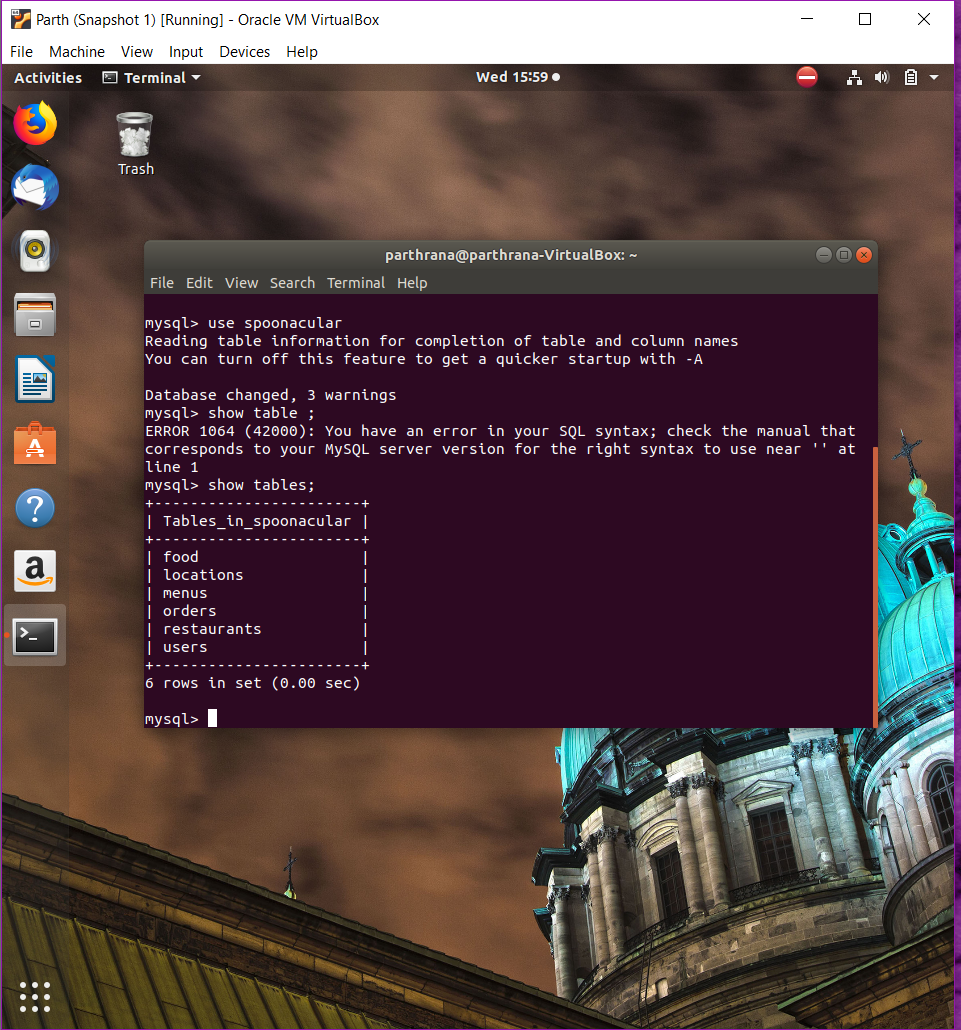
# Spoonacular

## Database

The database by the name ‘spoonacular’ has four tables; food, locations, menus, orders, restaurants and users.

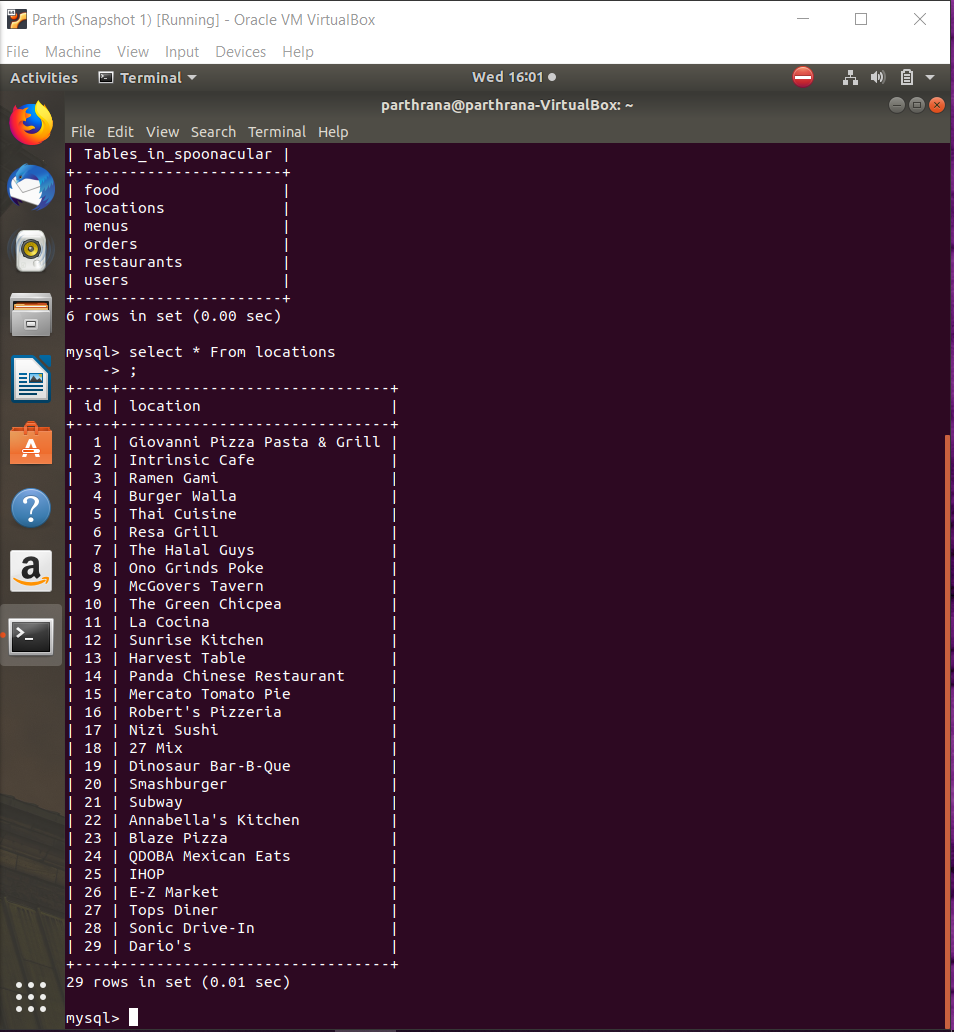
Food

Has the following columns; id, username, food, created\_at. id is the primary key. Saves favourite food for each particular user.



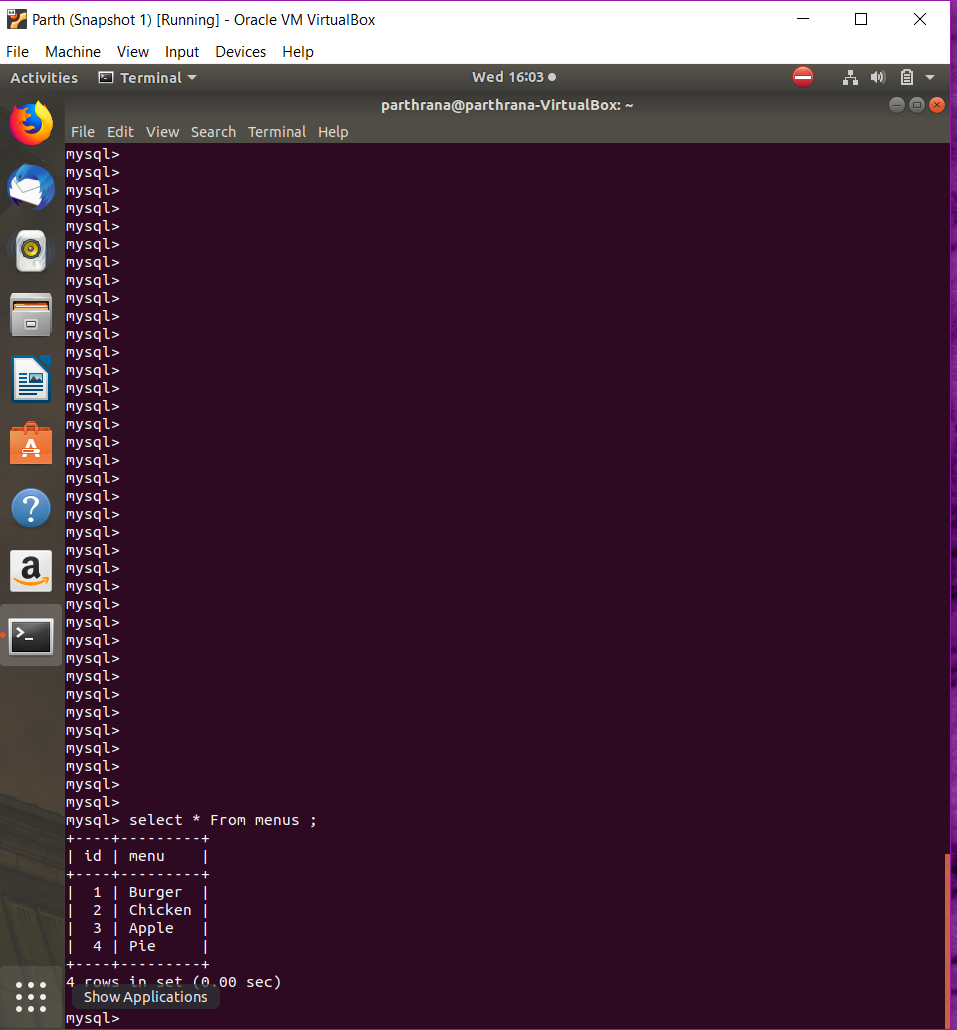
Locations

Has id and location as columns id is the primary key. Saves locations details.



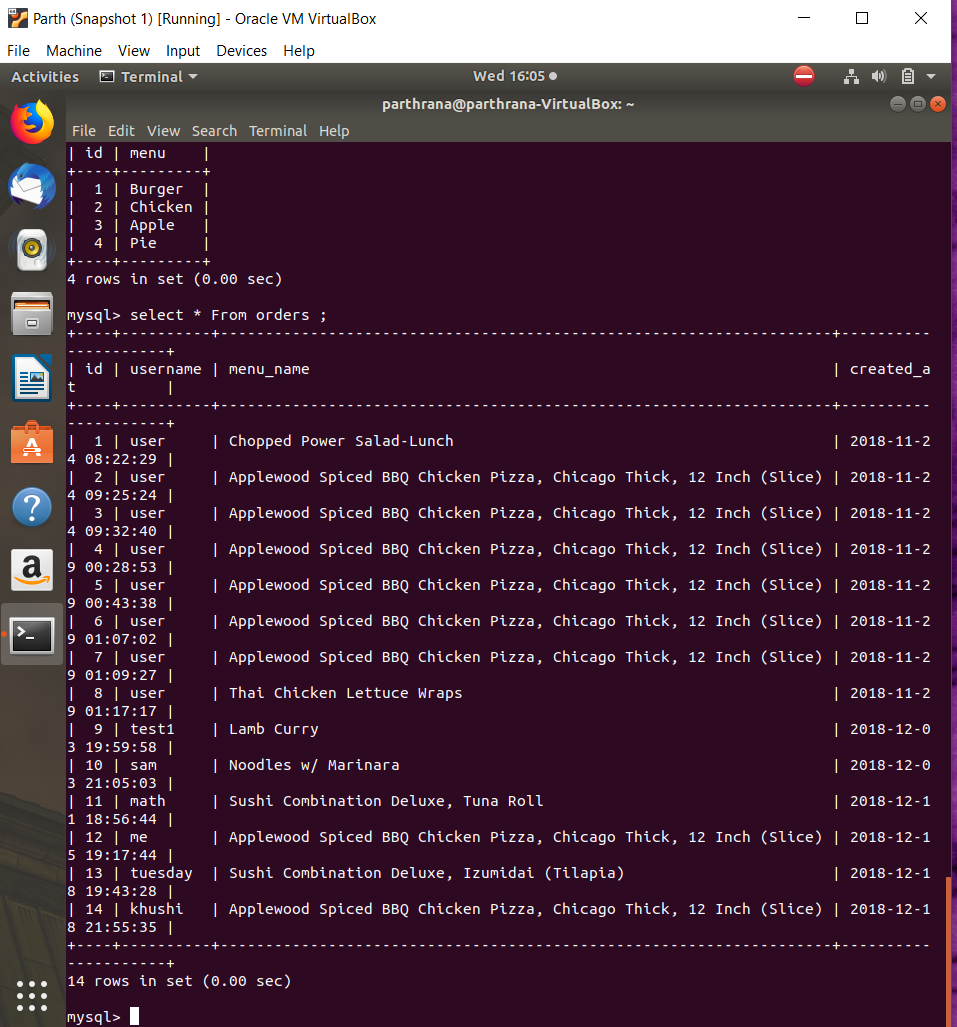
Menus

Has fields; id and menu as columns id is the primary key. Saves menus which is basically used to pull specific menu type from the Spoonacular API.



Orders

Has fields; Id, username, menu\_name, created\_at are its columns. id is the primary key. This saves details related to particular customer/user orders.

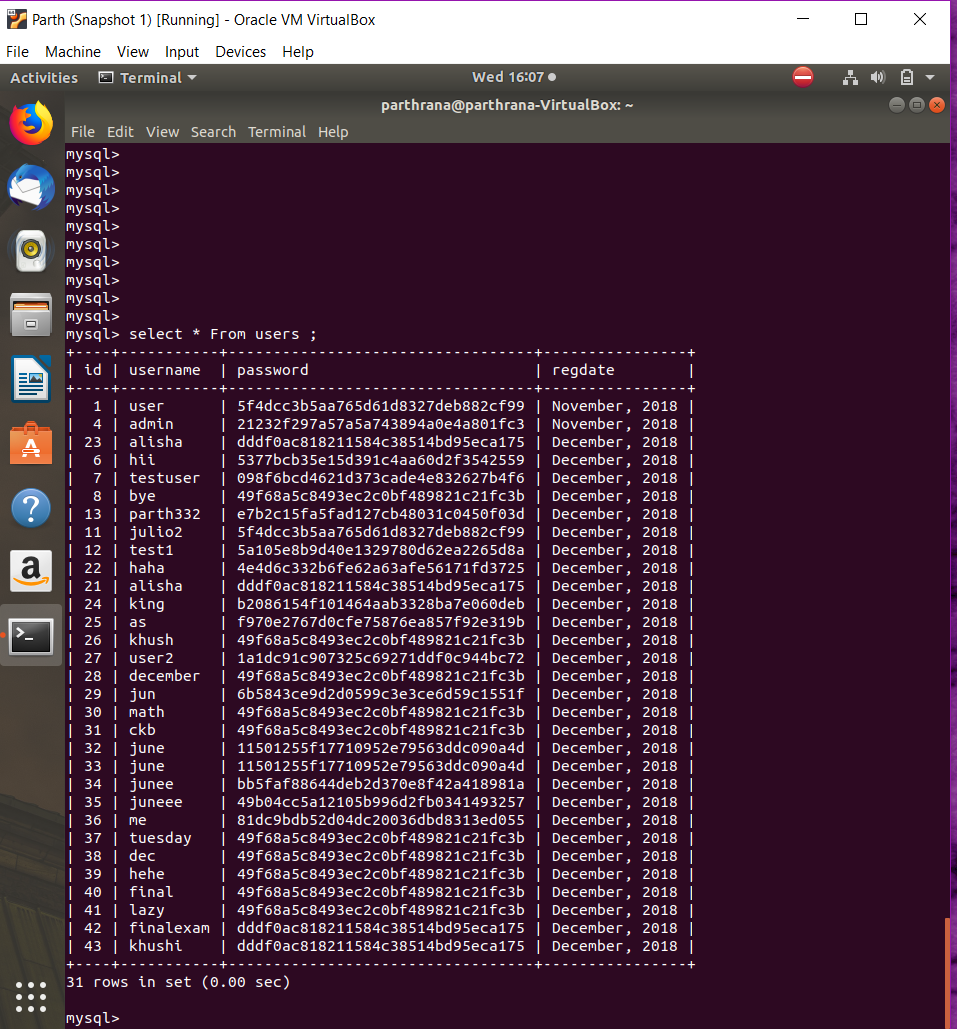


Restaurants

Has fields; Id, username, restaurant and created. id is the primary key. Used to save favourite restaurants for a particular user.

Users

Has fields; Id username, password & regdate are the columns. To save user data which is used to authenticate the users to the system.



## Spoonacular API.

Using the api key and a menu name, the list of similar menu types are fetched giving out a json of each menu details that are then output on the html page. The list menu is limited to ‘search word’ applied. Below is the Javascript code that interact with the API. For the API to work the server hosting the system must be connected to the internet.

function getMenuItem($item){

require\_once "./vendor/autoload.php";

// Disables SSL cert validation temporary

Unirest\Request::verifyPeer(false);

$response = Unirest\Request::get("https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/food/menuItems/search?query=".$item."&offset=0&number=10&minCalories=0&maxCalories=5000&minProtein=0&maxProtein=100&minFat=0&maxFat=100&minCarbs=0&maxCarbs=100",

array(

"X-Mashape-Key" => $GLOBALS['key'],

"X-Mashape-Host" => "spoonacular-recipe-food-nutrition-v1.p.rapidapi.com"

)

);

return $response->body->menuItems;

}

function searchMenu($query){

require\_once "./vendor/autoload.php";

// Disables SSL cert validation temporary

Unirest\Request::verifyPeer(false);

//fetch data

$response = Unirest\Request::get("https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/food/menuItems/suggest?query=".$query."&number=5",

array(

"X-Mashape-Key" => $GLOBALS['key'],

"X-Mashape-Host" => "spoonacular-recipe-food-nutrition-v1.p.rapidapi.com"

)

);

return $response->body->results;

}

function getMenuInfo($id){

// Disables SSL cert validation temporary

Unirest\Request::verifyPeer(false);

//fetch data

$response = Unirest\Request::get("https://spoonacular-recipe-food-nutrition-v1.p.rapidapi.com/food/menuItems/".$id,

array(

"X-Mashape-Key" => "ZbwYt8rrzGmshEmlKAsALgc4U1Q5p1p33UKjsnO2MXpmdT4w8H",

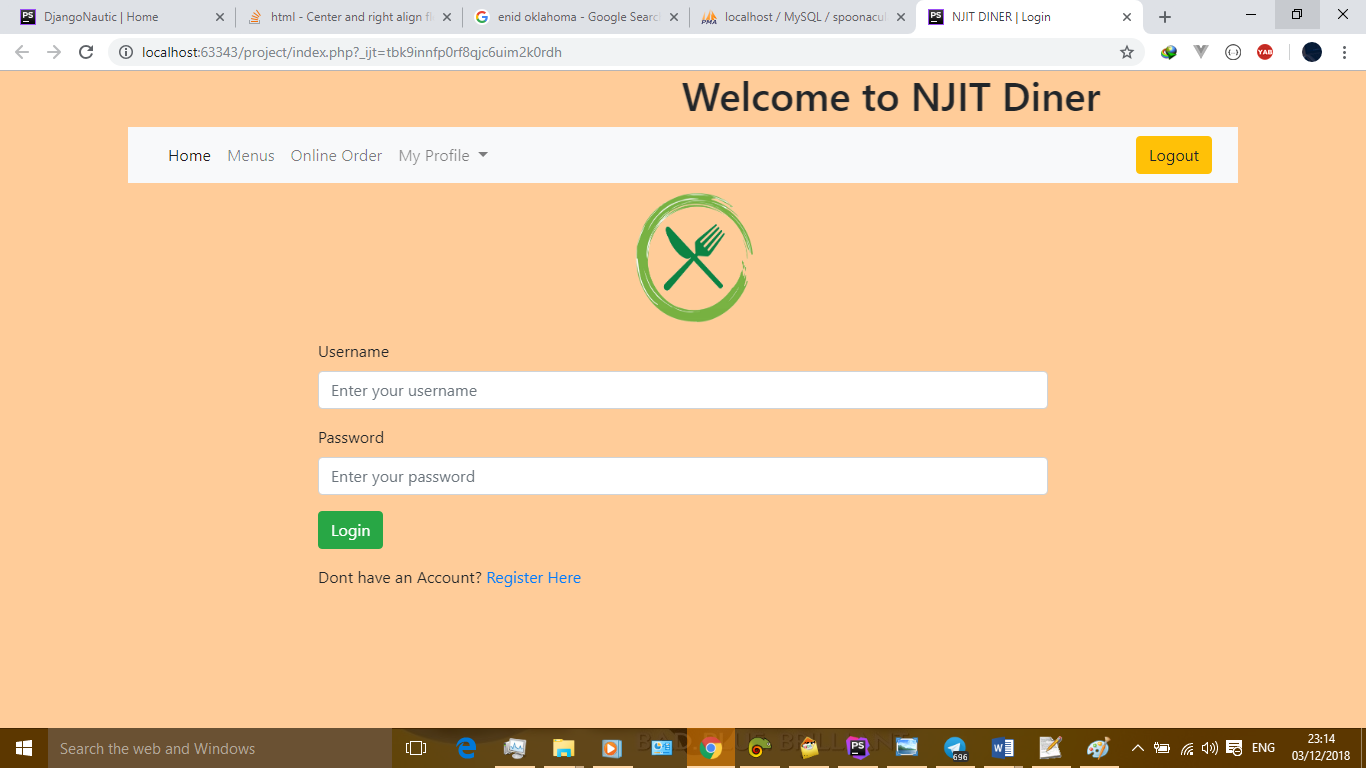
"X-Mashape-Host" => "spoonacular-recipe-food-nutrition-v1.p.rapidapi.com"

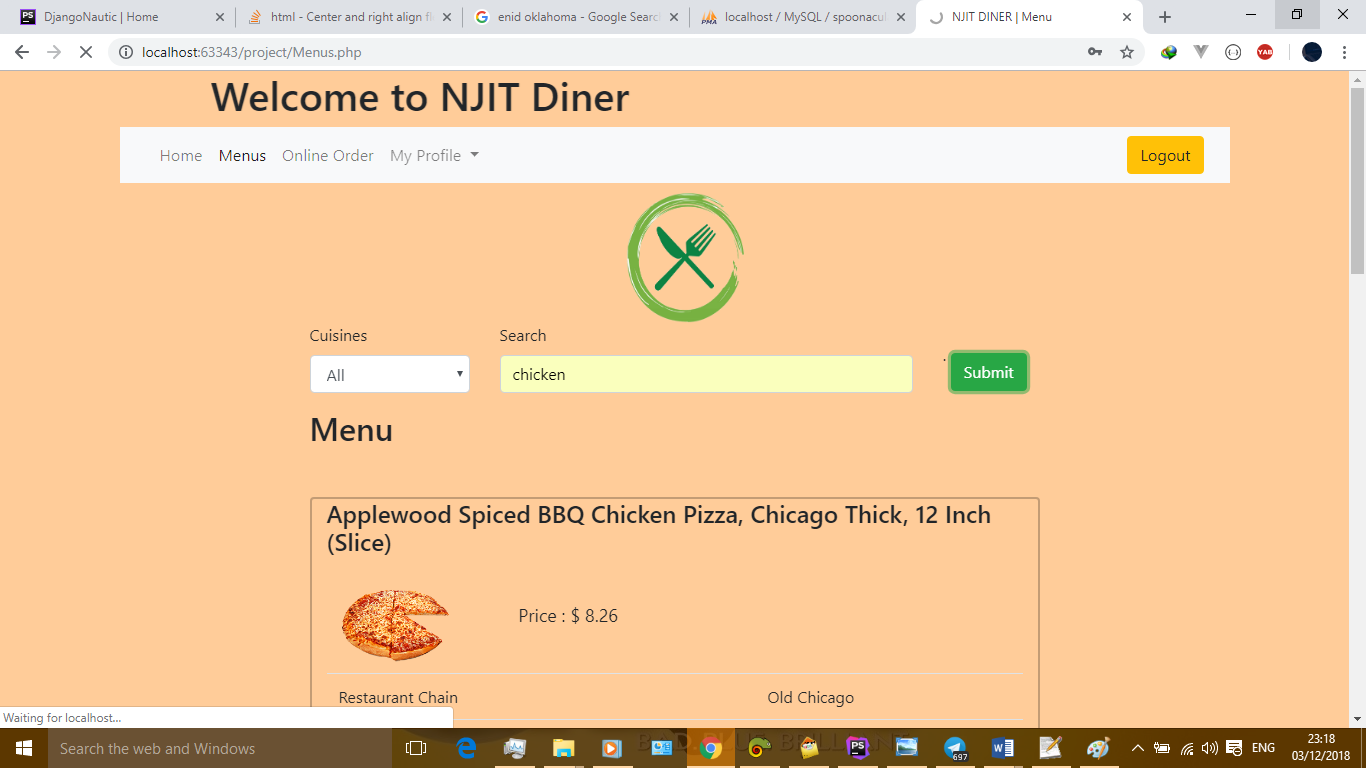
)

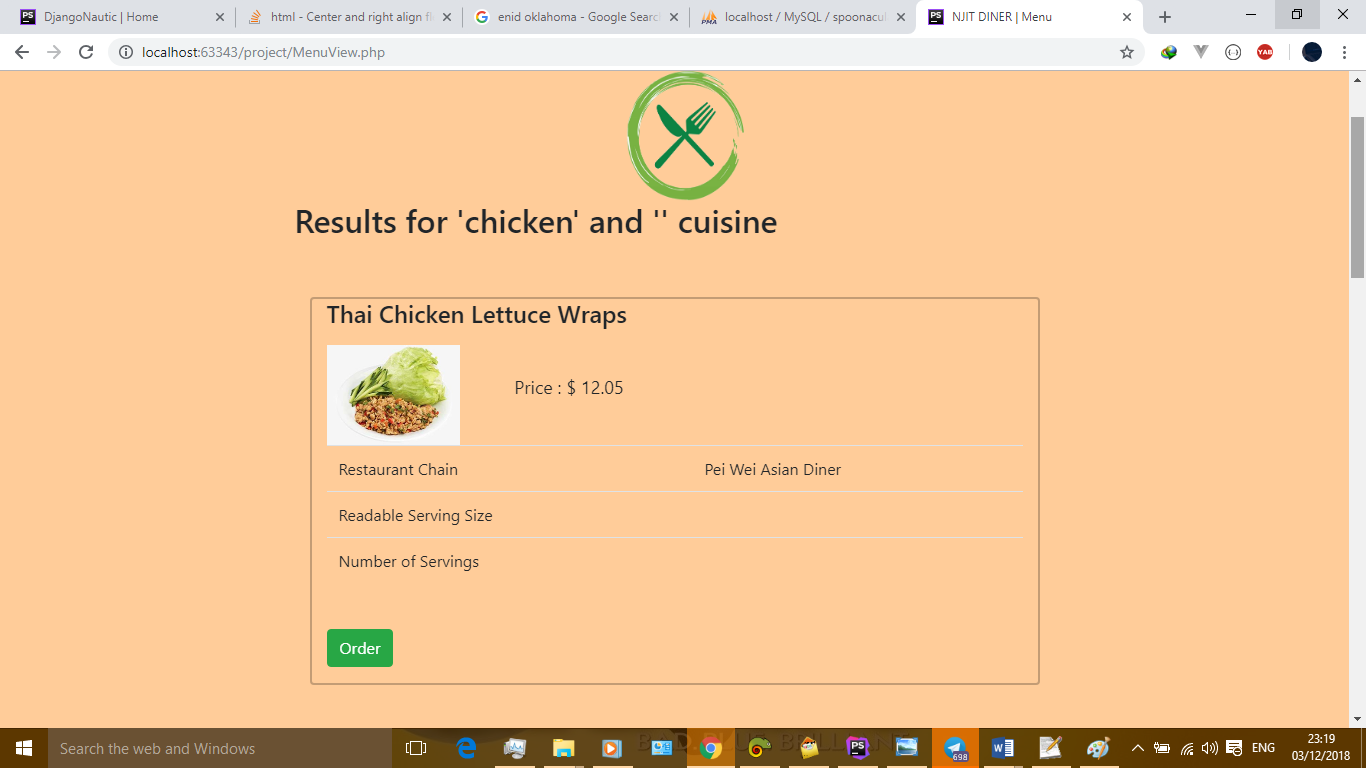
);

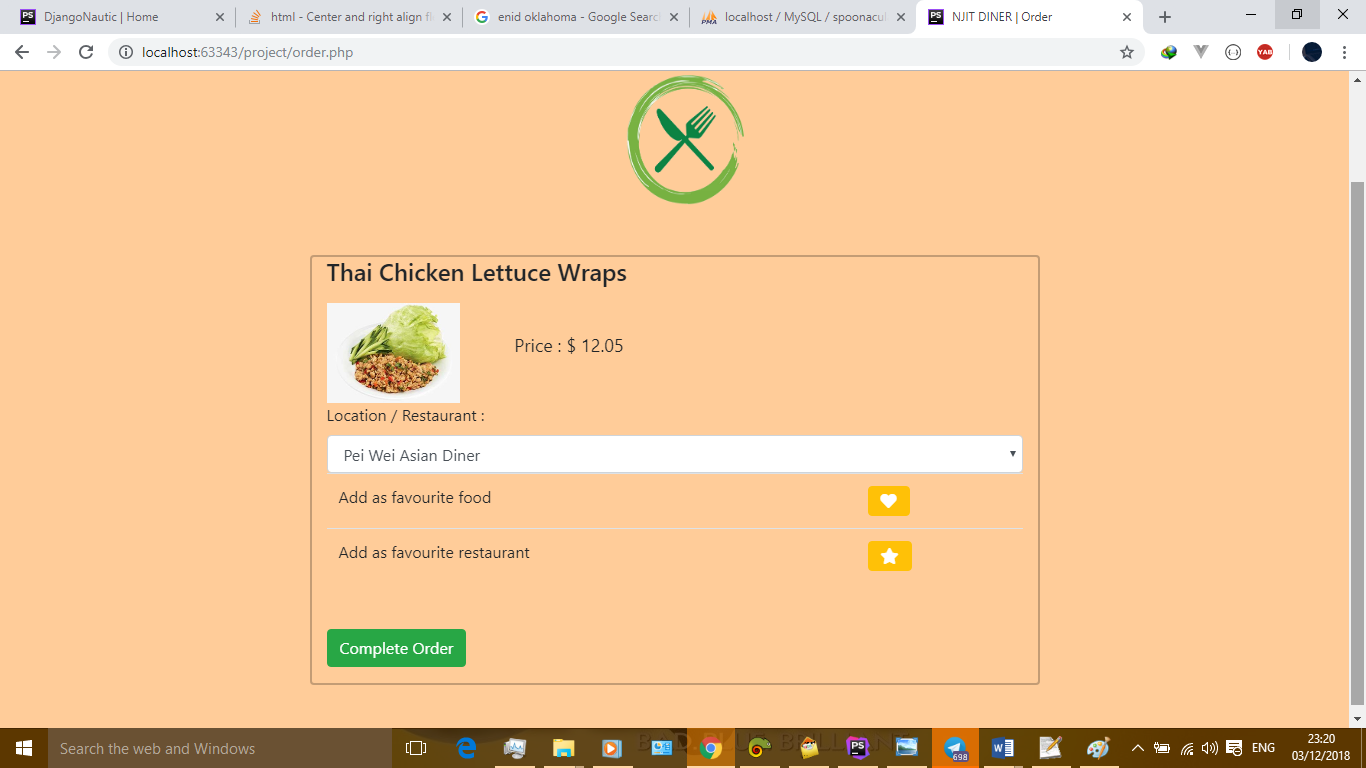
return $response->body;

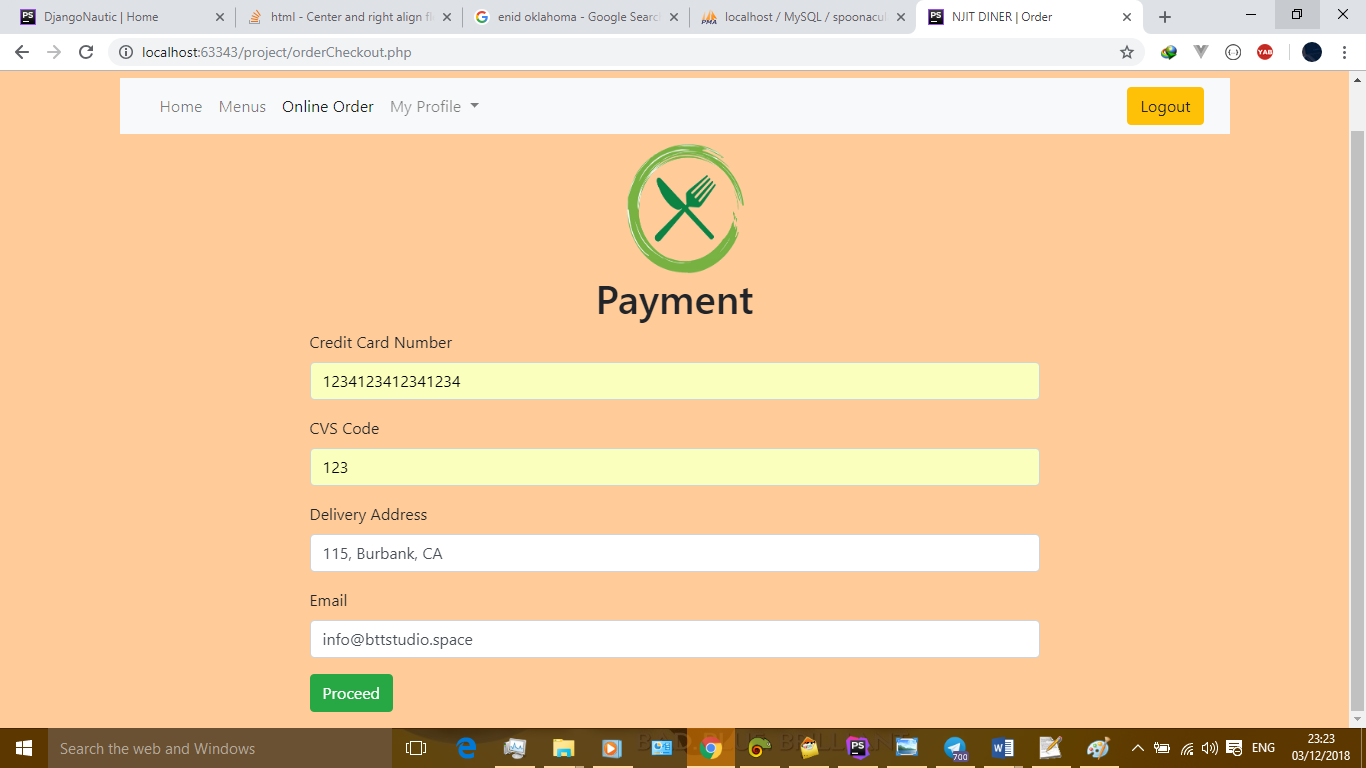
}

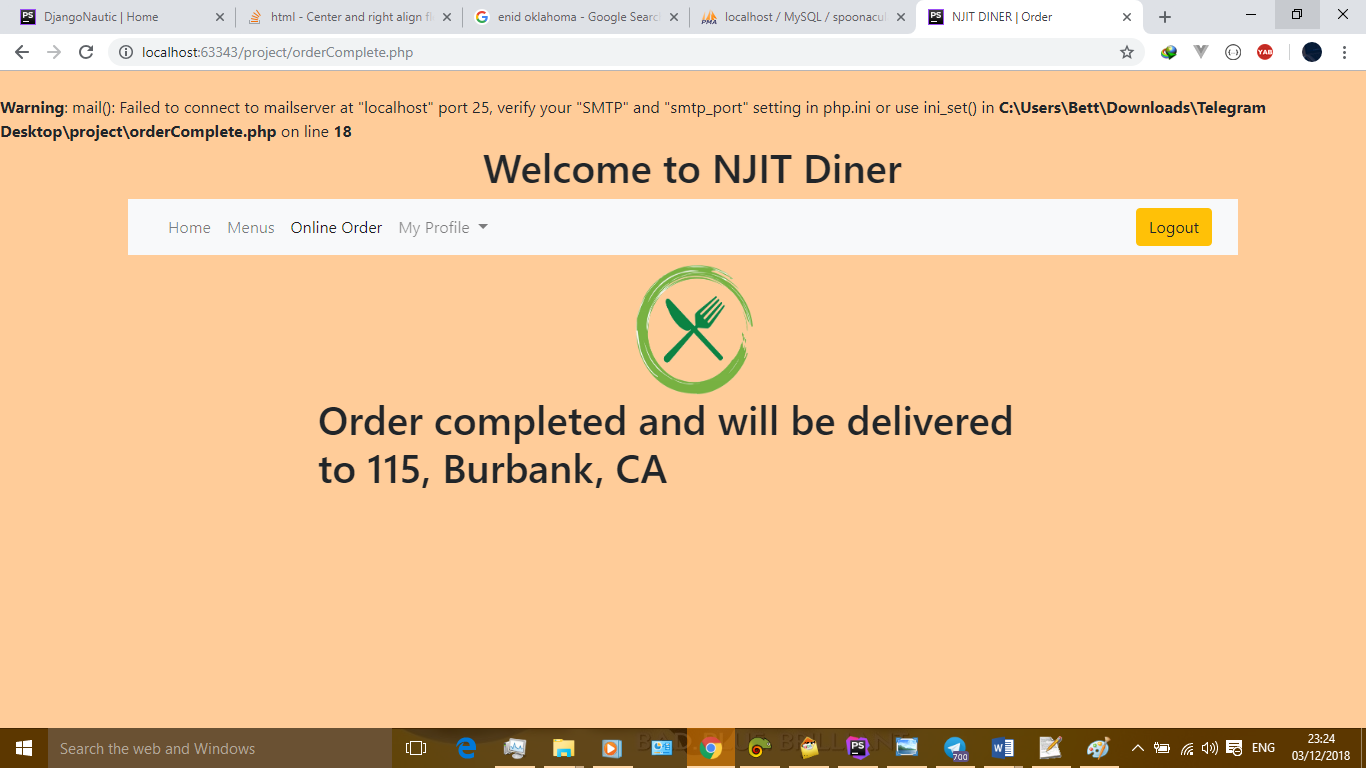












RabbitMQ:

### How does it work?

RabbitMQ works by offering an interface, connecting message senders (Publishers) with receivers (Consumers) through an exchange (Broker) which distributes the data to relevant lists (Message Queues).

APPLICATION       EXCHANGE TASK LIST         WORKER  
   [DATA] ------------> [DATA] ----------> [D]+[D][D][D] ------------>  [DATA]  
 Publisher        EXCHANGE Queue         Consumer

Installation Guides:

Linux, BSD, UNIX: [Debian, Ubuntu](https://www.rabbitmq.com/install-debian.html).

Windows: [Installer (recommended)](https://www.rabbitmq.com/install-windows.html) |  [Binary build](https://www.rabbitmq.com/install-windows-manual.html)

MacOS: [Homebrew](https://www.rabbitmq.com/install-homebrew.html) | [Generic binary build](https://www.rabbitmq.com/install-generic-unix.html)

### Installing on Ubuntu 13 / Debian 7 Based Systems

The process for downloading and installing RabbitMQ on Ubuntu and Debian will be similar to CentOS due to our desire of having a more recent version.

Let's begin with updating our system's default application toolset:

sudo apt-get update  
sudo apt-get -y upgrade

Enable RabbitMQ application repository:

echo "deb http://www.rabbitmq.com/debian/ testing main" >> /etc/apt/sources.list

Add the verification key for the package:

curl http://www.rabbitmq.com/rabbitmq-signing-key-public.asc | sudo apt-key add -

Update the sources with our new addition from above:

sudo apt-get update

And finally, download and install RabbitMQ:

sudo apt-get install rabbitmq-server

In order to manage the maximum amount of connections upon launch, open up and edit the following configuration file using nano:

sudo nano /etc/default/rabbitmq-server

Uncomment the limit line (i.e. remove #) before saving and exit by pressing CTRL+X followed with Y.

## Managing RabbitMQ

As we have mentioned before, RabbitMQ is very simple to get started with. Using the instructions below for your system, you can quickly manage its process and have it running at the system start-up (i.e. boot).

### Enabling the Management Console

RabbitMQ Management Console is one of the available plugins that lets you monitor the [RabbitMQ] server process through a web-based graphical user interface (GUI).

Using this console you can:

* Manage exchanges, queues, bindings, users
* Monitor queues, message rates, connections
* Send and receive messages
* Monitor Erlang processes, memory usage
* And much more

To enable RabbitMQ Management Console, run the following:

sudo rabbitmq-plugins enable rabbitmq\_management

Once you've enabled the console, it can be accessed using your favourite web browser by visiting: http://[your IP address]:15672/.

The default username and password are both set “guest” for the log in.

Note: If you enable this console after running the service, you will need to restart it for the changes to come into effect. See the relevant management section below for your operating system to be able to do it.

### Managing on Ubuntu / Debian Based Systems

To start, stop, restart and check the application status on Ubuntu and Debian, use the following:

# To start the service:  
service rabbitmq-server start  
  
# To stop the service:  
service rabbitmq-server stop  
  
# To restart the service:  
service rabbitmq-server restart  
  
# To check the status:  
service rabbitmq-server status

And that's it! You now have your own message queue working on your virtual server.

**References**

<https://market.mashape.com/spoonacular/recipe-food-nutrition#autocomplete-menu-item-search>

<https://www.rabbitmq.com/documentation.html>