# Web Services and Cloud Exam (June 2015) – Bid System

Design and implement **RESTful Web Services** based on ASP.NET Web API, Entity Framework Code First and MS SQL Server for a bid system. The bid system holds **users**, **offers** and **bids. Anonymous visitors** can view offers and bids, register and login. **Registered users** (after login) can submit offers and bid for existing offers.

**Offers** have **title**, **description** (optional), **seller** (registered user), **publish date**, **initial price**, **expiration date** (bidding end date), and set of bids (optionally). **Bids** belong to existing offers and have **bid price**, **bidder** (registered user), **date** and **comment** (optional).

### Compile and Run the Bid System Project

You are given a Visual Studio project "BidSystem" holding a **data layer** (EF data models and EF data context), **Web API application** (RESTful Web services) and automated **testing project** (designed to perform integration tests of the Web API application). You need to **compile and run** the Web API application and run the automated tests.

* The project **already holds the user login and user register** functionality. You will need to write the rest.
* Some of the **tests will pass** (user login / user register), but the others will **fail** because the tested functionality is missing.
* You are given the data model classes holding User, Offer and Bid, but these classes are unfinished. You will need to **finish the data layer** and make the EF Code First work correctly.
* You are given a **Postman collection** of HTTP requests to test your REST service. Play with it.

Use Visual Studio 2013 Update 4. Internet connection is required to download the referenced NuGet packages.

### List All Offers

Write a REST service to **list all offers**.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/offers/all | |
| Response | 200 OK  [{"Id":117,"Title":"My Offer (Active 6 months)","Description":"Some Description","Seller":"peter","DatePublished":"2015-06-23T21:20:16.677","InitialPrice":120.00,"ExpirationDateTime":"2015-12-23T23:20:16","IsExpired":false,"BidsCount":3,"BidWinner":null},{"Id":115,"Title":"Another Offer (Expired)","Description":null,"Seller":"nakov","DatePublished":"2015-06-23T21:20:16.657","InitialPrice":15.50,"ExpirationDateTime":"2015-06-23T00:20:16","IsExpired":true,"BidsCount":2,"BidWinner":"peter"}] | Returns the list of offers ordered by publish date from the latest to the earliest as JSON array. Each offer holds Id, Title, Description, Seller (username), DatePublished, InitialPrice, ExpirationDateTime, IsExpired, BidsCount and BidWinner. All dates are displayed in ISO 8601 format. An offer is expired if its expiration date has passed. If the offer is expired and when it has at least one bid, it has a bid winner – the username that has the maximum offers price (the biggest bid). |

5 score

### List Active Offers

Write a REST service to **list all active offers** (not expired).

|  |  |  |
| --- | --- | --- |
| Request | GET /api/offers/active | |
| Response | 200 OK  [{"Id":117,"Title":"My Offer (Active 6 months)","Description":"Some Description","Seller":"peter","DatePublished":"2015-06-23T21:20:16.677","InitialPrice":120.00,"ExpirationDateTime":"2015-12-23T23:20:16","IsExpired":false,"BidsCount":3,"BidWinner":null] | Returns the list of active offers ordered by expiration date from the earliest to the latest as JSON array. Each offer holds Id, Title, Description, Seller (username), DatePublished, InitialPrice, ExpirationDateTime, IsExpired, BidsCount and BidWinner. All dates are displayed in ISO 8601 format. |

5 score

### List Expired Offers

Write a REST service to **list all expired offers**.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/offers/expired | |
| Response | 200 OK  [{"Id":115,"Title":"Another Offer (Expired)","Description":null,"Seller":"nakov","DatePublished":"2015-06-23T21:20:16.657","InitialPrice":15.50,"ExpirationDateTime":"2015-06-23T00:20:16","IsExpired":true,"BidsCount":2,"BidWinner":"peter"}] | Returns the list of expired offers ordered by expiration date from the earliest to the latest as JSON array. Each offer holds Id, Title, Description, Seller (username), DatePublished, InitialPrice, ExpirationDateTime, IsExpired, BidsCount and BidWinner. All dates are displayed in ISO 8601 format. |

5 score

### Get Offer Details by ID

Write a REST service to get offer details by ID.

|  |  |  |  |
| --- | --- | --- | --- |
| Request | GET /api/offers/details/*{id}* | Example | GET /api/offers/details/115 |
| Response | 200 OK  {"Id":115,"Title":"Another Offer (Expired)","Description":null,"Seller":"peter","DatePublished":"2015-06-23T21:20:16.657","InitialPrice":15.50,"ExpirationDateTime":"2015-06-23T00:20:16","IsExpired":true,"BidWinner":"nakov","Bids":[{"Id":73,"OfferId":115,"DateCreated":"2015-06-23T21:26:36.64","Bidder":"nakov","OfferedPrice":1200.00,"Comment":"Let's bid!"},{"Id":72,"OfferId":115,"DateCreated":"2015-06-23T21:26:32.02","Bidder":"maria","OfferedPrice":700.00,"Comment":null}]} | Returned when the requested offer exists.  Returns the offer details as JSON array. It holds Id, Title, Description, Seller (username), DatePublished, InitialPrice, ExpirationDateTime, IsExpired, BidWinner and Bids. Dates are displayed in ISO 8601 format.  Bids are displayed as JSON array. Each bid holds Id, OfferId, DateCreated, Bidder (username), OfferedPrice and Comment.  Sort the bids from the latest to the earliest. If bids do not exist, return them as empty array: []. | |
| Error Response | 404 Not Found | Returned when the requested offer does not exist (invalid id). | |

10 score

### Register User

You are given a REST service to **register** a user account by **username** (unique) and **password**. When a user is registered, the service returns a bearer authorization access\_token to be used to authorize further requests to the other services. **Do not touch it**, just play with it to learn how it works:

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/user/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 issued 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. |

### Login User

You are given a REST service to **login** existing user by **username** and **password**. When a user is registered, the service returns a bearer authorization access\_token to be used to authorize further requests to the other services.  
**Do not touch it**, just play with it to learn how it works:

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/user/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 issued 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). |

### List User's Offers

Write a REST service to **list user's offers**.

This operation requires bearer authorization access\_token in the request headers to specify who is the user.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/offers/my  Authorization: Bearer *{access\_token}* | |
| Response | 200 OK  [{"Id":115,"Title":"Another Offer (Expired)","Description":null,"Seller":"nakov","DatePublished":"2015-06-23T21:20:16.657","InitialPrice":15.50,"ExpirationDateTime":"2015-06-23T00:20:16","IsExpired":true,"BidsCount":2,"BidWinner":"peter"}] | Returns the list of user's offers ordered by publish date from the earliest to the latest as JSON array. Each offer holds Id, Title, Description, Seller (username), DatePublished, InitialPrice, ExpirationDateTime, IsExpired, BidsCount and BidWinner. All dates are displayed in ISO 8601 format. |
| Error Response | 401 Unauthorized | Returned in case of invalid access\_token or invalid user specified in the access token. |

5 score

### Create a New Offer

Write a REST service to create a new offer. The offer **title**, **description** (optional), **initial price** and **expiration date** are posted in the HTTP body as form data (URL-encoded). New offers take as a **seller** the current user and the current date and time as **publish date**.

This operation requires bearer authorization access\_token in the request headers to specify who is the seller.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/offers  Content-Type: application/x-www-form-urlencoded  Authorization: Bearer *{access\_token}*  title=*{title}*&description=*{description}*&initialPrice=*{initial\_price}*&expirationDateTime=*{date\_time}* | | POST /api/offers  Authorization: Bearer iyA1OenDbE4aX…  title=Ford+Mondeo+2.0+5dr+for+Sale&description=2002%2C+diesel%2C+hatchback%2C+112+000+km%2C&initialPrice=500&expirationDateTime=3-Jul-2015+12%3A00 |
| Response | 201 Created  Location: http://localhost:8888/api/offers/details/9  {"Id":8,"Seller":"maria","Message":"Offer created."} | On success, when **user's offer** is added, the service returns 201 (Created) + a header "Location" holding the URL of the created offer + the offer Id + the Seller username + human readable message as JSON object in the response body. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid offer data (e.g. empty title). |
| Error Response | 401 Unauthorized | Returned in case of invalid access\_token or invalid user specified in the access token. |

15 score

### List User's Bids

Write a REST service to **list all user's bids**.

This operation requires bearer authorization access\_token in the request headers to specify who is the user.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/bids/my  Authorization: Bearer *{access\_token}* | |
| Response | 200 OK  [{"Id":75,"OfferId":117,"DateCreated":"2015-06-23T22:08:17.727","Bidder":"nakov","OfferedPrice":900.00,"Comment":null},{"Id":72,"OfferId":115,"DateCreated":"2015-06-23T21:26:32.02","Bidder":"nakov","OfferedPrice":700.00,"Comment":"Let's bid!"},{"Id":71,"OfferId":115,"DateCreated":"2015-06-23T21:25:41.65","Bidder":"nakov","OfferedPrice":650.33,"Comment":"Last chance, fellows"},{"Id":69,"OfferId":117,"DateCreated":"2015-06-23T21:25:31.833","Bidder":"nakov","OfferedPrice":800.00,"Comment":null}] | Returns the list of user's bids ordered by date from the latest to the earliest as JSON array.  Each bid holds Id, OfferId, DateCreated (in ISO 8601 format), Bidder (username), OfferedPrice and Comment. |
| Error Response | 401 Unauthorized | Returned in case of invalid access\_token or invalid user specified in the access token. |

5 score

### List User's Won Bids

Write a REST service to **list user's won bids**. A bid is **won** when it has offered the **highest price** for its offer and its offer has expired.

This operation requires bearer authorization access\_token in the request headers to specify who is the user.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/bids/won  Authorization: Bearer *{access\_token}* | |
| Response | 200 OK  [{"Id":14,"OfferId":112,"DateCreated":"2014-06-23T23:08:17.727","Bidder":"nakov","OfferedPrice":1450.00,"Comment":"My last bid"},{"Id":73,"OfferId":117,"DateCreated":"2015-06-23T21:48:31.833","Bidder":"nakov","OfferedPrice":1200.00,"Comment":null}] | Returns the list of user's won bids ordered by date from the earliest to the latest as JSON array.  Each bid holds Id, OfferId, DateCreated (in ISO 8601 format), Bidder (username), OfferedPrice and Comment. |
| Error Response | 401 Unauthorized | Returned in case of invalid access\_token or invalid user specified in the access token. |

10 score

### Bid for Existing Offer

Write a REST service to **bid for existing offer**. The **offer ID** is given in the request URI.

The **bid price** and **comment** (optional) are posted in the HTTP body as form data (URL-encoded).

This operation requires bearer authorization access\_token in the request headers to specify who is the bidder.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/offers/117/bid  Content-Type: application/x-www-form-urlencoded  Authorization: Bearer *{access\_token}*  bidPrice=*{price}*&comment=*{comment}* | | POST /api/offers/117/bid  Authorization: Bearer iyA1OenDbE4aX…  bidPrice=550&comment=Let's+bid! |
| Response | 200 OK  {"Id":78,"Bidder":"nakov","Message":"Bid created."} | On success, when **user's bid** is created, the service returns 200 (OK) + the bid Id + the Bidder username + human readable message as JSON object in the response body. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid offer data (e.g. missing bid price). |
| Error Response | 401 Unauthorized | Returned in case of invalid access\_token or invalid user specified in the access token. |
| Error Response | 404 Not Found | Returned when the requested offer does not exist (invalid offer id). |
| Error Response | 400 Bad Request  {"Message":"Offer has expired."} | Returned when the offer has expired (its expiration has was passed). |
| Error Response | 400 Bad Request  {"Message":"Your bid should be > 500"} | Returned when the bid price is invalid. This happens either when it is lower or equal to the initial price or when another higher bid already exists for the same offer. |

25 score

### Write Integration Tests for "Get Offer Details" Service

Write **integration tests** for the "**Get Offer Details**" REST service. Ensure you cover all interesting cases. Put your tests in a new class called "OfferDetailsIntegrationTests".

15 score

### Repository and Unit of Work

Before 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.

Bonus: 10 score

### Write Unit Tests for "My Bids" Service

Write **unit tests** with **mocking** for the "**My Bids**" 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 "MyBidsUnitTestsWithMocking".

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**.