# Web Services and Cloud Exam (April 2015) – Restaurants

Design and implement **RESTful Web Services** based on ASP.NET Web API, Entity Framework Code First and MS SQL Server for a nation-wide **restaurant system**. The restaurant system hold **towns**, **restaurants**, **meals, ratings** and **users**.   
**Anonymous visitors** view **restaurants** and the **meals** they offer.  
**Registered users** can:

* Create and rate restaurants.
* Create, edit and delete meals in restaurants.
* Make orders and view their own orders.

### Compile and Run the Project

You are given a Visual Studio project "Restaurants" holding a **data layer** (holding EF data models and EF data context) and a **Web API application** (RESTful Web services). You need to **compile and run** the Web API application.

* The project **already holds the user login and user register** functionality. You will need to write the rest.
* You are given the data model classes holding Restaurant, Town, Meal, MealType, Order and Rating, as well as a ready **DbContext**.
* You are given a **Postman collection** of HTTP requests to test your REST service. Play with it.

Internet connection is required to download the referenced NuGet packages.

### Get All Restaurants by Town

Write a REST service to list all Restaurants by town.

* Returns all restaurants by **town id**, ordered by **rating** (average of all given ratings, in descending order), then by **name** (ascending). Returns **id**, **name**, **rating** and **town** (id and name).

|  |  |  |
| --- | --- | --- |
| Request | GET /api/restaurants?townId={townId} | |
| Response | 200 OK  [{"id":1,"name":"Sushi Heaven",  "rating":7,"town":  {"id":2,"name":"Sofia"}},...] | Returns the list of all restaurants ordered alphabetically by rating as JSON array. |

5 score

### Create New Restaurant

Write a REST service to create a new restaurant.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/restaurants  Content-Type: application/x-www-form-urlencoded  name=*{restaurant-name}&townId={town-id}* | | POST /api/restaurants  name=Sushi+Place&townId=2 |
| Response | 201 Created  Location: http://localhost:1337/api/restaurants/2  {"id":7,"name":"Sushi Place","rating": null,"town":{"id":2,"name":"Sofia"}} | On success, the service returns **201** (**Created**) + a header "Location" holding the URL of the created restaurant + the restaurant data as JSON object in the response body (Id, Name, Rating and Town (id and name)). |
| Error Response | 400 Bad Request | Returned in case of missing or invalid restaurant data (e.g. empty restaurant name). |
| Error Response | 401 Unauthorized | Returned in case the user is logged in. |

5 score

### Rate an Existing Restaurant

Write a REST service to rate existing restaurants. A user may only give **one rating per restaurant**. If he rates twice, the old rating must be **overridden with the new value**.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/restaurants/{id}/rate  Content-Type: application/x-www-form-urlencoded  stars=*{stars-count}* | | POST /api/restaurants/2  stars=7 |
| Response | 200 OK | On success, the service returns 200 OK. |
| Error  Response | 400 Bad Request | Returned in case of **invalid rating** (must be in range 1…10) or if the **restaurant owner** rates his own restaurant. |
| Error Response | 404 Not Found | Returned in case of invalid restaurant id. |

10 score

### Get Restaurant Meals

Write a REST service to list all meals from a given restaurant.

* Returns all meals from a restaurant by **id**, ordered by **meal type order** (ascending), then by **name** (ascending). The meal type order is as follows: **salad** -> **soup** -> **main** -> **dessert**.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/restaurants/{id}/meals | Example |
| GET /api/restaurants/1/meals |
| Response | 200 OK  [{"id":3,"name":"Musaka","price":4.5,"type":"Main"},  {"id":2,"name":"ChocolateCake","price":2.6,"type":"Dessert"}] | Returns the list of meals by restaurant as JSON array. |
| Error Response | 404 Not Found | Returned when the requested restaurant does not exists (invalid id). |

5 score

### Create New Meal

Write a REST service to create a new meal.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/meals  Content-Type: application/x-www-form-urlencoded  name=*{meal-name}&price={price}&typeId={type-id}&restaurantId={restaurantId}* | | POST /api/meals  name=Shish%20Kebab&price=19.5 &typeId=3&restaurantId=2 |
| Response | 201 Created  Location: http://localhost:1337/api/meals/4  {"id":4,"name":"Shish Kebab","price":19.5,"type":"Main"} | On success, the service returns **201** (**Created**) + a header "Location" holding the URL of the created meal + the meal data as JSON object in the response body (Id, Name, Price and Type). |
| Error Response | 400 Bad Request | Returned in case of missing or invalid restaurant data (e.g. empty meal name), invalid type id or restaurant id (non-existent). |
| Error Response | 401 Unauthorized | Returned in case the user is not the restaurant owner. |

10 score

### Edit Existing Meal

Write a REST service to edit an existing meal. A meal can be edited only by the restaurant owner.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| PUT /api/meals/{id}  Content-Type: application/x-www-form-urlencoded  name=*{meal-name}&price={price}&typeId={type-id}* | | POST /api/meals/4  name=Chorba&price=2.5 &typeId=1 |
| Response | 200 OK  Location: http://localhost:1337/api/meals/4  {"id":4,"name":"Chorba","price":2.5,"type":"Soup"} | On success, the service returns **200 OK** + the meal data as JSON. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid restaurant data (e.g. empty meal name) or invalid type id (non-existent). |
| Error Response | 401 Unauthorized | Returned in case the user is not the restaurant owner. |
| Error Response | 404 Not Found | Returned when the requested meal does not exists (invalid meal id). |

10 score

### Delete Meal by ID

Write a REST service to delete a meal by ID. A meal can be deleted only by the restaurant owner.

|  |  |  |  |
| --- | --- | --- | --- |
| Request | DELETE /api/meals/*{id}* | Example | DELETE /api/meals/3 |
| Response | 200 OK  {"message":"Meal #3 deleted."} | On success, the service returns 200 (OK) + optional human-readable message, explaining that the meal was deleted. | |
| Error Response | 401 Unauthorized | Returned in case the user is not the restaurant owner. | |
| Error Response | 404 Not Found | Returned when the requested meal does not exists (invalid meal id). | |

5 score

### Create New Order

Write a REST service to create a new order of a **meal** in a specified **quantity**.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/meals/{id}/order  Content-Type: application/x-www-form-urlencoded  quantity={quantity} | | POST /api/meals/2/order  quantity=3 |
| Response | 200 OK | On success, the service returns **200** **OK** |
| Error Response | 400 Bad Request | Returned in case of missing or invalid restaurant data (e.g. missing quantity). |
| Error Response | 401 Unauthorized | Returned in case the user is not logged in. |
| Error Response | 404 Not Found | Returned when the requested meal does not exists (invalid meal id). |

10 score

### View Pending Orders

Write a REST service to get the **pending orders** of the **current user** with paging.

* Accepts query parameters in URL query string:
  + **startPage** (the page that should be fetched)
  + **limit** (the number of orders that should be returned in the range 2..10)
  + **mealId** (**optional**, id of a specific meal)
* Returns orders made by the **current user** (with the above filtering and paging applied), which are currently **pending**. Orders them by **date of creation** (from latest). Selects each order's **id**, **status**, **date** **of** **creation**, **meal** (id, name, price and type) and **quantity**.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| GET /api/orders?startPage={start-page}&limit={page-size}&mealId={mealId} | | GET /api/orders?startPage=0&limit=2&mealId=2 |
| Response | 200 OK  [{"id":2,"meal":{"id":3,"name":"Musaka","price":4.5,"type":"Main"},"quantity":5,"status":0,"createdOn":"2015-09-04T19:02:12.327"},…] | On success, the service returns **200** **OK** |
| Error Response | 401 Unauthorized | Returned in case the user is not logged in. |

10 score

### Register User

You are given a REST service to **register** a user account by **username** (unique) and **password**. Do not touch it, just play with it to learn how it works:

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/account/register Content-Type: application/x-www-form-urlencoded  username=*some\_username*&pasword=*some\_password* | | POST /api/user/register  username=maria&password=pAss123 |
| Response | 200 OK  {"access\_token":"VccMrKjEWki…", "token\_type":"bearer",  "userName":"maria", … } | On success, the service returns 200 (OK) + the registered username + the access\_token for bearer authorization. |
| Error Response | 400 Bad Request | Returned in case of **missing** or **invalid** user account data (e.g. empty password) or **duplicated** username. |

### User Login

You are given a REST service to **login** existing user by **username** and **password**. Do not touch it, just play with it to learn how it works:

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/account/login Content-Type: application/x-www-form-urlencoded  username=*some\_username*&pasword=*some\_password* | | POST /api/user/login  username=maria&password=pAss123 |
| Response | 200 OK  {"access\_token":"VccMrKjEWki…", "token\_type":"bearer",  "userName":"maria", … } | On success, the service returns 200 (OK) + the logged-in username + the access\_token for bearer authorization. |
| Error Response | 400 Bad Request | Returned in case of **missing** or **invalid** user account data (e.g. empty password or invalid username or password). |

### Write Integration Tests for "Edit Existing Meal" Service

Write **integration tests** for the "**Edit Existing Meal**" REST service. Ensure you cover all interesting cases. Put your tests in a new class called "MealsIntegrationTests".

10 score

### Repository and Unit of Work

Before you modifying your project first **backup your work**.

Implement the classical **Repository** and **Unit of Work patterns** to separate the EF data layer from the Web API controllers through interfaces and simplify the eventual unit testing of the Web API controllers. Use your new data layer in the Web API controllers instead of directly calling Entity Framework.

5 score

### Write Unit Tests for "Get All Restaurants by Town" Service

Write **unit tests** with **mocking** for the "**Get all restaurants by town**" REST service. Use a **fake or mocked repository** and unit of work implementations. Test the work of the Web API controller only. Your unit test should not access the database. Ensure you cover all interesting cases. Put your tests in a new class called "**Restaurants**ControllerTests".

15 score

### Write Unit Tests for "Get All Restaurants by Town" Service

Write **unit tests** with **mocking** for the "**Rate Restaurant**" REST service.

Bonus 10 score

## Exam Information

You are allowed to use any resources you have, e.g. Internet, software, existing code.

You are not allowed to get help from other people. Skype, ICQ, FB, email, talks, phone calls, etc. are forbidden.

Exam time: **6 hours**.