



UULebanon

A Capstone Web Project

By

Miguel Ibrahim

ID: 202005354

Capstone Professor: Dr. Azzam Mourad

CSC 599: Capstone Project

Lebanese American University

Arts & Science

Table of Contents

I.	Introduction:	3
II.	Technology Stack:	4
III.	Acknowledgements:	5
IV.	Explaining Key Terms:	6
V.	APIs and Frameworks Used:	7
VI.	Database Design:	11
A.	TABLE: USERS	13
B.	TABLE: RESTAURANTS	14
C.	TABLE: TOURISTIC_SITES	16
D.	Relational databases:	17
a.	Visited_sites:	17
b.	Rated_restaurants:	17
VII.	Application Flow:	18
i.	Index.php:	20
ii.	Touristic Attractions: Tourism.php	21
iii.	Anjar.php:	28
iv.	Restaurant.php:	30
v.	SignUp/ LogIn:	35
vi.	Uni-fied/php/Tourism.php:	41
vii.	Server-Side Automated Functions	44
viii.	Uni-fied/php/restaurants.php	57
ix.	Events.php	72
VIII.	Future Prospects:	78
IX.	References	80

I. Introduction:

As an introduction to the Capstone Project, one must start with the problem it is trying to solve. According to macrotrends Lebanon's tourism statistics for 2020 averaged around 2,369,000,000\$ being spent by tourists, a sharp decline of 72.82% from 2019. One of the major problems when it comes to searching for a country on google, Bing, or any other search engine, is that the first few pages does not sum up the information into a concise and clear page. Some results' origins net from Instagram such as the fact with the 961 page (<https://www.the961.com/touristic-sites-lebanon-this-summer/>), these images may be removed, altered, reported, or private to the uploader (if the user's page is private).

The idea sprouted while scrolling through the many restaurants on google maps. When searching for restaurants on google maps, the user is greeted with many results, and searching through them, and through their pictures may be daunting and time consuming for the user. Users these days want information to be available on a single page in a concise manner, and always available.

The Web Application: UULebanon looks to solve these problems by providing the necessary information, pictures, hosted in a clear and concise manner with no link to Instagram with a sole focus on the user as well as the performance of the application. The UU in UULebanon stands for Unique and Unified which are the main goals of this Web App, and of course Lebanon since it focuses on Lebanon.

Similar to trip advisor, this Web Application looks forward to providing a “local trip advisor”, such local trip advisor is worked in collaboration with experienced touristic companies in Lebanon, as well as the developer’s experience in Public Transport in Lebanon.

II. Technology Stack:

❖ Front End Stack:

- HTML5: HTML that has CSS styling embedded within, as well as improved semantic tags and mobile compatibility as well as the capability of incorporating it with Caching techniques.
- CSS3: New CSS Stack that serves for a better responsiveness for all devices. It also has advanced styling features, media queries, flexbox, and reduced file sizes.
- JavaScript: JavaScript is used in this application to allow users to see a responsive page as well as allowing code reusability when it comes to the search Bar as well as the Navbar.

❖ Back End Stack:

- PHP: PHP is used in this application to make the most of the website dynamic as well as allowing the website to reuse elements of code such as the navbar by calling functions in a php code that has the navbar.

- Python: Python is used in this Web Application to Web scrape information from the web and add it to the web application, the python code also allows recommending touristic sites to users based on their visited sites.

❖ Database Stack:

- MySQL: MySQL is a popular Database that is mostly used with Web Applications, this database allows the website to load the content dynamically as well as providing future incentives for the Web Application.

III. Acknowledgements:

I'd like to thank Dr. Azzam Mourad for advising and guidance, and Mr. Mario Chahoud for the ideas implemented into this website.

I'd like to also mention that some functionalities and bugs are solved with help from stack overflow and GitHub

There are some CSS, loading elements, AJAX, Bootstrap that are extracted online and downloaded.

The API keys are extracted from signing into an account and creating them

I used many Open-Source Software in this application to make it easier to develop this application, and keep it simple and cheap.

IV. Explaining Key Terms:

1. Web App: it is a software program that runs on a web server and can be accessed through a web browser. Unlike traditional desktop applications, web applications do not need to be installed on a user's device, and they can be accessed from any device with an internet connection on a web browser.
2. Caching: Caching is a technique used by web applications to improve performance and reduce server load. It involves storing frequently accessed data in fast storage mediums so it can be quickly retrieved without needed to generate it every time the website is called.
3. Responsive: A responsive web application means that the web application can run on TV, Laptop, Desktop, and mobile.
4. Dynamic: A dynamic web application is a web application that uses databases to load the information on the website. This can be useful if the developer wants a short code with no need to focus on styling everything.
5. Search Bar: a search bar is a convenient User Interface to all users, if the user wants to search for anything on all the database, they can input what they want to search for and return a result if it exists in the database.
6. Nav Bar: Nav Bar or Navigation Bar is an essential element to the usability of the web application; this allows the user to visit all the other websites the web application can offer. This Nav Bar can

transform into a drop nav when the screen width is mobile proportions.

The drop nav is similar to navbar; however, it has a recognizable burger line which is always recognizable as a drop nav.

7. Code reusability: This signifies that most of the code bases are similar to each other, thus there is no need to rewrite the same code in every file, instead these codes are written in a separate file, and called using a function making the code smaller.
8. Web Scraping: Web scraping is a technique used by Natural Language Processing, a branch of Artificial Intelligence, to scrape websites for essential information, which can prove useful for use later.

V. APIs and Frameworks Used:

An API (Application Programming Interface) is a set of protocols, routines, and tools that enable different software applications to communicate with each other. APIs are used to define how software components should interact, making it easier for developers to create complex applications by providing pre-built functionality.

APIs allow developers to interact with a service or application in a standardized way, without having to know the underlying implementation details. For example, a weather API may provide developers with access to current weather conditions by accepting a location parameter and returning weather data in a standardized format such as JSON or XML. (MuleSoft, n.d.).

APIs:

- Google API:
 - Google Maps API
 - Google Reviews API
 - Google Fonts API
- Other Maps:
 - Openstreetmap.org
 - Leaflet
- Weather API
 - OpenWeatherMap API

Python Imports:

- Mysql.connector
- Requests
- Bs4 → BeautifulSoup

Frameworks:

- Bootstrap
- jQuery
- CDN (Content Delivery Network) → AJAX

Explaining usage of APIs:

Openstreetmap + Leaflet:

1. Cost: Google Maps API usage can be expensive for websites that receive a large amount of traffic or require frequent updates.
2. Openness: OpenStreetMap data is open-source, meaning that it can be freely used, edited, and distributed by anyone. This allows for greater flexibility and customization than Google Maps, which has more restrictions on data usage.
3. Privacy: Some users may prefer to use a mapping solution that does not track their location or collect their data, and OpenStreetMap offers a more privacy-conscious option.
4. Customization: Leaflet is a lightweight and highly customizable JavaScript library for interactive maps, allowing developers to create unique and tailored mapping experiences for their users.

(Leaflet, 2022)

OpenWeatherMap API:

OpenWeather is a popular weather information provider used by many developers and businesses because of its extensive coverage and features. It provides up-to-date and accurate weather information for millions of locations worldwide, including current weather conditions, hourly and daily forecasts, historical data, and more. Additionally, OpenWeather provides easy-to-use APIs that developers can use to integrate weather data

into their applications quickly and easily. Other weather information providers may have their own unique features or advantages, but OpenWeather's coverage, accuracy, and API ease-of-use make it a popular choice for many developers. (Open Weather, 2023)

Content Delivery Network using CloudFlare

Cloudflare is a company that provides a content delivery network (CDN) to improve the performance, security, and reliability of websites and applications.

A CDN is a distributed network of servers that work together to deliver content to users from the server that is geographically closest to them. By caching content on servers located closer to end-users, a CDN can reduce the distance that data must travel, resulting in faster load times for websites and applications.

Cloudflare's CDN is designed to provide additional features beyond traditional CDN offerings. These features include DDoS protection, SSL/TLS encryption, web application firewall (WAF) protection, and caching optimization. Cloudflare also provides analytics and performance insights to help website owners optimize their content delivery.

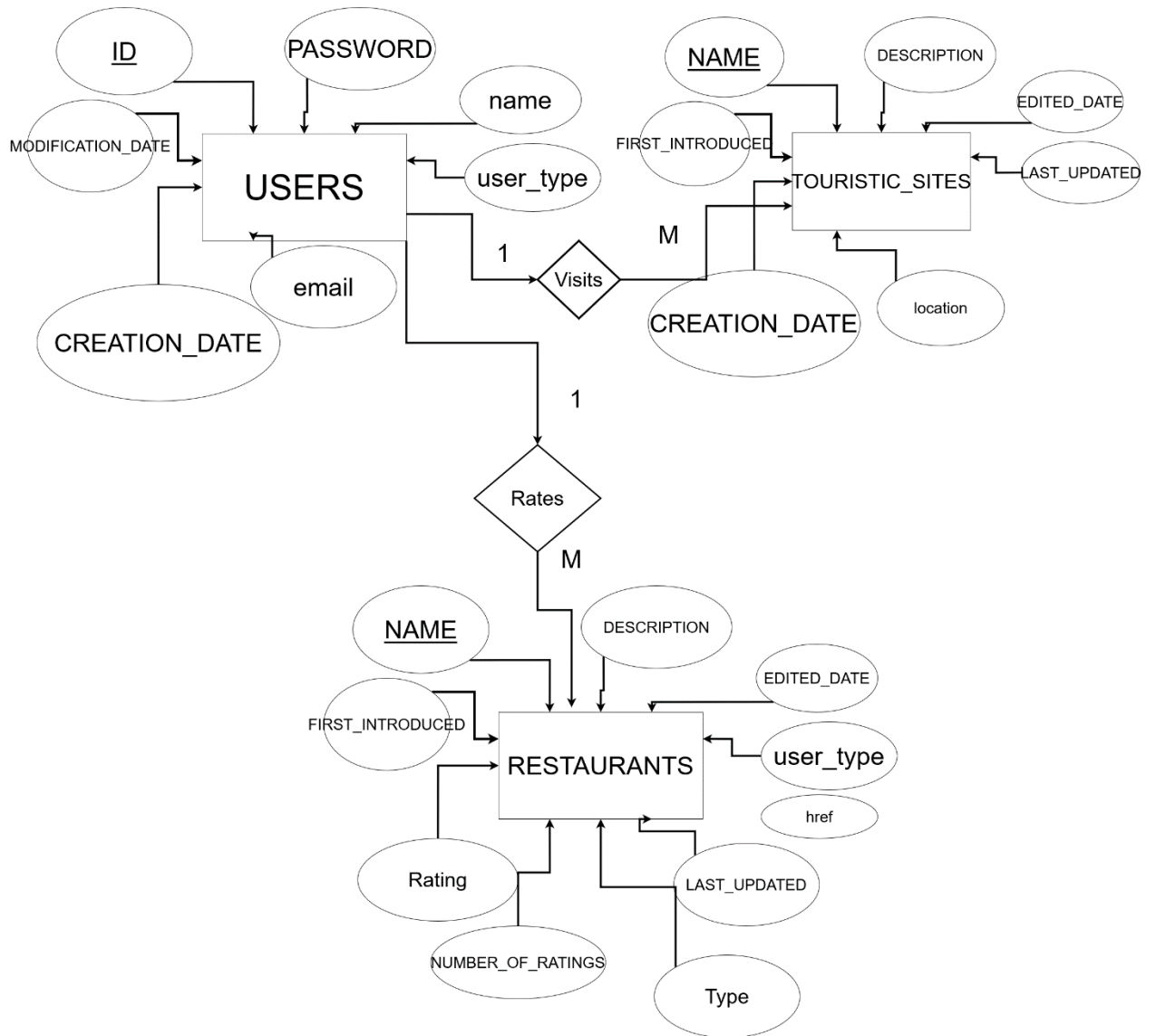
Overall, Cloudflare's CDN can help website owners improve their website's speed, security, and reliability while reducing their infrastructure costs. (Cloudflare, 2023)

VI. Database Design:


Legend:

- Rectangle: TABLES
- Circle/Ovals: Columns, attributes, assets.
- Rhombus (Tilted Squares): Relations

ER DIAGRAM :



A. TABLE: USERS

	#	Name	Type	Collation	Attributes	Null
<input type="checkbox"/>	1	ID 	int			No
<input type="checkbox"/>	2	PASSWORD	varchar(256)	utf8mb4_0900_ai_ci		No
<input type="checkbox"/>	3	CREATION_DATE	date			Yes
<input type="checkbox"/>	4	MODIFICATION_DATE	date			Yes
<input type="checkbox"/>	5	email	varchar(200)	utf8mb4_0900_ai_ci		No
<input type="checkbox"/>	6	name	varchar(200)	utf8mb4_0900_ai_ci		No
<input type="checkbox"/>	7	user_type	varchar(50)	utf8mb4_0900_ai_ci		No

The Table: Users is the table that stores signed up users inside it, the Primary key is **ID** to help interact with other tables, it is an int variable that is AUTO_INCREMENTed meaning that every time a new user is registered, the database gives them a new ID unique to them.

PASSWORD: password is a variable character of size 256, it uses a unique hashing technique for MySQL to store the user's password in the database

CREATION DATE: creation date is a date format that tracks if there is any breach in the database, it tracks the creation date of the user for this Web Application.

MODIFICATION DATE: modification date is a date format that tracks when the user changes his/her password and to detect the date of the potential data breach.

email: email is trivial to all web applications, instead of saving the ID of the user as a session, their email is stored in AES format on the session, this is a varchar of 200 to allow basic 128-bit AES encryption.

Name: Name is an essential part to keep track of whoever is active on the website, this web application can use the name of the person and the last activated time since the last cookie generated to see if the user is active or not

Usertype: This column is there to add more functionalities, websites and interfaces to the web application, this allows future improvements such as adding events, hotels, and organizations.

B. TABLE: RESTAURANTS

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 NAME	varchar(90)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	2 DESCRIPTION	varchar(300)	utf8mb4_0900_ai_ci		Yes	NULL			Change Drop More
<input type="checkbox"/>	3 FIRST INTRODUCED	date			No	None			Change Drop More
<input type="checkbox"/>	4 EDITED_DATE	date			No	None			Change Drop More
<input type="checkbox"/>	5 LAST_UPDATED	date			No	None			Change Drop More
<input type="checkbox"/>	6 Rating	float			No	None			Change Drop More
<input type="checkbox"/>	7 NUMBER_OF_RATINGS	int			No	None			Change Drop More
<input type="checkbox"/>	8 Type	varchar(50)	utf8mb4_0900_ai_ci		No	None			Change Drop More
<input type="checkbox"/>	9 href	varchar(300)	utf8mb4_0900_ai_ci		No	None			Change Drop More

NAME: name is the primary key of this table, it is a varchar of 90, it allows the user to interact with another table which is **rated_restaurants** using Table users and the logged in user.

DESCRIPTION: description is a varchar of 300 characters, in this application, it is used to describe restaurants to make the genre, food selection or filtering results easier.

FIRST INTRODUCED: this column is when the restaurant is first introduced into the database, it is a date format that allows the admin to know how long this restaurant has been in the database.

EDITED DATE, LAST UPDATED: it is similar to the table users, it has the same functionality.

















Rating: Rating is a floating point that tells the user how much the restaurant is rated.

NUMBER OF RATINGS: this keeps track of how many ratings the restaurant has received to not change the number of stars significantly each time a user rates a restaurant.

Type: Type is the type of restaurant it is, it is a variable character of size 50 allowing to determine the type of restaurant (food style) :Japanese, Lebanese, Indian, etc.

Href: Href allows the program to redirect the user to the main page of the restaurant which in this case is tourist advisor since it's a website that is always online unlike other web applications

C. TABLE: TOURISTIC SITES

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 NAME 	varchar(50)	utf8mb4_0900_ai_ci		No	None			 Change  Drop  More
<input type="checkbox"/>	2 DESCRIPTION	varchar(300)	utf8mb4_0900_ai_ci		Yes	NULL			 Change  Drop  More
<input type="checkbox"/>	3 FIRST INTRODUCED	date			Yes	NULL			 Change  Drop  More
<input type="checkbox"/>	4 EDITED_DATE	date			No	None			 Change  Drop  More
<input type="checkbox"/>	5 LAST_UPDATED	date			Yes	NULL			 Change  Drop  More



NAME: NAME is the primary key in this TABLE, it is a variable character of size 50, it allows the user to mark the site as visited according to the value of NAME stored in visited sites.

DESCRIPTION: This is the description of the site, it refers to the more information about the site, allowing the display of the information to be displayed.

FIRST INTRODUCED, EDITED DATE, LAST UPDATED: these are like the ones in users, and touristic sites.

D. Relational databases:



a. Visited_sites:

	#	Name	Type	Collation	Attributes	Null	Default
<input type="checkbox"/>	1	VISITED	tinyint(1)			No	None
<input type="checkbox"/>	2	USER_ID 	int			No	None
<input type="checkbox"/>	3	site_name 	varchar(200)	utf8mb4_0900_ai_ci		No	None

This relational database “**visited_sites**” is a product of having a relational aspect in this relational database, users visit touristic sites, this relation is one-to-many, meaning that one user (at a time) may visit many sites, and many sites may be visited by many (one at a time).

In this case, it takes the site_name as a foreign key to the table: touristic_sites’ “NAME” and allows the USER_ID to alter VISITED to visited or not visited according to sitename.

b. Rated_restaurants:

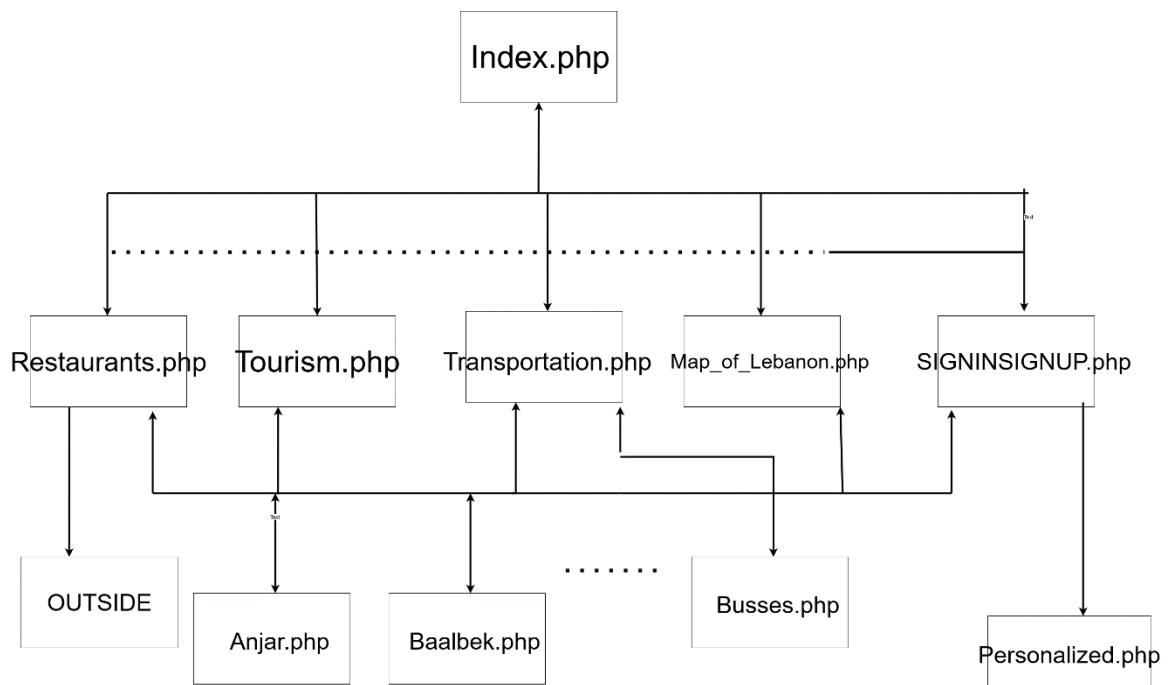
	#	Name	Type	Collation	Attributes	Null	Default
<input type="checkbox"/>	1	NAME 	varchar(100)	utf8mb4_0900_ai_ci		No	None
<input type="checkbox"/>	2	RATING	int			No	None
<input type="checkbox"/>	3	USER_ID 	int			No	None

This relational database “**rated_restaurants**” is a product of having a relational aspect in this relational database, users rates restaurants, this relation is one-to-many, meaning that

In this case, it takes the restaurant_name (NAME) as a foreign key to the table: restaurants “NAME” and allows the USER_ID to alter Rating to allow the user to input their own rating for a restaurant.

VII. Application Flow:

Version I:



Legend:

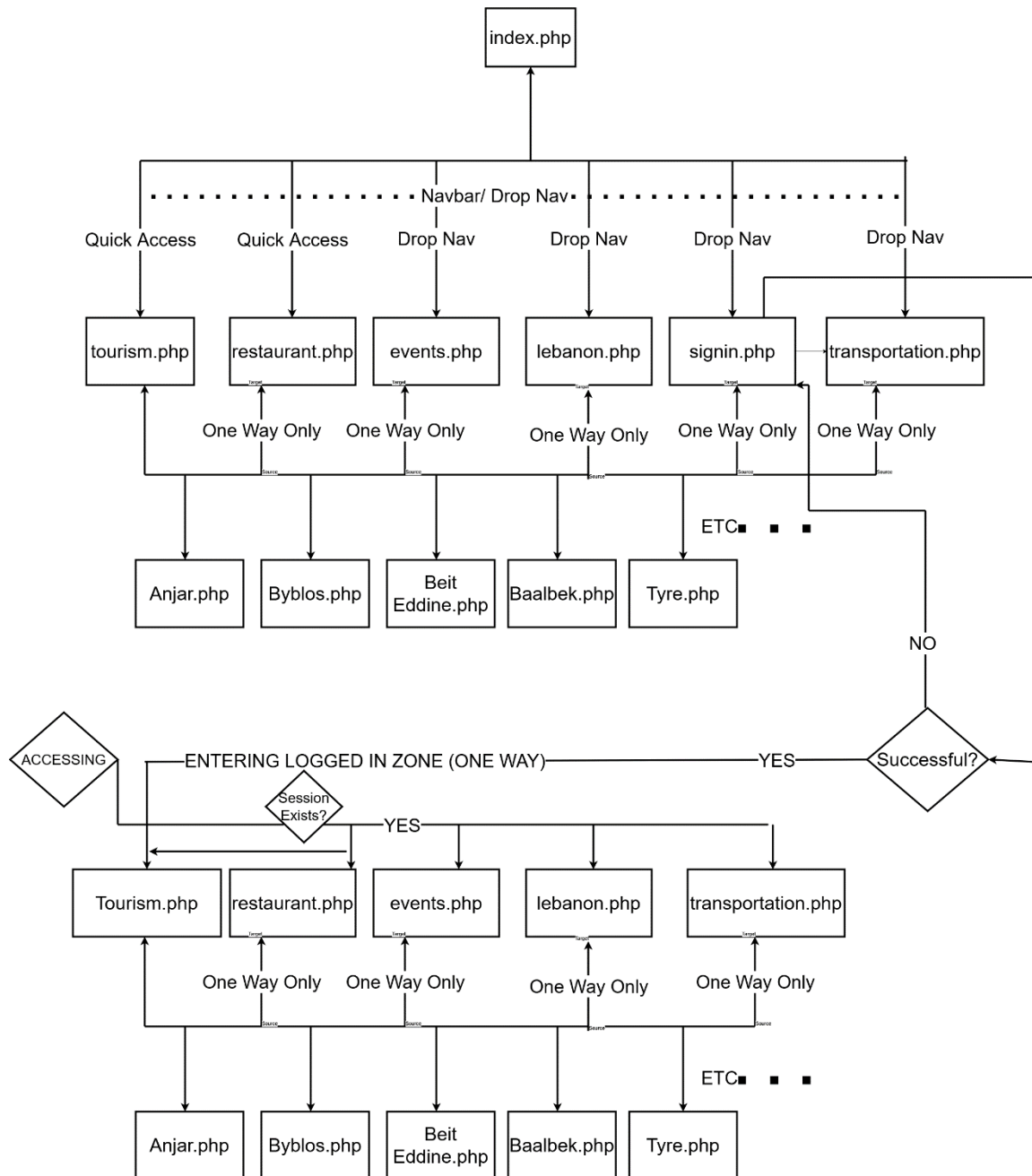
Dotted Lines: You can go from any of these places to any of the other places

Full line: One way only, only accepts incoming requests, does not reroute

Line with arrow: Allows movement of user from current place to place pointed to

Dotted Lines between Baalbek and Busses.php: There exists more Touristic places that are not mentioned.

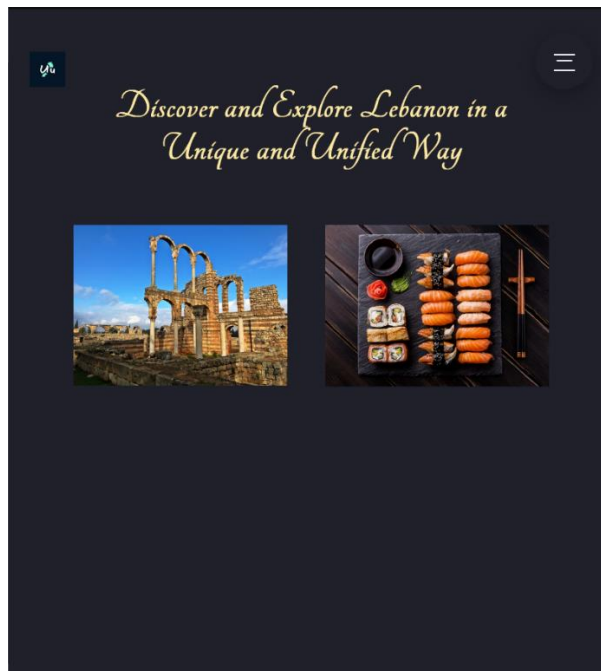
Version II:



The dotted lines and arrows are similar to version I, however, there is a lot more functionalities, and the web application is much larger than version I. In version II

however, the rhombus represents an if statement, so if the sign in was successful it will go to Tourism.php, if not, it will return to signin.php. Trying to access a logged in page should be impossible since it is a breach of the system, so the program checks for if the session exists or not, if it exists it will allow the user to access the page he/she typed in the browser, if it did not find it, it will reroute them to the main page which is index.php. The One Way Only represents that the application flow only goes one way from a place to another with no physical way back unless one presses the back button on the browser.

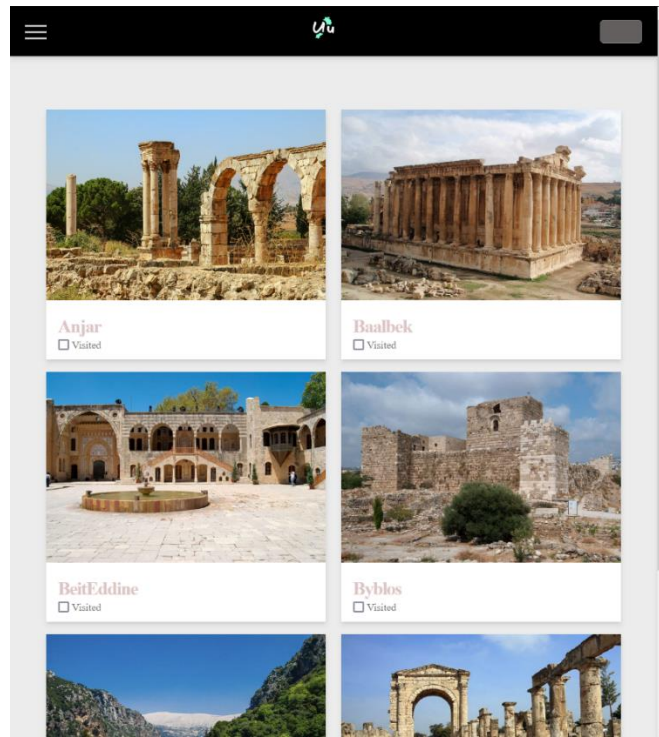
i. **Index.php:**



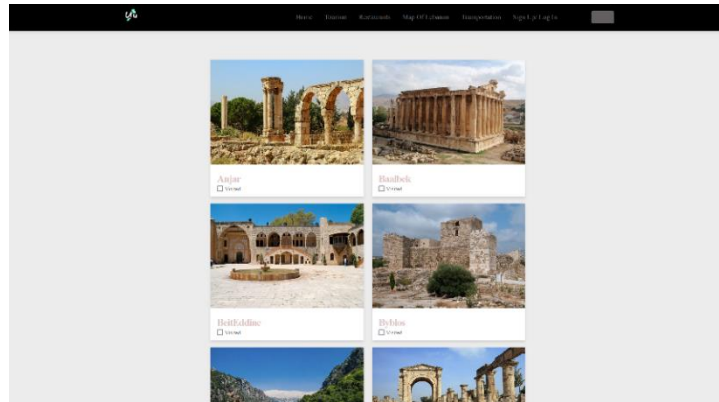
Index.html is the first page the user sees when they visit the Web Application, this php file is composed of a drop nav, stylings, APIs (Google fonts APIs), this page also contains means to directly access the sites using the pictures displayed on the main page, the first one on the left is touristic sites, and the one on the right is restaurants. This file has access to the styling sheet: indexstyles.css that has access to API fonts created by google, and

access to content delivery network (CDN). This website is responsive to all available screen sizes and is capable of rerouting to: Touristic Attractions, Restaurants, Transportation, Map of Lebanon, and Sign In.

ii. **Touristic Attractions: Tourism.php**



In this php file, the web application queries MySQL first to select all the results from touristic_sites and display them dynamically on the front end using php and css. This page also has a responsive navbar that is capable of morphing from a drop nav to a top nav according to the text size. Like so:



Code :

```
<?php

require_once "../navbar/navbar.php";

loadNavBar();

?>

<br><br><br><br>

<?php
require "../config/cfg.php";

try {

    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $pdo->query("SELECT NAME FROM touristic_sites");

    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

?>

<div class="cards">
```

```

<?php

if ($result) {

    foreach($result as $row) {

        $name = $row["NAME"];

        ?>

        <div class="card">

            <div class="card__image-holder">

                <?php echo'';?>

            </div>

            <div class="card-title">

                <h2>

                    <?php echo'<a href="/'.$name.'.php" target="_blank">'.$name.'</a>';?>

                </h2>

                <div>

                    <?php echo' <input type="checkbox" id="'.$name.'" name="'.$name.'"

onclick="run()">';?>

                    <?php echo' <label for="'.$name.'"><small>Visited</small></label>';?>

                </div>

            </div>

        </div>

    <?php }

} else {

    echo "0 results";

```

```

    }

    } catch(PDOException $e) {

        echo "Error: " . $e->getMessage();

    }

    $pdo = null;

?>

</div>

</body>

<script src="../JS/app.js"></script>

    <script src="../JS/SearchBar.js"></script>

<script>

    function run(){

        const id=document.getElementById();

        alert("You need to be logged in to mark a place as visited");

    }

</script>

</html>

```

This php file is extremely concise and straightforward, this php file queries mysql with the config file save on another folder, this query returns result and the php can display these results on the page in a card-like format. If this page were to be static, the code would have been 500 lines long, using this approach allows us to reduce the size of the code.


```
<?php  
require "../config/cfg.php";
```

In this code, the program is calling the config file for the configuration of mysql database name, location, username and password.

```
<?php  
  
require_once "../navbar/navbar.php";  
  
loadNavBar();  
  
?>
```

In this code, the function being called is in navbar, the navbar is being called dynamically from another file and displayed on the page.

```
<?php  
  
function loadNavBar(){  
  
?>  
  
<header class="header" id="header">  
  
  <nav class="navbar container">  
  
    <a href="../index.html" class="brand"></a>  
  
    <div class="burger" id="burger">
```

```

    <span class="burger-line"></span>

    <span class="burger-line"></span>

    <span class="burger-line"></span>

</div>

<span class="overlay"></span>

<div class="menu" id="menu">

    <ul class="menu-inner">

        <li class="menu-item"><a class="menu-link"
href="../index.php">Home</a></li>

        <li class="menu-item"><a class="menu-link"
href="../Tourism.php">Tourism</a></li>

        <li class="menu-item"><a class="menu-link"
href="./Restaurants.php">Restaurants</a></li>

        <li class="menu-item"><a class="menu-link" href="./Lebanon.php">Map of
Lebanon</a></li>

        <li class="menu-item"><a class="menu-link"
href="./Transportation.php">Transportation</a></li>

        <li class="menu-item"><a class="menu-link" href="./index.php">Sign Up/ Log
In</a></li>

    </ul>

</div>

<span><i class="bx bx-search search-toggle"></i></span>

<div class="search-block">

```

```

        <form class="search-form">

            <span><i class="bx bx-arrow-back search-cancel"></i></span>

            <input type="search" name="search" class="search-input" placeholder="Search
here...">

        </form>

        <div id="search-results"></div>

    </div>

</nav>

</header>

<?php
}

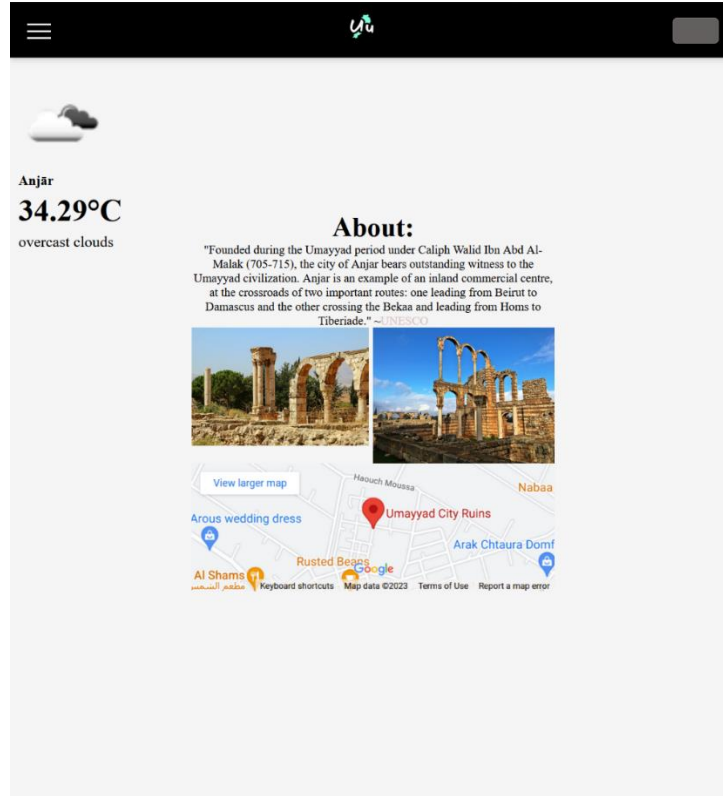
?>

```

The above is the code for the navbar, instead of using the same coded in each page, this function is being called to reduce the size of the files dramatically.

Tourism.php allows the user to click on the name of the site which leads them to their dedicated page as follows:

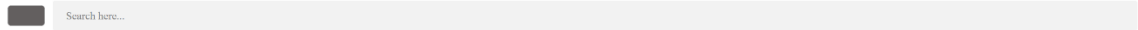
iii. Anjar.php:



This format is used by all the additional websites pages. The following web page presents images of the site taken by hand as well as its location relative to the map, the user can read an “about” usually extracted from UNESCO’s website, this page also has a weather information about that site; however, open-source information like these do not provide precipitation or chances of precipitations, thus take caution, and prepare in advance.

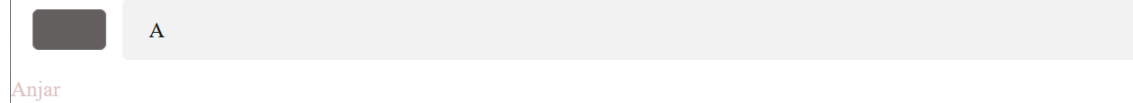


One might wonder: “What might this button do?”, this button is a button that expands into a search bar as so:



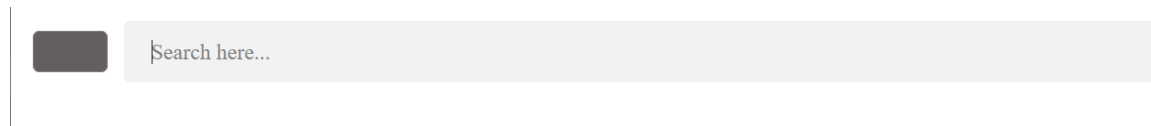
In this embedded page, the user can search for anything that exists in the whole site, before that, the input is cleaned and checked for any SQL injections or XSS attacks as so:

```
function sanitizeText(text) {  
    return text.replace(/&/g, '&amp;').replace(/</g, '&lt;').replace(/>/g, '&gt;').replace(/"/g, '&quot;').replace(/'/g, '&#x27;');  
}
```



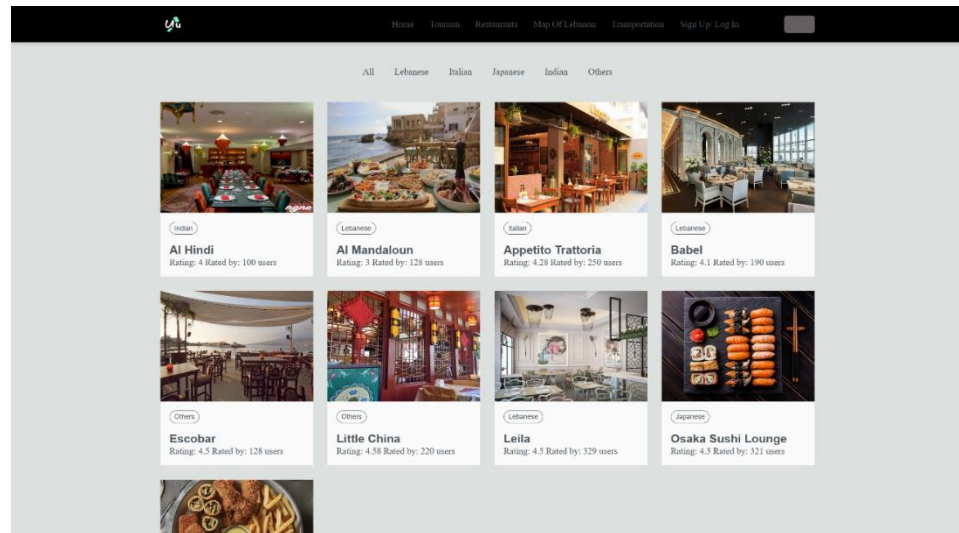
This is an example, later, this page would be capable of displaying an image within the search results, as well as a better orientation that can come out under the searchbar.

Ofcourse, if the searchbar is empty, it does not return anything



Clicking on the button again leads us back to the page it was opened from.

iv. Restaurant.php:



This php page is dedicated towards displaying restaurants on the front end. This php code is also dynamic, meaning it queries the database for all the restaurants available in the database and returns its name, rating, number of ratings, and type.

The code:

```
<?php

require_once "../navbar/navbar.php";

loadNavBar();

?>

<br><br><br><br>

<?php

require "../config/cfg.php";

?>

<section class="author-archive">
```

```
<div class="container">

  <input type="radio" id="All" name="categories" value="All" checked>

  <input type="radio" id="Lebanese" name="categories" value="Lebanese">

  <input type="radio" id="Italian" name="categories" value="Italian">

  <input type="radio" id="Japanese" name="categories" value="Japanese">

  <input type="radio" id="Indian" name="categories" value="Indian">

  <input type="radio" id="Others" name="categories" value="Others">

  <ol class="filters">

    <li>

      <label for="All">All</label>

    </li>

    <li>

      <label for="Lebanese">Lebanese</label>

    </li>

    <li>

      <label for="Italian">Italian</label>

    </li>

    <li>

      <label for="Japanese">Japanese</label>

    </li>

    <li>

      <label for="Indian">Indian</label>

    </li>
```

```

        </li>

        <label for="Others">Others</label>

    </li>

</ol>

<?php

try {

    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);

    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $pdo->query("SELECT NAME,Rating,NUMBER_OF_RATINGS,Type
FROM restaurants");

    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

?>

<ol class="posts">

    <?php

    if ($result) {

        foreach($result as $row) {

            $name = $row["NAME"];

            $ratings= $row["Rating"];

            $nmb_rating=$row["NUMBER_OF_RATINGS"];

            $type=$row["Type"];

            ?>

```



```

<li class="post" data-category=<?php echo ".$type?">

    <article>

        <figure>

            <?php echo'<a href="">?>

            <?php echo' '?>

            </a>

            <figcaption>

                <ol class="post-categories">

                    <li>

                        <a href=""><?php echo ".$type?"</a>

                    </li>

                </ol>

                <h2 class="post-title">

                    <a href="https://www.instagram.com/nasmabeyrout/?hl=en"
target="_blank"><?php echo ".$name ?>

                    </a>

                </h2>

                <?php

                    $floor_num = floor($ratings);

                    echo '<p> Rating: '.$ratings.' Rated by: '.$nmb_rating.' users</p>';?>

                </figcaption>

            </figure>

        </article>

```

```

        </li>

<?php }

} else {

    echo "0 results";

}

} catch(PDOException $e) {

    echo "Error: " . $e->getMessage();

}

$pdo = null;

?>

</ol>

</body>

<script src="../../JS/app.js"></script>

<script src="../../JS/SearchBar.js"></script>

<script>

function run(){

    const id=document.getElementById();

}

</script>

</html>

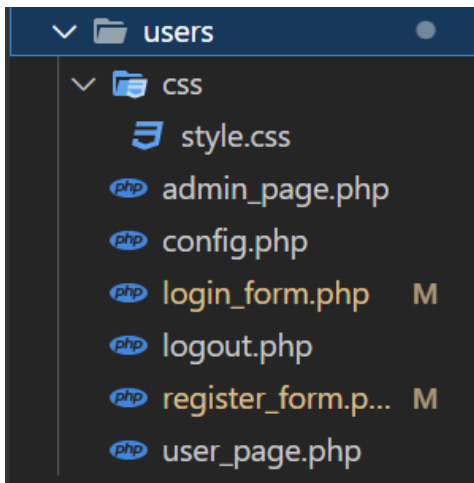
```

This php allows the filtering of data presented on the page, as well as viewing the rating of the website (extracted from google manually) as well as the number of ratings

(extracted from google manually). The users of this website after login are capable of rating the restaurant once, increasing/decreasing their rating according to their input provided when pressing on the RATE button.

v. **SignUp/ LogIn:**

Sign Up and Login is by far the largest file that based on these files:



The main file: register_form composes of sign up and login code that sends the results to another file, which sends the results to the database.

```
<?php
@include './config.php';

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

session_start();

function isEmailValid($email) {
```

```

return preg_match("/^[^\\s@]+@[^\\s@]+\\.([^\\s@]+)$/", $email);
}

function isPasswordValid($password) {

    return preg_match("/^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)[a-zA-Z\\d]{8,}$/", $password);
}

if(isset($_POST['submit'])){

    $current_time = date('Y-m-d H:i:s');

    $current_time2 = date('Y-m-d H:i:s');

    $name = mysqli_real_escape_string($conn, $_POST['name']);

    $email = mysqli_real_escape_string($conn, $_POST['email']);

    $pass = md5($_POST['password']);

    $cpass = md5($_POST['cpassword']);

    $user_type = $_POST['user_type'];

    $select = "SELECT * FROM users WHERE email = ? AND PASSWORD = ?";

    $stmt = $conn->prepare($select);

    $stmt->bind_param("ss", $email, $pass);

    $stmt->execute();

    $result = $stmt->get_result();

    if(mysqli_num_rows($result) > 0){

        $error[] = 'user already exists!';
    }
}

```

```

    } else {

        if($pass != $cpass){

            $error[] = 'password not matched!';

        } else if (!isEmailValid($email)) {

            $error[] = 'invalid email format!';

        } else if (!isPasswordValid($_POST['password'])) {

            $error[] = 'password must be at least 8 characters, one uppercase, one lowercase,
and one number!';

        } else {

            $insert = "INSERT INTO users(name, email, PASSWORD, user_type,
CREATION_DATE, MODIFICATION_DATE)

            VALUES('$name', '$email', '$pass', '$user_type', '$current_time', '$current_time2')";

            mysqli_query($conn, $insert);

            $user_select = "SELECT ID FROM users WHERE email = ? AND PASSWORD =
?";

            $stmt = $conn->prepare($user_select);

            $stmt->bind_param("ss", $email, $pass);

            $stmt->execute();

            $user_result = $stmt->get_result();

            $user_row = mysqli_fetch_assoc($user_result);

```

```

$user_id = $user_row['ID'];

$sites_select = "SELECT NAME FROM touristic_sites";

$sites_result = mysqli_query($conn, $sites_select);

while($site_row = mysqli_fetch_assoc($sites_result)){

    $site_name = $site_row['NAME'];

    $site_insert = "INSERT INTO visited_sites(USER_ID, site_name, VISITED)
VALUES('$user_id', '$site_name', 0)";

    mysqli_query($conn, $site_insert);

}

header('location:login_form.php');

}

}

};

?>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>register form</title>

    <link rel="stylesheet" href="css/style.css">

```

```

</head>

<body>

<div class="form-container">

    <form action="" method="post">

        <h3>register now</h3>

        <?php

            if(isset($error)){

                foreach($error as $error){

                    echo '<span class="error-msg">'.$error.'</span>';

                };

            };

        ?>

        <input type="text" name="name" required placeholder="enter your name">

        <input type="email" name="email" required placeholder="enter your email">

        <input type="password" name="password" required placeholder="enter your
password">

        <input type="password" name="cpassword" required placeholder="confirm your
password">

        <select name="user_type">

            <option value="user"></option>

            <option value="admin"></option>

        </select>

```

```

<input type="submit" name="submit" value="register now" class="form-btn">

<p>already have an account? <a href="login_form.php">login now</a></p>

</form>

</div>

</body>

</html>

```

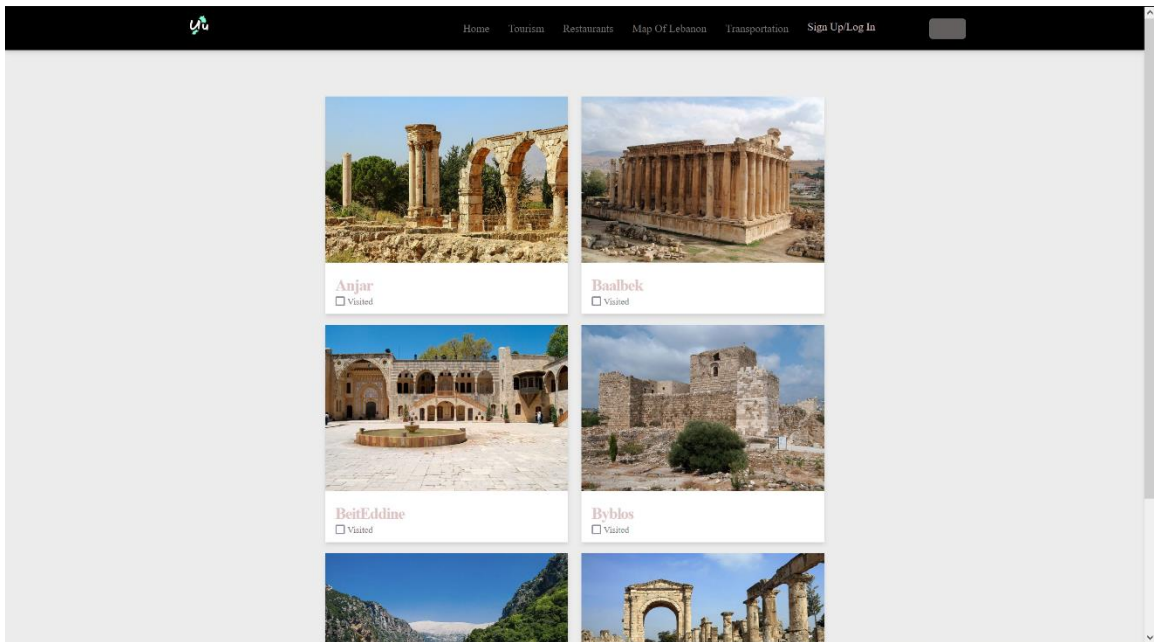
This code is a complex code that does the following:

- Create a user in the database with AES encryption based off PASSWORD function in mysql, when creating a user, this program also initializes many other databases by initializing visited_sites to have all the UserIDs and tourism names as the following:
- This code also prevents potential SQLi attacks and checks for email regex to verify h authenticity of the user.

VISITED	USER_ID	site_name
0	7	Anjar
0	7	Baalbek
0	7	BeitEddine
0	7	Byblos
0	7	Kadisha
0	7	Tyre
0	8	Anjar
0	8	Baalbek
0	8	BeitEddine
0	8	Byblos
0	8	Kadisha
0	8	Tyre

The following is 2 test users, these test users were created on the basis of testing purposes, this information is basically tied to the information of the logged in user. To explain this process, we need to see the page first.

vi. **Uni-fied/php/Tourism.php:**



This page might seem very similar to the not logged in user; however, this code is not similar.

This is the code:

```
<?php
require "../config/cfg.php";
try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->query("SELECT NAME FROM touristic_sites");
```

```

$result = $stmt->fetchAll(PDO::FETCH_ASSOC);

?>

<div class="cards">

<?php
if ($result) {

    foreach($result as $row) {

        $name = $row["NAME"];

        ?>

        <div class="card">

            <div class="card__image-holder">

                <?php echo'';?>

            </div>

            <div class="card-title">

                <h2>

                    <?php echo'<a href="/'.$name.'.php" target="_blank">'.$name.'</a>';?>

                </h2>

                <div>

                    <?php echo'<input type="checkbox" id="'.$name.'" name="'.$name.'"

onclick="run()">';?>

                    <?php echo'<label for="'.$name.'"><small>Visited</small></label>';?>

                </div>

```

```

        </div>

    </div>

    <?php }

    } else {

        echo "0 results";

    }

} catch(PDOException $e) {

    echo "Error: " . $e->getMessage();

}

$pdo = null;

?>

</div>

</body>

<script>

function run(){

    const id=document.getElementById();

    alert("You need to be logged in to mark a place as visited");

}

</script>

</html>

```

An extensive code for a simple website is what makes the experience great, this web application can update the database value VISITED (Which is a Boolean) to either 1 or 0 signifying that it is visited by the user.

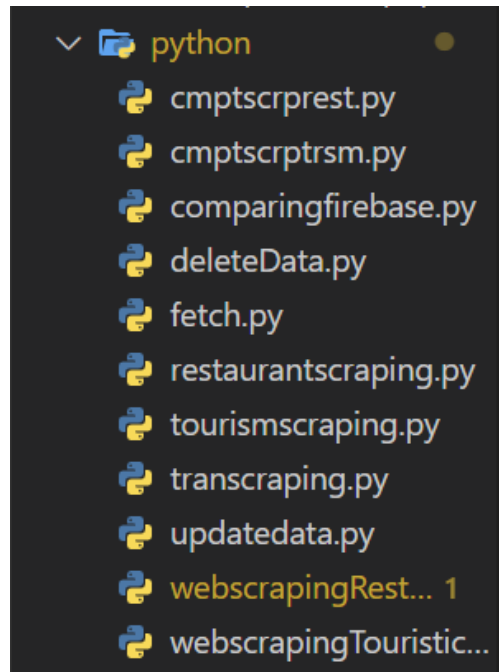
Code process:

1. Query the database to enable the user to create an account, as well as that, it needs to initialize the database: visited sites, everytime a user is created, it should query the database to use the username to fetch the ID, and it should create a series of data in visited sites to 0 where the columns are Full texts. → register_form.php
2. Extract Session: User_name (email) which is an AES encrypted plain text, decrypt it using AES as well to decrypt the hash and to use it later on.
3. This decrypted hash will be used to query the database users for the ID of the user currently logged in, this will return their ID. At the same time, the database queries the table: touristic_sites to display all the touristic_sites on the page. The ID of the user is used to retrieve all the VISITED information from the USER. This information is displayed in front of the user and viewed responsively and dynamically.
4. The checkbox is interacted by using an AJAX form as a future goal of this project: keep the database updated, remove users, and admin page.

All the other pages are similar to the ones in the not logged in user.

vii. Server-Side Automated Functions

Many of the functions of the web application are done automatically by a detector software done as a vendor option; however, for the sake of simplicity, it is important to mention these functions.



This is the folder that governs the automated functions on the server.

Cmptscrprest.py or Compute Scraping for Restaurant, is a python program that does web scraping every midnight for new restaurants in Lebanon. The rampant issue in Lebanon means that new companies and touristic sites don't surface frequently. If some do surface, it is important to know when they started:

```
import mysql.connector
import requests
from bs4 import BeautifulSoup
# Establish connection
cnx = mysql.connector.connect(user='root', password='',
                               host='localhost', database='users')
```

```

# Create cursor object

cursor = cnx.cursor()

# execute a query

query = "SELECT LAST_UPDATED,NAME FROM restaurants"

cursor.execute(query)

array=[]

# Iterate over results

for result in cursor:

    array.append(result)

    print(result)

query = 'Best restaurants in Lebanon'

num_results = 1

url = f'https://www.google.com/search?q={query}&num={num_results}'

headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'}

response = requests.get(url, headers=headers)

# Parse the HTML response using BeautifulSoup

soup = BeautifulSoup(response.content, 'xml')

# Locate the search results container

```

```

search_results = soup.find_all('div', class_='g')

# Extract the necessary data from each search result

urls = []

queries=[]

for result in search_results:

    # Extract the URL of the search result

    url = result.find('a')['href']

    # Check if the URL is a real link and add it to the list of URLs

    if url.startswith('http'):

        urls.append(url)

# Query each website and extract the last updated time

for url in urls:

    # Send a GET request to the website

    response = requests.get(url)

    # Parse the HTML response using BeautifulSoup

    soup = BeautifulSoup(response.content, 'xml')

    # Locate the last updated time, if available

    time_tag = soup.find('time')

```

```

if time_tag:

    last_updated_time = time_tag.text

    print(fURL: {url}, Last Updated Time: {last_updated_time}')

    queries.append(last_updated_time)

else:

    print(fURL: {url}, Last Updated Time not found')

for db_result, web_result in zip(array, queries):

    # Compare the last updated time values

    if db_result[0] != web_result:

        # Print the differences

        print(f"Name: {db_result[1]}, Last Updated (DB): {db_result[0]}, Last Updated
(Web): {web_result}")

cursor.close()

cnx.close()

```

This python code manually shows what are new, putting this file into an ipynb file allows the full execution of the code. Explaining the code however, this code is tied to mysql database as well as google search engine. This program queries the database for the name, and last updated information from restaurants, this information is compared to the information extracted from web scraping the results of the first 10 results of the first page of google. In this case, there are no new restaurants in Lebanon.

Cmptscrptrsm.py or Compute Scraping Tourism, is similar to Cmptscrprest.py but with touristic sites instead, running the code on an ipynb file shows the following:

```
.. URL: https://www.the961.com/touristic-sites-lebanon-this-summer/, Last  
   Name: Anjar, Last Updated (DB): 1980-01-01, Last Updated (Web): May 16,
```

The site shows that Anjar is newly updated since an initial time of 1980 (Which is an initialization time writes for all restaurants and touristic sites). Since the name is similar, there is no need to update the database.

Please ignore the other files, they are files to check for integrity of items in the database

New functionalities include a custom based recommendation algorithm unique to each user, this is the code:

```
import mysql.connector  
  
import smtplib  
  
from email.mime.multipart import MIMEMultipart  
from email.mime.base import MIMEBase  
from email.mime.text import MIMEText  
from email.utils import COMMASPACE  
from email import encoders  
  
import configparser  
  
config = configparser.ConfigParser()
```

```

# Connect to MySQL server

config.read('config.ini')

mydb = mysql.connector.connect(

    mysql_host = config['mysql']['host'],

    mysql_user = config['mysql']['user'],

    mysql_password = config['mysql']['password'],

    mysql_database = config['mysql']['database']

)

smtp_server = config['smtp']['server']

smtp_port = config['smtp']['port']

smtp_username = config['smtp']['username']

smtp_password = config['smtp']['password']

# Create cursor to execute queries

mycursor = mydb.cursor()

# Query 'users' table to get email and ID

mycursor.execute("SELECT email, ID FROM users")

# Fetch all rows of the query result

users = mycursor.fetchall()

# Loop through each user

for user in users:

```

```

# Query 'visited_sites' table to get VISITED value of each site_name for the user

mycursor.execute("SELECT site_name, VISITED FROM visited_sites WHERE
user_ID = %s", (user[1],))

visited_sites = mycursor.fetchall()

# Create a list of not visited sites

not_visited_sites = []

for site in visited_sites:

    if not site[1]:

        not_visited_sites.append(site[0])

# If the user has not visited any sites, send email with list of sites and a picture of each
site

if not_visited_sites:

    # Format email content

    email_content = f"Dear {user[0]},\n\nYou have not yet visited:\n"

    for site_name in not_visited_sites:

        email_content += f"- {site_name}\n"

    # Get the location of the visited site from the touristic_sites table

```

```

        mycursor.execute("SELECT location FROM touristic_sites WHERE name =
%s", (site_name,))

        location = mycursor.fetchone()[0]

        # Get all the sites that are close to the visited site from the touristic_sites table

        mycursor.execute("SELECT name FROM touristic_sites WHERE location = %s
AND name != %s", (location, site_name))

        nearby_sites = mycursor.fetchall()

        if nearby_sites:

            email_content += "You may also be interested in visiting these nearby sites:\n"

            for nearby_site in nearby_sites:

                email_content += f"- {nearby_site[0]}\n"

            email_content += "\nPlease take the opportunity to visit these sites. Otherwise,
please enjoy a picture of them:\n"

            # Send email with pictures of sites

            for site_name in not_visited_sites:

                email_content += f"{site_name}.jpg\n"

            # Create a MIME message object and add sender, recipient and subject

            msg = MIMEMultipart()

            msg['From'] = smtp_username

            msg['To'] = user[0]

```

```

msg['Subject'] = 'Sites to visit'

# Add text content to the message

text = MIMEText(email_content)

msg.attach(text)


# Add pictures of sites to the message

for site_name in not_visited_sites:

    filepath = f'../Heritage/{site_name}.jpg'

    with open(filepath, 'rb') as f:

        img = MIMEBase('application', 'octet-stream')

        img.set_payload(f.read())

        encoders.encode_base64(img)

        img.add_header('Content-Disposition', f'attachment;
filename="{site_name}.jpg"')

        msg.attach(img)


# Send the message using SMTP server

with smtplib.SMTP(smtp_server, smtp_port) as server:

    server.ehlo()

    server.starttls()

    server.ehlo()

```

```
server.login(smtp_username, smtp_password)

server.sendmail(smtp_username, user[0], msg.as_string())

print(f"Email sent to {user[0]}")
```

Explaining the code:

The script uses the following libraries: `mysql.connector` for connecting to a MySQL database, `smtplib` for sending emails using the Simple Mail Transfer Protocol (SMTP), and `email` for creating and formatting email messages.

The script reads configuration information from a `config.ini` file using the `configparser` library. The configuration file contains information such as the MySQL server hostname, username, password, and database name, as well as the SMTP server hostname, port number, username, and password.

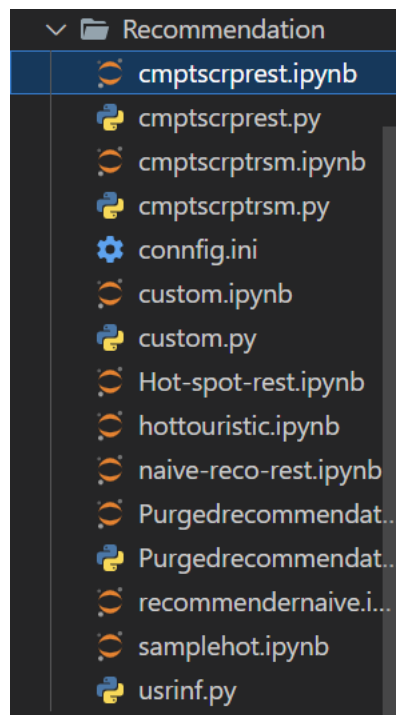
The script first connects to the MySQL database using the configuration information read from the file. It then queries the `users` table to get the email address and ID of each user. For each user, the script queries the `visited_sites` table to get the `VISITED` value of each site that the user has visited. If a user has

not visited any sites, the script creates a list of not visited sites, and for each site, it queries the `touristic_sites` table to get the location of the site and a list of nearby sites.

The script then creates an email message for each user containing the list of not visited sites and the pictures of those sites. It attaches the pictures to the email message using the `email.mime.base.MIMEBase` class. Finally, the script sends the email to the user using the SMTP server specified in the configuration file.

Overall, this script automates the process of sending personalized emails to users based on their visiting habits, encouraging them to visit the touristic sites they have not yet explored.

Recommendation systems:



This folder governs all the recommendations the server size computes, the following folder has 3 different recommendation algorithms that it follows:

1. Purged Type Recommendation, File: **naïve-reco.ipynb**

The Purged Type Recommendation basically means that there is not enough data and interaction in the database resulting in a naïve recommendation, the user will be recommended all the restaurants as well as all the touristic sites, the following recommendation is done via parsing through the relations databases and locating for presence of data inside it, if there is no data inside the relational database, the user will receive an email through SPMD which is an emailing system used by google's gmail noting them the sites available on the website.

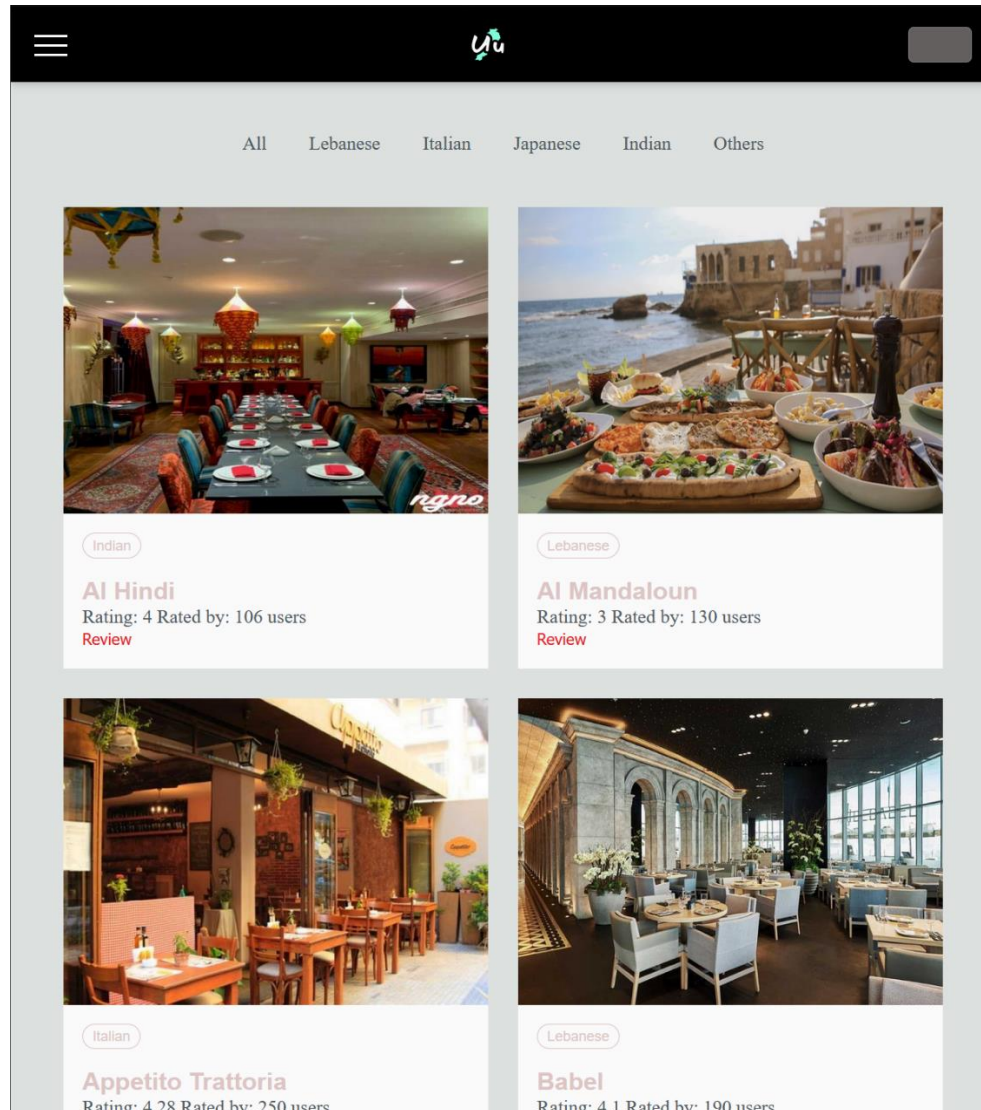
2. Nearest Neighbor, File: **nearest.ipynb**

The Nearest neighbor recommendation uses the fact that some restaurants are close to other restaurants, and some touristic sites close to another, thus if a user visited: Baalbek and not Anjar, they will be recommended to visit Anjar in this case since they know the area now and are familiar with the people as well.

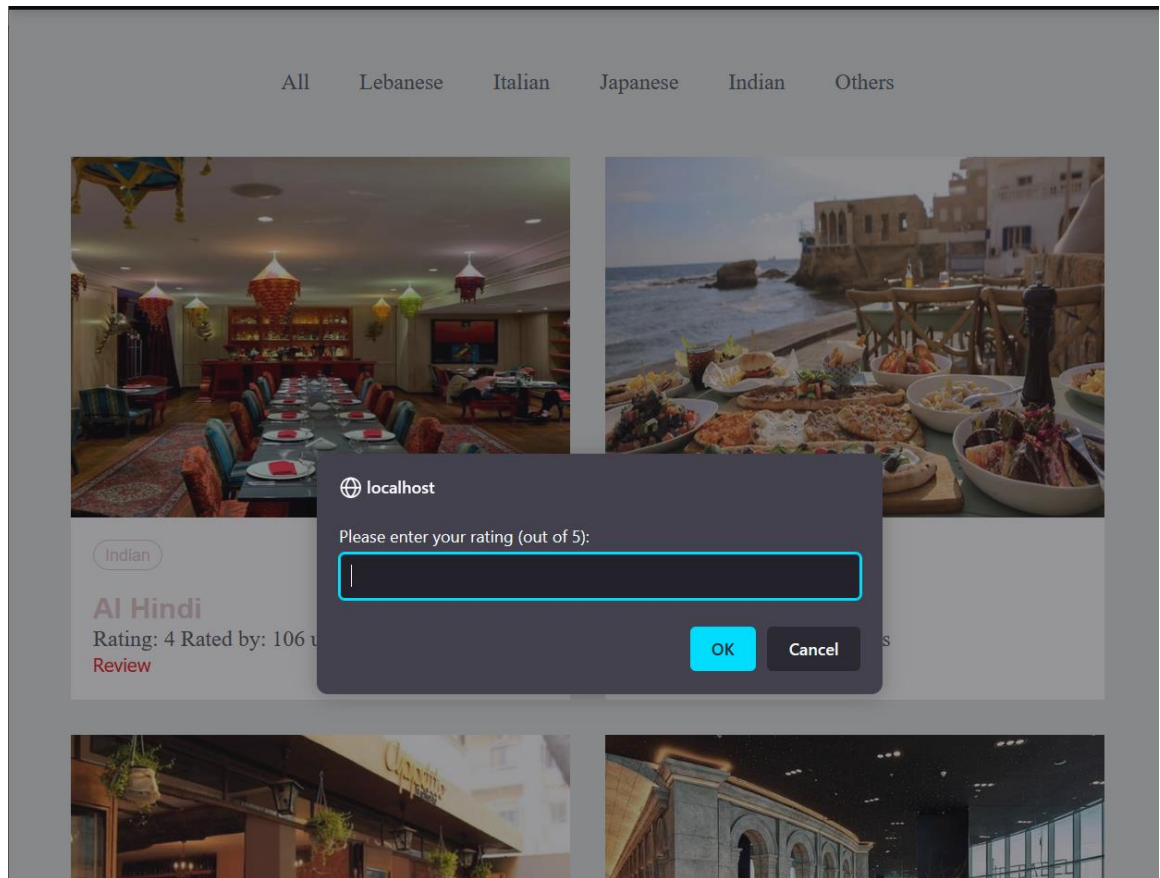
3. Hotspot, File: **Hot-spot.ipynb**

A hotspot is a restaurant/touristic site that is receiving so much interaction and high reviews, that you cannot but recommend it to new users, the following recommendation is done using SPMD as well

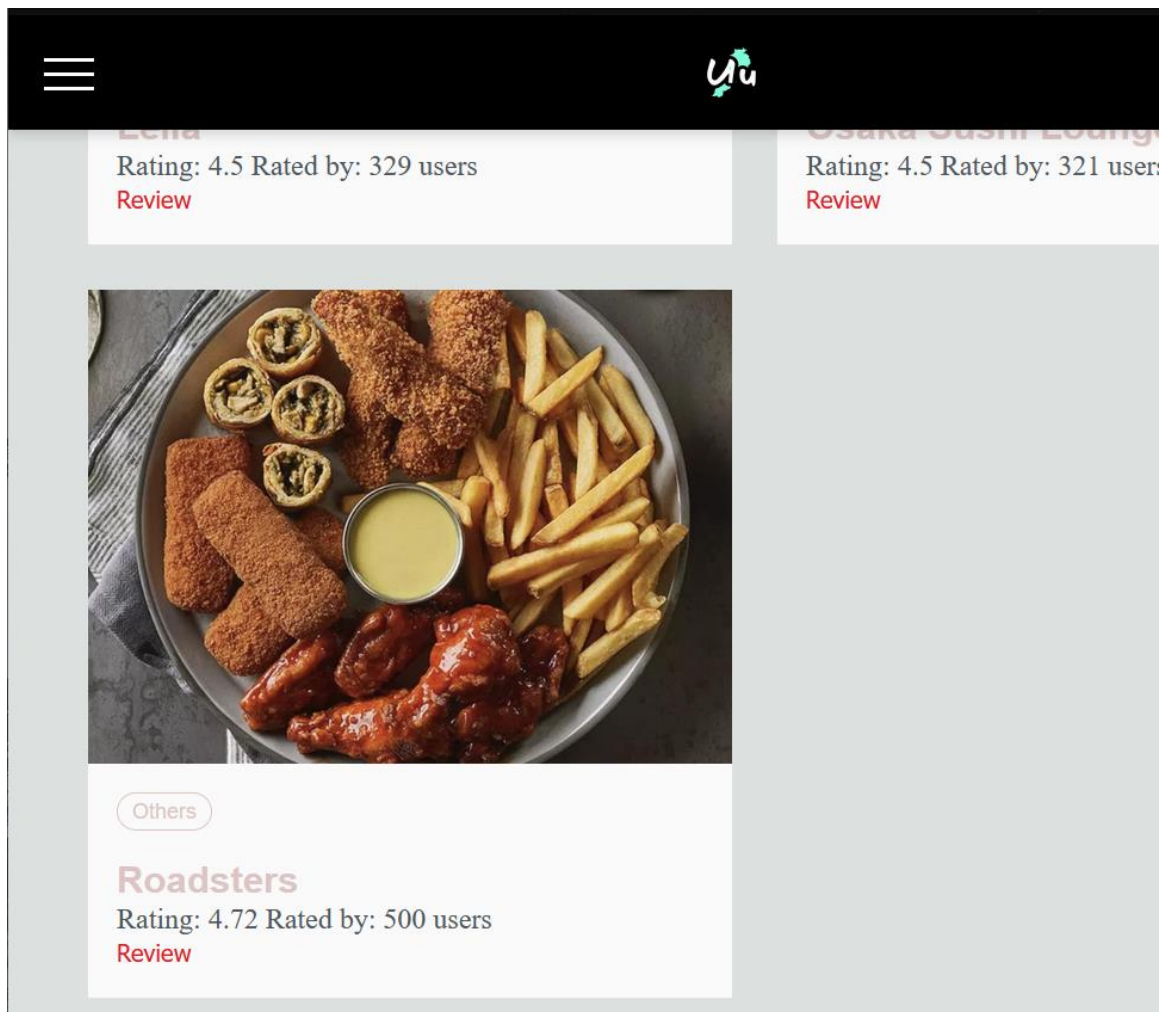
viii. Uni-fied/php/restaurants.php



A logged-in user has the capability to review a restaurant he/she has visited once and only once, the user will be pressing on Review and the following will show up:



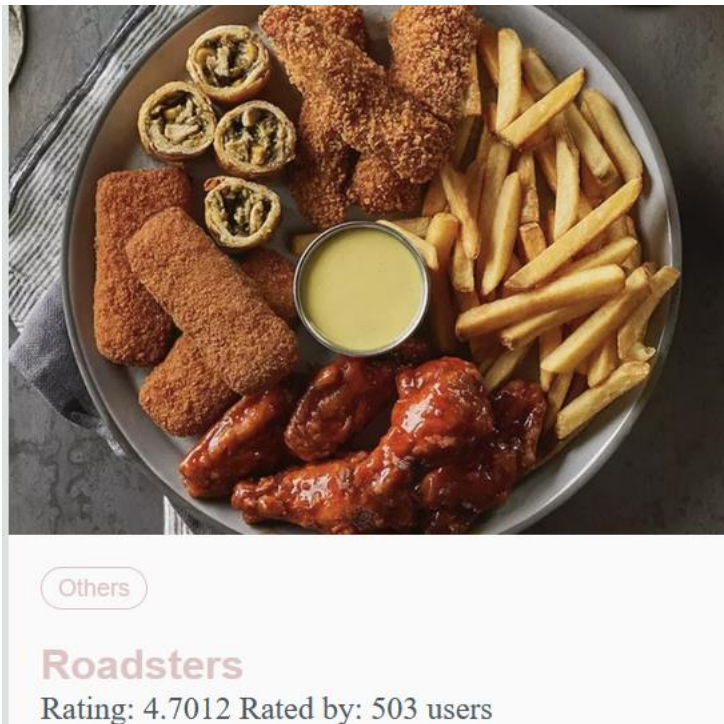
A pop up on the screen will show up and it will query the user to input a value between 1 and 5 and wil update the database, since Al Hindi is already updated by the user signed in, it wont be updated again, lets update another restaurant



Let's update this restaurant by clicking on Review

A screenshot of a rating dialog box. The title bar says 'localhost'. The text inside says 'Please enter your rating (out of 5):'. Below this is a text input field containing the number '5'. At the bottom right, there are two buttons: 'OK' (blue) and 'Cancel' (grey).

I am going to give it a rating of 5 to see what happens



The Application is well updated, now since the application is online and being updated multiple times, results may vary a lot from time to time.

The code:

```
<?php

if (session_status() == PHP_SESSION_NONE) {

    session_start();

}

if(!isset($_SESSION["user_name"])){

    header("Location: ../index.php");

    exit();

}

$usrnm = $_SESSION["user_name"];
```

```

?>

<script>

function review(name) {

var rating = prompt("Please enter your rating (out of 5):");

if (rating != null && rating != "" && !isNaN(rating)) {

rating = Math.min(Math.max(parseInt(rating), 1), 5);

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function() {

if (xhr.readyState == 4 && xhr.status == 200) {

var result = JSON.parse(xhr.responseText);

alert(result.message);

}

};

xhr.open("POST", "review.php", true);

xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

xhr.send("name=" + name + "&rating=" + rating);

}

}

</script>

<body>

<!-- Start of HTML and NavBar-->

```

```

<?php

require_once "../navbar/logedinbar.php";

loadNavBar();

?>

<br><br><br><br>

<?php

require "../config/cfg.php";

?>

<section class="author-archive">

    <!-- HTML -->

    <div class="container">

        <!--Defining set elements-->

        <input type="radio" id="All" name="categories" value="All" checked>

        <input type="radio" id="Lebanese" name="categories" value="Lebanese">

        <input type="radio" id="Italian" name="categories" value="Italian">

        <input type="radio" id="Japanese" name="categories" value="Japanese">

        <input type="radio" id="Indian" name="categories" value="Indian">

        <input type="radio" id="Others" name="categories" value="Others">

        <!-- Enabling filtering and how to define them and display them-->

        <ol class="filters">

            <li>

                <label for="All">All</label>

            </li>

```

```

        <li>

            <label for="Lebanese">Lebanese</label>

        </li>

        <li>

            <label for="Italian">Italian</label>

        </li>

        <li>

            <label for="Japanese">Japanese</label>

        </li>

        <li>

            <label for="Indian">Indian</label>

        </li>

        <li>

            <label for="Others">Others</label>

        </li>

    </ol>

<?php
try {

    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);

    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $pdo->query("SELECT NAME,Rating,NUMBER_OF_RATINGS,Type
FROM restaurants");

    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

```

```

?>

<ol class="posts">

  <?php

    if ($result) {

      foreach($result as $row) {

        $name = $row["NAME"];

        $ratings= $row["Rating"];

        $nmb_rating=$row["NUMBER_OF_RATINGS"];

        $type=$row["Type"];

        ?>

        <li class="post" data-category=<?php echo ".$type?">

          <!--1024x740!-->

          <article>

            <figure>

              <?php echo'<a href=f="'.$name.'.php" target="_blank">'?>

              <?php echo' '?>

            </a>

            <figcaption>

              <ol class="post-categories">

                <li>

                  <a href=""><?php echo ".$type?"</a>

                </li>

              </ol>

```



```

        <h2 class="post-title">

        <?php echo ' <a href=./'. $name. '.php target="_blank">';?><?php
echo". $name ?>

        </a>

        </h2>

        <?php

        $floor_num = floor($ratings);

        echo ' <p> Rating: ' . $ratings. ' Rated by: ' . $nmb_rating. ' users</p>';?>

        <button onclick="review('<?php echo $name; ?>')">Review</button>

        </figcaption>

        </figure>

        </article>

    </li>

    <?php }

    } else {

        echo "0 results";

    }

} catch(PDOException $e) {

    echo "Error: " . $e->getMessage();

}

$pdo = null;

?>

</ol>

```

```

</body>

<script src="../../JS/app.js"></script>

<script src="../../JS/SearchBar.min.js"></script>

<script>

function run(){

    const id=document.getElementById();

}

</script>

</html>

```

Review.php:

```

<?php

$user_name = $_SESSION['user_name'];

// Connect to the database

$db = new mysqli('localhost', 'root', '', 'users');

// Check for errors

if ($db->connect_errno) {

    die('Failed to connect to database: ' . $db->connect_error);

}

if ($_SERVER["REQUEST_METHOD"] == "POST") {

    require_once "../config/cfg.php";

    $name = $_POST["name"];

    $rating = $_POST["rating"];

```

```

// Check if the rating value is valid

if (!is_numeric($rating) || $rating < 1 || $rating > 5) {

    $response = array("status" => "error", "message" => "Invalid rating value. Please enter
a value between 1 and 5.");

    echo json_encode($response);

    exit;

}

try {

    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);

    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $pdo->prepare("SELECT Rating, NUMBER_OF_RATINGS FROM
restaurants WHERE NAME = ?");

    $stmt->execute([$name]);

    $result = $stmt->fetch(PDO::FETCH_ASSOC);

    if ($result) {

        // Get the user ID from the database

        $user_query = "SELECT ID FROM users WHERE name = '$user_name'";

        $user_result = $db->query($user_query);

        $user_row = $user_result->fetch_assoc();

        $user_id = $user_row['ID'];

        // Check if the user has already rated this restaurant

        $check_query = "SELECT * FROM rated_restaurants WHERE NAME = ? AND
USER_ID = ?";

```

```

$stmt_check = $db->prepare($check_query);

$stmt_check->bind_param("si", $name, $user_id);

$stmt_check->execute();

$check_result = $stmt_check->get_result(); $check_result = $db->query($check_query);

if ($check_result->num_rows > 0) {

    $response = array("status" => "error", "message" => "You have already rated this
restaurant.");

    echo "<script>alert('You have already rated this restaurant.');

```

```

    } else {

        $response = array("status" => "success", "message" => "Review added
successfully.");

        echo "<script>alert('Review added successfully.');

```

Explaining Restaurants.php code:

The first block of code checks if a PHP session is already started, and if not, starts a new session. Then, it checks if the session has a variable named "user_name" set. If not, it redirects the user to the index.php page and exits the script.

1. The next section contains HTML code for the web page, including a navbar, a list of restaurants, and a search bar.
2. The PHP code starts with a try-catch block that connects to a database using PDO and queries a table named "restaurants" to get information about each restaurant. It then loops through the results and displays the information for each restaurant in a list item.
3. For each restaurant, the PHP code displays the name, image, category, rating, and the number of users that have rated it. It also displays a "Review" button for each restaurant that calls a JavaScript function named "review" and passes the restaurant name as a parameter.
4. The "review" function prompts the user to enter a rating (out of 5) for the restaurant. It then sends an AJAX request to the "review.php" file with the restaurant name and rating as parameters. If the request is successful, it displays a message to the user with the response from the server.
5. The HTML code also includes a set of radio buttons for filtering the list of restaurants by category. It uses JavaScript to apply the filter and display only the restaurants that match the selected category.

Explaining review.php:

This PHP code is a script that handles the review form submission for a restaurant rating system.

The first few lines retrieve the user name from the session and create a connection to the database. Then, it checks if the request method is POST and if it is, it retrieves the name and rating values from the form using the `$_POST` superglobal variable.

The code then checks if the rating value is valid by ensuring that it is a number between 1 and 5. If it is not valid, an error response is returned in JSON format and the script exits.

If the rating is valid, the script attempts to establish a PDO database connection and retrieve the current rating and number of ratings for the restaurant specified in the form. If the restaurant is not found in the database, an error response is returned.

If the restaurant is found, the script checks if the user has already rated the restaurant by querying the `rated_restaurants` table using the user ID and restaurant name. If the user has already rated the restaurant, an error response is returned and the script exits.

If the user has not rated the restaurant, the script calculates the new rating by taking the current rating and number of ratings, adding the new rating, and

dividing by the new number of ratings. The restaurants table is updated with the new rating and number of ratings, and the user's rating is inserted into the rated_restaurants table. If the insertion is successful, a success response is returned in JSON format and an alert is displayed in the browser. If the insertion fails, a database error message is returned.

Finally, the script closes the database connection and exits.

ix. Events.php

Events.php is a relatively new idea that does the following:

```
<header class="header" id="header">

  <nav class="navbar container">

    <a href="../index.php" class="brand"></a>

    <div class="burger" id="burger">

      <span class="burger-line"></span>

      <span class="burger-line"></span>

      <span class="burger-line"></span>

    </div>

    <span class="overlay"></span>

    <div class="menu" id="menu">

      <ul class="menu-inner">

        <li class="menu-item"><a class="menu-link"
href="../index.php">Home</a></li>
```



```

        <li class="menu-item"><a class="menu-link"
href="/Tourism.php">Tourism</a></li>

        <li class="menu-item"><a class="menu-link"
href="/Restaurants.php">Restaurants</a></li>

        <li class="menu-item"><a class="menu-link" href="/Lebanon.php">Map of
Lebanon</a></li>

        <li class="menu-item"><a class="menu-link"
href="/Transportation.php">Transportation</a></li>

        <li class="menu-item"><a class="menu-link"
href="/users/register_form.php">Sign Up/Log In</a></li>
    </ul>
</div>

<span><i class="bx bx-search search-toggle"></i></span>

<div class="search-block">

    <form class="search-form">

        <span><i class="bx bx-arrow-back search-cancel"></i></span>

        <input type="search" name="search" class="search-input" placeholder="Search
here...">

    </form>

    <div id="search-results"></div>

</div>

</nav>

```

```

</header>

<script src="../JS/SearchBar.min.js"></script>

<br>

<table>

  <thead>

    <tr>

      <th>Month</th>

      <th>Dates</th>

      <th>Name</th>

      <th>Description</th>

    </tr>

  </thead>

  <tbody>

    <?php

      include('simple_html_dom.php');

      $url = 'https://www.ticketingboxoffice.com/';

      $html = file_get_contents($url);

      $html = str_get_html($html); // Updated to use str_get_html function

      $events_array = array(); // Initialize the array outside of the foreach loop

```

```

foreach ($html->find('div.containerMix') as $event) { // Changed the variable name to
$event to be more descriptive

$event_data = array();

$event_data['Fday'] = $event->find('.Fday', 0) ? $event->find('.Fday', 0)->plaintext : "";

$event_data['FMonth'] = $event->find('.FMonth', 0) ? $event->find('.FMonth', 0)-
>plaintext : "";

$event_data['FMonth2'] = $event->find('.FMonth2.mb5M', 0) ? $event-
>find('.FMonth2.mb5M', 0)->plaintext : "";

$event_data['eventFTitle'] = $event->find('.eventFTitle', 0) ? $event->find('.eventFTitle',
0)->plaintext : "";

$event_data['eventDdesc'] = $event->find('.eventDdesc', 0) ? $event->find('.eventDdesc',
0)->plaintext : "";

$events_array[] = $event_data;
}

$html->clear(); // Free up memory by clearing the HTML object
unset($html); // Unset the HTML object variable
?>

<?php foreach ($events_array as $event) : ?>

    <tr>

        <td><?php echo $event['FMonth']; ?></td>

        <td><?php echo $event['Fday']; ?> <?php echo $event['FMonth2']; ?></td>

```

```
<td><?php echo $event['eventFTitle']; ?></td>

<td><?php echo $event['eventDdesc']; ?></td>

</tr>

<?php endforeach; ?>

</tbody>

</table>

</body>

</html>
```

Explaining the code:

This code is a webscraping script that does the following:

- Use Virgin Megastore's Ticketing Box Office to query the events available of their website using php.
- The <script> tag contains JavaScript code that sets the Content-Security-Policy header. This header helps to prevent cross-site scripting attacks by specifying which resources are allowed to be loaded from which origins.

- The scraping:

The first line includes a PHP file called "simple_html_dom.php" which provides functions to parse and manipulate HTML documents.

The second line defines the URL of the webpage to be scraped.

The third line retrieves the HTML content of the webpage using the PHP function `file_get_contents`.

The fourth line creates an instance of `simple_html_dom` and passes the HTML content retrieved in step 3 to the `str_get_html` function which converts the HTML content into a `simple_html_dom` object.

The fifth line initializes an empty array called `$events_array` to hold the data of each event.

The sixth line starts a `foreach` loop to iterate through each `div` element on the webpage with a class of "containerMix". For each element, it extracts the relevant data using `simple_html_dom` functions and saves it in an associative array called `$event_data`.

The last line within the `foreach` loop appends the `$event_data` array to the `$events_array` array.

The `$html` object is cleared and the memory allocated to it is released to avoid memory leaks.

The events data is displayed in an HTML table using a `foreach` loop that iterates through each event in the `$events_array` array and displays it in a table row.

Image:



Title	Date	Location	Image	Day	Month	Time	Title	-	Name	Description
	10			May		21:45	Beirut Club vs Zobahan Club		Beirut Club vs Zobahan Club	

The code outputs the above as result.

VIII. Future Prospects:

The Future Prospects of this capstone project is to take this web application further by adding a better interaction with the site. Currently the site is up and running on Wamp, and the next move is to host it online on a paid or a free hosting service, the easiest thing to do is to export the database using SQL queries to 000webhost (which is the current deployment website) and to change the configuration to 000webhost instead. The images are generated statically, downloading the images is necessary to keep the website online and accessible, a website with images that don't lead anywhere is a failed website. This leads to more problems however, if the image does not exist, and the web scraping finds new restaurants in Lebanon, it needs access to that image as well as the rating, and number of ratings.

Currently, the checkbox method is the most complex method of updating the VISITED section of visited_sites/ sites_visited, if the checkbox is not working, updating the database may not work. Sometimes however, the database updates, but the checkbox remains the same, problems like these may be due to multiple users creating and logging into the same website at the same laptop/desktop/device, a dynamic session is necessary to know exactly which user_id to grab at the time. Another solution is to add a up_time for the users database to know exactly who is logged in and from where and when.

Another suggestion for the website is to add Lebanese Culture as well as hotels, with each having their own mechanisms and functionalities. Other suggestions include adding more events to the page using methods of python instead of php since php was not made for web scraping, even if I were to download the dom parser function and use it with php, it wouldn't be better than to use python; however, I will need to convert the whole application to a framework that suits machine learning such as Django which is a whole new type of framework that supports web development, and that I am not familiar with at all.

Another idea is to recommend another site on the spot, when selecting and successfully updating the database, the site will display a pop up that you successfully visited the site/ rated the restaurant, and it will recommend other sites under it directly, to do that, we must change our entire project to another framework that is made for Machine Learning which is Django in this case.

Hope you enjoyed my project and report.

IX. References

Lebanon Tourism Statistics 1995-2023. (n.d.). MacroTrends.

<https://www.macrotrends.net/countries/LBN/lebanon/tourism-statistics>

What is an API? (Application Programming Interface) / MuleSoft. (n.d.). MuleSoft.

<https://www.mulesoft.com/resources/api/what-is-an-api>