Remote Hosts and Web Servers

ITX2000 - CW2

SUBMITTED BY

Mithun Shanoj - M00739358

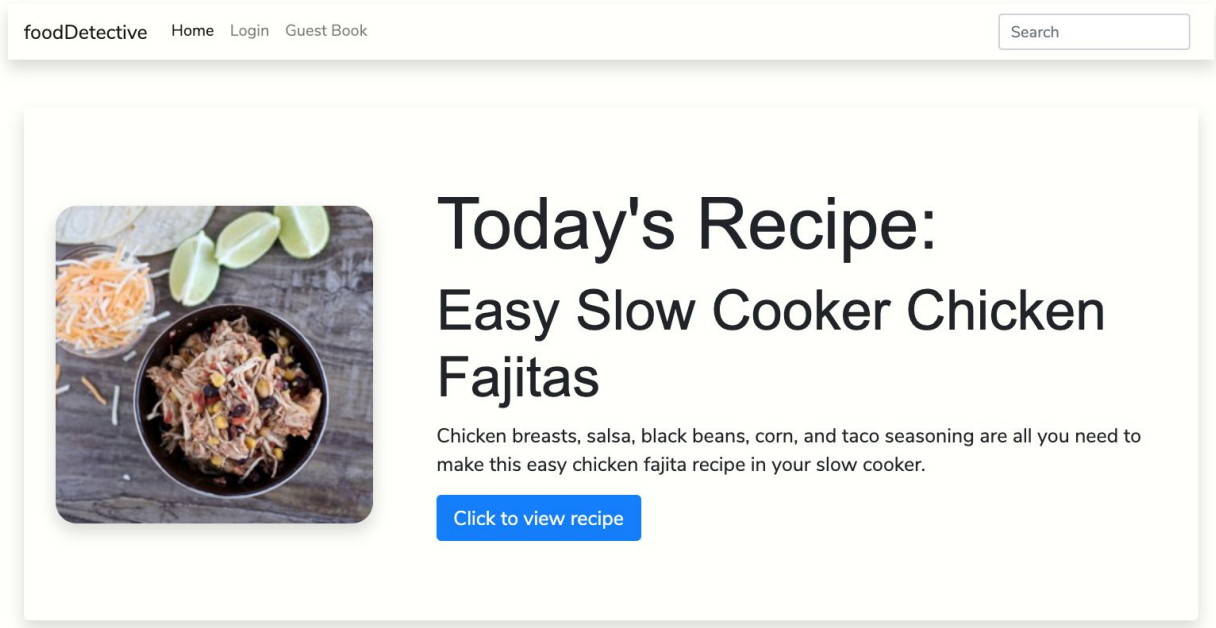
Description

foodDetective is a food review website that serves as a blog for the admin to talk about their favourite dishes. Bootstrap and advanced CSS animations are used in the website to make it look and feel as clean and seamless as possible. Colours and other design elements of the website were also chosen carefully to make sure everything looks fluid and consistent from one another. Each post displays the admin's rating as well as a quick review on how the dish felt to them along with a picture of the dish adjacent to it. Users are given an option to comment on each post as well as make comments on the overall website on the Guestbook.

Dynamic Scraping

Using BeautifulSoup, 3 main blocks of data were scrapped - each from different websites. Scraped data is stored into separate tables with checks for duplication to make sure there is no redundancy in the tables.

1. Recipe of the Day



This block is shown straight on the index page, it displays the recipe name, a short description about it, an image as well as a link that will take you straight to the page that shows additional information on how to cook that dish.

```python
# establish connection with the recipe website
url = requests.get("https://www.allrecipes.com/recipes/")
txt = url.text
soup = BeautifulSoup(txt, 'html.parser')

# scrape recipe of the day
rotd= soup.find('article',class_='fixed-recipe-card' )
f_name = rotd.find('span',class_='fixed-recipe-card__title-link').text
f_desc = rotd.find('div',class_='fixed-recipe-card__description').text
f_link = rotd.a['href']
f_img = rotd.a.img['data-original-src']

# save into db
connection = sqlite3.connect(DB_FILE)
cursor = connection.cursor()
params = {'name': f_name, 'desc': f_desc, 'link': f_link, 'img': f_img}
try:
    cursor.execute("insert into recipeOfTheDay VALUES (:name, :desc, :link, :img)", params)
except:
    print("Already in DB; Not updating.")
connection.commit()
cursor.close()

# display index page with recipe of the day
return render_template('index.html', f_name=f_name, f_desc=f_desc, f_link=f_link, f_img=f_img)
```

```html
<div class="jumbotron shadow shadow" style="background-color: #FFFFFB;">
    <div class="d-flex row">
        <div class="col-4 d-flex align-items-center">
            <img class="shadow" style="border-radius: 20px;" src="{{ f_img }}">
        </div>
        <div class="col-8">
            <h1 class="display-3">Today's Recipe:</h1>
            <h1 class="display-4">{{ f_name }}</h1>
            <p class="lead"> {{ f_desc }} </p>
            <p class="lead">
                <a class="btn btn-primary btn-lg" href="{{f_link}}" role="button">Click to view recipe</a>
            </p>
        </div>
    </div>
</div>
```

Code snippet above shows how the data was scraped and then stored into table as well as how it is displayed on the index page.

2. Quote of the day

> Quote of the day: Thousands of candles can be lighted from a single candle, and the life of the candle will not be shortened. Happiness never decreases by being shared.

For this, a bunch of quotes were extracted from BrainyQuotes and then put into a Python list. From there a random number generator was used to pick a random quote and displayed as the quote of the day on the Guestbook page.

```
# connect and scrape random qoute to show as quote of the day from brainyquote
q_url = "https://www.brainyquote.com/topics/inspirational-quotes"
response = requests.get(q_url)
soup = BeautifulSoup(response.content,'html.parser')
quotes = soup.find_all('a', attrs={"title": "view quote"})
quotesList = []
for quote in quotes:
    quotesList.append(quote.text)

# random quote
num = random.randint(0,len(quotesList))
qotd = quotesList[num]

# save into db
connection = sqlite3.connect(DB_FILE)
cursor = connection.cursor()
params = {'qotd': qotd}
try:
    cursor.execute("insert into quoteOfTheDay VALUES (:qotd)", params)
except:
    print("Already in DB; Not updating.")
connection.commit()
cursor.close()
```

```
<!--Guestbook-->
<div class="row shadow alert alert-warning my-5 w3-animate-opacity">
    <h3>Quote of the day: {{ qotd }}</h3>
</div>
```

Code snippet above shows how the data was scraped and then stored into table as well as how it is displayed on the Guestbook page.

3. Weather

Date	Description	Temperature	Wind	Humidity
18 MAR	Partly Cloudy	--23°	E 16 km/h	52%
19 MAR	Partly Cloudy	29°22°	SSW 20 km/h	51%
20 MAR	Sunny	29°23°	WNW 17 km/h	53%
21 MAR	T-Storms	30°23°	ESE 21 km/h	55%
22 MAR	Isolated T-Storms	30°21°	SW 28 km/h	63%
23 MAR	Mostly Sunny / Wind	24°20°	WNW 38 km/h	47%
24 MAR	Mostly Sunny	25°18°	WNW 24 km/h	50%
25 MAR	Partly Cloudy	26°21°	NE 18 km/h	51%
26 MAR	Partly Cloudy	26°21°	ENE 20 km/h	53%
27 MAR	AM Showers	27°22°	NE 17 km/h	59%
28 MAR	Partly Cloudy	30°23°	ENE 17 km/h	52%
29 MAR	AM T-Storms	30°22°	SE 17 km/h	52%
30 MAR	Partly Cloudy	29°22°	WSW 21 km/h	55%
31 MAR	Partly Cloudy	28°22°	W 22 km/h	54%
1 APR	Partly Cloudy	28°22°	W 21 km/h	52%

Weather data from the site was scraped from weather.com. This was the hardest of the bunch to scrape as I wanted to write a loop to extract the whole table instead of scraping everything row by row. This is shown on the Guestbook.

```python
# establish connection with the weather website
page=requests.get("https://weather.com/en-IN/weather/tenday/l/af60f113ba123ce93774fed531be2e1e51a1666be5d6012f129cfa27bae1ee6c")
content=page.content
soup=BeautifulSoup(content,"html.parser")
l=[]
weather=""
all=soup.find("div",{"class":"locations-title ten-day-page-title"}).find("h1").text

# scrape the weather table
table=soup.find_all("table",{"class":"twc-table"})
for items in table:
    for i in range(len(items.find_all("tr"))-1):
        d = {}
        try:
            d["date"]=items.find_all("span",{"class":"day-detail"})[i].text
            d["desc"]=items.find_all("td",{"class":"description"})[i].text
            d["temp"]=items.find_all("td",{"class":"temp"})[i].text
            d["wind"]=items.find_all("td",{"class":"wind"})[i].text
            d["humidity"]=items.find_all("td",{"class":"humidity"})[i].text

            # save into db
            connection = sqlite3.connect(DB_FILE)
            cursor = connection.cursor()
            params = {'date': d["date"], 'desc': d["desc"], 'temp': d["temp"], 'wind': d["wind"], 'humidity': d["humidity"]}
            try:
                cursor.execute("insert into weather VALUES (:date, :desc, :temp, :wind, :humidity)", params)
            except:
                print("Already in DB; Not updating.")
            connection.commit()
            cursor.close()

        except:
            d["date"]="None"
            d["desc"]="None"
            d["temp"]="None"
            d["wind"]="None"
            d["humidity"]="None"
        l.append(d)
```

```html
<div class="row shadow p-5 w3-animate-opacity">
    <table class="table table-striped justify-content-around" style="min-width: 100%;">
        <thead>
            <tr>
                <th>Date</th>
                <th>Description</th>
                <th>Temperature</th>
                <th>Wind</th>
                <th>Humidity</th>
            </tr>
        </thead>
        <tbody>
            {% for day in weather %}
        <tr>
            <th>{{ day['date'] }}</th>
            <th>{{ day['desc'] }}</th>
            <th>{{ day['temp'] }}</th>
            <th>{{ day['wind'] }}</th>
            <th>{{ day['humidity'] }}</th>
        </tr>
        {% endfor %}
        </tbody>
    </table>
</div>
```

SQL Tables for Dynamic Scraping

Quote of the day

```
sqlite> select *
   ...> from quoteOfTheDay;
 qotd = If I have seen further than others, it is by standing upon the shoulders of giants.

 qotd = What lies behind you and what lies in front of you, pales in comparison to what lies inside of you.

 qotd = As we express our gratitude, we must never forget that the highest appreciation is not to utter words, but to live by them.

 qotd = What we think, we become.

 qotd = You must do the things you think you cannot do.
```

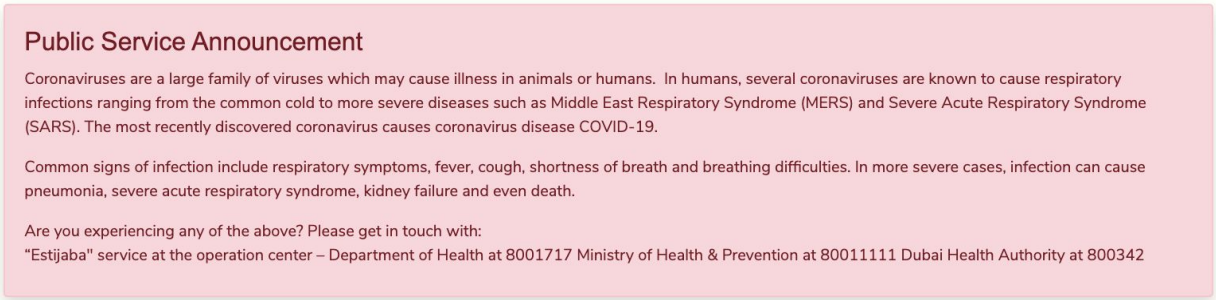
Weather

```
sqlite> select *
   ...> from weather;
    date = 18 MAR
    desc = Partly Cloudy
    temp = --23°
    wind = E 20 km/h
humidity = 53%

    date = 19 MAR
    desc = Partly Cloudy
    temp = 29°22°
    wind = SW 21 km/h
humidity = 51%

    date = 20 MAR
    desc = Sunny
    temp = 29°23°
    wind = WNW 17 km/h
humidity = 53%
```

Recipe of the day

```
sqlite> select *
   ...> from recipeOfTheDay;
 name = Easy Slow Cooker Chicken Fajitas
 desc = Chicken breasts, salsa, black beans, corn, and taco seasoning are all you need to make this easy chicken fajita recipe in yo
 link = https://www.allrecipes.com/recipe/259768/easy-slow-cooker-chicken-fajitas/
  img = https://images.media-allrecipes.com/userphotos/300x300/4932240.jpg
```

Static Scraping



For this, I chose to make a PSA - Public Service Announcement about COVID-19 with information from three different sites. The first is a definition of the virus, which was taken from WHO's official site, the second are the symptoms which are taken from another WHO page and last is emergency contact taken from Dubai government's official corona help-line page.

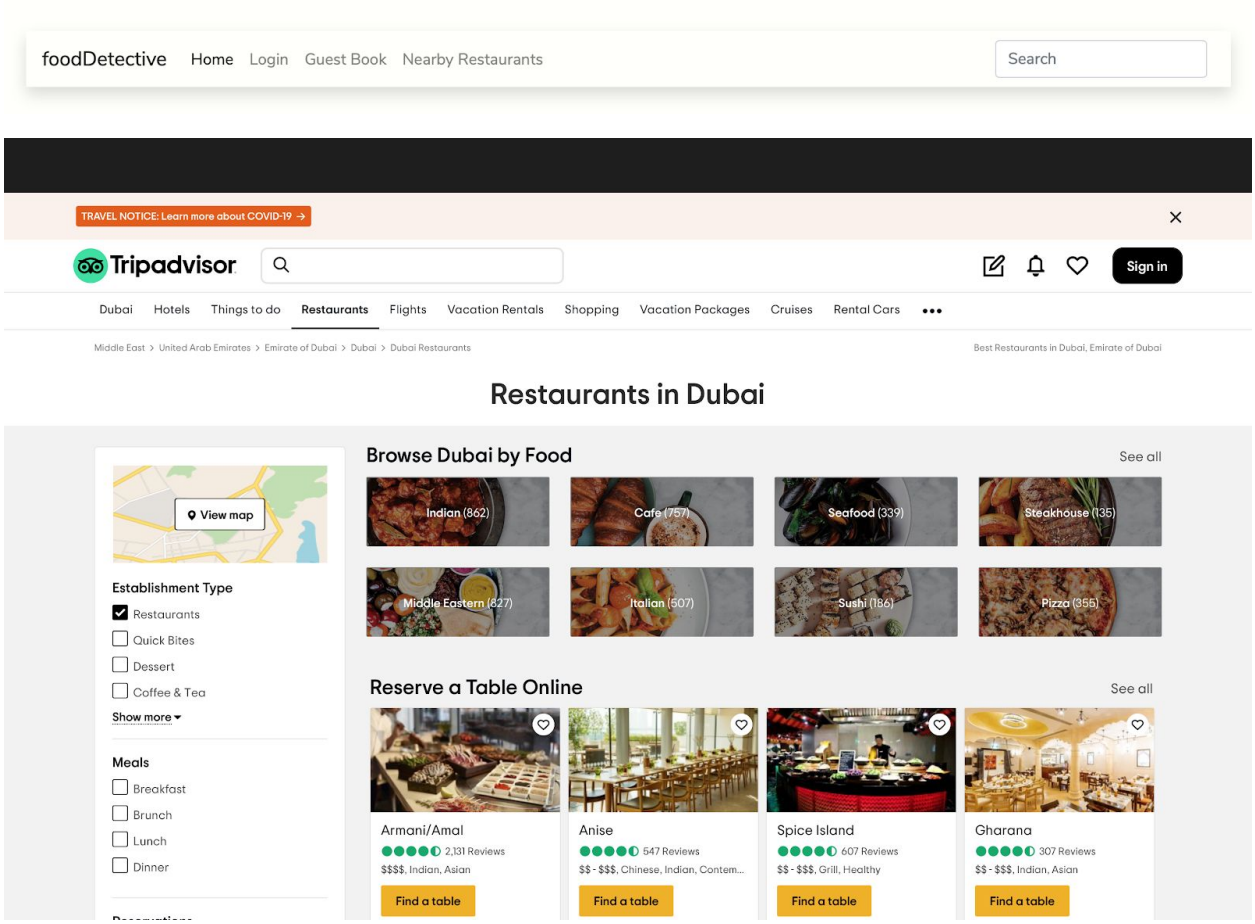
```
# connect and scrape corona related data / what is corona?
c_url = "https://www.who.int/news-room/q-a-detail/q-a-coronaviruses"
response = requests.get(c_url)
soup = BeautifulSoup(response.content,'html.parser')
whatC = soup.find("div",{"class":"sf-accordion__content"}).find("p").text

# connect and scrape corona related data / corona symptoms
c_url = "https://www.who.int/health-topics/coronavirus"
response = requests.get(c_url)
soup = BeautifulSoup(response.content,'html.parser')
symptomsC = soup.find_all('span', attrs={"style": "background-color:transparent;text-align:inherit;text-transform:inherit;
white-space:inherit;word-spacing:normal;caret-color:auto;"})[0].text

# connect and scrape corona related data / who to contact?
c_url = "https://www.dha.gov.ae/Covid19/Pages/home.aspx"
respons  BeautifulSoup t(c_url)
soup = BeautifulSoup(response.content,'html.parser')
contactC = soup.find_all('ul', attrs={"style": "list-style-type:disc;margin-left:22.15px;"})[1].text
```

```
<div class="row shadow alert alert-danger my-5 w3-animate-opacity">
   <h3>Public Service Announcement</h3>
   <p>{{ whatC }}</p>
   <p>{{ symptomsC }}</p>
   <p>Are you experiencing any of the above? Please get in touch with: <br>
      {{ contactC }}
   </p>
</div>
```

Code snippet above shows how the data was scraped and displayed on Guestbook page

New Concepts / Ideas

1. Nearby Restaurants - Google Library



Nearby Restaurants is a tab on my Navbar that will run a google search with the query 'Restaurants Near Me' and takes you to the first result. This works dynamically so say, if the user is in Iceland and uses that tab, the link would be of restaurants in Iceland near the user and not from Dubai since Google results are personalised.

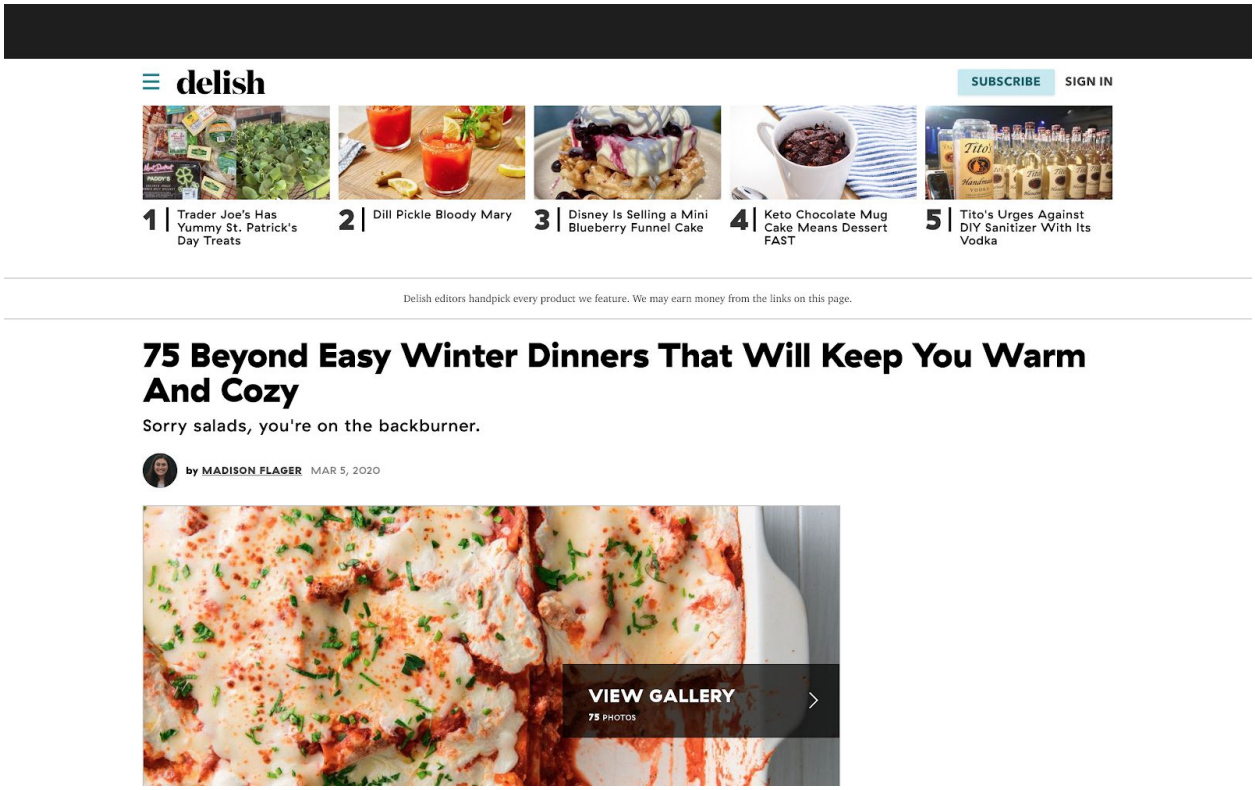
Relevant code snippet on the next page.

```
# google search for nearby restaurants
query = "restaurants near me"
for j in search(query, tld="ae", num=1, stop=1, pause=2):
    restaurant = j
```

```
    <li class="nav-item">
        <a class="nav-link" href="{{ restaurantnearme }} ">Nearby Restaurants</a>
    </li>
</ul>
```

2. Seasonal Recipe Recommendations

This Winter, have fun cooking with these recipes selected just for you: Click ME!



This uses Google library again but with added logic to find the current season using datetime library.

```
if (currentMonth >= 5 and currentMonth <10):
    season = "Summer"
    query2 = "cold recipe for hot weather"

else:
    season = "Winter"
    query2 = "Winter recipe for cold weather"

for k in search(query2, tld="ae", num=1, stop=1, pause=2):
    recipe = k
```

Code snippet behind seasonal recipe recommendation logic.

3. Efficient Table Extraction - Loops, Ranges & Lists

While trying to make the code efficient, I found that by using loops, ranges and Python lists I was able to shorten the code by a huge margin.

```
# establish connection with the weather website
page=requests.get("https://weather.com/en-IN/weather/tenday/l/af60f113ba123ce93774fed531be2e1e51a1666be5d6012f129cfa27bae1ee6
content=page.content
soup=BeautifulSoup(content,"html.parser")
l=[]
weather=""
all=soup.find("div",{"class":"locations-title ten-day-page-title"}).find("h1").text

# scrape the weather table
table=soup.find_all("table",{"class":"twc-table"})
for items in table:
    for i in range(len(items.find_all("tr"))-1):
        d = {}
        try:
            d["date"]=items.find_all("span",{"class":"day-detail"})[i].text
            d["desc"]=items.find_all("td",{"class":"description"})[i].text
            d["temp"]=items.find_all("td",{"class":"temp"})[i].text
            d["wind"]=items.find_all("td",{"class":"wind"})[i].text
            d["humidity"]=items.find_all("td",{"class":"humidity"})[i].text

            # save into db
            connection = sqlite3.connect(DB_FILE)
            cursor = connection.cursor()
            params = {'date': d["date"], 'desc': d["desc"], 'temp': d["temp"], 'wind': d["wind"], 'humidity': d["humidity"]}
            try:
                cursor.execute("insert into weather VALUES (:date, :desc, :temp, :wind, :humidity)", params)
            except:
                print("Already in DB; Not updating.")
            connection.commit()
            cursor.close()

        except:
            d["date"]="None"
            d["desc"]="None"
            d["temp"]="None"
            d["wind"]="None"
            d["humidity"]="None"
        l.append(d)
```

What I do is - instead of manually scraping weather data row by row, my loop runs though and and creates bs4 objects that it later appends to a Python list. This way if the page I'm scraping from decides to increase their forecast from a current 10-day to maybe, say 15 -days I wouldn't have to make any change to my code as well.

4. Random Quote Generator

Using the Python lists and Random library, I was able to write a random quote generator for my 'Thought of the day' page. It wasn't something entirely new but I think using it in this way rather than directly scraping the quote from a 'quote of the day' website was a different approach.

```
q_url = "https://www.brainyquote.com/topics/inspirational-quotes"
response = requests.get(q_url)
soup = BeautifulSoup(response.content,'html.parser')
quotes = soup.find_all('a', attrs={"title": "view quote"})
quotesList = []
for quote in quotes:
    quotesList.append(quote.text)

# random quote
num = random.randint(0,len(quotesList))
qotd = quotesList[num]
```

Additional Notes

- All Dynamic Scraped data are stored into tables without duplication of data.
- In case of duplicate data, the website prints that the data is already in the DB on to the console.

```
(env) mythxn@MacBook-Pro-2 foodDetective % flask run
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Already in DB; Not updating.
Already in DB; Not updating.
Already in DB; Not updating.
```

- Everything was written using Try and Catch blocks to make sure there are no surprise bugs in the code.