

Faculty of Computer Science

Prof. Alexander Adam, Daniel Richter

Datenbanken und Web Techniken

Project Report (SS 2018)

**Restaurant Menu Fetcher!!!**

Chemnitz, 01. July 2018

|  |  |
| --- | --- |
| **Team members** | **Matriculation no.** |
| Anubhav Gupta (Master in Web Engineering) | 486044 |
| Paurush Vishnoi (Master in Web Engineering) | 486789 |

Contents

[1. Introduction 3](#_Toc518249895)

[2. Problem Statement: 4](#_Toc518249896)

[2.1 Frontend: 4](#_Toc518249897)

[2.2 Backend: 4](#_Toc518249898)

[3. Technologies Used: 5](#_Toc518249899)

[3.1 SQLite: 5](#_Toc518249900)

[3.1.1 Main Features 5](#_Toc518249901)

[3.1.2 Why SQLite? 5](#_Toc518249902)

[4. WebCrawler: 6](#_Toc518249903)

[4.1 Overview 6](#_Toc518249904)

[4.2 Crawling Policy 6](#_Toc518249905)

[5. Python: 7](#_Toc518249906)

[5.1 Indentation 7](#_Toc518249907)

[5.2 Libraries 7](#_Toc518249908)

[6. REST API: 8](#_Toc518249909)

[7. HTML - JavaScript: 9](#_Toc518249910)

[6.1 Why JS? 9](#_Toc518249911)

[8. ER Representation: 10](#_Toc518249912)

[9. Use Case Scenario 11](#_Toc518249913)

[10. UML Representation: 12](#_Toc518249914)

[11. Practical Demonstration 13](#_Toc518249915)

[11.1 .Python File Details: 14](#_Toc518249916)

[11.2 .HTML File Details: 18](#_Toc518249917)

[12. Conclusion 20](#_Toc518249918)

[13. APPENDIX 21](#_Toc518249919)

[13.1 Show All Dishes 21](#_Toc518249920)

[13.2 Show Dishes Based on Search 22](#_Toc518249921)

# 1. Introduction

Many restaurants offer their menu in the internet. These menus are mainly available as normal Web Pages. Aggregators, such as e. g. pizza.de or dominos.de,

to name some of them, require the participating restaurants to provide their menu in a machine executable format. So the restaurants must provide their data, additional to their own website.

We have developed a web application in Python Language, that scraps different publically viewable menus and presents them in a common user interface. The only advantage of Web crawler is a program that acts as an automated script which browses through the internet in a systematic way. The web crawler looks at the tags for the elements in the pages, the kind of content each page has and the links, before returning the information to the search engine.

The advantage for the user, amongst other things, to not have to search through all the menus for a special dish. User is presented with a menu aggregated of all the single menus of the respective restaurants and also will see which restaurants has the most inexpensive price. The exact task description follows in the next section.

# 2. Problem Statement:

Our Task is to Develop a web application, that fulfills the following functionalities described as below:

## 2.1 Frontend:

1.At least of three real restaurants (or other dishes providers), each restaurant must contain at least 7 dishes, the menu should be available.

2.All available dishes that are in the database can be listed with the search button used in UI.

3.There should be detail information for dishes:

a. Ingredients per restaurant

b. price per restaurant

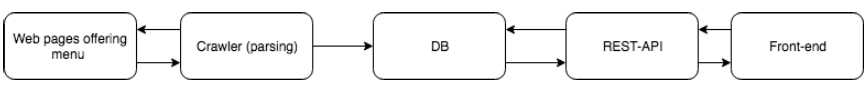
4.All the restaurants should have necessary information like address, contact number and opening hours.

## 2.2 Backend:

1. The processing part Crawler (parsing) to fetch data and save it in your chosen database.

2. REST-API to communicate with your database for the necessary data and provides them to the client. For this project there is no need to create a proactive service, that always queries the menu data.

Desired project structure is also shown below:



# 3. Technologies Used:

## 3.1 SQLite:

A Database is a collection of data, which is organized to be easily accessed, managed and updated. It is relational database management system contained in C programming library.  In contrast to many other database management systems, SQLite is not a [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server) database engine. Rather, it is embedded into the end program.

### 3.1.1 Main Features

SQLite is [ACID](https://en.wikipedia.org/wiki/ACID)-compliant and implements most of the [SQL](https://en.wikipedia.org/wiki/SQL) standard, using a dynamically and weakly [typed](https://en.wikipedia.org/wiki/Data_type) SQL [syntax](https://en.wikipedia.org/wiki/Syntax) that does not guarantee the [domain integrity](https://en.wikipedia.org/wiki/Domain_integrity).

SQLite is a popular choice as [embedded database](https://en.wikipedia.org/wiki/Embedded_database) software for local/client storage in [application software](https://en.wikipedia.org/wiki/Application_software) such as [web browsers](https://en.wikipedia.org/wiki/Web_browser). It is arguably the most widely deployed [database engine](https://en.wikipedia.org/wiki/Database_engine), as it is used today by several widespread browsers, [operating systems](https://en.wikipedia.org/wiki/Operating_system), and [embedded systems](https://en.wikipedia.org/wiki/Embedded_system) (such as mobile phones), among others. SQLite has [bindings](https://en.wikipedia.org/wiki/Language_binding) to many programming languages.

### 3.1.2 Why SQLite?

SQLite uses dynamic types for tables. It means you can store any value in any column, regardless of the data type.

SQLite allows a single database connection to access multiple database files simultaneously. This brings many nice features like joining tables in different databases or copying data between databases in a single command.

SQLite is capable of creating in-memory databases which are very fast to work with.

# 4. WebCrawler:

A Web crawler, sometimes called a spider, is an [Internet bot](https://en.wikipedia.org/wiki/Internet_bot) that systematically browses the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), typically for the purpose of [Web indexing](https://en.wikipedia.org/wiki/Web_indexing) (web spidering).

[Web search engines](https://en.wikipedia.org/wiki/Web_search_engine) and some other sites use Web crawling or spidering software to update their [web content](https://en.wikipedia.org/wiki/Web_content) or indices of others sites' web content. Web crawlers copy pages for processing by a search engine which [indexes](https://en.wikipedia.org/wiki/Index_(search_engine)) the downloaded pages so users can search more efficiently.

Crawlers can validate [hyperlinks](https://en.wikipedia.org/wiki/Hyperlink) and [HTML](https://en.wikipedia.org/wiki/HTML) code. They can also be used for [web scraping](https://en.wikipedia.org/wiki/Web_scraping).

## 4.1 Overview

A Web crawler starts with a list of [URLs](https://en.wikipedia.org/wiki/Uniform_Resource_Locator) to visit, called the seeds. As the crawler visits these URLs, it identifies all the [hyperlinks](https://en.wikipedia.org/wiki/Hyperlink) in the page and adds them to the list of URLs to visit, called the [crawl frontier](https://en.wikipedia.org/wiki/Crawl_frontier). URLs from the frontier are [recursively](https://en.wikipedia.org/wiki/Recursion) visited according to a set of policies. If the crawler is performs or archive [websites](https://en.wikipedia.org/wiki/Website) it copies and saves the information as it goes. The archives are usually stored in such a way they can be viewed, read and navigated as they were on the live web, but are preserved as ‘snapshots'.

## 4.2 Crawling Policy

The behavior of a Web crawler is the outcome of a combination of policies:

* Selection policy: Which states the pages to download.
* Re-visit policy : Which states when to check for changes to the pages.
* Politeness policy : That states how to avoid overloading [Web sites](https://en.wikipedia.org/wiki/Web_sites).
* Parallelization policy: That states how to coordinate distributed web crawlers.

# 5. Python:

Python is a [high-level interpreted programming language](https://en.wikipedia.org/wiki/High-level_programming_language) for [general-purpose programming](https://en.wikipedia.org/wiki/General-purpose_programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python has a design philosophy that emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability), notably using [significant whitespace](https://en.wikipedia.org/wiki/Significant_whitespace). It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

## 5.1 Indentation

Python uses [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation, rather than [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) or keywords, to delimit [blocks](https://en.wikipedia.org/wiki/Block_(programming)). An increase in indentation comes after certain statements, a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed as [off-side rule](https://en.wikipedia.org/wiki/Off-side_rule).

## 5.2 Libraries

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks.

The language's core philosophy is summarized as below:

* Beautiful is better than ugly
* Explicit is better than implicit
* Simple is better than complex
* Complex is better than complicated
* Readability counts

# 6. REST API:

Representational State Transfer (REST) is an architectural style that defines a set of constraints and properties based on [HTTP](https://en.wikipedia.org/wiki/HTTP). Web Services that conform to the REST architectural style, or RESTful [web services](https://en.wikipedia.org/wiki/Web_service), provide interoperability between computer systems on the [Internet](https://en.wikipedia.org/wiki/Internet).

The concept of REST API was defined in 2000 by [Roy Fielding](https://en.wikipedia.org/wiki/Roy_Fielding) in his doctoral dissertation

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. The service used in this project is GET

The GET method means retrieve whatever information ([...]) is identified by the Request-URI. Here is an example of using GET method.

**Request:**

Specify the GET method and forward slash (/):

GET /

**Data:**

None.

**Response:**

Here is a successful response:

[ "name" , "ESPRESSO" ]

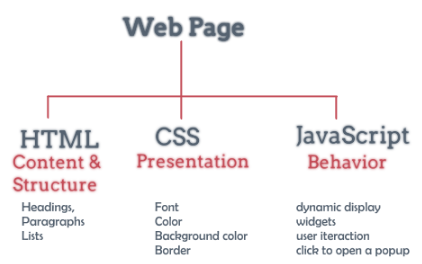
# 7. HTML - JavaScript:

JavaScript often abbreviated as JS, is a high-level, interpreted programming language. It is a language which is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm.

Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web. JavaScript enables interactive web pages and thus is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it.

## 6.1 Why JS?

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. It has an API for working with text, arrays, dates, regular expressions, and basic manipulation of the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.



# 8. ER Representation:

An ER model is a pictorial representation of any kind of business process or a relation between different entities.

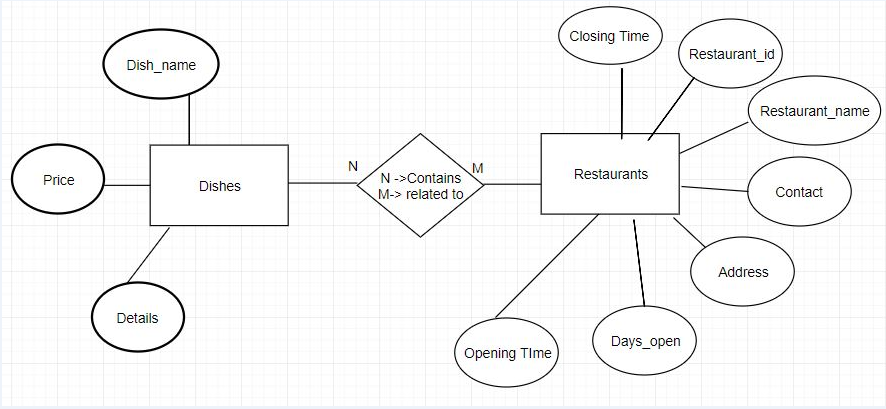
Entities might be described by connections, as well as by extra properties (traits), which incorporate identifiers called "primary keys". Graphs made to speak to traits and additionally elements and connections might be called entity characteristic relationship charts, instead of entity– relationship models.

**Restaurants and Dishes**

Selection of the three random restaurants among many is done and then the dishes present in each restaurant are defined according to their characteristics.

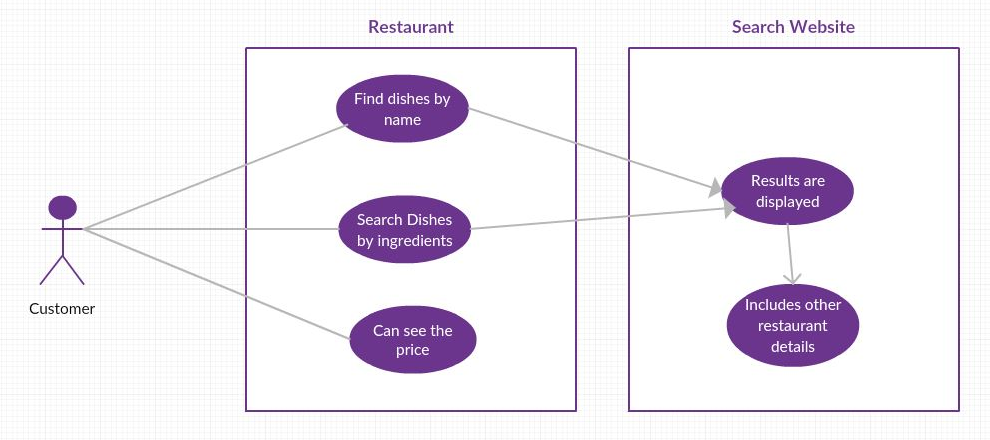
**Entity1** – Restaurant **Entity 2** – Dishes

**Relationship** – Each restaurant can have one or many dishes.



# 9. Use Case Scenario:

Use case model is sub representation of UML diagram. It is used to conglomerate different entities and represent them in the form a flowchart. The process flow generally represents a user , the system and the processes under them and bestow the relationship between them.



* A restaurant contains details of all the dishes along with their price and ingredient details.
* The details are then stored in a database table which can further be used to display on the front-end.
* A customer can search the details of particular dish as well as see the complete details of all the dishes present in all restaurants.

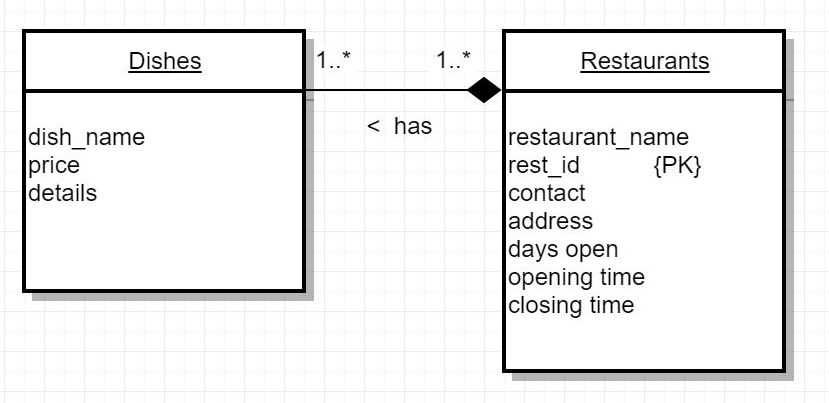
Entity - Guest

System - Restaurant

Processes - Searching the dish name, Searching the ingredients, See results at front-end

# 10. UML Representation:

The basic module of object oriented model is UML (Unified Modeling language), also known as Class diagram, represents main classes their relationships, their modularity as well as the dependency of the child over parent class.



The relational schemas for the Restaurant database are:

**Restaurants (rest\_id, restaurant\_name, contact, address, days open, opening time, closing time)**

**Dishes (dish\_name, price, details)**

# 11. Practical Demonstration

In the Interface which is "Restaurant dish details Fetcher" which displays the particular dish with its restaurant details based on any keyword based on the dish name or its ingredients. There are two search button implemented on our website. The first button takes the input from the search button and then based on that particular string it fetches the details directly from the database while the second button once clicked prints all the dish details from all the restaurants on our webpage.



Fig: Displaying the menu

## 11.1 .Python File Details:

**File 1 -> DemoHotel.py**

**Description:** This file uses python inbuilt library Beautiful Soup to crawl the website and then scrap and save the data to the particular variable locally. We are scraping three restaurant URLs as below:

url1 = 'http://cafeturtle.com/menu/'

url2 = 'http://theburgerhouse.com/food/'

url3 = 'http://www.thebaluchi.com/chandigarh/vegeterian-menu.php'

Also it creates the Table name " RESTAURANT\_DATA " on our Database i.e. tutorial.db file on SQLite.

In particular instant of time, the data will be scrapped again from the restaurants reason being the update of any dish in the restaurant menu. This functionality is working with "scheduler" library.

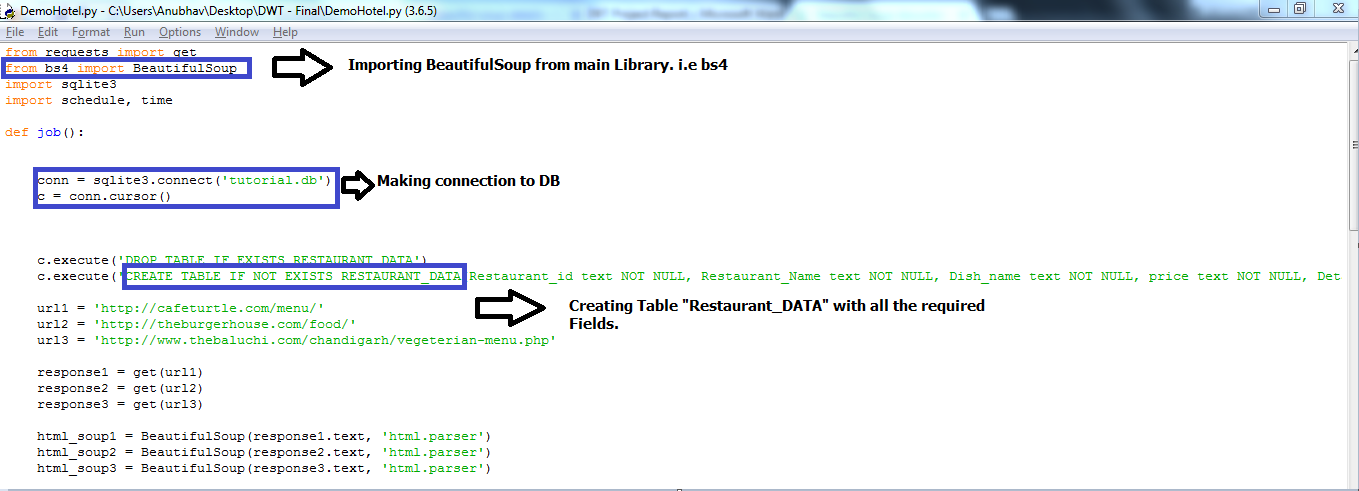
**Beautiful Soup:**

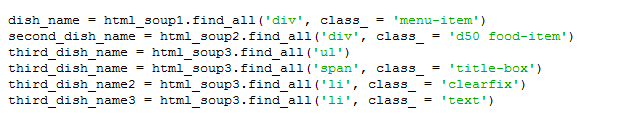
Beautiful Soup is a Python package and in-built library (bs4) for parsing HTML and XML documents (including having malformed markup, i.e. non-closed tags, so named after tag soup). It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

**Parsing HTML**

Use Beautiful Soup to parse an HTML document. Here are some of the things that Beautiful Soup knows:

* Table and list tags have a natural nesting order. For instance, <TD> tags go inside <TR> tags, not the other way around.
* The contents of a <SCRIPT> tag should not be parsed as HTML.
* A <META> tag may specify an encoding for the document.



****

Parsing the HTML based on classes on the tags like: div, span etc. and then storing it to local variables which we can use it in different fields of our Database. Beautiful Soup defines a lot of methods for searching the parse tree, but they’re all very similar.  The two most popular methods: **find()** and **find\_all().**

**File 2 -> demoREST.py**

**Description**: The function of this file is to use the scrapped data from the database and use it to display the data on the front-end. The data is sent to the front-end by using the REST API.

"**flask**" is a Python framework which has a python class **FLASK**.

The library used for this process is **flask\_restful** under **flask. Flask-RESTful** is an extension for Flask that helps to build REST APIs without any difficulty. It is a lightweight abstraction that works with user's existing ORM/libraries.

From flask, libraries like request , response and render\_template are also imported for calling the REST APIs and sending the response on the web page "**Hotel.html**".

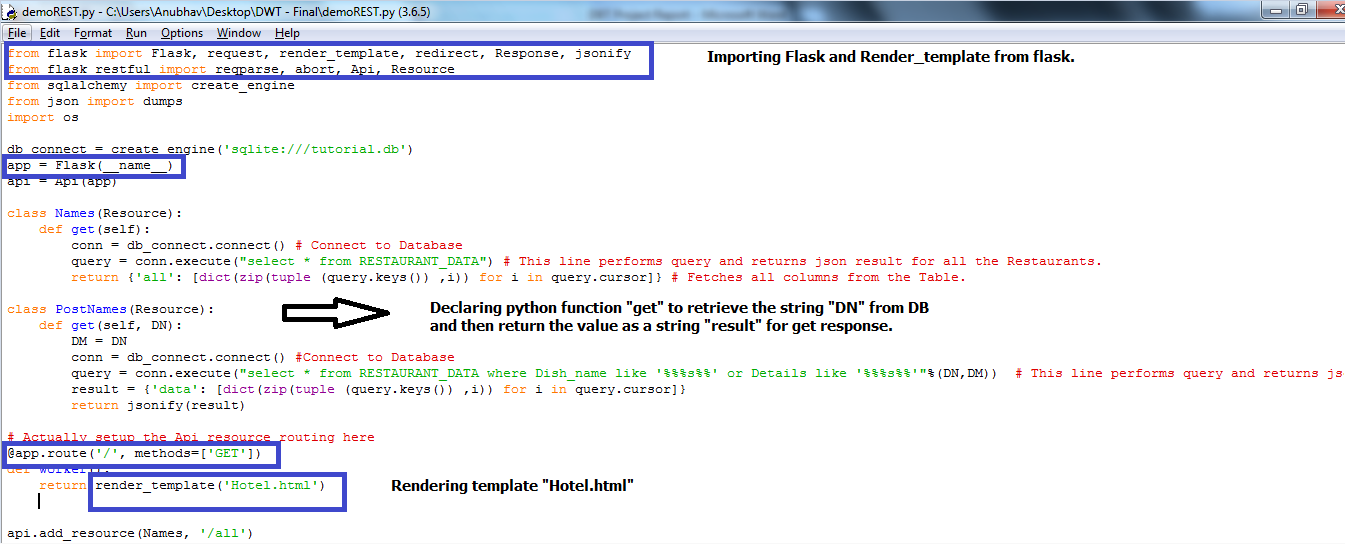
For calling a template, user needs to create a folder called '**templates**' in the same folder where the python file has been kept. when the python file runs, the html file can automatically be called from the templates folder and displayed on the localhost as a template.

**app** is the default Flask application object

**@app.route** tells the Flask class which URL should be triggered when the function is called.

**api.add\_resource** is used to add a resource to the API.

"**if \_\_name\_\_ == '\_\_main\_\_':"** is used to assign a name to the API script**.** Python assigns the name "\_\_main\_\_" to the script when the script is executed. If the script is imported from another script, the script keeps it given name



JSONify is a minimal HTML-form to JSON to HTML-form converting plug-in for jQuery. It creates JSON string from the name-and-value pair of form fields, and can perform JSON to form initialization.

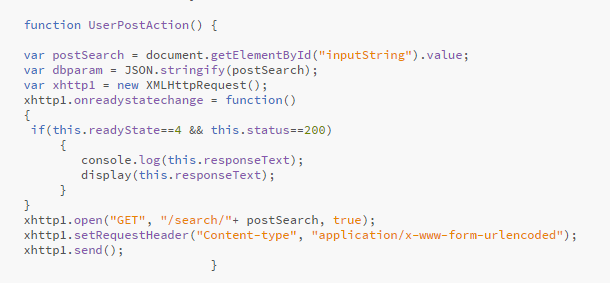
**jsonify()** method converts form field values into standard JSON object, it can be called on any valid form element, which returns JSON string representing the form fields. Note that form fields must have name attribute in order to be included in JSON string.

## 11.2 .HTML File Details:

**File 3 -> Hotel.html**

We have developed out front-end API with HTML JavaScript. Through which we have implemented one search box to process the string, i.e. **"postSearch",** two buttons with id **"b1"** and **"b2"**.

To make the connection from frontend to our REST Api we have used two functions named **UserPostAction()** and **UserAction().**



A new **XMLHttpRequest** request is getting created and then we are checking the status of the page with attributes **ready State = 4 & status = 200** for our webpage.

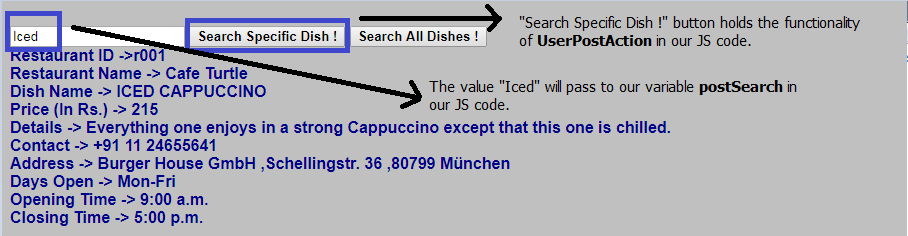
Then the searched string i.e. " **postSearch**" being passed to our REST Api through **open("GET", "/search/"+ postSearch, true);**

and it is sent out directly using **xhttp1.send().**

Once REST Api processes the output based on the particular query like Dish Name or Ingredient details being passed already, then in order to print on our frontend we have created another function named **display(result).**



REST Api then fetches the query in JSON format as we have used **jsonify()** in our REST python code. As JSON works on key value pairs, we have declared one variable **"data1"** to divide the output according to the different columns of our table and then display the data on front end as below.



# 12. Conclusion

Web crawling is a concept of dealing with large data sets and getting the data into the database by crawling it. The data can be further stored in the database and used to provide the information to the user.

One of the most challenging things in the [web crawling](https://www.promptcloud.com/blog/web-crawling-for-retail-business/) space is to deal with coordination of successive crawls. However, identifying the proper tags for the elements can remove the impediments which user can face during the process.

REST Api is an application program interface ([API](https://searchmicroservices.techtarget.com/definition/application-program-interface-API)) that uses [HTTP](https://searchwindevelopment.techtarget.com/definition/HTTP) requests to GET, PUT, POST and DELETE data. The database results can be adjured through REST to bestow the details, which further can be imparted to the end-user.

For the project, basically three processes have been followed :-

* Crawling the data and storing it in database.
* REST API requesting the database table to provide the result and store it on Localhost.
* Stored data can finally be exhibited on the front-end along with all details and sub-details.

# 13. APPENDIX

## 13.1 Show All Dishes

Returns an array of JSON data about all the available dishes in DB.

* **URL**

/all

* **Method**

GET

* **URL Params**

None

* **Data Params**

None

* **Success Response:**
* Code: 200

Content:

[

{

"Restaurant\_id": "r001",

"Restaurant\_Name": "Cafe Turtle",

"Dish\_name": "RISTRETTO",

"price": "145",

"Details": "A small, very strongly flavoured coffee.",

"Contact": "+91 11 24655641",

"Address": "Burger House GmbH ,Schellingstr. 36 ,80799 M\u00fcnchen",

"Days\_Open": "Mon-Fri",

"Opening\_time": "9:00 a.m.",

"Closing\_time": "5:00 p.m."

},

{}, {},{},....so on

]

* **Error Response:**

None

* **Sample Call:**

****

## 13.2 Show Dishes Based on Search

Returns Array of JSON data about dishes from the Database based on keyword Provided by the User.

* **URL**

/search/

* **Method**

GET

* **URL Para**

+postSearch

* **Data Params**

{“name”: “RISTRETTO”}

{“details”: “small”}

* **Success Response:**
* Code: 200

Content:

[

{

"Address": "Burger House GmbH ,Schellingstr. 36 ,80799 M\u00fcnchen",

"Closing\_time": "5:00 p.m.",

"Contact": "+91 11 24655641",

"Days\_Open": "Mon-Fri",

"Details": "A small, very strongly flavoured coffee.",

"Dish\_name": "RISTRETTO",

"Opening\_time": "9:00 a.m.",

"Restaurant\_Name": "Cafe Turtle",

"Restaurant\_id": "r001",

"price": "145"

}

]

{}, {},{},....so on

* **Error Response:**

None

* **Sample Call:**

