**Iraqi Journal of Intelligent Computing and Informatics (IJICI)**
Vol. 2, 1, June 2023, pp. 26~35
ISSN: 2791-2868, DOI: 10.52940/ijici.v2i1.31

# Sidr tree disease diagnosis system Python programming

**Zainab M. Jiwar, Zainab I. Othman**
Department of Computer Information Systems, College of Computer Science and Information Technology, University of Basrah, Basrah, Iraq

**Article Info**

***Article history:***

Received February 26, 2023
Revised March 23, 2023
Accepted March 27, 2023

***Keywords:***

Semantic Web
Ontology
Query engine
Python Web Framework
Flask min-framework

**ABSTRACT**

The Sidr tree is an important fruit tree in many parts of the world, but it is often affected by various diseases and pests that can impact the quality and quantity of its fruit production. This paper proposes the use of ontology and semantic web technologies with Python programing language to program a diagnosis service for the Sidr tree diseases. Programming a Semantic Web application system and services requires knowledge of many technologies such as Resource Description Framework (RDF), Web Ontology Language (OWL), and SPARQL Query. The underline Web service (Sidr tree disease Web service), has to be programmed in connection to a pre-existing ontology and knowledge base. The service program is made of three main components. These components are the user interface, diagnosis service-related functions, Knowledge base, and query engine. The Python 3.10 code listing in this paper represents only the code related to disease diagnosing service and the Sidr tree ontology (SidrTreeonto). In addition to the listed statements and functions, the complete diagnosing service code contains some other unlisted HTML static and dynamic Web pages. The code is run and tested and found to be simple and easy to use even by unexpert users. Overall, this application will provide a useful tool for Sidr tree farmers and researchers to diagnose and manage diseases affecting their trees.

***Corresponding Author:***

Zainab M. Jiwar
Department of Computer Information Systems, College of Computer Science and Information Technology, University of Basrah, Basrah, Iraq
Email: itpg.zainab.jiwar@uobasrah.edu.iq

## 1. INTRODUCTION

The Semantic Web is an extension of the World Wide Web that aims to make data more interoperable, machine-readable, and accessible [1]-[4]. The Semantic Web achieves this by using metadata and ontologies to represent data in a structured and standardized manner. With the Semantic Web, machines can understand and interpret data, which allows for the development of intelligent applications that can automatically process and analyze data.

Programming a Semantic Web application system requires knowledge of various and special technologies such as Resource Description Framework (RDF)[5], Web Ontology Language (OWL)[6],[7], and SPARQL Query Language [8]. RDF is a standard for representing information in the Semantic Web, and it provides a way to describe resources and their relationships. OWL is a language for creating ontologies that define concepts (classes), relationships (properties), and rules in a particular domain [9], [10]. SPARQL Query Language is used to manipulate and retrieve data stored in RDF format.

To develop Semantic Web Applications, developers can use various programming languages such as Python, Java, and JavaScript [11]. Each one of these languages has its own strengths and weaknesses, and the choice of programming language depends on the requirements of the application. Python is a popular language for Semantic Web programming due to its ease of use and availability of libraries for manipulating ontologies [12]. It has several powerful libraries, frameworks, and tools that can be used to build efficient and robust semantic web applications. By leveraging Python's strong typing, object-oriented features, and community support, developers can create scalable and maintainable semantic web applications that integrate machine learning, NLP, and other technologies to enhance functionality and improve data analysis.

Frameworks (Web frameworks or Web application frameworks) stand for a group of libraries and modules that let programmers create web applications without having to be concerned with low-level issues like protocol, thread management, etc.), as Flask can be used to develop Semantic Web applications [13]. Flask is a lightweight and flexible web framework that is ideal for developing small to medium-sized applications.

Plant diseases are a major problem for farmers, leading to crop loss and reduced yields. Early identification and diagnosis of plant diseases are critical to preventing their spread and reducing their impact on crop yields. However, identifying plant diseases can be a difficult and time-consuming task, requiring specialized knowledge and expertise. In this paper, we propose the use of ontology and semantic web technologies with Python to develop a Web diagnosis service that can help farmers diagnose Sidr tree diseases quickly and accurately. The current diagnosis service is in fact only part of a bigger system called the Sidr Tree Disease Diagnosis System (STDDS). The STDDS system is developed using Python programming language, Flask framework, HTML, and CSS. The STDDS system uses an ontology to represent knowledge about plant (Sidr tree) diseases, symptoms, and treatments. The system uses SPARQL Query Language to retrieve and analyze data from the ontology.

The system allows users to input symptoms of a plant disease, and the system will analyze the symptoms and provide a diagnosis along with recommended treatments. The system can also be used to search for information about specific plant diseases, symptoms, and treatments.

Building an ontology-based web application for diagnosing Sidr tree disease has several contributions. One of the main contributions is the potential to improve the accuracy and efficiency of Sidr tree disease diagnosis, which can lead to more effective disease management and higher crop yields [14].

The use of an ontology to represent Sidr tree disease information provides the following features:

- A standardized and structured way to organize and analyze data. This can help to improve the accuracy of disease diagnosis by enabling a more consistent and systematic analysis of disease symptoms and causes.
- The usage of an ontology makes it easier to combine data from many sources., including different languages and domains, which can provide a more comprehensive and holistic view of plant diseases.
- Support decision-making processes in agriculture. By providing a tool for farmers and other stakeholders to access and analyze plant disease data, the application can help them make more informed decisions about disease management strategies, including selecting appropriate control measures, such as pesticide application, and managing plant populations.
- Contribute to the advancement of knowledge in the field of agriculture. By organizing and representing plant disease information in a standardized way, the application can enable researchers to conduct more systematic and comprehensive analyses of plant disease data. This can lead to new insights and discoveries about plant diseases and their causes, as well as more effective disease management strategies.

In summary, building an ontology-based web application for plant disease diagnosis can contribute to improving the accuracy and efficiency of disease diagnosis, supporting decision-making processes in agriculture, and advancing knowledge in the field. As such, it has the potential to provide significant benefits for farmers, researchers, and other stakeholders in the agriculture industry.

## 2. RESEARCH PROBLEM

There are many ways (mathematics, visual processing, and pattern recognition techniques) and many programming languages (Python, PHP, Java, and JavaScript) to code and program any traditional computerized application with widely used data formats. But, for those systems and apps that use a special data format such as RDF and ontology in Web apps, many frameworks, and special libraries mixed with static and dynamic Web pages need to be incorporated in a definite way. Hence, the research problem can be enclosed in how can we program a Sidr tree disease diagnosis system using Python and Semantic Web methods and technologies.

## 3. STDDS SYSTEM COMPONENTS

The STDDS system used in this paper is based on the work in the same field in my master’s degree project. It is made of the following main components:

- User interface.
- Two basic module Knowledge base, and Query engine modules.
- Services modules (diagnosing, searching, monitoring, pathogen, treatment).

System components are shown in the diagram of Figure 1.

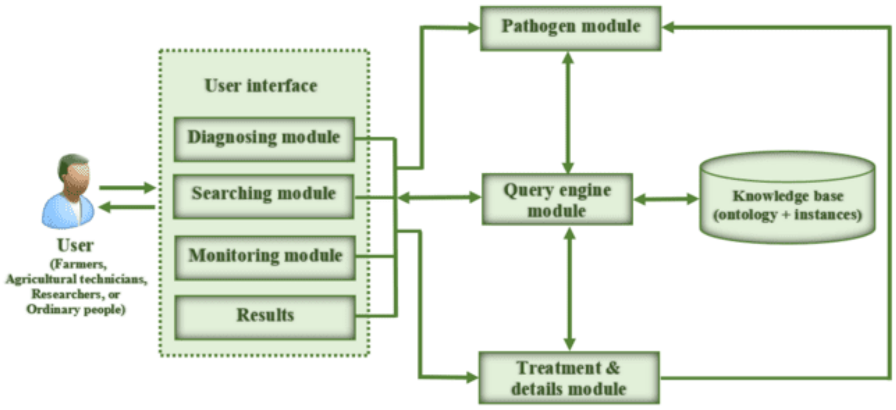
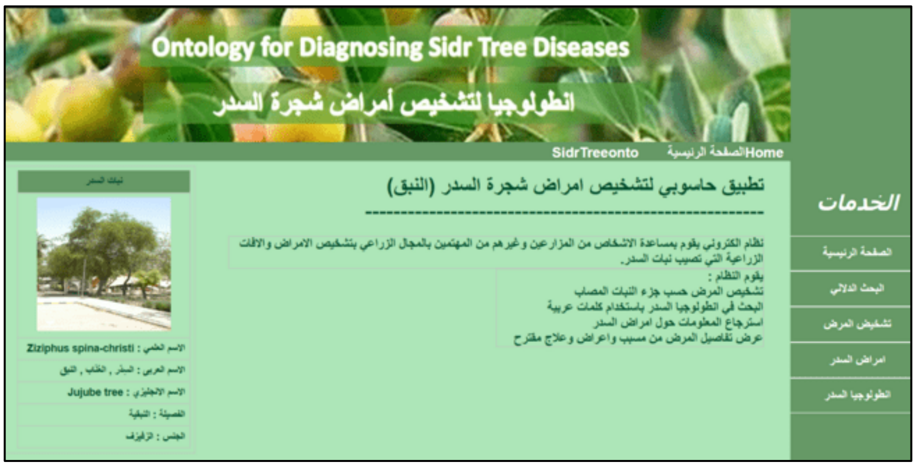

User interface
Diagnosing module
Searching module
Monitoring module
Results
User
(Farmers, Agricultural technicians, Researchers, or Ordinary people)
Pathogen module
Query engine module
Knowledge base (ontology + instances)
Treatment & details module

Figure 1. STDDS system architecture.

### 3.1. User Interface (UI)

The system user interface (UI) is used to make it easier for users to engage in the system. The UI is usually in the form of a dynamic HTML page (index.html) to enable the user to request the desired system service and display the request results. Figure 2 shows the index.html page for the STDDS system UI.



تطبيق حاسوبي لتشخيص امراض شجرة السدر (النبق)

نظام الكتروني يقوم بمساعدة الاشخاص من المزارعين و غيرهم من المهتمين بالمجال الزراعي بتشخيص الامراض والافات الزراعية التي تصيب نبات السدر.

يقوم النظام :
تشخيص المرض حسب جزء النبات المصاب
البحث في انطولوجيا السدر باستخدام كلمات عربية
استرجاع المعلومات حول امراض السدر
عرض تفاصيل المرض من مسبب واعراض وعلاج مقترح

الاسم العلمي : Ziziphus spina-christi
الاسم العربي : السدر , الكنار , النبق
الاسم الانجليزي : Jujube tree
الفصيلة : النبقية
الجنس : الزفيزف

Figure 2. The STDDS system user interface.

### 3.2. Knowledge base and Query engine modules

The knowledge base (ontology) and query engine modules are the heart core modules of any Semantic Web system because all functions and services need to go through and between both modules.

The knowledge base (domain ontology with instances) is the most important component of the Semantic Web [15]. In this paper, the used ontology is called SidrTreeonto-Ontology. It has been taken from our previous research paper [16]. SidrTreeonto ontology represents the knowledge base to provide the information needed to diagnose the Sidr tree diseases system such information as abnormalities, pathogens, tree parts, etc. It was built by Protégé ontology editor [17],[18] by following Noy and McGuinness' methodology [19]. Part of SidrTreeonto ontology is shown in Figure 3. The SidrTreeonto is made of 7 superclasses (*SidrDisease*, *Diagnosis*, *SidrPart*, *Abnormality*, *Pathogens*, *Treatment*, and *Cases*), 26 subclasses (*BacterialDiseases*, *Fungus*, *Root*, …), 13 object properties (*factorOf*, *hasDiagnosis*, *hasTreatment*, …), 6 data properties (*Name*, *scientific_name*, *description*, …), and around 185 instances have been defined of all classes.

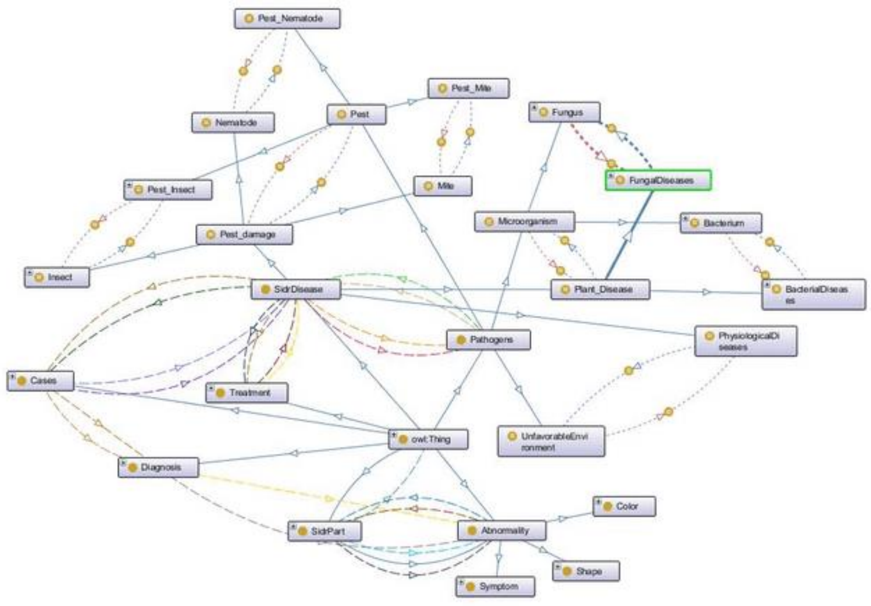


Figure 3. SidrTreeonto ontology.

The query engine is a framework to accepts user input queries and responses to questions through the I/O interface and uses this dynamic information together with the static knowledge stored in the knowledge base [20]. The knowledge in the knowledge base is used to derive conclusions about the current case or situation as presented by the user's input.

### 3.3. Services modules

The STDDS system was built with many services (diagnosing, searching, monitoring, pathogen, treatment), that are needed in accomplishing the intended goals of the system. In this paper, we are going to show in detail how to program one of these services using Python programing language with the aid of Semantic Web technologies [21]. It is the Sidr tree disease diagnosing service.

## 4. DISEASE DIAGNOSIS SERVICE

The most important service provided by the STDDS system is the disease diagnosis service. To run this service, first, the user (the farmer), has to select the infected Sidr tree part, such as leaves, root, stem, etc. from the knowledge base, the system then displays all the abnormalities that may appear on the selected part. Secondly, the system sends the user-selected abnormalities (farmer-observed abnormalities), through the query engine module, which executes special SPARQL queries to the knowledge base to diagnose diseases related to the given abnormalities. The query results may be one or more diseases. The query engine module returns the results to the diagnostic module to display on the user interface in ascending order based on the percentage values computed from the matching process between the number of abnormalities entered and the number of real disease abnormalities stored in the knowledge base. The user selects the disease diagnosis with the highest percentage and the system returns abnormalities, cases, pathogens, treatments, and other details. The flowchart of this service is shown in Figure 4.

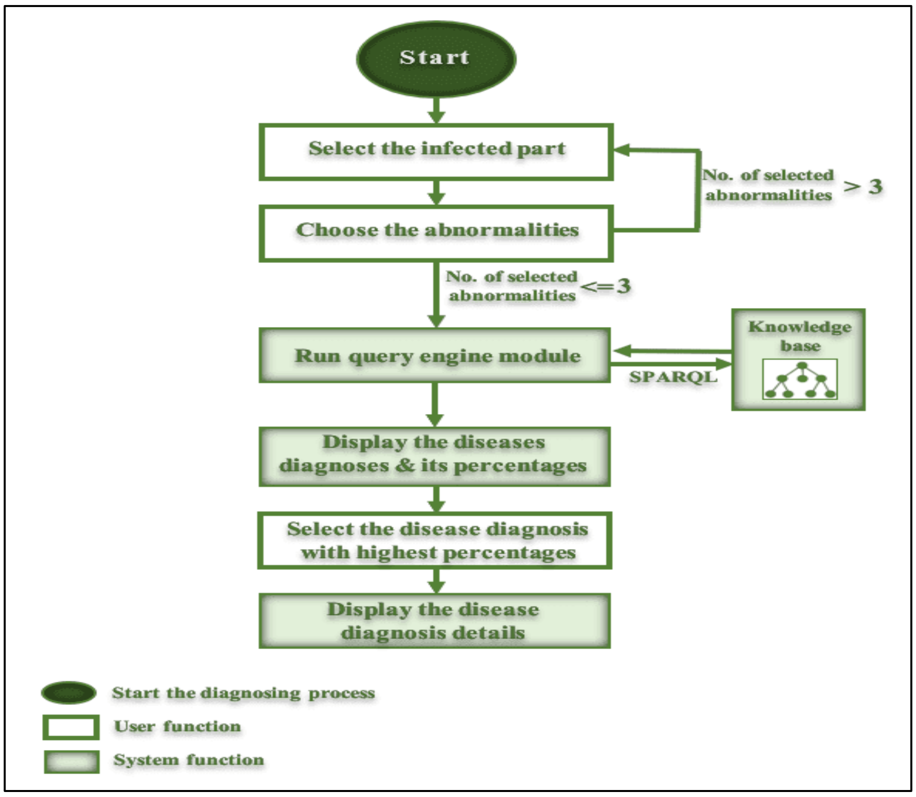
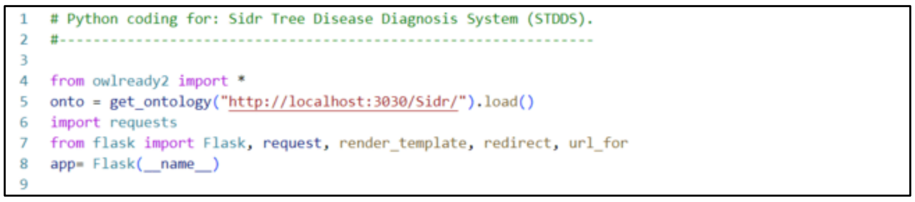

Start
Select the infected part
No. of selected abnormalities > 3
Choose the abnormalities
No. of selected abnormalities <=3
Run query engine module
SPARQL
Knowledge base
Display the diseases diagnoses & its percentages
Select the disease diagnosis with highest percentages
Display the disease diagnosis details
Start the diagnosing process
User function
System function

Figure 4. The diagnosis of disease or pest.

## 4. THE DISEASE DIAGNOSIS SERVICE PROGRAMMING

The following Paragraphs show the complete Python 3.10 [22] code listing of the disease diagnosis service of the STDDS system. To simplify the explanation of the program, the code listing is fragmented into pieces. Each program fragment (or segment) is made of numbered lines.

The first fragment is shown in Listing 1. Statements and the coding lines of the first fragment must be at the top of any Python Semantic Web program that deals with the Owlready2 library [23] and Flask mini framework.



```
1  # Python coding for: Sidr Tree Disease Diagnosis System (STDDS).
2  #-------------------------------------------------------------
3
4  from owlready2 import *
5  onto = get_ontology("http://localhost:3030/Sidr/").load()
6  import requests
7  from flask import Flask, request, render_template, redirect, url_for
8  app= Flask(__name__)
9
```

Listing 1. First STDDS program fragment.

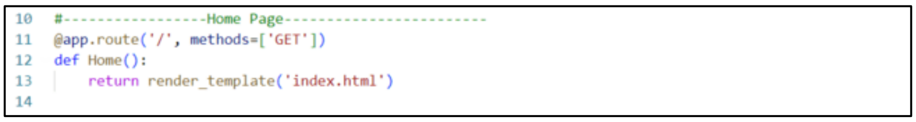
Line (4) is used to show how to import the **Owlready2** library. The import of the contents of the module with "from owlready2 import *" is required and it has to be at the beginning of the program. This is because Owlready2 redefines some Python functions, such as the issubclass () function.

Line (5) represents the statement for loading the pre-created domain ontology. Note that the domain ontology (SidrTreeonto) is created by the **protégé** editor and stored as a triple store on the "http://localhost:3030/" under the directory "Sidr/".

Line (6) is to import the **requests** library [24]. The requests library is the de facto standard for making HTTP requests in Python. It abstracts the complexities of making requests behind a beautiful, simple API so that you can focus on interacting with services and consuming data in your application.

Lines (7-8): both lines are used only once at the beginning of the main program code. They show the way of calling **Flask** mini framework. Flask is a Python module used to build web apps, it allows the connection of the URL path of the web application page with a Python function; when this path is requested, the function is run, and it returns the correct HTML page. On the line before the method, add **@app.route('/path')** to define the path (it is a Python function decorator). The parameters that are supplied as arguments to the Python method, can be found in the pathways and are denoted by the angle brackets <…> as will be shown in the following program fragments.

The second STDDS program fragment is shown in Listing 2. It represents the general way of calling any HTML page by using Flask. In this segment, the called page is the home page ‘index.html’.

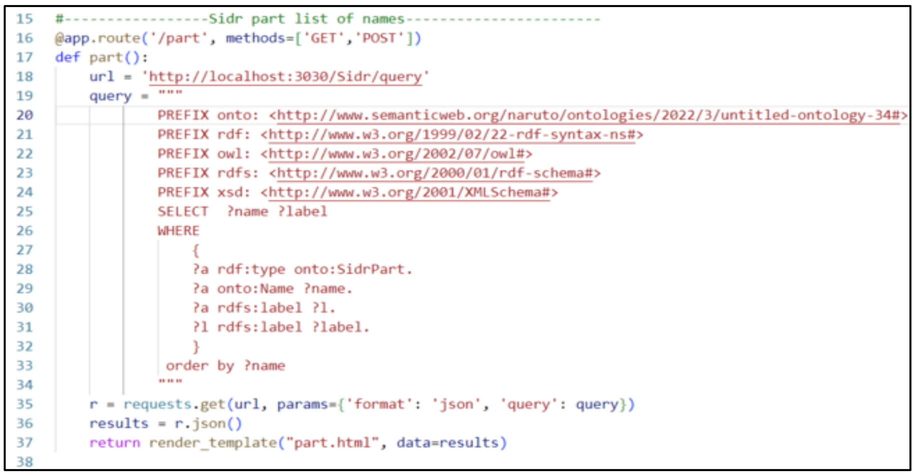


```
10  #----------------Home Page-----------------------
11  @app.route('/', methods=['GET'])
12  def Home():
13      return render_template('index.html')
14
```

Listing 2. Second STDDS program fragment.

Lines (11-13) show the way that Flask calls any HTML page ‘index.html’. it defines a function Home () for calling the ‘index.html’ that represents the home page of the system.

The third STDDS program segment is shown in Listing 3. The output of this segment is a popup list of all Sidr tree parts which are stored in the knowledge base (SidrTreeonto ontology).



```
15  #----------------Sidr part list of names----------------------
16  @app.route('/part', methods=['GET','POST'])
17  def part():
18      url = 'http://localhost:3030/Sidr/query'
19      query = """
20              PREFIX onto: <http://www.semanticweb.org/naruto/ontologies/2022/3/untitled-ontology-34#>
21              PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
22              PREFIX owl: <http://www.w3.org/2002/07/owl#>
23              PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
24              PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
25              SELECT  ?name ?label
26              WHERE
27                  {
28                  ?a rdf:type onto:SidrPart.
29                  ?a onto:Name ?name.
30                  ?a rdfs:label ?l.
31                  ?l rdfs:label ?label.
32                  }
33               order by ?name
34              """
35      r = requests.get(url, params={'format': 'json', 'query': query})
36      results = r.json()
37      return render_template("part.html", data=results)
38
```

Listing 3. Third STDDS program fragment.

Its lines are as follows:

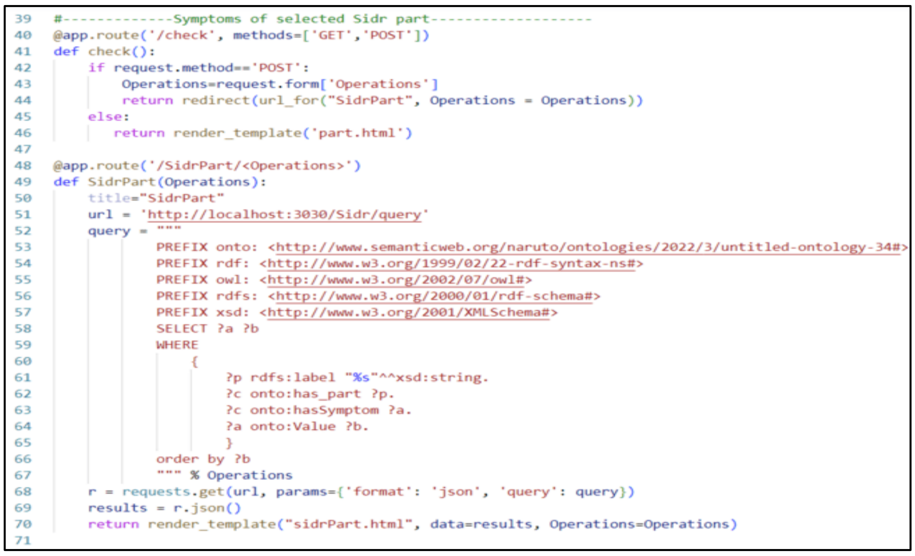
Line (16) represent the Python standard way of defining a function. It is the part () function.

Lines (17-34) represent the action needed to be performed by the part () function. It is in a form of SPARQL query to get the names of the Sidr tree parts that are specified and stored in the knowledge base.

Lines (35-36) are used to assign the function returned values to the variable (r) and then convert the results into a JSON format [25] as a variable (results).

Line (37) is used to send the function’s return value (results) to be displayed on a new HTML page called part.html. and let the user select one tree part-name from the displayed popup list.

The fourth STDDS program segment is shown in Listing 4. In the ontology, one or more abnormalities are assigned to each tree part. This segment is used to display the assigned abnormalities (symptoms) to the selected tree part obtained by the third program segment as in above.



```
39  #------------Symptoms of selected Sidr part------------------
40  @app.route('/check', methods=['GET','POST'])
41  def check():
42      if request.method=='POST':
43          Operations=request.form['Operations']
44          return redirect(url_for("SidrPart", Operations = Operations))
45      else:
46          return render_template('part.html')
47
48  @app.route('/SidrPart/<Operations>')
49  def SidrPart(Operations):
50      title="SidrPart"
51      url = 'http://localhost:3030/Sidr/query'
52      query = """
53                PREFIX onto: <http://www.semanticweb.org/naruto/ontologies/2022/3/untitled-ontology-34#>
54                PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
55                PREFIX owl: <http://www.w3.org/2002/07/owl#>
56                PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
57                PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
58                SELECT ?a ?b
59                WHERE
60                    {
61                        ?p rdfs:label "%s"^^xsd:string.
62                        ?c onto:has_part ?p.
63                        ?c onto:hasSymptom ?a.
64                        ?a onto:Value ?b.
65                        }
66                order by ?b
67                """ % Operations
68      r = requests.get(url, params={'format': 'json', 'query': query})
69      results = r.json()
70      return render_template("sidrPart.html", data=results, Operations=Operations)
71
```

Listing 4. Fourth STDDS program fragment.

The fourth program segment is called from the part.html page. Two functions are defined in it as follows:

Lines (40-46) are used to create the first function check () on the path to '/check'. If the user selects any tree parts, then the function lets the next query be performed else it will return the response back again to the part.html page.

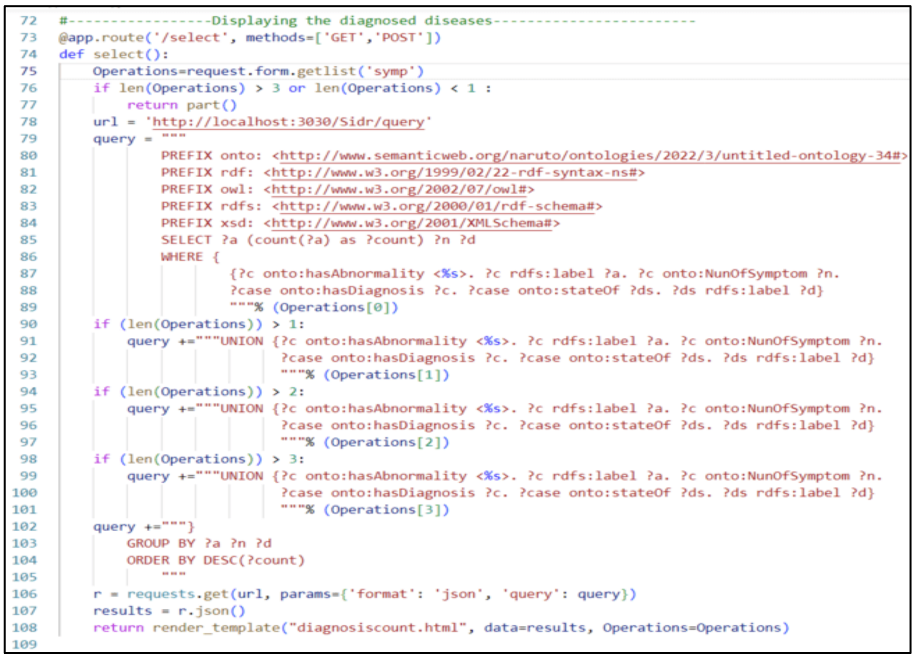
Lines (48-70) represent the second function SidrPart () on the path '/SidrPart'. The query results are assigned to the variable (r) and then convert the query output into a JSON format in the variable (results). Then the function returned values (results) are displayed on a new HTML page called SidrPart.html which let the user check at most three abnormalities from the displayed popup list.

The fifth STDDS program segment is shown in Listing 5. This segment is used to display a list of one or more disease diagnoses and their matching percentages between the selected abnormalities (from the fourth program fragment) and the abnormalities assigned to that disease in the ontology. The fifth program segment is called from the SidrPart.html page.

Its program lines are as follows:

Lines (73-77) are used to create the function select () on the path to '/select'. If the user selects at most three abnormalities, then the function lets the next query be performed else it will return the response back again to part.html.

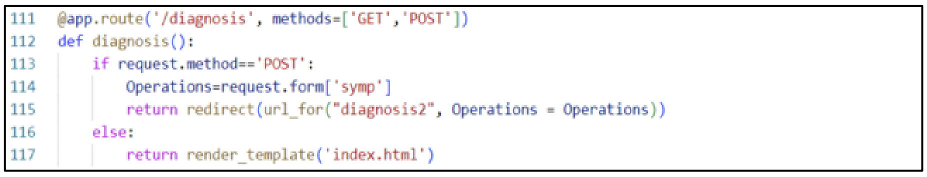
Lines (78-108) represent the query that returns the disease diagnosis if there is a matching between selected abnormalities and the abnormalities assigned to those diseases in the ontology. The query results are assigned to the variable (r) and then convert into a JSON format as a variable (results). After that, the function returned values (results) are displayed on a new HTML page called diagnosiscount.html. And let the user check disease diagnosis from the displayed popup list (it is better to check the disease with the highest matching percentage).



```
72  #-----------------Displaying the diagnosed diseases------------------------
73  @app.route('/select', methods=['GET','POST'])
74  def select():
75      Operations=request.form.getlist('symp')
76      if len(Operations) > 3 or len(Operations) < 1 :
77          return part()
78      url = 'http://localhost:3030/Sidr/query'
79      query = """
80              PREFIX onto: <http://www.semanticweb.org/naruto/ontologies/2022/3/untitled-ontology-34#>
81              PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
82              PREFIX owl: <http://www.w3.org/2002/07/owl#>
83              PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
84              PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
85              SELECT ?a (count(?a) as ?count) ?n ?d
86              WHERE {
87                      {?c onto:hasAbnormality <%s>. ?c rdfs:label ?a. ?c onto:NunOfSymptom ?n.
88                      ?case onto:hasDiagnosis ?c. ?case onto:stateOf ?ds. ?ds rdfs:label ?d}
89                      """% (Operations[0])
90      if (len(Operations)) > 1:
91          query +="""UNION {?c onto:hasAbnormality <%s>. ?c rdfs:label ?a. ?c onto:NunOfSymptom ?n.
92                            ?case onto:hasDiagnosis ?c. ?case onto:stateOf ?ds. ?ds rdfs:label ?d}
93                            """% (Operations[1])
94      if (len(Operations)) > 2:
95          query +="""UNION {?c onto:hasAbnormality <%s>. ?c rdfs:label ?a. ?c onto:NunOfSymptom ?n.
96                            ?case onto:hasDiagnosis ?c. ?case onto:stateOf ?ds. ?ds rdfs:label ?d}
97                            """% (Operations[2])
98      if (len(Operations)) > 3:
99          query +="""UNION {?c onto:hasAbnormality <%s>. ?c rdfs:label ?a. ?c onto:NunOfSymptom ?n.
100                           ?case onto:hasDiagnosis ?c. ?case onto:stateOf ?ds. ?ds rdfs:label ?d}
101                           """% (Operations[3])
102     query +="""}
103         GROUP BY ?a ?n ?d
104         ORDER BY DESC(?count)
105             """
106     r = requests.get(url, params={'format': 'json', 'query': query})
107     results = r.json()
108     return render_template("diagnosiscount.html", data=results, Operations=Operations)
109
```

Listing 5. Fifth STDDS program fragment.

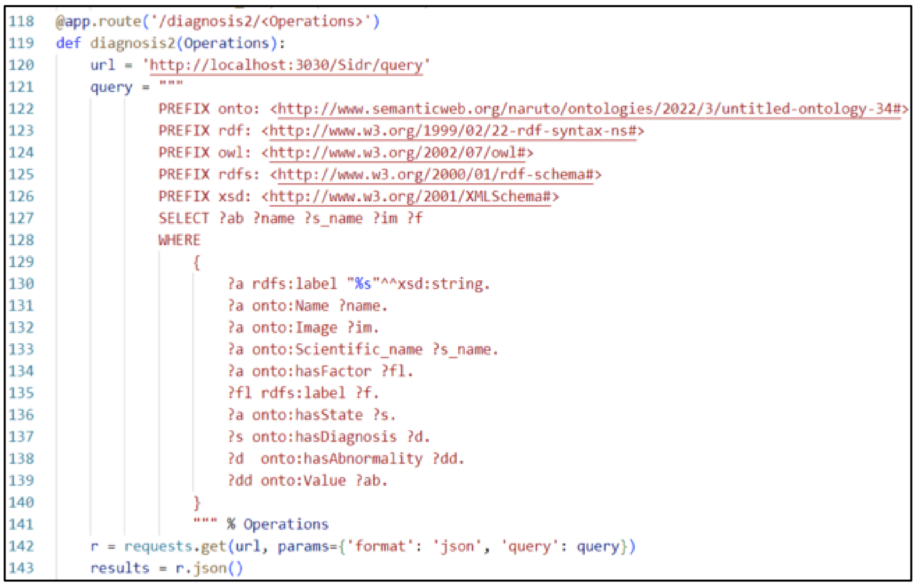
The sixth STDDS program fragment is shown in Listing 6. The function of this segment (lines (111-117)) is to make sure that the user selects a disease from the displayed list or not. And if not then return the control to the home page else run the next fragment (the seventh fragment).



```
111 @app.route('/diagnosis', methods=['GET','POST'])
112 def diagnosis():
113     if request.method=='POST':
114         Operations=request.form['symp']
115         return redirect(url_for("diagnosis2", Operations = Operations))
116     else:
117         return render_template('index.html')
```

Listing 6. Sixth STDDS program fragment.

The seventh fragment (lines (118-143)), as shown in Listing 7 is used to display disease details (information and its abnormalities).



```
118 @app.route('/diagnosis2/<Operations>')
119 def diagnosis2(Operations):
120     url = 'http://localhost:3030/Sidr/query'
121     query = """
122             PREFIX onto: <http://www.semanticweb.org/naruto/ontologies/2022/3/untitled-ontology-34#>
123             PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
124             PREFIX owl: <http://www.w3.org/2002/07/owl#>
125             PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
126             PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
127             SELECT ?ab ?name ?s_name ?im ?f
128             WHERE
129                 {
130                     ?a rdfs:label "%s"^^xsd:string.
131                     ?a onto:Name ?name.
132                     ?a onto:Image ?im.
133                     ?a onto:Scientific_name ?s_name.
134                     ?a onto:hasFactor ?fl.
135                     ?fl rdfs:label ?f.
136                     ?a onto:hasState ?s.
137                     ?s onto:hasDiagnosis ?d.
138                     ?d  onto:hasAbnormality ?dd.
139                     ?dd onto:Value ?ab.
140                 }
141                 """ % Operations
142     r = requests.get(url, params={'format': 'json', 'query': query})
143     results = r.json()
```

Listing 7. Seventh STDDS program fragment.

The eighth fragment (lines (144-161)), as shown in Listing 8 is used to display disease treatment.

```
    url2 = 'http://localhost:3030/Sidr/query'
    query2 = """
            PREFIX onto: <http://www.semanticweb.org/naruto/ontologies/2022/3/untitled-ontology-34#>
            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#>
            SELECT ?s
            WHERE
                {
                    ?a rdfs:label "%s"^^xsd:string.
                    ?a onto:hasTreatment ?c.
                    ?c onto:Description ?s.
                }
                """ % Operations
    r2 = requests.get(url2, params={'format': 'json', 'query': query2})
    results2 = r2.json()
    return render_template("diseaseAT.html", data=results, data2=results2, Operations=Operations)
```

Listing 8. Eighth STDDS program fragment.

The ninth fragment (lines (393-394)), as shown in Listing 9, is used as a server startup point.

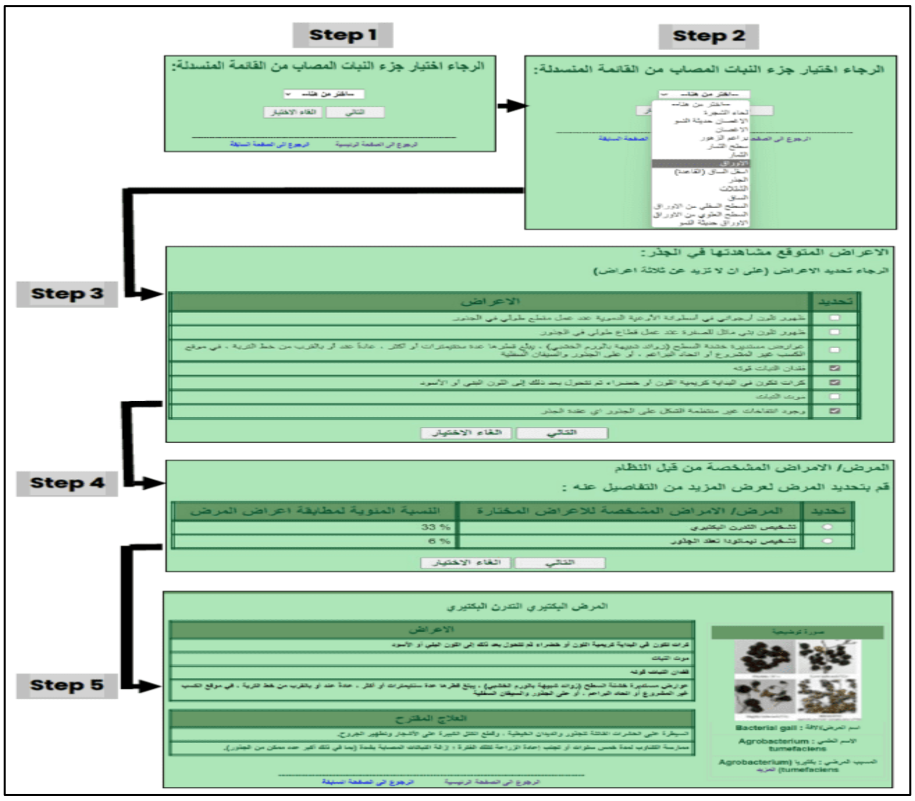
```
392 #----------------Running flask app-----------------------
393 if __name__=='__main__':
394     app.run(debug=True)
```

Listing 9. Ninth STDDS program fragment.

The application instance has a run method that launches Flask's integrated development web server: Line (393) represents a Python condition. It is used here to ensure that the development web server is started only when the script is executed directly. And when the script is imported by another script, it is assumed that the parent script will launch a different server, so the Line (394) call is skipped.

Note that this fragment always should be at the end of the application code. Once the server starts up, it goes into a loop that waits for requests and services them. This loop continues until the application is stopped, for example by hitting Ctrl-C.

Figure 5 visually depicts the steps for an instance of running the diagnosis service of all pseudocodes in the above coding lists.


Step 1
الرجاء اختيار جزء النبات المصاب من القائمة المنسدلة:
التالي
الغاء الاختيار
الرجوع الى الصفحة الرئيسية
الرجوع الى الصفحة السابقة
Step 2
الرجاء اختيار جزء النبات المصاب من القائمة المنسدلة:
الاوراق
الجذر
الساق
Step 3
Step 4
Step 5

الاعراض المتوقع مشاهدتها في الجذر:

الرجاء تحديد الاعراض (على ان لا تزيد عن ثلاثة اعراض)

| تحديد | الاعراض |
|---|---|
| ☐ | ظهور اللون ارجواني في اسطوانة الأوعية الدموية عند عمل مقطع طولي في الجذور |
| ☐ | ظهور اللون بني مائل للصفرة عند عمل قطاع طولي في الجذور |
| ☐ | [illegible] |
| ☑ | فقدان النبات قوته |
| ☑ | كرات تكون في البداية كريمية اللون أو خضراء ثم تتحول بعد ذلك إلى اللون البني أو الأسود |
| ☐ | موت النبات |
| ☑ | وجود انتفاخات غير منتظمة الشكل على الجذور اي عقد الجذر |

المرض/ الامراض المشخصة من قبل النظام

قم بتحديد المرض لعرض المزيد من التفاصيل عنه :

| تحديد | المرض/ الامراض المشخصة للاعراض المختارة | النسبة المئوية لمطابقة اعراض المرض |
|---|---|---|
| ○ | تشخيص التدرن البكتيري | 33 % |
| ○ | تشخيص نيماتودا تعقد الجذور | 6 % |

المرض البكتيري التدرن البكتيري

| الاعراض |
|---|
| كرات تكون في البداية كريمية اللون أو خضراء ثم تتحول بعد ذلك إلى اللون البني أو الأسود |
| موت النبات |
| فقدان النبات قوته |
| [illegible] |

| العلاج المقترح |
|---|
| [illegible] |
| [illegible] |

Bacterial gall
Agrobacterium tumefaciens

Figure 5. The diagnosis service running steps.

## 4. CONCLUSION

Coding any ontology-based Web application requires sufficient familiarity with the Semantic Web technologies and frameworks related to the Semantic Web programming method. Web apps usually utilize a pre-existing ontology and the programmer can not start the system programming or application coding without it. And because Python programing language contains many statements, functions, and Semantic Web-related libraries, Python is implemented in the work coding.

The code listing shown in this paper is only one service of the many in the original STDDS. It is the Sidr tree disease diagnosis service. Only the main code from the main diagnosing service Python program is shown and not including the mentioned HTML pages since it is easy to code such of these pages.

The code above is tested and evaluated using hypothetical Sidr tree disease and the resultant evaluation was excellent. The system is simple and easy to use by any user.

## REFERENCES

[1] P. Hitzler, “A review of the semantic web field”. Communications of the ACM, vol. 64, pp.76-83, 2021.
[2] M. M. Taye, “Understanding semantic web and ontologies: Theory and applications,” J Comput, vol. 2, no. 6, 2010.
[3] R. Y. Alsalhee and A. M. Abdullah, “Building Quranic stories ontology using MappingMaster domain-specific language,” International Journal of Electrical and Computer Engineering, vol. 12, no. 1, p. 684, 2022.
[4] A. M. Abdullah and M. Raisan, “Building a core Arabic ontology about iraqi news,” International Journal of Engineering and Advanced Technology (IJEAT), vol. 4, no. 6, pp. 171–177, 2015.
[5] G. Antoniou and F. van Harmelen, “A semantic web primer”. MIT press, 2004.
[6] S. Bechhofer et al., “OWL web ontology language reference,” W3C recommendation, vol. 10, no. 2, pp. 1–53, 2004.
[7] T. Segaran, C. Evans, and J. Taylor, “Programming the semantic web: build flexible applications with graph data”. O’Reilly Media, Inc., 2009.
[8] J. Pérez, M. Arenas, and C. Gutierrez, “Semantics and Complexity of SPARQL,” in International semantic web conference, pp. 30–43, 2006.
[9] L. Liu and M. T. Özsu, “Encyclopedia of database systems”, vol. 6. Springer, 2009.
[10] K. Munir and M. S. Anjum, “The use of ontologies for effective knowledge modelling and information retrieval,” Applied Computing and Informatics, vol. 14, no. 2, pp. 116–126, 2018.
[11] John Hebeler, Matthew Fisher, Ryan Blace, and Andrew Perez-Lopez, “Semantic Web Programming”, Published by Wiley Publishing, Inc., Indianapolis, Indiana, ISBN: 978-0-470-41801-7, 2009.
[12] Lamy Jean-Baptiste , “Ontologies with Python”, Université Sorbonne Paris Nord, LIMICS, Sorbonne Université, ISBN-13 (pbk): 978-1-4842-6551-2, 2021.
[13] M. Grinberg, “Flask web development: developing web applications with python”. “ O’Reilly Media, Inc.,” 2018.
[14] K. Lagos-Ortiz, J. Medina-Moreira, M. A. Paredes-Valverde, W. Espinoza-Morán, and R. Valencia-García, “An ontology-based decision support system for the diagnosis of plant diseases,” Journal of Information Technology Research (JITR), vol. 10, no. 4, pp. 42–55, 2017.
[15] Kuo, K.-L.; Fuh, C.-S. “A rule-based clinical decision model to support the interpretation of multiple data in health examinations”. J. Med. Syst., 35, 1359–1373, 2011.
[16] Zainab M. Jiwar, Zainab I. Othman, “Building an ontology for diagnosing Sidr tree diseases”, Journal of Al-Qadisiyah for Computer Science and Mathematics Vol. 15(1), pp Comp. 68–78, 2023.
[17] Knublauch, H.; Fergerson, R.W.; Noy, N.F.; Musen, M.A. “The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications”. Springer: Berlin/Heidelberg, Germany; pp. 229–243, 2004.
[18] “protégé.” https://protege.stanford.edu/ (accessed Jan. 03, 2023).
[19] N. F. Noy and D. L. Mcguinness, “Ontology Development 101: A Guide to Creating Your First Ontology,” Mar, [Online]. Available: www.unspsc.org, 2001.
[20] Mahmoud A. El-Askary, “An Ontology-Based Approach for Diagnosing Date Palm Diseases”, thesis Submitted to Islamic University of Gaza, 2015.
[21] Gerald Aistleitner, “Design and implementation of a prototypical decision support system for communication”, PhD Thesis, Universität Linz, 2020.
[22] “python 3.10”, https://www.python.org/downloads/release/python-3100/ (accessed Jan. 03, 2023).
[23] Jean-Baptiste Lamy. “Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies”. In: Artificial Intelligence in Medicine 80, DOI: 10.1016/j.artmed, 2017.
[24] Kenneth REITZ, “Requests Documentation Release 2.28.2”, [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/requests/latest/requests.pdf, 2020.
[25] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč, “Foundations of JSON schema”, In Proceedings of the 25th international conference on World Wide Web (pp. 263-273), 2016.