



UNIVERSIDAD
DE MÁLAGA



E.T.S. INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería de la Salud Mención Bioinformática

Proyecto Final

Aplicación web para Centros de Salud Mental

Adrián Segura Ortiz
Marina Pacheco Rojas

Asignatura Estándares de datos
abiertos e integración de datos

COURSE 2020/2021

Objetivo

Nuestro proyecto se centra en el diseño y construcción de una aplicación web cuyas funcionalidades sirvan de soporte informático a Centros de Salud Mental. El objetivo principal de nuestra aplicación consiste en tanto facilitar la labor de los psiquiatras y administrativos que trabajen en el centro, como optimizar la atención médica recibida por los pacientes y familiares.

Para alcanzar esta meta nuestro sistema le permitirá a los psiquiatras el uso de diferentes secciones, las cuales se encuentran destinadas a agilizar las tareas asociadas a las diferentes fases que constituyen las consultas médicas de los pacientes.

Por otro lado a los pacientes y familiares se les proporcionará un alto nivel de interacción en el sistema con el fin de que puedan manejar dentro de lo posible sus respectivos seguimientos médicos.

Además, el sistema contemplará un apartado informativo donde se expongan datos del centro en cuestión junto con descripciones, asociaciones de pacientes y consejos de carácter general asociados a determinadas enfermedades mentales.

Índice

1. Requisitos	5
1.1. Usuarios	5
1.2. Requisitos Funcionales	6
1.3. Requisitos No Funcionales	8
2. Diseño BD	9
2.1. Psiquiatra	10
2.2. Administrativo	11
2.3. Paciente	12
2.4. Familiar Autorizado	14
2.5. Formulario consulta	15
2.6. Mensaje	16
3. Población BD	17
4. Vistas XML	21
4.1. Vista 1	25
4.2. Vista 2	28
4.3. Vista 3	32

4.4. Vista 4	37
5. Vistas RDF	41
5.1. Generación del código RDF	42
5.2. Subida de código rdf a virtuoso	52
5.3. Lanzamiento de consultas SPARQL	52
6. Web Application	65
6.1. Implementación	65
6.2. Interfaz	68
6.3. Ejecución	71
6.4. Vídeo	72

1

Requisitos

1.1. Usuarios

- **Psiquiatra:** Se trata del usuario principal de nuestra aplicación y dado que su frecuencia de interacción es superior a la del resto, nuestro sistema girará en torno a él. Su principal acción en el sistema estará relacionada con la escritura y actualización de datos ya que será el responsable de la elaboración de informes, la petición de pruebas complementarias y la proporción de consejos médicos.
- **Administrativo:** Su papel en el sistema viene definido por su propio nombre, es decir, se encargará de la gestión administrativa de los datos en el sistema. En muchas ocasiones su interacción en el sistema se encuentra relacionada con acción en posiciones intermedias de distintos procesos, como la petición de pruebas complementarias, la programación de citas o el proceso de facturación.
- **Paciente:** Las operaciones de este usuario son principalmente consultivas, sin embargo, tendrá a su disposición una gran variedad de datos relacionados con su paso por el centro. Estos datos contemplan asociaciones de pacientes, consejos útiles de carácter general, información del centro y facturas de pago.

- **Familiar autorizado:** Su función principal en el sistema consiste en ejercer de puente entre el psiquiatra y el paciente mediante la lectura de los consejos proporcionados por el especialista, así como cubrir las carencias de gestión que el enfermo no sea capaz de abordar.

1.2. Requisitos Funcionales

Psiquiatra

- **RF01:** El psiquiatra podrá consultar sus datos personales y profesionales registrados en el sistema.
- **RF02:** El psiquiatra podrá consultar el horario de citas previsto en su agenda.
- **RF03:** El psiquiatra podrá consultar y elaborar informes clínicos de pacientes.
- **RF04:** El psiquiatra podrá solicitar pruebas clínicas complementarias asociadas a pacientes.
- **RF05:** El psiquiatra podrá consultar los resultados de las pruebas clínicas previamente solicitadas.
- **RF06:** El psiquiatra podrá enviar a familiares autorizados consejos para abordar el trato cotidiano con el paciente.

Administrativo

- **RF07:** El administrativo podrá crear, borrar y modificar registros de psiquiatras, pacientes y familiares en el sistema.

- **RF08:** El administrativo podrá añadir y cancelar citas de consulta.
- **RF09:** El administrativo podrá consultar las solicitudes de pruebas clínicas complementarias con el fin de gestionar oficialmente su trámite.
- **RF10:** El administrativo podrá enviar al psiquiatra correspondiente los resultados de pruebas recibidos en el sistema.
- **RF11:** El administrativo podrá elaborar y registrar facturas de pago.

Paciente

- **RF12:** El paciente podrá solicitar, cancelar y visualizar citas de consulta.
- **RF13:** El paciente podrá consultar información profesional acerca de los psiquiatras registrados en el sistema.
- **RF14:** El paciente podrá consultar sus datos personales registrados en el sistema.
- **RF15:** El paciente podrá consultar y descargar sus facturas de pago.

Familiar autorizado

- **RF16:** El familiar autorizado podrá consultar los consejos proporcionados por el psiquiatra acerca del trato con el paciente.
- **RF17:** El familiar autorizado podrá solicitar citas de consulta en caso de que el paciente esté incapacitado para hacerlo o le otorgue oficialmente al familiar ese poder.

Todos los usuarios

- **RF18:** El usuario podrá consultar la información proporcionada en el sistema sobre el centro de salud mental.
- **RF19:** El usuario podrá consultar información acerca de ciertas enfermedades mentales, así como de sus descripciones, asociaciones de pacientes y consejos de carácter general.
- **RF20:** El usuario podrá modificar su contraseña de acceso en caso de que lo desee.

1.3. Requisitos No Funcionales

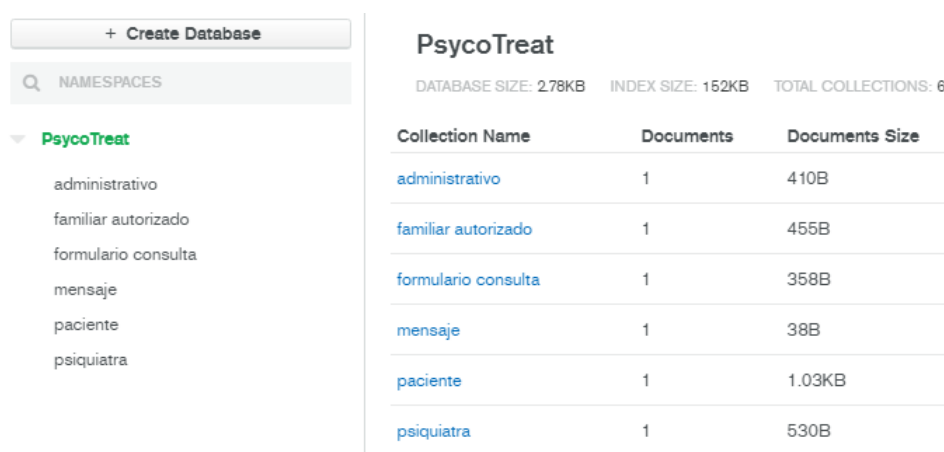
- **RNF01:** Todos los datos del sistemas serán almacenados en la nube.
- **RNF02:** Toda la información personal de los usuarios estará encriptada.
- **RNF03:** El sistema debe garantizar el cumplimiento de la LOPD (Ley Orgánica de la Ley de Protección de Datos).
- **RNF04:** El sistema proporcionará una comunicación cifrada entre los usuarios implicados en una operación.
- **RNF05:** La base de datos de nuestro servidor tendrá una capacidad de 1 TB ampliable.
- **RNF06:** El sistema proporcionará una disponibilidad del 97 %.
- **RNF07:** Las imágenes recibidas en el sistema usarán el formato DICOM.
- **RNF08:** Se utilizará el formato HL7 para compartir datos.

2

Diseño BD

En este apartado trataremos de exponer el proceso de construcción y diseño de nuestra base de datos. Ésta se encontrará situada en **MongoDB Atlas** con la finalidad de ser accesible desde cualquier dispositivo sin la necesidad de tener que descargar su contenido.

Su nombre, al igual que nuestra aplicación, es **PsycoTreat** y estará formada por seis colecciones que se corresponden a los datos de cada uno de los tipos de usuario junto con los formularios de consulta y los mensajes de psiquiatras dirigidos a los familiares.



The screenshot displays the MongoDB Atlas interface for a database named 'PsycoTreat'. On the left, a sidebar shows the database's namespaces, including 'administrativo', 'familiar autorizado', 'formulario consulta', 'mensaje', 'paciente', and 'psiquiatra'. The main panel on the right provides a summary of the database's size (2.78KB), index size (152KB), and total collections (6). Below this, a table lists the collections and their document counts and sizes.

Collection Name	Documents	Documents Size
administrativo	1	410B
familiar autorizado	1	455B
formulario consulta	1	358B
mensaje	1	38B
paciente	1	1.03KB
psiquiatra	1	530B

Una vez conocida la estructura principal de nuestra base de datos, nos centraremos en mostrar mediante un lenguaje similar a **Json Schema**, los diferentes campos que van a contener los objetos pertenecientes a cada colección.

2.1. Psiquiatra

Para este tipo de usuario tendremos en primera instancia un conjunto de **datos de ámbito personal**, y en segundo lugar una serie de campos relacionados con su **información profesional y actividad en el centro**.

El campo **fotografía** consiste en una cadena de caracteres que indica el fichero png donde se almacena la imagen.

```
1  {
2    "_id" : STRING,
3    "nombre" : STRING,
4    "apellidos" : STRING,
5    "dni" : STRING,
6    "fecha de nacimiento" : DATE,
7    "sexo" : STRING,
8    "email" : STRING,
9    "telefono" : STRING,
10   "direccion" : STRING,
11   "codigo postal" : STRING,
12   "localidad" : STRING,
13   "provincia" : STRING,
14   "fotografia" : STRING,
15   "titulacion" : STRING,
16   "universidad" : STRING,
17   "especialidad" : STRING,
18   "años antigüedad en centro" : INTEGER,
19   "numero de pacientes" : INTEGER
20 }
```

2.2. Administrativo

Este usuario tendrá una estructura similar al anterior, sólo que omitiendo cierta información correspondiente a los últimos campos del objeto.

No obstante, tendrá un campo adicional que hará referencia a un array de objetos que contengan los identificadores de los **mensajes recibidos** por parte del psiquiatra para la solicitud de pruebas complementarias u otras necesidades.

```
1  {
2    "_id" : STRING,
3    "nombre" : STRING,
4    "apellidos" : STRING,
5    "dni" : STRING,
6    "fecha de nacimiento" : DATE,
7    "sexo" : STRING,
8    "email" : STRING,
9    "telefono" : STRING,
10   "direccion" : STRING,
11   "codigo postal" : STRING,
12   "localidad" : STRING,
13   "provincia" : STRING,
14   "fotografia" : STRING,
15   "titulacion" : STRING,
16   "años antigüedad en centro" : INTEGER,
17   "mensajes recibidos" : [{"mensaje" : STRING}]
18 }
```

2.3. Paciente

Los pacientes serán los objetos que mayor número de campos van a poseer puesto que son la principal fuente de datos en nuestra aplicación. Su estructura se puede dividir en tres partes:

- **Datos personales:** Idénticos a los vistos hasta ahora en el resto de usuarios.
- **Patologías previas:** Array de objetos referentes a cada uno de los tratamientos ya finalizados que han sido dirigidos en el centro.
- **Patología actual:** Aquella que causa la presente atención del paciente.

Una vez finalizado un tratamiento la patología actual abandona su posición y se añade al array de patologías previas, introduciendo en el campo **fecha fin** la fecha actual de modificación. Tras ello, la patología actual queda vacía y por tanto preparada para ser rellenada en el próximo tratamiento del paciente.

El campo **formularios consultas** estará asociado a cada patología y contendrá un array con los identificadores de los formularios que se almacenan en su colección correspondiente.

Además, los pacientes serán los que se encarguen a través del campo **psiquiatra asignado** de indicar la relación entre ellos y los psiquiatras que los hayan atendido. Esta relación al igual que la anterior, será específica de cada patología y se indicará mediante la escritura del identificador del correspondiente psiquiatra en dicho atributo.

```

1 {
2   "_id" : STRING,
3   "nombre" : STRING,
4   "apellidos" : STRING,
5   "dni" : STRING,
6   "fecha de nacimiento" : DATE,
7   "sexo" : STRING,
8   "email" : STRING,
9   "telefono" : STRING,
10  "direccion" : STRING,
11  "codigo postal" : STRING,
12  "localidad" : STRING,
13  "provincia" : STRING,
14  "fotografia" : STRING,
15  "patologias previas" : [
16    {
17      "patologia" : STRING,
18      "fecha inicio" : DATE,
19      "fecha final" : DATE,
20      "psiquiatra asignado" : STRING,
21      "formularios consultas" : [{ "formulario" : STRING }],
22      "medicamentos asignados" : [{ "medicamento" : STRING }],
23      "pruebas complementarias" : [{ "prueba" : STRING }]
24    }
25  ],
26  "patologia actual" : STRING,
27  "fecha inicio" : STRING,
28  "psiquiatra asignado" : STRING,
29  "formularios consultas" : [{ "formulario" : STRING }],
30  "medicamentos asignados" : [{ "medicamento" : STRING }],
31  "pruebas complementarias" : [{ "prueba" : STRING }]
32 }

```

2.4. Familiar Autorizado

El familiar autorizado tendrá la **información personal** habitual junto con una serie de campos que lo relacionen con el **paciente** correspondiente y los **mensajes recibidos** por parte del psiquiatra del mismo.

Además, se añadirá un **nivel de autorización** que será **total** si puede efectuar las mismas acciones que el paciente, **parcial** si realiza alguna de ellas o **normal** si su interacción en el sistema es la justamente necesaria.

```
1 {  
2   "id" : STRING,  
3   "nombre" : STRING,  
4   "apellidos" : STRING,  
5   "dni" : STRING,  
6   "fecha de nacimiento" : DATE,  
7   "sexo" : STRING,  
8   "email" : STRING,  
9   "telefono" : STRING,  
10  "direccion" : STRING,  
11  "codigo postal" : STRING,  
12  "localidad" : STRING,  
13  "provincia" : STRING,  
14  "fotografia" : STRING,  
15  "paciente asociado" : STRING,  
16  "relacion" : STRING,  
17  "nivel de autorizacion" : STRING,  
18  "mensajes recibidos" : [{"mensaje" : STRING}]  
19 }
```

2.5. Formulario consulta

Esta colección almacenará la información reflejada en los informes de consulta realizados por el **psiquiatra**.

Por ello, se especificará el médico que lo redactó y el **paciente** al que se evaluaba, introduciendo una serie de información acerca del inicio, desarrollo y desenlace de la consulta.

Una vez acabado, su identificador será añadido en el campo correspondiente situado en la colección de pacientes. En caso de que el tratamiento requiera algún tipo de medicación está deberá ser adicionalmente añadida en el perfil del paciente de forma manual.

```
1 {  
2   "_id" : STRING,  
3   "paciente" : STRING,  
4   "psiquiatra" : STRING,  
5   "fecha consulta" : DATE,  
6   "motivo consulta" : STRING,  
7   "desencadenante" : STRING,  
8   "observaciones" : STRING,  
9   "diagnostico" : STRING,  
10  "tratamiento" : STRING,  
11  "frecuencia tratamiento" : STRING  
12 }
```


2.6. Mensaje

Esta colección contendrá objetos que representen los distintos mensajes que son intercambiados entre diversos usuarios del sistema.

Los dos casos más comunes serán:

- Mensajes de **psiquiatras a familiares** que indiquen pautas de actuación sobre el cuidado diario de los pacientes asociados.
- Mensajes de **psiquiatras al administrativo** informando de peticiones de pruebas clínicas complementarias u otras necesidades.

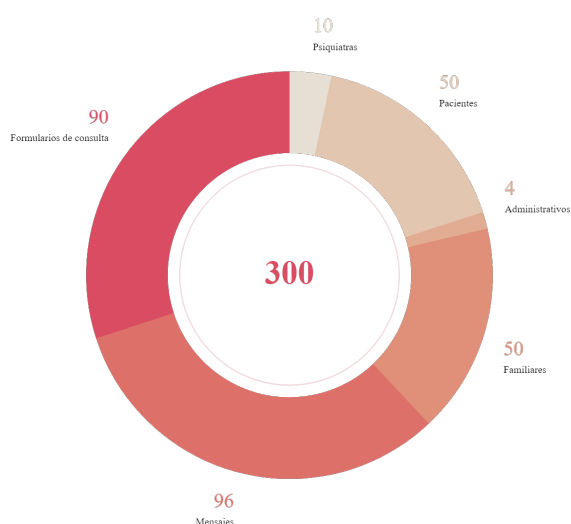
Tras la recepción de un mensaje este es almacenado en el perfil del receptor dentro del array correspondiente reservado para este fin.

```
1 {  
2   "id" : STRING,  
3   "emisor" : STRING,  
4   "receptor" : STRING,  
5   "fecha de envio" : DATE,  
6   "contenido" : STRING  
7 }
```

3

Población BD

Una vez establecida la estructura de cada una de nuestras colecciones, el siguiente paso se centra en poblar nuestra base de datos. Para ello, introduciremos un total de **300 entradas** que repartiremos de la siguiente forma:



Donde aproximadamente el 30 % de los **mensajes** se corresponderán a **peticiones de pruebas** y mantenimiento dirigidas a administrativos, y el 70 % serán mensajes enviados a familiares para proporcionarles **consejos** e información ligada a pacientes.

Para simular de la mejor manera posible una **distribución real**, es lógico que cada paciente haya realizado al menos una o dos consultas por patología, y a su vez los familiares hayan recibido unos cuantos mensajes cada uno. Es por esto, que dichas colecciones duplican en tamaño al resto, ya que además van a ser nuestras principales fuentes de datos a la hora de analizar información útil cuyo resultado mostraremos en la aplicación web.

Para llevar a cabo esta población se ha hecho uso de un **procedimiento totalmente manual** que se justifica por varios motivos:

1. A pesar de que los **datos personales** de los pacientes sí podrían haber sido generados aleatoriamente, habría dificultado la posterior tarea de:
 - Elaboración de familiares teniendo en cuenta la necesidad de coincidencia del apellido correspondiente así como la fecha de nacimiento y la localización de los mismos.
 - Asignación de la fecha de sus propios tratamientos teniendo en cuenta la edad aleatoria automáticamente generada. Que además afecta de forma directa a la de los formularios de consulta y a la de los mensajes que reciben los familiares respecto a la evolución de los pacientes.
2. Respecto a sus **datos clínicos**, además del peligro de generar enfermedades mentales de forma aleatoria nos encontramos con el problema de flexibilidad de estructura que tiene esta colección. Es decir, no todos los pacientes tienen el mismo número de patologías previas y además no todos se encuentran en tratamiento actualmente, por lo que la generación automática en este caso era completamente inviable. Además, aunque fuese posible hay algunas combinaciones de enfermedades que resultan muy improbables y que por tanto no pueden coincidir en el historial de un mismo paciente.
3. Tanto los **formularios de consulta** como los **mensajes** resultan imposibles de generar automáticamente si queremos mantener un mínimo de coherencia temporal y semántica respecto a lo descrito en los pacientes.

Además, entre los propios formularios y mensajes debe haber cierta relación así como con los medicamentos asignados en la parte clínica de los pacientes y las diversas peticiones de pruebas complementarias.

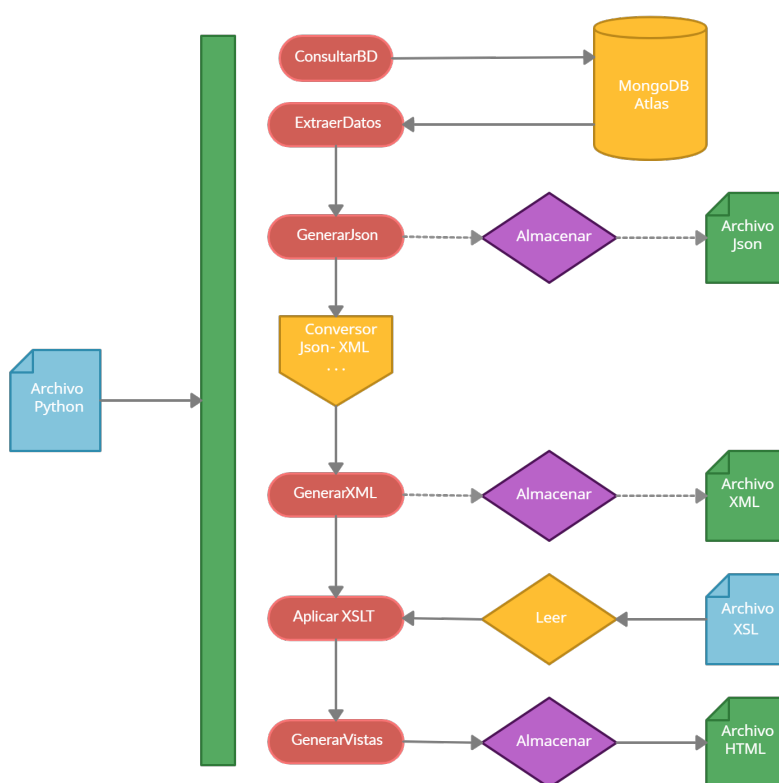
En resumen, a pesar de haber podido comenzar con una pequeña generación automática de datos personales de 50 pacientes (lo cual era solo una pequeña parte de una colección), no llegaba a compensar el posterior esfuerzo de cuadrar el resto de atributos en las demás colecciones. Por ello, decidimos diseñar **50 casos de pacientes** más o menos reales desde cero, que nos permitiese generar un total de 300 entradas mediante la asignación de una serie de formularios, familiares, mensajes y medicamentos acordes a la localización temporal e historia que queríamos mostrar en ellos.

4

Vistas XML

En esta sección del trabajo trataremos de generar diferentes vistas de la población anteriormente introducida en la base de datos. Para ello, hemos realizado un **procedimiento** lo más **automático** posible que nos permitiese en un futuro poner en marcha nuestro flujo de trabajo desde cualquier terminal.

Todo el proceso queda incorporado dentro del fichero python denominado **programa.py**, el cual contiene el código necesario para abarcar las tareas desde la consulta de la base de datos hasta la final generación de vistas en html.



A continuación, nos disponemos a exponer de forma ordenada las principales secciones de código que hacen posible la eficacia de nuestro programa:

- Establecimiento de la **conexión** a la base de datos y lectura de colecciones para su posterior uso.

```
# Establecemos conexión
myclient = pymongo.MongoClient("mongodb+srv://adriansegura:
    adrianseguraortiz1999@psicotreatbd.e4kvw.mongodb.net/PsycoTreat?
    retryWrites=true&w=majority")
# Guardamos las colecciones en variables
mydb = myclient["PsycoTreat"]
psiquiatras = mydb["psiquiatra"]
administrativos = mydb["administrativo"]
pacientes = mydb["paciente"]
familiares = mydb["familiar autorizado"]
formularios = mydb["formulario consulta"]
mensajes = mydb["mensaje"]
```

- Aplicación de **consultas** para obtener la información que mostraremos en las distintas vistas resultantes.

Este es el caso de las funciones **Consulta_n()** cuyas implementaciones se comentarán en las correspondientes secciones de cada vista.

```
# Vista 1: Psiquiatras y sus pacientes.
def Consulta_1 (): . . .
# Vista 2: Pacientes con sus formularios y pruebas complementarias
    clasificados por patología.
def Consulta_2 (): . . .
# Vista 3: Familiares de pacientes y sus mensajes.
def Consulta_3 (): . . .
# Vista 4: Informe enfermedades mentales
def Consulta_4 (): . . .
```

Su principal tarea se basa en aplicar comandos propios de Mongo a través de agregaciones que nos permitan combinar datos procedentes de distintas colecciones.

- Generación de **vistas** haciendo uso de la información extraída en las anteriores consultas.

Antes de ilustrar el código de la función principal asociada a esta tarea veremos dos funciones complementarias que tendrán un uso recurrente durante su ejecución. La primera de ellas se denomina **AplicaXSLT ()** y se corresponde con el código necesario para realizar la habitual transformación XSLT.

```
def AplicaXSLT(res_consulta_xml, archivo_xsl):  
    et_xslt = ET.parse(archivo_xsl)  
    transform = ET.XSLT(et_xslt)  
    newdom = transform(res_consulta_xml)  
    return newdom
```

La segunda se llama **GuardaArchivo ()** y se va a encargar tanto del almacenamiento opcional de los ficheros json y xml resultantes de cada consulta, como de la escritura de los fichero html que contendrán las distintas vistas.

```
def GuardaArchivo(cadena, nombre_archivo):  
    f = open(nombre_archivo, "w", encoding = "utf 8")  
    f.write(cadena)  
    f.close()
```

Finalmente tenemos la función principal **GeneraVista ()**, la cual se encargará de realizar la consulta especificada y llamar a todas las funciones mencionadas anteriormente con el fin de proporcionarnos una solución.


```

def GeneraVista(num_consulta, guardar_json, guardar_xml):
    # Obtenemos el resultado de nuestra consulta
    if(num_consulta == 1):
        res_consulta_json = Consulta_1()
    elif(num_consulta == 2):
        res_consulta_json = Consulta_2()
    elif(num_consulta == 3):
        res_consulta_json = Consulta_3()
    elif(num_consulta == 4):
        res_consulta_json = Consulta_4()
    else:
        return "consulta no implementada"

    # Construimos el objeto json
    json_docs = []
    for doc in res_consulta_json:
        json_docs.append(doc)

    # Almacenamos fichero.json en caso de que se solicite
    if(guardar_json):
        with open('Consultas json/resultado_consulta_' + str(num_consulta)
            + '.json', 'w', encoding="utf 8") as file:
            str(json_docs).encode('utf 8')
            json.dump(json_docs, file, indent=4, ensure_ascii=False)

    # Convertimos el resultado a formato xml
    res_consulta_xml_string = json2xml.Json2xml(json_docs, attr_type=False)
        .to_xml()

    # Almacenamos fichero.xml en caso de que se solicite
    if(guardar_xml):
        GuardaArchivo(res_consulta_xml_string[0:1], "Consultas xml/
            resultado_consulta_" + str(num_consulta) + ".xml")

    # Construimos objeto xml
    res_consulta_xml = ET.XML(res_consulta_xml_string)

    # Aplicamos el archivo xsl previamente diseñado
    html = AplicaXSLT(res_consulta_xml, "Consultas xsl/vista_" + str(
        num_consulta) + ".xsl")

    # Almacenamos la vista en fichero.html
    GuardaArchivo(str(html), "Vistas html/vista_" + str(num_consulta) + ".
        html")

```

4.1. Vista 1

La primera vista trata de mostrarnos los **psiquiatras** junto con la información asociada a todos sus **pacientes**, tanto actuales como anteriores. Para ello aplicaremos una doble función de agregación sobre la colección de psiquiatras en la que añadiremos dos nuevos campos:

- **Pacientes actuales:** Aquellos cuyo psiquiatra asignado en la patología actual se corresponda con el médico listado en la colección de psiquiatras.
- **Pacientes antiguos:** Aquellos que tengan al psiquiatra en cuestión asociado en alguna de sus patologías previas.

Por tanto, la función **Consulta_1 ()** tiene la siguiente implementación:

```
def Consulta_1 ():
    result = psiquiatras.aggregate([
        {
            '$lookup': {
                'from': 'paciente',
                'localField': '_id',
                'foreignField': 'psiquiatra asignado',
                'as': 'pacientes actuales'
            }
        },
        {
            '$lookup': {
                'from': 'paciente',
                'localField': '_id',
                'foreignField': 'patologias previas.psiquiatra asignado',
                'as': 'pacientes antiguos'
            }
        }
    ])
    return result
```

Cuya aplicación nos devuelve una salida en formato json que es almacenada de forma opcional en el fichero **resultado_consulta_1.json** con la siguiente estructura:

```
1  [ {
2    __id : "g1_id_psiquiatria_00N",
3    ... : "...",
4    pacientes actuales: [ { __id : "g1_id_paciente_00M",
5                          ... : "..."} ],
6    pacientes antiguos: [ { __id : "g1_id_paciente_00L",
7                          ... : "..."} ]
8  } ]
```

La cual es analizada por el conversor para generar el correspondiente código xml que nuevamente es opcionalmente almacenado en el fichero **resultado_consulta_1.xml** y cuya estructura me dispongo a mostrar a continuación:

```
<all>
  <item>
    <_id>g1_id_psiquiatria_00N</_id>
    <...>...</...>
    <pacientes_actuales>
      <item>
        <_id>g1_id_paciente_00M</_id>
        <...>...</...>
      </item>
    </pacientes_actuales>
    <pacientes_antiguos>
      <item>
        <_id>g1_id_paciente_00L</_id>
        <...>...</...>
      </item>
    </pacientes_antiguos>
  </item>
</all>
```

Finalmente realizamos la transformación XSLT a dicho resultado mediante la aplicación del archivo **vista_1.xsl**, el cual tendrá que realizar una acción extra de filtrado en los pacientes antiguos.

Esto se debe a que para cada psiquiatra tenemos almacenados en dicho campo todos los pacientes tal que alguna de sus patologías previas haya sido tratada por el psiquiatra en cuestión. No obstante, para sólo hacer visibles aquellas patologías en las que haya intervenido nuestro psiquiatra, realizaremos lo siguiente:

```
<...>...</...>
<xsl:for each select = "all/item">
  <xsl:variable name = "id_psiquiatra" select = "_id"/>
  <...>...</...>
  <xsl:for each select = "pacientes_antiguos/item">
    <...>...</...>
    <xsl:for each select = "patologias_previas/item">
      <xsl:if test = "psiquiatra_asignado = $id_psiquiatra">
        <xsl:value of select = "patologia"/>
      </xsl:if>
    </xsl:for each>
  <...>...</...>
</xsl:for each>
<...>...</...>
</xsl:for each>
```

Es decir, guardamos para cada psiquiatra el valor de su identificador dentro de la variable **\$id_psiquiatra** para poder consultarla antes de la inclusión de cualquier patología previa del paciente.

Finalmente, la vista generada de forma automática es almacenada en el archivo **vista_1.html** y tiene el siguiente aspecto para un psiquiatra de la lista capturado al azar:

• Roberto Fernández Rojas

◦ Información personal

Nombre	Apellidos	DNI	Fecha de Nacimiento	Sexo	Email	Teléfono
Roberto	Fernández Rojas	32568415D	26/03/1970	Hombre	roberto_hr90@gmail.com	645852112

◦ Información profesional

Titulación	Universidad	Especialidad	Años de antigüedad en el centro	Número de pacientes
Graduado en Medicina	Universidad de Granada (UGR)	Psiquiatría general	24	6

◦ Antiguos pacientes

Nombre	Apellidos	DNI	Patología: Fecha
David	Lasa Ordoñez	23567210K	▪ Trastorno alimenticio: 10/03/2016-17/08/2016
Carmen	Castro Romero	23446631C	▪ Depresión: 12/10/2009-03/02/2010 ▪ Depresión: 14/07/2012-15/01/2013
Eduardo	Valderrama Ortiz	23767219K	▪ Deficit de atención: 15/12/2008-01/04/2009
Jimena	Benitez Castro	23717219K	▪ Hiperactividad: 15/08/1996-19/01/1998

◦ Pacientes actuales en tratamiento

Nombre	Apellidos	DNI	Patología	Fecha Inicio
David	Lasa Ordoñez	23567210K	Trastorno de ansiedad	11/09/2020
Carmen	Castro Romero	23446631C	Trastorno de ansiedad	07/08/2020
Eduardo	Valderrama Ortiz	23767219K	Trastorno alimenticio	14/07/2020
Clara	Labrador Vera	23763319X	Depresión	11/09/2020
Jimena	Benitez Castro	23717219K	Alcoholemia	08/09/2020
Adrián	Pacheco Segura	23223319J	Drogadicción	13/11/2020

4.2. Vista 2

La siguiente vista resulta ser la más completa de todas. En ella se pretende mostrar a cada **paciente** junto con sus **formularios**, **medicamentos** y **pruebas complementarias** clasificadas dentro de cada patología.

No obstante, parecido al caso de los psiquiatras, una parte del proceso de clasificación será realizado en el archivo xls ya que las consultas no nos permiten elaborar tal tarea dentro de un margen considerable de simplicidad.

De toda la información que deseamos mostrar, los formularios son los únicos que se encuentran contenidos en otra colección. Por tanto, nuestra consulta se basará en obtener los datos de los mismos mediante la comparación de identificadores.

Para ello, realizaremos una consulta similar a la anterior donde añadiremos nuevamente dos campos como consecuencia de la sentencia de agregación.

- **Formularios actuales:** Aquellos cuyo identificador se encuentra mencionado en la patología actual del paciente.
- **Formularios previos:** Aquellos que aparezcan en alguna patología previa del mismo.

Por tanto, la función **Consulta_2 ()** tiene la siguiente implementación:

```
def Consulta_2 ():
    result = pacientes.aggregate([
        {
            '$lookup': {
                'from': 'formulario consulta',
                'localField': 'formularios consultas.formulario',
                'foreignField': '_id',
                'as': 'formularios actuales'
            }
        },
        {
            '$lookup': {
                'from': 'formulario consulta',
                'localField': 'patologias previas.formularios consultas.
                    formulario',
                'foreignField': '_id',
                'as': 'formularios previos'
            }
        }
    ])
    return result
```

Cuya aplicación nos devuelve una salida en formato json que es almacenada de forma opcional en el fichero **resultado_consulta_2.json** con la siguiente estructura:

```

1  [ {
2    __id : "g1_id_paciente_00N",
3    ... : "...",
4    formularios actuales : [ { __id : "g1_id_form_00M",
5                               ... : "..."} ],
6    formularios previos : [ { __id : "g1_id_form_00L",
7                             ... : "..."} ]
8  } ]

```

La cual es analizada por el conversor para generar el correspondiente código xml que nuevamente es almacenado de forma opcional en el fichero **resultado_consulta_2.xml** y cuya estructura me dispongo a mostrar a continuación:

```

<all>
  <item>
    <_id>g1_id_paciente_00N</_id>
    <...>...</...>
    <formularios_actuales>
      <item>
        <_id>g1_id_form_00M</_id>
        <...>...</...>
      </item>
    </formularios_actuales>
    <formularios_previos>
      <item>
        <_id>g1_id_form_00L</_id>
        <...>...</...>
      </item>
    </formularios_previos>
  </item>
</all>

```

Este resultado será a continuación sometido a la transformación XSLT mediante la aplicación del archivo **vista_2.xsl**.

Sin embargo, tal y como adelantábamos antes, este fichero tendrá que realizar una labor extra parecida a la elaborada con los psiquiatras. En este caso la tarea consistirá en una clasificación más que en un proceso de filtrado.

Esto se debe a que dentro del campo formularios_previos nos encontramos con todos aquellos que hayan aparecido en alguna patología previa del paciente pero sin distinguir en cual de ellas. Para solucionar este problema el archivo xsl realiza lo siguiente:

```
<...>...</...>
<xsl:for each select = "all/item">
  <...>...</...>
  <xsl:if test = "patologias_previas">
    <xsl:for each select = "patologias_previas/item">
      <...>...</...>
      <xsl:for each select = "formularios_consultas/item">
        <xsl:variable name="formulario" select="formulario"/>
        <xsl:for each select = "../..../formularios_previos
        /item">
          <xsl:if test = "$formulario = _id">
            <...>...</...>
          </xsl:if>
        </xsl:for each>
      </xsl:for each>
    <...>...</...>
  </xsl:for each>
</xsl:if>
<...>...</...>
</xsl:for each>
```

Por tanto, lo que estamos haciendo es un proceso reiterativo en el que para cada formulario encontrado dentro de una patología, se busca dentro del array de formularios_previos para así poder mostrar su información.

Una vez hecho esto, la vista resultante es almacenada dentro del archivo **vista_2.html**, el cual tiene la siguiente apariencia para cada paciente:

• Mario Martín Pérez

◦ Información personal

Nombre	Apellidos	DNI	Fecha de Nacimiento	Sexo	Email	Teléfono
Mario	Martín Pérez	85457744B	13/06/1960	Hombre	mariomp1998@gmail.com	652225874

◦ Patologías Previas

▪ Depresión

Patología	Fecha Inicio	Fecha Final	Medicamentos Asignados
Depresión	19/03/2011	01/02/2012	▪ Protriptilina

▪ Formularios

Fecha	Motivo Consulta	Desencadenante	Observaciones	Diagnóstico	Tratamiento	Frecuencia del tratamiento
19/03/2011	Intento de suicidio	Fallecimiento de su cónyuge	El paciente está decaído y presenta un muy bajo estado de ánimo	Depresión	Se recomienda tomar Protriptilina	1 vez al día durante 12 meses
01/02/2012	Revisión		El paciente ha superado la enfermedad		Se recomienda ir al psicólogo para evitar una recaída	1 vez a la semana

▪ Pruebas Complementarias

Prueba
Prueba función tiroidea

◦ Patología Actual: Demencia

Patología	Fecha Inicio	Medicamentos Asignados
Demencia	01/11/2020	▪ Donepezilo ▪ Memantina

▪ Formularios

Fecha	Motivo Consulta	Desencadenante	Observaciones	Diagnóstico	Tratamiento	Frecuencia del tratamiento
01/11/2020	Olvido de la ruta de vuelta a casa		El paciente muestra pérdidas de memoria, dificultad para comunicarse y dificultad para manejar tareas complejas	Demencia	Se recomienda tomar Donepezilo para mejorar la memoria y Memantina para el aumento de la actividad cerebral	2 veces al día

▪ Pruebas Complementarias

Prueba
Prueba de estimulación con TRH

4.3. Vista 3

En esta vista trataremos de mostrar a todos los **familiares** junto con los **mensajes recibidos** por parte de los psiquiatras, de los cuales también mostraremos un mínimo de información. Para ello, el procedimiento se basará nuevamente en la función de agregación que nos proporciona python y que nos permite seleccionar información de varias colecciones.

No obstante, a diferencia de las demás consultas, tendremos que extraer datos de cuatro colecciones distintas: mensajes, familiares, pacientes y psiquiatras. La agregación será realizada sobre la colección de familiares ya que estos se tratan de los protagonistas de esta vista.

Por tanto, la función **Consulta_3 ()** hará uso de una secuencia de varios \$lookup de tal forma que el resultado de uno siempre sea la entrada del siguiente para así ir consolidando nuestro resultado. Finalmente podemos exponer la siguiente implementación de código:

```
def Consulta_3():
    result = familiares.aggregate([
        {
            '$lookup': {
                'from': 'paciente',
                'localField': 'paciente asociado',
                'foreignField': '_id',
                'as': 'paciente asociado'
            }
        },
        {
            '$lookup': {
                'from': 'mensaje',
                'localField': 'mensajes recibidos.mensaje',
                'foreignField': '_id',
                'as': 'mensajes recibidos'
            }
        },
        {
            '$lookup': {
                'from': 'psiquiatra',
                'localField': 'mensajes recibidos.emisor',
                'foreignField': '_id',
                'as': 'emisores'
            }
        }
    ])
    return result
```

Tras la consulta se generarán tres nuevos campos que me dispongo a describir a continuación:

- **Paciente asociado:** Aquel cuyo identificador se encuentra mencionado dentro del campo paciente asociado del familiar.
- **Mensajes recibidos:** Aquellos cuyo identificador se encuentra contenido en la lista de mensajes del familiar.
- **Emisores:** Aquellos psiquiatras que constan como emisores en los mensajes obtenidos anteriormente.

Todo ello provocará la generación de un código en formato json que es almacenado de forma opcional en el fichero **resultado_consulta_3.json** con la siguiente estructura:

```
1  [ {  
2    __id : "g1_id_familiar_00N",  
3    ... : "..."  
4    paciente asociado : [ { __id : "g1_id_paciente_00M",  
5                          ... : "..."} ],  
6    mensajes recibidos : [ { __id : "g1_id_mensaje_00L",  
7                           ... : "..."} ],  
8    emisores : [ { __id : "g1_id_psiquiatra_00P",  
9                 ... : "..."} ]  
10 } ]
```

El conversor analiza el resultado para generar el código xml que se almacena de forma opcional en el fichero **resultado_consulta_3.xml**

```

<all>
  <item>
    <_id>g1_id_familiar_00N</_id>
    <...>...</...>
    <paciente_asociado>
      <item>
        <_id>g1_id_paciente_00M</_id>
        <...>...</...>
      </item>
    </paciente_asociado>
    <mensajes_recibidos>
      <item>
        <_id>g1_id_mensaje_00L</_id>
        <...>...</...>
      </item>
    </mensajes_recibidos>
    <emisores>
      <item>
        <_id>g1_id_psiquiatria_00P</_id>
        <...>...</...>
      </item>
    </emisores>
  </item>
</all>

```

Finalmente realizamos la transformación XSLT a dicho resultado mediante la aplicación del archivo **vista_3.xsl**, el cual al igual que en el caso anterior, deberá realizar una actividad de clasificación adicional en la que relacione los psiquiatras de la lista de emisores con cada uno de los mensajes recibidos contemplados en su correspondiente campo.

Para ello, el procedimiento consistirá en un proceso reiterativo en el que para cada mensaje se realice la lectura del emisor, para así almacenarla en una variable y poder buscar este identificador dentro de la lista de emisores del familiar. Esto nos permitirá conocer el nombre, los apellidos y el cargo de la persona que envió cada mensaje.

```

<...>...</...>
<xsl:for each select = "all/item">
  <...>...</...>
  <xsl:for each select = "mensajes_recibidos/item">
    <xsl:variable name="id_emisor" select="emisor"/>
    <xsl:for each select = "../..emisores/item">
      <xsl:if test = "_id = $id_emisor">
        <...>...</...>
      </xsl:if>
    </xsl:for each>
  </xsl:for each>
  <...>...</...>
</xsl:for each>

```

Una vez hecho esto, la vista resultante es almacenada dentro del archivo **vista_3.html**, el cual tiene la siguiente apariencia para cada familiar:

• Francisca Pacheco Pérez

◦ Información personal

Nombre	Apellidos	DNI	Fecha de Nacimiento	Sexo	Email	Teléfono
Francisca	Pacheco Pérez	02058965R	18/09/1974	Mujer	franciscapacheco74@gmail.com	671234489

◦ Relación con paciente

Relación	Paciente asociado				Nivel de autorización
	Nombre	Apellidos	DNI	Fecha de Nacimiento	
Madre	Marta	Romero Pacheco	23456671C	21/02/1992	Parcial

◦ Mensajes recibidos

Emisor			Fecha	Contenido
Nombre	Apellidos	Cargo en el centro		
Sara	Sánchez Ortiz	Psiquiatra	27/12/2005	Marta presenta claros síntomas de padecer trastorno de estrés postraumático. Usted le puede servir de gran ayuda si se encarga de apartar de su vida cotidiana todos aquellos objetos, temas de conversación, programas de televisión, etc que le recuerden lo sucedido.
Sara	Sánchez Ortiz	Psiquiatra	24/10/2020	Marta necesita comer y sobre todo querer hacerlo. Intentar no forzar ingestas de comidas por obligación, si no demostrarle y hacerle ver que es lo mejor para ella. También es recomendable realizar paseos diarios de baja intensidad para que reconozca su agotamiento físico y la causa del mismo.

Tal y como se observa en la imagen, se puede contemplar la aparición de datos procedentes de las cuatro colecciones que hemos consultado durante el proceso de agregación.

4.4. Vista 4

Por último, hemos diseñado una vista adicional que nos proporcionase una visión de los datos desde una perspectiva distinta a las demás. En ella, trataremos de exponer para cada **enfermedad mental** contemplada en el sistema, el **total de pacientes** que las hayan padecido.

El procedimiento de consulta resulta bastante distinto a los anteriores si tenemos en cuenta que no tenemos una colección protagonista ni tampoco varias de las que elaborar una agregación. Es por esto, que para generar esta vista haremos un uso principal de la función de agrupación **\$group**.

No obstante, dado que nuestras enfermedades se alojan en varias posiciones distintas dentro de los datos de los pacientes, será necesario realizar dos agrupaciones distintas. Por un lado agruparemos todas las patologías previas de todos los pacientes, y por otro, el total de las actuales.

Esto va a provocar la posterior necesidad de unir ambos resultados de agrupación para así otorgarle un solo número de recuento a cada enfermedad. Sin embargo, esta operación no es posible realizarla con comandos tradicionales de Mongo. Por ello, dicha parte del proceso será elaborada **manualmente** para construir el resultado json paso a paso.

El procedimiento consistirá en realizar una lista de enfermedades uniendo lo leído en ambas agrupaciones, sumar el recuento obtenido en cada caso y finalmente contruir un array de objetos json cuyo contenido va a ser únicamente el nombre de la enfermedad y su recuento.

```

def Consulta_4():
    result1 = pacientes.aggregate([
        {
            '$unwind': '$patologias previas'
        },
        {
            '$group': {
                '_id': '$patologias previas.patologia',
                'count': { '$sum': 1 }
            }
        }
    ])
    result2 = pacientes.aggregate([
        {
            '$group': {
                '_id': '$patologia actual',
                'count': { '$sum': 1 }
            }
        }
    ])
    json_res = '['
    lista_enfermedades = list()
    docs = list()
    for doc in result1:
        if (doc['_id'] != None):
            docs.append(doc)
            lista_enfermedades.append(doc['_id'])
    for doc in result2:
        if (doc['_id'] != None):
            docs.append(doc)
            lista_enfermedades.append(doc['_id'])
    lista_enfermedades = list(set(lista_enfermedades))
    vector_count = list((0,)*len(lista_enfermedades))
    for i in range(0,len(lista_enfermedades)):
        for doc in docs:
            if(doc['_id'] == lista_enfermedades[i]):
                vector_count[i] += int(doc['count'])
            json_res = json_res + '{"enfermedad":"' + lista_enfermedades[i] + '",'
            json_res = json_res + 'total":"' + str(vector_count[i]) + '"},'
    json_res = json_res[0:1] + ']'
    result = json.loads(json_res)
    return result

```

Tras ello, nuestro programa se encarga de generar el código en formato json que es almacenado de forma opcional en el fichero **resultado_consulta_4.json** con la siguiente estructura:

```
1  [ {  
2    enfermedad: "...",  
3    total: ...  
4  } ]
```

Este resultado es analizado por el conversor para generar el correspondiente código xml y almacenarlo de forma opcional en el fichero **resultado_consulta_4.xml**. A continuación, me dispongo a mostrar su estructura:

```
<all>  
  <item>  
    <enfermedad>...</enfermedad>  
    <total>...</total>  
  </item>  
</all>
```

Finalmente realizamos la transformación XSLT a dicho resultado mediante la aplicación del archivo **vista_4.xsl** cuya unica función será mostrar en una tabla los datos obtenidos.

```
<...>...</...>  
<xsl:for each select="all/item">  
  <tr>  
    <td><xsl:value of select = "enfermedad"/></td>  
    <td><xsl:value of select = "total"/></td>  
  </tr>  
</xsl:for each>  
<...>...</...>
```


Una vez hecho esto, la vista resultante es almacenada dentro del archivo **vista_4.html**, cuyo contenido ilustro a continuación:

Informe enfermedades mentales:

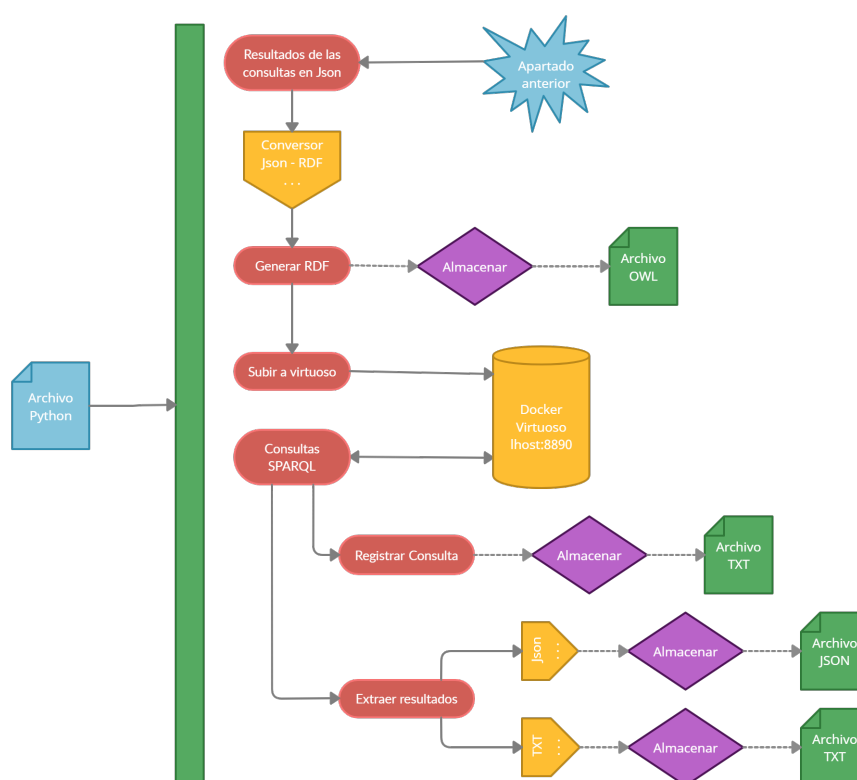
Enfermedad Mental	Total de casos
Alcoholemia	1
Drogadicción	7
Alcoholismo	4
Ansiedad	4
Ninfomania	1
Trastorno de estrés postraumático (TEPT)	2
Deficit de atención	1
Esquizofrenia	1
Trastorno de la personalidad	3
Depresión	12
Trastorno de ansiedad	4
Trastorno bipolar	3
Trastorno obsesivo-compulsivo (TOC)	1
Trastorno alimenticio	6
Depresión post-parto	1
Hiperactividad	1
Trastorno de ansiedad social	1
Demencia	3
Trastorno del Espectro Autista	2
Déficit de Atención e Hiperactividad	1
Síndrome de Asperger	1
Retraso mental severo	1
Síndrome de Diógenes	1
Psicosis	3

5

Vistas RDF

En este apartado del proyecto trataremos de utilizar las consultas anteriormente ilustradas con el fin de generar un **archivo rdf** para cada una de ellas. El resultado se almacenará en **formato owl**, de tal forma que podamos visualizar su contenido en **protégé**. Una vez comprobado que los archivos se han generado correctamente, cogeremos el código rdf generado en el programa para subirlo a **virtuoso** y poder así lanzar **consultas SPARQL** desde python.

Por tanto, al flujo de trabajo de nuestro programa, se le suma lo siguiente:



Con el fin de poder comprender mejor el programa, me dispongo a mostrar a continuación el procedimiento detallado del mismo junto con los resultados obtenidos en cada paso.

5.1. Generación del código RDF

Esta parte del código está compuesta por funciones que tratan de producir paso a paso el string correspondiente al rdf del resultado de cada consulta. Por ello, tendremos métodos especializados en:

- **Producción de clases:** Para ello tendremos una función que denominaremos **MeteClases()** a la que le pasaremos una lista con las clases que queremos definir junto con la URI del rdf que estemos construyendo.

```
def MeteClases (clases, URI):  
    res = ''  
    for c in clases:  
        res = res + '''        <!      ''' + URI + c + '''      >  
                                <owl:Class rdf:about="''' + URI + c + '''"/>  
'''  
    return res
```

- **Producción de object properties:** En este caso tendremos una función similar a la anterior solo que le proporcionaremos un diccionario cuyas claves especifiquen el nombre de las relaciones y los valores sean una lista especificando el dominio y el rango de las mismas.

```

def MeteOP(d_op, URI):
    res = ''
    for i in range(0, len(d_op)):
        res = res + '''
            <!''' + URI + list(d_op)[i] + ''' >
            <owl:ObjectProperty rdf:about="''' + URI + list(d_op)[i] + '''">
                <rdfs:domain rdf:resource="''' + URI + d_op[list(d_op)[i]
                    ][0] + '''"/>
                <rdfs:range rdf:resource="''' + URI + d_op[list(d_op)[i]
                    ][1] + '''"/>
            </owl:ObjectProperty>
'''
    return res

```

- **Producción de data properties:** Esta función resultará parecida a la anterior solo que en este caso el diccionario tomará como valores otros diccionarios, ya que a diferencia de antes sí se da el caso de tener múltiples dominios por lo que una estructura en forma de lista no es suficiente.

```

def MeteDP(d_dp, URI):
    res = ''
    for i in range(0, len(d_dp)):
        obj_dominio = d_dp[list(d_dp)[i]][ 'dominio' ]
        string_dominio = ''
        if (isinstance(obj_dominio, list)):
            for dom in obj_dominio:
                string_dominio = string_dominio + '
                    <rdfs:
                    domain rdf:resource=" ' + URI + dom + '"/> \n'
            else:
                string_dominio = '
                    <rdfs:domain rdf:resource=" '
                    + URI + obj_dominio + '"/> \n'
        res = (res + '
            <! ' + URI + list(d_dp)[i] + ' > \n'
            + '
                <owl:DatatypeProperty rdf:about=" ' + URI +
                list(d_dp)[i] + '"> \n'
            + string_dominio +
            '
                <rdfs:range rdf:resource="http://www.w3.org
                /2001/XMLSchema#' + d_dp[list(d_dp)[i]][ 'tipo' ] + '"/> \n
            + '
                </owl:DatatypeProperty> \n')
    return res

```

- **Producción de individuos:** Este caso es mucho más complejo puesto que existe una mayor diversidad de posibilidades y entra en juego la lectura de los códigos json resultantes de las consultas. Para abordar esta parte de la producción del archivo rdf, hemos diseñado una función para cada una de las clases que van a existir en nuestros documentos. No obstante, aunque en algunos casos pueda coincidir la existencia de la misma clase en dos archivos distintos, no se mostrará la misma información en cada uno de ellos. Es este el motivo de que en ocasiones estas funciones necesitan parámetros adicionales que les ayuden a tomar decisiones sobre qué mostrar y qué no.

Para no alargar la redacción de esta memoria mostraré simplemente las cabeceras de estas funciones cuya implementación completa se almacena en el script de python adjunto.

```
# Psiquiatras:
def CreaRDFPsiquiatra (psiquiatra, URI, muestra_datos_profesionales):

# Pacientes:
def CreaRDFPaciente (paciente, URI, muestra_patologias):

# Patologías:
def CreaRDFPatologia (patologia, ident, URI, formularios):

# Formularios de consultas:
def CreaRDFFormulario(formulario, URI):

# Familiares autorizados:
def CreaRDFFamiliar(familiar, URI):

# Mensajes:
def CreaRDFMensaje(mensaje, URI, emisor):

# Enfermedades:
def CreaRDFEnfermedad(enfermedad, URI):
```

- **Producción de comentarios:** El principal objetivo de esta función consiste en introducir comentarios que auncien cambios de apartado dentro del documento rdf. No afectarán a su visualización en protégé pero sí facilitan la lectura manual del mismo.

```
def MeteComentario (apartado):  
    res = ('' <!  
          ///////////////////////////////////  
          //  
          '' + apartado + ''  
          //  
          ///////////////////////////////////  
          > '' )  
    return res
```

Además de estas funciones, tenemos las correspondientes a cada consulta que se encargarán de hacer un uso adecuado de las anteriores con el fin de obtener el código rdf asociado a su documento json.

En ellas se definen las listas de clases, los diccionarios de las propiedades, los comentarios a introducir y la llamada a la función que le corresponda. Esta función será la referente a la creación de la clase principal de cada consulta, es decir, aquella asociada a la colección a la que se le han realizado todas las agregaciones necesarias durante su extracción en MongoDB.

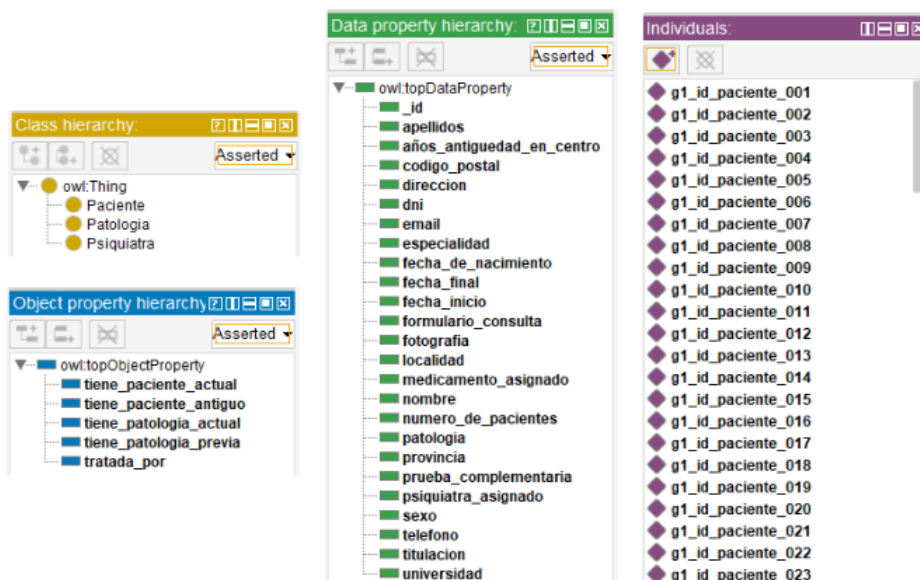
Será este método el que en función de los campos que contenga el diccionario proporcionado, decida llamar al resto de funciones creadoras para formar las demás entidades.

- **Consulta 1:** Psiquiatras junto con la información asociada a todos sus pacientes, tanto actuales como anteriores.

Por ello, la función principal a llamar será `CreaRDFPsiquiatra()`, la cual invocará a `CreaRDFPaciente()` cuando lea el campo agregado en la consulta y esta a su vez llame a `CreaRDFPatologia()` cuando observe el campo correspondiente a las patologías en el diccionario del paciente.

```
def GeneraRDFConsulta1():
    res_consulta = Consulta_1()
    URI='http://www.semanticweb.org/usuario/ontologies/2020/11/consulta_1#'
    clases = ['Psiquiatra', 'Paciente', 'Patologia']
    d_dp = {'_id':{'dominio':['Psiquiatra', 'Paciente'], 'tipo':'string'},
            'nombre':{'dominio':['Psiquiatra', 'Paciente'], 'tipo':'string'},
            ...}
    d_op = {'tiene_paciente_actual' : ['Psiquiatra', 'Paciente'], '
            tiene_paciente_antiguo' : ['Psiquiatra', 'Paciente'],
            'tiene_patologia_actual' : ['Paciente', 'Patologia'], '
            tiene_patologia_previa' : ['Paciente', 'Patologia'],
            'tratada_por' : ['Patologia', 'Psiquiatra']}
    rdf = '''<?xml version="1.0"?>
    <rdf:RDF xmlns="''' + URI + '''"
            xml:base="http://www.semanticweb.org/usuario/ontologies/2020/11/
            consulta_1"
            xmlns:owl="http://www.w3.org/2002/07/owl#" ...'''
    # Definimos las object properties
    rdf = rdf + MeteComentario('Object Properties')
    rdf = rdf + MeteOP(d_op, URI)
    # Definimos las data properties
    rdf = rdf + MeteComentario('Data Properties')
    rdf = rdf + MeteDP(d_dp, URI)
    # Definimos las clases
    rdf = rdf + MeteComentario('Classes')
    rdf = rdf + MeteClases(clases, URI)
    # Definimos los individuos
    rdf = rdf + MeteComentario('Individuals')
    for doc in res_consulta:
        lista = CreaRDFPsiquiatra(doc, URI, True)
        psiquiatra = lista[0]
        pacientes = lista[1]
        patologias = lista[2]
        rdf = rdf + psiquiatra + pacientes + patologias
    rdf = rdf + '</rdf:RDF>'
    GuardaArchivo(rdf, "Consultas rdf/consulta_1.owl")
    return rdf
```

Esta función nos devuelve un código rdf que subiremos posteriormente a virtuoso y que además es almacenado en disco en el fichero con extensión owl. Si abrimos dicho archivo en protégé podemos observar cómo se han creado correctamente todos los elementos que constituyen nuestra vista.



- **Consulta 2:** Pacientes junto a sus medicamentos, formularios y pruebas complementarias clasificadas por patología.

En este caso llamaremos a la función `CreaRDFPaciente()` que se encargará de utilizar la función `CreaRDFPatologia()` proporcionándole los diccionarios de los formularios correspondientes a la patología que estemos construyendo para que durante su formación se las transfiera al método `CreaRDFFormulario()`.

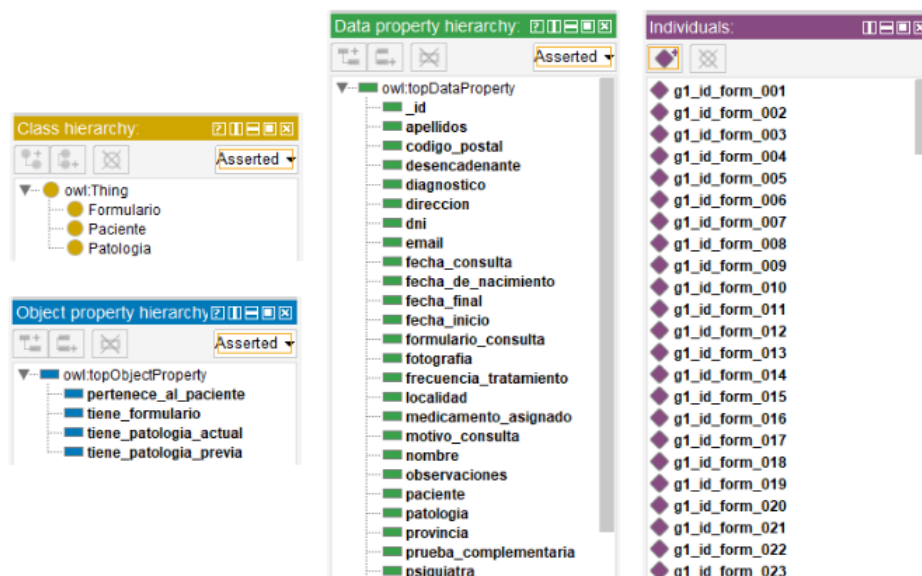
Cabe mencionar que al igual que antes puesto que las patologías no representan ninguna colección de nuestra base de datos se les asignará un identificador autogenerado que hace uso del id de los pacientes a los que pertenecen.


```

def GeneraRDFConsulta2():
    res_consulta = Consulta_2()
    URI='http://www.semanticweb.org/usuario/ontologies/2020/11/consulta_2#'
    clases = ['Paciente', 'Patologia', 'Formulario']
    d_dp = {'_id':{'dominio':['Paciente', 'Formulario'], 'tipo':'string'},
            'nombre':{'dominio':'Paciente', 'tipo':'string'},
            'apellidos':{'dominio':'Paciente', 'tipo':'string'},
            ...
            'patologia':{'dominio':'Patologia', 'tipo':'string'},
            'fecha_inicio':{'dominio':'Patologia', 'tipo':'dateTime'},
            'fecha_final':{'dominio':'Patologia', 'tipo':'dateTime'},
            ...
            'paciente':{'dominio':'Formulario', 'tipo':'string'},
            'psiquiatra':{'dominio':'Formulario', 'tipo':'string'},
            'fecha_consulta':{'dominio':'Formulario', 'tipo':'dateTime'},
            ...}
    d_op = {'tiene_formulario' : ['Patologia', 'Formulario'],
            'tiene_patologia_actual' : ['Paciente', 'Patologia'],
            'tiene_patologia_previa' : ['Paciente', 'Patologia'],
            'pertenece_al_paciente' : ['Formulario', 'Paciente']}
    rdf = '''<?xml version="1.0"?>
    <rdf:RDF xmlns="''' + URI + '''"
        xml:base="http://www.semanticweb.org/usuario/ontologies/2020/11/
        consulta_2"
        xmlns:owl="http://www.w3.org/2002/07/owl#" ...'''
    # Definimos las object properties
    rdf = rdf + MeteComentario('Object Properties')
    rdf = rdf + MeteOP(d_op, URI)
    # Definimos las data properties
    rdf = rdf + MeteComentario('Data Properties')
    rdf = rdf + MeteDP(d_dp, URI)
    # Definimos las clases
    rdf = rdf + MeteComentario('Classes')
    rdf = rdf + MeteClases(clases, URI)
    # Definimos los individuos
    rdf = rdf + MeteComentario('Individuals')
    for doc in res_consulta:
        lista = CreaRDFPaciente(doc, URI, True)
        pacientes = lista[0]
        patologias = lista[1]
        formularios = lista[2]
        rdf = rdf + pacientes + patologias + formularios
    rdf = rdf + '</rdf:RDF>'
    GuardaArchivo(rdf, "Consultas rdf/consulta_2.owl")
    return rdf

```

Su resultado al igual que el resto es posteriormente subido a virtuoso y almacenado en el archivo owl correspondiente. Si abrimos dicho fichero en protégé podemos observar la correcta construcción de clases, propiedades e individuos:



- **Consulta 3:** Familiares junto con información asociada a los pacientes así como los mensajes recibidos, de los cuales se expone cierta información de los emisores.

Es por esto, que en este caso la función principal será `CreaRDFFamiliar()`, la cual llama a `CreaRDFPaciente()` y `CreaRDFMensaje()` cuando observa los campos correspondientes. Además, a esta última se le proporcionará el diccionario del emisor, que al ser siempre un psiquiatra hará uso de la función `CreaRDFPsiquiatra()` para generar su rdf a la vez que excluye información profesional del mismo, puesto que no nos interesa su especificación en esta vista. El código de la función asociada es similar a las anteriores y se muestra a continuación:

```

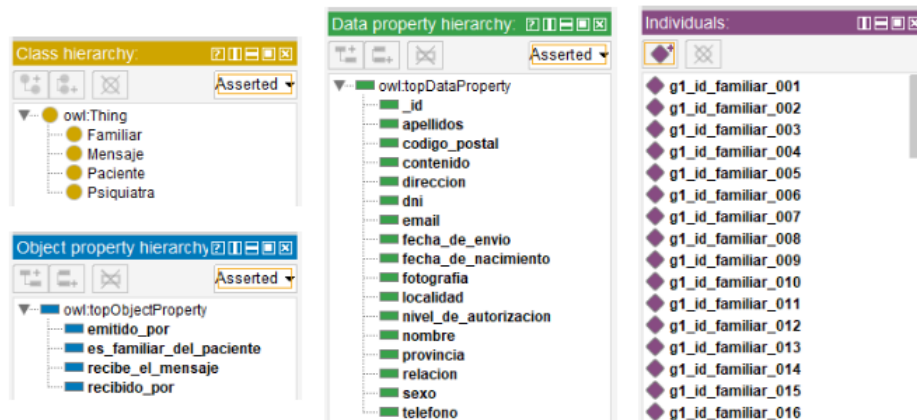
def GeneraRDFConsulta3():
    res_consulta = Consulta_3()
    URI='http://www.semanticweb.org/usuario/ontologies/2020/11/consulta_3#'
    clases = ['Familiar', 'Paciente', 'Mensaje', 'Psiquiatra']
    d_dp = {'_id':{'dominio':['Familiar', 'Paciente', 'Mensaje', 'Psiquiatra']},
            'tipo':'string'},
            'nombre':{'dominio':['Familiar', 'Paciente', 'Psiquiatra'], 'tipo':
                        'string'},
            'apellidos':{'dominio':['Familiar', 'Paciente', 'Psiquiatra'], '
                        tipo':'string'},
            ...
            'relacion':{'dominio':'Familiar', 'tipo':'string'},
            'nivel_de_authorized':{'dominio':'Familiar', 'tipo':'string'},
            'fecha_de_envio':{'dominio':'Mensaje', 'tipo':'dateTime'},
            'contenido':{'dominio':'Mensaje', 'tipo':'string'}}
    d_op = {'es_familiar_del_paciente' : ['Familiar', 'Paciente'],
            'recibe_el_mensaje' : ['Familiar', 'Mensaje'],
            'emitido_por' : ['Mensaje', 'Psiquiatra'],
            'recibido_por' : ['Mensaje', 'Familiar']}

    rdf = '''<?xml version="1.0"?>
    <rdf:RDF xmlns="''' + URI + '''"
        xml:base="http://www.semanticweb.org/usuario/ontologies/2020/11/
        consulta_3"
        xmlns:owl="http://www.w3.org/2002/07/owl#" ...'''

    # Definimos las object properties
    rdf = rdf + MeteComentario('Object Properties')
    rdf = rdf + MeteOP(d_op, URI)
    # Definimos las data properties
    rdf = rdf + MeteComentario('Data Properties')
    rdf = rdf + MeteDP(d_dp, URI)
    # Definimos las clases
    rdf = rdf + MeteComentario('Classes')
    rdf = rdf + MeteClases(clases, URI)
    # Definimos los individuos
    rdf = rdf + MeteComentario('Individuals')
    for doc in res_consulta:
        lista = CreaRDFFamiliar(doc, URI)
        familiares = lista[0]
        pacientes = lista[1]
        mensajes = lista[2]
        psiquiatras = lista[3]
        rdf = rdf + familiares + pacientes + mensajes + psiquiatras
    rdf = rdf + '</rdf:RDF>'
    GuardaArchivo(rdf, "Consultas rdf/consulta_3.owl")
    return rdf

```

Una vez generado el rdf resultante abrimos el fichero con extensión owl para poder visualizarlo en protégé.

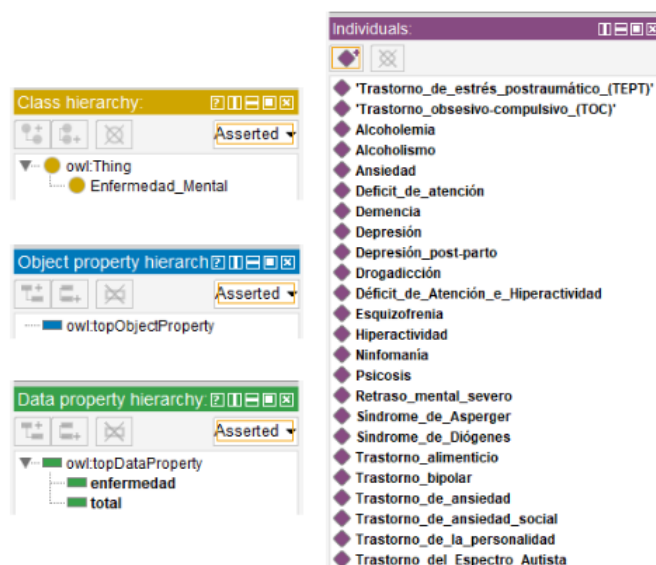


- **Consulta 4:** Se muestran todas las enfermedades mentales junto con el total de casos registrados en el sistema.

Para esta última vista utilizaremos sólo la función `CreaRDFEnfermedad()`.

```
def GeneraRDFConsulta4():
    res_consulta = Consulta_4()
    URI='http://www.semanticweb.org/usuario/ontologies/2020/11/consulta_4#'
    clases = ['Enfermedad_Mental']
    d_dp = {'enfermedad':{'dominio':'Enfermedad_Mental','tipo':'string'},
            'total':{'dominio':'Enfermedad_Mental','tipo':'int'}}
    rdf = '''...'''
    # Definimos las data properties
    rdf = rdf + MeteComentario('Data Properties')
    rdf = rdf + MeteDP(d_dp, URI)
    # Definimos las clases
    rdf = rdf + MeteComentario('Classes')
    rdf = rdf + MeteClases(clases, URI)
    # Definimos los individuos
    rdf = rdf + MeteComentario('Individuals')
    for doc in res_consulta:
        rdf = rdf + CreaRDFEnfermedad(doc, URI)
    rdf = rdf + '</rdf:RDF>'
    GuardaArchivo(rdf, "Consultas rdf/consulta_4.owl")
    return rdf
```

Al igual que siempre comprobamos la correcta ejecución de nuestro código mediante la revisión del fichero en protégé.



5.2. Subida de código rdf a virtuoso

PROVISIONAL: A la espera de obtener respuesta de Antonio Benitez este proceso ha sido realizado manualmente de forma provisional.

5.3. Lanzamiento de consultas SPARQL

Para poder realizar consultas a los grafos situados en virtuoso, hemos construido una función que nos permite de forma adicional registrar la consulta realizada en un txt así como almacenar los resultados tanto en formato json como de texto plano. En caso de no querer cargar nada en disco, se especifica el parámetro guardar como false para que así muestre de forma alternativa los resultados obtenidos por consola.

```

def SPARQL(query, enunciado, num_consulta, num_subconsulta, guardar):
    endpoint = "http://127.0.0.1:8890/sparql"
    sparql = SPARQLWrapper(endpoint)
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    str_res = enunciado + '\n'
    for result in results["results"]["bindings"]:
        str_res = str_res + '\n'
        for columna in list(result):
            str_res = str_res + result[columna]["value"] + "    "
        str_res = str_res[0:len(str_res) 4]
    if(guardar):
        GuardaArchivo(enunciado + '\n\n' + query, "Consultas rdf/
            Subconsultas C" + str(num_consulta) + "/query_subconsulta_" +
            str(num_subconsulta) + ".txt")
        GuardaArchivo(json.dumps(results, indent = 4, ensure_ascii = False)
            , "Consultas rdf/Subconsultas C" + str(num_consulta) + "/"
            resultado_subconsulta_" + str(num_subconsulta) + ".json")
        GuardaArchivo(str_res, "Consultas rdf/Subconsultas C" + str(
            num_consulta) + "/resultado_subconsulta_" + str(num_subconsulta)
            + ".txt")
    else:
        pp = pprint.PrettyPrinter(indent = 4)
        pp.pprint(results)
        print(str_res)

```

A estas consultas SPARQL les llamaremos subconsultas ya que estamos extrayendo información del resultado de consultas anteriores realizadas a nuestra base de datos original. Por otro lado, dado que en nuestro caso dichas subconsultas ya se encuentran prefijadas, las hemos almacenado en un diccionario para proporcionarselas de forma secuencial a la función anterior.

Para ello, definimos el siguiente método al que le pasamos dicho diccionario.

```

def EjecutaTodasSubconsultas (subconsultas, guardar):
    for subconsulta in subconsultas:
        SPARQL(subconsulta['query'], subconsulta['enunciado'], subconsulta[
            'num_consulta'], subconsulta['num_subconsulta'], guardar)

```

A continuación me dispongo a mostrar algunos ejemplos junto con los resultados obtenidos:

- **Sub1Con1:** Los psiquiatras junto con las patologías que están tratando actualmente.

```
PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
1#>
SELECT DISTINCT ?nombre_psiquiatra ?apellidos_psiquiatra ?
nombre_patologia_actual
WHERE {{?psiquiatra rdf:type esq:Psiquiatra .
?psiquiatra esq:nombre ?nombre_psiquiatra .
?psiquiatra esq:apellidos ?apellidos_psiquiatra .
?patologia_actual rdf:type esq:Patologia .
?patologia_actual esq:tratada_por ?psiquiatra .
?patologia_actual esq:patologia ?nombre_patologia_actual }
FILTER NOT EXISTS {?patologia_actual esq:fecha_final ?fecha_final}}
ORDER BY ?nombre_psiquiatra ?nombre_patologia_actual
```

Ana – Sánchez Tomás – Alcoholismo

Ana – Sánchez Tomás – Drogadicción

Antonio Jesús – Ramírez Loto – Depresión

Francisco – Sepúlveda Rey – Depresión

José María – Leiva Jiménez – Ansiedad

José María – Leiva Jiménez – Demencia

José María – Leiva Jiménez – Drogadicción

Lorena – González Seco – Depresión

Lorena – González Seco – Trastorno de la personalidad

...

- **Sub2Con1:** Psiquiatras que hayan tratado anteriormente Alcoholismo.

```

PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
1#>
SELECT ?nombre_psiquiatra ?apellidos_psiquiatra
WHERE {?psiquiatra rdf:type esq:Psiquiatra .
      ?psiquiatra esq:nombre ?nombre_psiquiatra .
      ?psiquiatra esq:apellidos ?apellidos_psiquiatra .
      ?patologia rdf:type esq:Patologia .
      ?patologia esq:tratada_por ?psiquiatra .
      ?patologia esq:patologia "Alcoholismo" .
      ?patologia esq:fecha_final ?fecha_final}

```

Lucía – Manrique Pérez

Miguel – Pérez López

- **Sub3Con1:** Los psiquiatras con sus especialidades.

```

PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
1#>
SELECT ?nombre ?apellidos ?especialidad
WHERE {?psiquiatra rdf:type esq:Psiquiatra .
      ?psiquiatra esq:nombre ?nombre .
      ?psiquiatra esq:apellidos ?apellidos .
      ?psiquiatra esq:especialidad ?especialidad}

```

Sara – Sánchez Ortiz – Psiquiatría general

Roberto – Fernández Rojas – Psiquiatría general

Lucía – Manrique Pérez – Neuropsiquiatría

Francisco – Sepúlveda Rey – Psiquiatría infanto-juvenil

José María – Leiva Jiménez – Psiquiatría general

Ana – Sánchez Tomás – Psiquiatría general

Manuel – Gómez Cabello – Neuropsiquiatría

Lorena – González Seco – Psiquiatría general

Antonio Jesús – Ramírez Loto – Psiquiatría general

Miguel – Pérez López – Psiquiatría general

- **Sub4Con1:** Psiquiatras con el total de patologías tratadas, tanto previas como actuales.

```
PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
1#>
SELECT ?nombre_psiquiatra ?apellidos_psiquiatra (count(?patologia) as ?
total_patologias)
WHERE {?psiquiatra rdf:type esq:Psiquiatra .
      ?psiquiatra esq:nombre ?nombre_psiquiatra .
      ?psiquiatra esq:apellidos ?apellidos_psiquiatra .
      ?patologia rdf:type esq:Patologia .
      ?patologia esq:tratada_por ?psiquiatra }
GROUP BY ?nombre_psiquiatra ?apellidos_psiquiatra
```

José María – Leiva Jiménez – 5

Ana – Sánchez Tomás – 5

Miguel – Pérez López – 4

Francisco – Sepúlveda Rey – 4

Roberto – Fernández Rojas – 11

Lorena – González Seco – 4

Antonio Jesús – Ramírez Loto – 4

Sara – Sánchez Ortiz – 16

Lucía – Manrique Pérez – 8

Manuel – Gómez Cabello – 4

- **Sub1Con2:** Todos los pacientes que hayan tenido o tengan actualmente depresión.

```
PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
2#>
SELECT ?nombre ?apellidos ?inicio ?final
  WHERE {{?paciente rdf:type esq:Paciente .
          ?paciente esq:tiene_patologia_previa ?patologia_previa .
          ?patologia_previa esq:patologia "Depresión" .
          ?paciente esq:nombre ?nombre .
          ?paciente esq:apellidos ?apellidos .
          ?patologia_previa esq:fecha_inicio ?inicio .
          ?patologia_previa esq:fecha_final ?final}}
 UNION { ?paciente rdf:type esq:Paciente .
          ?paciente esq:tiene_patologia_actual ?patologia_actual .
          ?patologia_actual esq:patologia "Depresión" .
          ?paciente esq:nombre ?nombre .
          ?paciente esq:apellidos ?apellidos .
          ?patologia_actual esq:fecha_inicio ?inicio }}
```

Marta – Romero Pacheco – 23/08/2009 – 03/02/2010

Mario – Martín Pérez – 19/03/2011 – 01/02/2012

Carmen – Castro Romero – 12/10/2009 – 03/02/2010

Carmen – Castro Romero – 14/07/2012 – 15/01/2013

Marta – Cantos Martín – 15/09/2016 – 03/01/2017

Álvaro – Torres Sánchez – 13/02/2017 – 15/08/2017

Alberto – Álvarez Martín – 18/08/2018 – 02/01/2019

Clara – Labrador Vera – 11/09/2020

Raquel – Sutton Rivera – 12/09/2020

Jorge – Vázquez Rivera – 12/11/2020

Jorge – Cañete Vera – 15/09/2020

Ana – Soria Vera – 05/07/2020

- **Sub2Con2:** Los pacientes que hayan sido diagnosticados de trastorno alimenticio junto con los formularios de las consultas donde se haya especificado dicho diagnóstico.

```
PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
2#>
SELECT ?nombre ?apellidos ?fecha_consulta ?motivo_consulta ?observaciones
WHERE { ?paciente rdf:type esq:Paciente .
        ?paciente esq:nombre ?nombre .
        ?paciente esq:apellidos ?apellidos .
        ?formulario rdf:type esq:Formulario .
        ?formulario esq:pertenece_al_paciente ?paciente .
        ?formulario esq:diagnostico "Trastorno alimenticio" .
        ?formulario esq:fecha_consulta ?fecha_consulta .
        ?formulario esq:motivo_consulta ?motivo_consulta .
        ?formulario esq:observaciones ?observaciones}
```

Marta – Romero Pacheco – 24/10/2020 – Preocupante pérdida de peso – La paciente presenta palidez, falta de energía y delgadez extrema

David – Lasa Ordoñez – 13/05/2016 – Revisión – El paciente presenta un claro aumento de peso y gran mejoría psicológica

Inmaculada – Reyes Sánchez – 27/09/2019 – Extrema delgadez – La paciente presenta un temor intenso al sobrepeso

Mireya – Grande Salado – 17/09/2016 – Inanición voluntaria – La paciente asegura que hasta que no tenga el cuerpo de las modelos que ve en redes sociales no comerá

...

- **Sub3Con2:** Los pacientes junto con su patología actual y el tratamiento indicado en las consultas.

```
PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
2#>
SELECT ?nombre ?apellidos ?nombre_patologia_actual ?tratamiento_asignado
WHERE { ?paciente rdf:type esq:Paciente .
        ?paciente esq:nombre ?nombre .
        ?paciente esq:apellidos ?apellidos .
        ?paciente esq:tiene_patologia_actual ?patologia_actual .
        ?patologia_actual esq:patologia ?nombre_patologia_actual .
        ?patologia_actual esq:tiene_formulario ?formulario .
        ?formulario esq:tratamiento ?tratamiento_asignado}
```

Marta – Romero Pacheco – Trastorno alimenticio – Se recomienda tomar ciproheptadina para estimular el apetito

Olivia – Cañete Domínguez – Trastorno bipolar – Se recomienda tomar Olanzapina para estabilizar el ánimo

Lucas – Romero Padilla – Trastorno de la personalidad – Se recomienda tomar Amitriptilina

Mario – Martín Pérez – Demencia – Se recomienda tomar Donepezilo para mejorar la memoria y Memantina para el aumento de la actividad cerebral

...

- **Sub1Con3:** Pacientes junto con sus familiares y el nivel de autorización.

```
PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
3#>
SELECT ?nombre_paciente ?apellidos_paciente ?relacion ?nombre_familiar ?
apellidos_familiar ?nivel_de_autorizacion
WHERE { ?paciente rdf:type esq:Paciente .
?paciente esq:nombre ?nombre_paciente .
?paciente esq:apellidos ?apellidos_paciente .
?familiar rdf:type esq:Familiar .
?familiar esq:es_familiar_del_paciente ?paciente .
?familiar esq:nombre ?nombre_familiar .
?familiar esq:apellidos ?apellidos_familiar .
?familiar esq:relacion ?relacion .
?familiar esq:nivel_de_autorizacion ?nivel_de_autorizacion}
```

Marta – Romero Pacheco – Madre – Francisca – Pacheco Pérez – Parcial

Olivia – Cañete Domínguez – Marido – Paco – Segura Vera – Total

Lucas – Romero Padilla – Padre – Fran – Romero Sánchez – Normal

Mario – Martín Pérez – Hermana – Ana – Martín Pérez – Normal

Silvia – Román Huerta – Madre – María – Huerta Ordoñez – Parcial

Lucas – Pirlo Molina – Hermano – Juan – Pirlo Molina – Normal

José Ramón – López Galiana – Madre – Nerea – Galiana Trujillo – Normal

Melania – Castillo del Río – Novio – Jesús – Ruiz Segura – Normal

...

- **Sub2Con3:** Los familiares junto con los mensajes recibidos y el paciente al que se hace referencia.

```

PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
3#>
SELECT ?nombre_familiar ?apellidos_familiar ?fecha ?contenido ?
nombre_paciente ?apellidos_paciente
WHERE { ?familiar rdf:type esq:Familiar .
?familiar esq:nombre ?nombre_familiar .
?familiar esq:apellidos ?apellidos_familiar .
?mensaje rdf:type esq:Mensaje .
?mensaje esq:recibido_por ?familiar .
?mensaje esq:contenido ?contenido .
?mensaje esq:fecha_de_envio ?fecha .
?paciente rdf:type esq:Paciente .
?familiar esq:es_familiar_del_paciente ?paciente .
?paciente esq:nombre ?nombre_paciente .
?paciente esq:apellidos ?apellidos_paciente}

```

Francisca – Pacheco Pérez – 27/12/2005 – Marta presenta claros síntomas de padecer trastorno de estrés postraumático. Usted le puede servir de gran ayuda si se encarga de apartar de su vida cotidiana todos aquellos objetos, temas de conversación, programas de televisión, etc que le recuerden lo sucedido. – Marta – Romero Pacheco

Ana – Martín Pérez – 02/11/2020 – Si Mario sufre un episodio de demencia, debe tomar Donepezilo 1gr junto con un sobre de Memantina. – Mario – Martín Pérez

Fran – Romero Sánchez – 02/11/2020 – Lucas padece un leve trastorno, sin embargo, vemos beneficioso tratar su desencadenante más que su propia enfermedad. Nos gustaría citarle a usted para poder hablar respecto ciertos asuntos. – Lucas – Romero Padilla

Ángela – Sánchez Ruiz – 10/11/2020 – Ana María ha empeorado su estado y

su forma de afrontar las cosas. Le hemos aumentado la dosis hasta que
consigamos estabilizarla – Ana María – Nevado Sánchez

...

■ **Sub3Con3:** Psiquiatras con el número de mensajes enviados.

```
PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
3#>
SELECT ?nombre_psiquiatra ?apellidos_psiquiatra (count(?mensaje) as ?
total_mensajes_enviados)
WHERE { ?psiquiatra rdf:type esq:Psiquiatra .
?psiquiatra esq:nombre ?nombre_psiquiatra .
?psiquiatra esq:apellidos ?apellidos_psiquiatra .
?mensaje rdf:type esq:Mensaje .
?mensaje esq:emitido_por ?psiquiatra }
GROUP BY ?nombre_psiquiatra ?apellidos_psiquiatra
```

José María – Leiva Jiménez – 4

Ana – Sánchez Tomás – 5

Miguel – Pérez López – 4

Francisco – Sepúlveda Rey – 4

Roberto – Fernández Rojas – 12

Lorena – González Seco – 4

Antonio Jesús – Ramírez Loto – 4

Sara – Sánchez Ortiz – 17

Lucía – Manrique Pérez – 10

Manuel – Gómez Cabello – 4

- **Sub1Con4:** Enfermedades ordenadas por el número de casos.

```
PREFIX rdf: <http://www.w3.org/1999/02/22 rdf syntax ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX esq: <http://www.semanticweb.org/usuario/ontologies/2020/11/consulta
4#>
SELECT ?nombre_enfermedad ?apariciones
WHERE {?enfermedad rdf:type esq:Enfermedad_Mental .
      ?enfermedad esq:enfermedad ?nombre_enfermedad .
      ?enfermedad esq:total ?apariciones }
ORDER BY DESC (?apariciones)
```

Depresión – 12

Drogadicción – 7

Trastorno alimenticio – 6

Alcoholismo – 4

Ansiedad – 4

Trastorno de ansiedad – 4

Demencia – 3

...

6

Web Application

En este apartado trataremos de exponer de forma clara y concisa los principales aspectos de la construcción de nuestra aplicación web definitiva. Cabe destacar que finalmente hemos logrado cumplir con todos los requisitos inicialmente establecidos además de incorporar funcionalidades adicionales relacionadas con el análisis computacional de los informes llevados a cabo por el psiquiatra. Esto a provocado la inclusión de nuevas colecciones en nuestra base de datos de Mongo y por tanto la incorporación de nuevos datos compatibles con la población establecida previamente.

6.1. Implementación

PsycoTreat ha sido implementada a través de python mediante la elaboración de una API Rest con Flask. Este código se encuentra en el script principal denominado **app.py** y se nutre de archivos html, jinja2, javascript y hojas de estilo css para conformar la totalidad de la aplicación web. Dado que el código es proporcionado como parte de la entrega adjunta a este documento y que su extensión es extremadamente amplia, sólo mostraré los fragmentos de código que me parezcan imprescindibles para la comprensión de las posteriores explicaciones.

El primer paso consiste en la importación de librerías que se encuentran detalladas en el archivo **requirements.txt** de nuestro entorno. Además, en dichas importaciones se incluyen dos scripts complementarios relacionados con las funciones adicionales que han sido separadas del código principal con el fin de conseguir su clara distinción.

```
from flask import Flask, render_template, url_for, request, session,
    redirect
from flask_restful import reqparse, abort, Resource
from flask_pymongo import pymongo
import json
import base64
import datetime
import enfermedades
import formulario_aprox
```

Tras esto, debemos conectarnos a las distintas colecciones de la base de MongoDB Atlas construida en apartados anteriores, con el fin de poder acceder a sus datos y realizar operaciones con los mismos desde la interfaz de nuestra aplicación.

```
CONNECTION_STRING_MY_DB = "mongodb+srv://adriansegura:
    adrianseguraortiz1999@psicotreatbd.e4kvw.mongodb.net/PsycoTreat?
    retryWrites=true&w=majority"
my_client = pymongo.MongoClient(CONNECTION_STRING_MY_DB)
my_db = my_client.get_database('PsycoTreat')
```

Además, para poder gestionar el inicio de sesión de los distintos usuarios debemos acceder a la base de datos elaborada conjuntamente”donde se almacenan los nombres de usuario a las contraseñas de nuestra población.

```

CONNECTION_STRING_GROUP_DB = "mongodb+srv://adriansegura:
    adrianseguraortiz1999@cluster0.k8hpd.mongodb.net/<Estandares>?
    retryWrites=true&w=majority"
group_client = pymongo.MongoClient(CONNECTION_STRING_GROUP_DB)
group_db = group_client.get_database("Estandares")

```

Tras acceder a nuestra información sólo nos queda poner en marcha las funciones de flask y definir una sucesión de direcciones web asociadas a distintas páginas html mediante el uso de **render_template()**. Muestro a continuación, el ejemplo de nuestra página de inicio:

```

app = Flask(__name__)
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

@app.route('/')
def pagina_principal():
    return render_template("home.html")

```

Esta operación se complica a medida que vamos añadiendo funcionalidades a nuestras páginas. Por ejemplo, en caso de tener un formulario necesitaremos diferenciar los métodos GET y POST, lo cual se realiza de la siguiente forma:

```

@app.route('/psiquiatra/<id_psiquiatra>/mis_pacientes/anadir_consulta/<id_paciente>', methods = ['GET', 'POST'])
def psiquiatra_mis_pacientes_anadir_consulta(id_psiquiatra, id_paciente):
    if request.method == 'GET':
        return render_template("formularios/formulario_consulta.html.j2",
            id_psiquiatra = id_psiquiatra)
    elif request.method == 'POST':
        id = "g1_id_form_" + str(col_formularios.count() + 1).zfill(3)
        fecha = datetime.date.today().strftime('%Y %m %d')
        motivo = request.form['motivo']
        desencadenante = request.form['desencadenante']
        observaciones = request.form['observaciones']
        diagnostico = request.form['diagnostico']
        tratamiento = request.form['tratamiento']
        frecuencia = request.form['frecuencia']

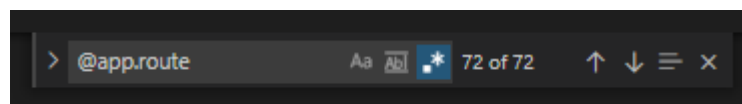
```

```

aproximacion = formulario_aprox.damePorcentajes(motivo,
    desencadenante, observaciones, diagnostico)
formulario = {"_id" : id, "psiquiatra" : id_psiquiatra, "paciente"
    : id_paciente, "fecha consulta" : fecha,
    "motivo consulta" : motivo, "desencadenante" : desencadenante, "
    observaciones" : observaciones, "diagnostico" : diagnostico,
    "tratamiento" : tratamiento, "frecuencia tratamiento" : frecuencia,
    "aproximacion computacional" : aproximacion}
col_formularios.insert_one(formulario)
col_pacientes.update_one({"_id" : id_paciente}, {"\ $push" : {"
    formularios consultas" : {"formulario" : id}}})
return psiquiatra_mis_pacientes_tratar(id_psiquiatra, id_paciente)

```

Nuestra aplicación cuenta con un total de 72 direcciones, asociadas por tanto a 72 páginas distintas donde se intentan abarcar todas las funcionalidades de nuestro trabajo.



Finalmente, comentar que todas las páginas html y sus correspondientes hojas de estilo css han sido elaboradas manualmente salvo la página de inicio de sesión donde se ha hecho uso de una plantilla de bootstrap.

6.2. Interfaz

A continuación, muestro algunas capturas de las páginas principales de cada usuario con el fin de tener una primera impresión del trabajo elaborado.

■ Administrativo

Desde esta página el administrativo podrá seleccionar a cualquier tipo de usuario para gestionar sus registros. Se contemplan la visualización, edición, eliminación y creación de los distintos usuarios del sistema.

Registro de Usuarios

Seleccione el tipo de usuario:

[Paciente](#)
[Psiquiatra](#)
[Familiar](#)
[Administrativo](#)

© Universidad de Málaga

[Privacidad](#) [Aviso legal](#)

Eligiendo a los pacientes obtendríamos el siguiente listado:

Listado de pacientes

	Nombre	Apellidos	Email	Localidad	Teléfono			
001	Marta	Romero Pacheco	martaromero92@gmail.com	Málaga	669245569	Ver	Editar	Borrar
002	Olivia	Cañete Domínguez	oliviacanete_d90@gmail.com	Málaga	695874412	Ver	Editar	Borrar
003	Lucas	Romero Padilla	lucas_romero95@gmail.com	Málaga	632545885	Ver	Editar	Borrar
004	Mario	Martín Pérez	mariomp1998@gmail.com	Málaga	652225874	Ver	Editar	Borrar
005	Silvia	Román Huerta	silvia_roman00@gmail.com	Málaga	698554782	Ver	Editar	Borrar

■ Psiquiatra

Esta página define la zona en la que podrá tratar a sus pacientes actuales mediante la elaboración de nuevos informes, petición de pruebas clínicas, asignación de medicamentos, etc. Además, podrá finalizar y comenzar los tratamientos que desee, así como consultar historiales médicos.

Pacientes

Pacientes actuales

	Nombre	Apellidos	Email	Localidad	Teléfono			
024	Raquel	Sutton Rivera	raquelsutton97@gmail.com	Málaga	619565731	Ver	Tratar	Finalizar tratamiento

Pacientes antiguos

	Nombre	Apellidos	Email	Localidad	Teléfono			
022	Inmaculada	Reyes Sánchez	inmareyes99@gmail.com	Málaga	634528215	Ver	Empezar tratamiento	
023	Paula	Pestaña Cochar	paulapesti99@gmail.com	Málaga	633128221	Ver	Empezar tratamiento	
025	Paloma	Trinidad Conejo	palematrinidad96@gmail.com	Málaga	621321780	Ver	Empezar tratamiento	

© Universidad de Málaga

[Privacidad](#) [Aviso legal](#)

El tratamiento actual de un paciente tendría el siguiente aspecto:

Paciente: Raquel Sutton Rivera

Tratar

Patología: **depresión (F33.0)** [Editar patología](#)
Fecha de inicio: **2020-09-12**
Formularios: **Formulario 056** [Nuevo formulario](#)
Medicamentos: **Sertralina** [Añadir medicamento](#)
Registrar primera prueba clínica: [Añadir prueba](#)

Solicitud de pruebas complementarias


Nombre	Apellidos	Titulación	
Pedro	Ruiz Benitez	técnico informático	Enviar mensaje
Gloria	Duarte López	técnico informático	Enviar mensaje
Jesús	Romero Castillo	Auxiliar administrativo	Enviar mensaje
María Isabel	Del Campo Padilla	Auxiliar administrativo	Enviar mensaje

Proporción de consejos a familiares

Nombre	Apellidos	Relación	Nivel de autorización	
Roberto	Sutton Rivera	Hermano	Normal	Enviar mensaje

■ Paciente

En esta página el paciente podrá gestionar sus citas para acceder a las consultas médicas de su psiquiatra asociado. Podrá tanto añadir como eliminar citas.



PsycoTreat

[Home](#)[Citas](#)[Mis datos](#)[Facturas](#)[Psiquiatras](#)[Salir](#)

Listado de citas

Citas Programadas		
2021-06-18	09:00-09:30	Borrar
2021-07-20	13:00-13:30	Borrar

[Añadir](#)

© Universidad de Málaga

[Privacidad](#) [Aviso legal](#)

■ Familiar autorizado

El familiar podrá acceder a los mensajes recibidos por parte del psiquiatra y obtener una visualización detallada de los mismos.

Mensajes

Mensajes recibidos

De parte de			Fecha
Nombre	Apellidos	Tipo de usuario	
Sara	Sánchez Ortiz	psiquiatra	2005-12-27 Leer
Sara	Sánchez Ortiz	psiquiatra	2020-10-24 Leer

Por ejemplo, si quisiéramos visualizar el primero de ellos:


PsycoTreat

[Home](#)
[Mensajes](#)
[Citas](#)
[Facturas](#)
[Salir](#)

Emisor:

- Nombre: Sara
- Apellidos: Sánchez Ortiz
- Tipo usuario: psiquiatra

Receptor:

- Nombre: Francisca
- Apellidos: Pacheco Pérez
- Tipo usuario: familiar

Fecha de envío: 2005-12-27

Contenido: Marta presenta claros síntomas de padecer trastorno de estrés postraumático. Usted le puede servir de gran ayuda si se encarga de apartar de su vida cotidiana todos aquellos objetos, temas de conversación, programas de televisión, etc que le recuerden lo sucedido.

© Universidad de Málaga

[Privacidad](#) [Aviso legal](#)

6.3. Ejecución

Para ejecutar nuestro programa será necesario abrirlo en una herramienta similar a VSC y realizar los siguientes pasos:

1. Activación del entorno escribiendo `.\Scripts\activate`
2. Instalar paquetes mediante la ejecución de la sentencia `pip install requirements.txt`
3. Poner en marcha el script mediante la especificación de `python .\app.py`
4. Hacer `Ctrl + Click` sobre el enlace generado.

Iniciamos sesión con las siguientes credenciales:

■ **Administrativo**

- Usuario: g1_id_admin_001
- Contraseña: admin_54

■ **Psiquiatra**

- Usuario: g1_id_psiquiatra_001
- Contraseña: psico_32

■ **Paciente**

- Usuario: g1_id_paciente_001
- Contraseña: pac_83

■ **Familiar** (Nivel de autorización Parcial)

- Usuario: g1_id_familiar_001
- Contraseña: fami_21

6.4. Vídeo

Además de lo descrito hasta ahora se adjunta en la entrega un vídeo que trata de exponer el funcionamiento de nuestra aplicación.