/recurso/sector-publico/medio-ambiente/pliniancore/DistributionType/2832"/>
</Plinian:Taxon_Record>
</rdf:RDF>
```

Figura 8: Resultado de solicitar contenido RDF/XML para una URI utilizando un *profile* RDF