# Zappos Ordering System API documentation

Overview: This API is organized around REST. It lets you the lookup following information related to restaurants-

* List all restaurants
* List all menus in a restaurant
* List all menu items in a restaurant

It also enables a person to delete information from the aforementioned lists.

Resources: This API is developed using the following-

* Java (Spring Framework)
* Hibernate
* MySQL
* Tomcat
* Redis

Installation: In order to use this API start the MySQL server and Redis server and install the java project on Tomcat.

**MySQL**: The configuration details of MySQL are stored in src/main/java/resources/properties/database.properties file.

**Redis**: The configuration details of Redis are stored in src/main/java/com/orderingsystem/config/CacheConfig.java

URL configuration:

The URL to get response from the REST API should look something like:

http://{tomcat\_url}/{application\_context}/{requested\_API}

The application\_context for this API is /OrderingSystem

The requested\_API parameter varies as follows:

Restaurants: http://localhost:8080/OrderingSystem/restaurants

Menus: http://localhost:8080/OrderingSystem/restaurants/{restaurant\_id}/menus

Menu Items: [http://localhost:8080/OrderingSystem/restaurants/{restaurant\_id}/menus/{menu\_id}/menuitems](http://localhost:8080/OrderingSystem/restaurants/%7Brestaurant_id%7D/menus/%7Bmenu_id%7D/menuitems)

Where restaurant\_id is the id (reference to the restaurant table in MySQL) of the restaurant for which the menus are requested.

Where restaurant\_id is the id of the restaurant and menu\_id is the id of the menu to which the menu item belongs to.

Database Structure: The database that was used while developing this API contained 3 tables with the following structure:

**Database name:**zappos

**Table 1**: restaurants

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Primary Key |
| restaurant\_id | int | Yes |
| name | varchar | No |

**Table 2:** menus

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Primary Key |
| menu\_id | int | Yes |
| restaurant\_id | int | Yes |
| name | varchar | No |

**Table 3:** menu\_items

|  |  |  |
| --- | --- | --- |
| Column Name | Type | Primary Key |
| menu\_item\_id | int | Yes |
| menu\_id | int | Yes |
| restaurant\_id | int | Yes |
| name | varchar | No |
| price | decimal | No |

Methods:

|  |  |
| --- | --- |
| Method Name | Description |
| getRestaurants() | Parameters: None  It fetches a list of restaurants from the database if it is not already cached in redis template, otherwise redis returns the list of restaurants. |
| getMenus() | Parameters: restaurantId  It fetches all the menus of the restaurant with restaurantId as id from the database if it is not already cached in redis, otherwise redis returns the cached menus for that restaurantId. |
| getMenuItems() | Parameters: menuId, restaurantId  It fetches all the menu items of the menu with given menuId and restaurantId from the database if it is not already cached in redis, otherwise redis returns the cached menu items for that menuId and restaurantIDd. |
| deleteRestaurants() | Parameters: None  It deletes all the restaurants and related information from the database and invalidates the redis cache. |
| deleteMenu() | Parameters: menuId, restaurantId  It deletes all the menu and related information for given menuId and restaurantId from the database and invalidates the redis cache. |
| deleteMenuItem() | Parameters: itemId, menuId, restaurantId  It deletes all the items for the given itemId, menuId and restaurantId from the database and invalidates the redis cache. |

## Future Work:

* To reduce the response time, pagination can be used with each request where in only that amount of data is sent to the front end which can be handled in one page.
* Use of distributed databases since the API should be able to handle a lot of data.