ZAPPOS MENU API

DOCUMENTATION

Git hub: ***https://github.com/KshitijChauhan/ZapposFoodAPI.git***

The API has been deployed on Heroku cloud with the endpoint as: ***https://polar-stream-29751.herokuapp.com***

You can view the documentation of all the requests here(Postman) (Heroku Cloud): ***https://documenter.getpostman.com/view/4179013/zapposrestaurantapi/RW1UPNpy***

The API developed uses 7 classes (Separation of Concerns) namely:

RestaurantWebService – This is a Java servlet that accepts restaurant search requests and processes them as per the user's request. This returns a JSONObject as the response.

MenuWebService – This is a java servlet that accepts menu display search requests and processes them as per the user's request. This returns a JSONObject as the response.

DataExtractSQL – This is a java class used by the doGet method of the API. This class is used for both restaurants search as well as menu search and fetches data from the MySQL database as per user's request. This returns a JSON object as response.

DataAdd – This java class used by the doPost method of the API. This class is used for both restaurant and menu post operations and inserts data to the MySQL as per the user's request. This returns a JSONObject.

DataPut – This java class is used by the doPut method of my API. This class is used for both restaurant and menu put operations and updates data to the MySQL database as per the user's request. This returns a JSONObject.

Utils – This is a java class used to provide response to the user as per the request and result. The data returned is in the form of JSONObject

GetKey – This method is used to generate API key to use the restaurantWebService.

Tables used: restaurants, menu_table and user_table

The table currently contains data from 3 cities: Pittsburgh, Las Vegas and New York.

Steps for using the API:

Using doGet(Postman could be used to access the endpoints)

1) The first step is to access the endpoint: https://polar-stream-29751.herokuapp.com/GetKey or http://localhost:8080/RestaurantWeb/GetKey
   This endpoint would provide you with a unique key to access the API. This is used to enhance the security and store the data for the users who access the API for analytical purposes.
   This page requires 2 inputs from the user: name and email ID and provides a unique key based on SHA-256 hashing algorithm. This key is required to access the restaurants_id which is used to access the menu.
2) Once you have a key, you need to get the restaurant_id to further access the menu for that particular restaurant. To access and find out the restaurant_id you have to provide the API key and either the city or the restaurant name or both.
   This would provide you with the details about the restaurants situated in Pittsburgh along with their ID and other details. Please make user to note down the ID of the restaurant you wish to have the menu for.

   Example 1: To access the restaurants based in Pittsburgh the end point you can use is: http://localhost:8080/RestaurantWeb/restaurants/apikey=hCibWTwGaM/city=pittsburgh or https://polar-stream-29751.herokuapp.com/restaurants/apikey=hCibWTwGaM/city=pittsburgh

   Example 2: To access the restaurant with the name Caribbean Taste the end point you can use is: https://polar-stream-29751.herokuapp.com/restaurants/apikey=hCibWTwGaM/restaurant_name=Caribbean%20Taste or http://localhost:8080/RestaurantWeb/restaurants/apikey=hCibWTwGaM/restaurant_name=Caribbean%20Taste

   Example 3: To access a restaurant in Pittsburgh with the name China House and city Pittsburgh, the end point to use is: http://localhost:8080/RestaurantWeb/restaurants/apikey=hCibWTwGaM/city=pittsburgh&restaurant_name=china%20house
   Or
   https://polar-stream-29751.herokuapp.com/restaurants/apikey=hCibWTwGaM/city=pittsburgh&restaurant_name=china%20house

3) Once you have the restaurant_id with you, you need to use that id to fetch the menu of that restaurant. You can use only just the restaurant_id to get the menu for lunch, breakfast and dinner. In case you are looking for a specific menu you may add the menu_category to get the menu of a particular category.

Example 1: To access the menu of China House(restaurant_id = 8), you can use the endpoint as: https://polar-stream-29751.herokuapp.com/Menu/restaurant_id=8
Or
http://localhost:8080/RestaurantWeb/Menu/restaurant_id=8

Example 2: To get a specific menu you can use the endpoint as:
http://localhost:8080/RestaurantWeb/Menu/restaurant_id=8&menu_category=lunch
or
https://polar-stream-29751.herokuapp.com/Menu/restaurant_id=8&menu_category=lunch

Using doPost (Use PostMan)

1) To access the doPost method which allows users to add data to the database you will require a key. If you do not have the key, please visit the endpoint: https://polar-stream-29751.herokuapp.com/GetKey or http://localhost:8080/RestaurantWeb/GetKey.
This key is required to use doPost operation on the restaurants table but is not required to use the doPost operation on the menu_table because to access menu_table you need the restaurant_id and restaurant_id can only be found using the API key.

2) You can use Postman to test the result of the doPost operations on the restaurants table. The raw body has to follow the patter as below to access the database or else an error will be displayed.

Example 1: Using the entire set of parameters:
```
{
        "api_key": "hCibWTwGaM",
        "restaurant_name": "frank Pizzeria",
        "city": "Seattle",
        "price_range": 4,
        "rating": 3.7,
        "contact_number": "6178768987"
}
```

Output :

```
{
  "data": "The restaurant frank Pizzeria has been added.",
  "statusCode": 200
}
```

Example 2: not using the entire list of parameter:

```
{
            "api_key": "hCibWTwGaM",
            "restaurant_name": "Taco Bell",
            "city": "Seattle",
            "price_range": 4,
            "rating": 3.7
}
```

Notice the missing contact_number

Output:

```
{
   "message": "Please provide the full set of data",
   "statusCode": 400
}
```

Example 3: If the restaurant you are adding already exists:

```
{
            "api_key": "hCibWTwGaM",
            "restaurant_name": "Frank Pizzeria",
            "city": "Seattle",
            "price_range": 4,
            "rating": 3.7,
            "contact_number": "4126543456"
}
```

Notice the name of the restaurant, Frank Pizzeria has already been added

Output:

```
{
   "message": "The restaurant Frank Pizzeria is already present in the database",
   "statusCode": 400
}
```

3) You can use Postman to test the result of the doPost operations on the menu_table. The raw body has to follow the patter as below to access the database or else an error will be displayed.

   Example 1: Adding a dish to the menu using the entire set of parameters:

```
{
  "restaurant_id" : 1,
  "dish_name" : "french fries",
  "dish_category": "appetizer",
  "dish_price": 5.99,
  "dish_popularity": 4,
  "menu_category": "lunch",
  "dish_id": 1
}
```

   Output :

```
{
  "data": "The item curly fries has been added.",
  "statusCode": 200
}
```

   Example 2: Adding a dish to menu but with incomplete parameters:

```
{
  "restaurant_id" : 1,
  "dish_name" : "french fries",
  "dish_category": "appetizer",
  "dish_price": 5.99,
  "menu_category": "lunch",
  "dish_id": 1
}
```

   Notice the field dish_popularity missing

   Output:

```
{
  "message": "Please provide the full set of data",
  "statusCode": 400
}
```

   Example 3: trying to add a dish already present for a given restaurant:

```
{
   "restaurant_id" : 1,
   "dish_name" : "french fries",
   "dish_category": "appetizer",
   "dish_price": 5.99,
   "dish_popularity": 4,
   "menu_category": "lunch",
   "dish_id": 1
}
```

Notice we are trying to add French fries which has already been added

Output:

```
{
   "message": "The dish entered already exist",
   "statusCode": 400
}
```

Using doPut(Use Postman)

1) To access the doPut method which allows users to update data to the database you will require a key. If you do not have the key, please visit the endpoint: https://polar-stream-29751.herokuapp.com/GetKey or http://localhost:8080/RestaurantWeb/GetKey.
   This key is required to use doPut operation on the restaurants table but is not required to use the doPut operation on the menu_table because to access menu_table you need the restaurant_id and restaurant_id can only be found using the API key.

2) You can use Postman to test the result of the doPut operations on the restaurants table. The raw body has to follow the patter as below to access the database or else an error will be displayed.

   Example 1: using doPut with all the required parameters:

```
{
        "api_key": "hCibWTwGaM",
        "restaurant_id": 14,
        "restaurant_name": "Frank Pizzeria",
        "city": "Seattle",
        "price_range": 4,
        "rating": 3.7,
        "contact_number": "4126511122"
}
```

If you will notice, I updated the contact_number of this restaurant

Output :

```
{
   "data": "The restaurant Frank Pizzeria has been updated.",
   "statusCode": 200
}
```

Example 2: Using doPut on a restaurant that does not exist

```
{
        "api_key": "hCibWTwGaM",
        "restaurant_id": 25,
        "restaurant_name": "Frank Pizzeria",
        "city": "Seattle",
        "price_range": 4,
        "rating": 3.7,
        "contact_number": "4126511122"
}
```

Notice the restaurant_id, since the table contains restaurant_id till 14, using an ID greater than 14 would give the result as:

```
{
   "message": "Restaurant does not exist in table",
   "statusCode": 400
}
```

3) You can use Postman to test the result of the doPut operations on the menu_table. The raw body has to follow the patter as below to access the database or else an error will be displayed.

   Example 1: doPut used with correct parameters:

```
{
   "restaurant_id" : 1,
   "dish_name" : "french fries",
   "dish_category": "appetizer",
   "dish_price": 5.99,
   "dish_popularity": 4,
   "menu_category": "lunch",
   "dish_id": 1
}
```

Output:

```
{
   "data": "The item french fries has been updated.",
   "statusCode": 200
}
```

Example 2: If you try and update a dish with dish_id greater than what is present in the table you can expect to see the following as the response:

```
{
   "restaurant_id" : 1,
   "dish_name" : "french fries",
   "dish_category": "appetizer",
   "dish_price": 5.99,
   "dish_popularity": 4,
   "menu_category": "lunch",
   "dish_id": 10000
}
```

Notice that the dish_id used is 10000, which does not exist in the table as table contains data till ~450 dishes and hence the output is as follows:

```
{
   "message": "Dish in this restaurant does not exist in table",
   "statusCode": 400
}
```

Concepts and resources Used:

1) **API Key authentication:** I have enabled the API key authentication to make sure that only authentic requests are made and false requests could be monitored. SHA-256 algorithm is used to generate the key. This key is stored in the user table along with the name and email ID of the user.
2) **AWS Cloud RDS MySQL Instance:** I have hosted my MySQL database on AWS RDS Instance. This is done to provide remote access to whosoever uses this API wherever in the world.
3) **Mitigation of SQL injection:** Made use of Prepared Statement to execute my SQL queries instead of forming SQL String queries. This would enable the API to mitigate the impact of SQL Injections. Also, Prepared Statement allows for the per query caching which would increase the optimization of the API.
4) **Separation of Concerns:** I have made use of different classes to handle different API operations such as doPost, doPut, doGet
5) **Hosting on Heroku Cloud:** I have hosted my API on the Heroku cloud so that the API could be utilized even without having access to the backend code.

6) **Glass fish server –** I have made use of glassfish server instead of Tomcat as glass fish is the major server that is being used in my distributed systems classes.

What I missed (Will be implemented in version 2):

1) **Redis Instance** – I did create a redis instance on AWS Elasticache but due to time constraint was not able to implement redis into the API this time around. I will continue working on the redis part of API and improve my API with time.
2) **DynamoDB** – I feel I missed all the noSQL stuff. DynamoDB is something that I have not made use of and I wanted to make use of MySQL first, get the API running and then implement DynamoDB, but unfortunately time constraint got me. I will try and implement DynamoDB in the future release.
3) **LRU caching** – I wished to implement LRU caching to the API and would be doing that in the future.
4) **Parallel SQL threading** – Since the queries used in this API were not complex and did not use joins I did not implement SQL parallel threading but in case in future we tend to use big queries, implementing Parallel SQL threading would be a great option to optimize the time consumed in fetching data from the tables.
5) **Implementation of a more robust security system –** An API key generation using timestamps could be put into place to further increase the security and OAuth1/OAuth2 could also be implemented.
6) **Hibernate ORM –** Hibernate could be used for mapping of java model to the relational database and will be implemented in the next version

Limitations of the API:

1) **DoDelete** – Due to a constraint on the server I was not able to implement the doDelete operation on my API. Glass fish server didn't allow me to access the doDelete and hence I had to skip it. I have implemented doPut instead of doDelete for this API.

Thank you for providing me this opportunity to present my work to you. I hope you like it.

That's all folks!!!!