# Web Services and Cloud Exam (April 2015) – Bug Tracker

Design and implement **RESTful Web Services** based on ASP.NET Web API, Entity Framework Code First and MS SQL Server for a bug tracking system. The bug tracking system holds **users**, **bugs** and **comments. All users** of the service can list, filter, submit, edit and delete bugs, list comments and add comments. **Anonymous visitors** can also register and login. **Registered users** (after login) can submit bugs and add comments **with author**.

**Bugs** have **title**, **description** (optional), **status** (**Open** / **InProgress** / **Fixed** / **Closed**), **author** (registered user, optional) and submit **date**. **Comments** belong to existing bugs and have **text**, **author** and publish **date**.

### Compile and Run the Bug Tracker Project

You are given a Visual Studio project "BugTracker" 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, Bug and Comment, 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 Bugs

Write a REST service to list all bugs.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/bugs | |
| Response | 200 OK  [{"Id":24, "Title":"broken link", "Status":"Open", "Author":"nakov", "DateCreated":"2015-04-18T10:02:23.883"}, {"Id":11, "Title":"incorrect error message", "Status":"Fixed", "Author":null, "DateCreated":"2015-04-18T09:57:30.587"}, {"Id":7, "Title":"app hangs", "Status":"Closed", "Author":"maria", "DateCreated":"2015-03-11T17:33:00"}] | Returns the list of bugs sorted by date from the latest to the earliest as JSON array. Each bug holds Id, Title, Status, Author and DateCreated. Author can be existing username or null (anonymous bug). All dates are displayed in ISO 8601 format. |

5 score

### Get Bug Details by ID

Write a REST service to get bug details by ID.

|  |  |  |  |
| --- | --- | --- | --- |
| Request | GET /api/bugs/*{id}* | Example | GET /api/bugs/1 |
| Response | 200 OK  {"Id":3, "Title":"Hangs on login", "Description":null, "Status":"Fixed", "Author":"nakov", "DateCreated":"2015-03-11T11:02:00", "Comments":[{"Id":8, "Text":"seems like the login service hangs", "Author":"nakov", "DateCreated":"2015-01-27T18:00:00"}, {"Id":5, "Text":"Yeah, it was the login service. Fixed.", "Author":null, "DateCreated":"2014-01-01T12:38:00"}]} | Returned when the requested bug exists. Holds the bug details as JSON object: Id, Title, Description, Status, Author, DateCreated and Comments (as JSON array). Each comment holds Id, Text, Author and DateCreated. Sort the comments by date from the latest to the earliest. If comments do not exist, return them as empty array: []. | |
| Response | 200 OK  {"Id":4, "Title":"Bug with no comments", "Description":null, "Status":"Fixed", "Author":"nakov", "DateCreated":"2015-03-11T11:02:00", "Comments":[]} | If comments do not exist, return them as empty array: []. | |
| Error Response | 404 Not Found | Returned when the requested bug 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. |

### User Login

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). |

### Submit a New Bug

Write a REST service to submit a new bug. The bug **title** and **description** are posted in the HTTP body as form data (URL-encoded). The **title** is required. The **description** is optional. New bugs take a status "Open" and the current date and time as creation date.

If a valid user's bearer authorization access\_token is given in the request headers, the bug author should be the logged-in user. Otherwise, the bug author is null (anonymous bug submission).

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/bugs  Content-Type: application/x-www-form-urlencoded  Authorization: Bearer *{access\_token}*  title=*{title}*&description=*{description}* | | POST /api/bugs  Authorization: Bearer iyA1OenDbE4aX…  title=broken+link&description=The+%22Learn+more%22+button+is+broken. |
| Response | 201 Created  Location: http://localhost:5555/api/bugs/8  {"Id":8,"Message":"Anonymous bug submitted."} | On success, when **anonymous bug** is added, the service returns 201 (Created) + a header "Location" holding the URL of the created bug + the bug Id + human readable message as JSON object in the response body. |
| Response | 201 Created  Location: http://localhost:5555/api/bugs/9  {"Id":8,"Author":"maria","Message":"User bug submitted."} | On success, when **user's bug** is added, the service returns 201 (Created) + a header "Location" holding the URL of the created bug + the bug Id + the bug Author username + human readable message as JSON object in the response body. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid bug data (e.g. empty title). |

15 score

### Edit Existing Bug

Write a REST service to edit existing bug. Bug id is passed as part of the URI. The changed bug properties are sent in the HTTP body as form data (URL-encoded). Only title, description and status might be changed. If the HTTP body holds non-empty title, it should be updated, but if title is missing, it stays unchanged. The same applies for the other properties. Don not change properties not listed in the request body.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| PATCH /api/bugs/*{id}*  Content-Type: application/x-www-form-urlencoded  title=*{new-title}*&description=*{new-description}*&status=*{new-status}* | | PATCH /api/bugs/8  description=Patched+description&status=Closed |
| Response | 200 OK  {"Message":"Bug #8 patched."} | On success, the service returns 200 (OK) + optional human-readable message, explaining that the bug was edited. |
| Error Response | 404 Not Found | Returned when the requested bug does not exists (invalid id). |
| Error Response | 400 Bad Request | Returned in case of empty request body or invalid status. |

10 score

### Delete Bug by ID

Write a REST service to delete a bug by ID. When a bug is deleted, all its comments are also deleted.

|  |  |  |  |
| --- | --- | --- | --- |
| Request | DELETE /api/bugs/*{id}* | Example | DELETE /api/bugs/8 |
| Response | 200 OK  {"Message":"Bug #8 deleted."} | On success, the service returns 200 (OK) + optional human-readable message, explaining that the bug was deleted. | |
| Error Response | 404 Not Found | Returned when the requested bug does not exists (invalid id). | |

5 score

### List Bugs by Filter

Write a REST service to list **all bugs matching given filter**. The **filter** is given as set of URL parameters. It may hold keyword, statuses and author. If a **keyword** is given, only bugs holding this keyword in its title should be returned. **Statuses** may consist of one or several statuses, separated by |, e.g. Closed|InProgress. If statuses are given, the result returned bugs should be in one of these statuses. If an **author** username is given, only bugs from the specified author should be returned.

|  |  |  |  |
| --- | --- | --- | --- |
| Request | | Example | |
| GET /api/bugs/filter  keyword=*{keyword}*&statuses=*{status1|status2|…}*&author=*{author-username}* | | GET /api/bugs/filter?keyword=link&statuses=Open|Closed&author=nakov | |
| Response | 200 OK  [{"Id":24, "Title":"broken link", "Status":"Open", "Author":"nakov", "DateCreated":"2015-04-18T10:02:23.883"}, {"Id":11, "Title":"incorrect link", "Status":"Close", "Author":"nakov",…] | | Returns the list of bugs sorted by date from the latest to the earliest as JSON array. Each bug holds Id, Title, Status, Author and DateCreated. |

10 score

### Get All Comments

Write a REST service to list all comments for all bugs.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/comments | |
| Response | 200 OK  [{"Id":20,"Text":"this works, it is not a bug", "Author":"maria", "DateCreated":"2015-04-18T10:03:07.11", "BugId":1, "BugTitle":"broken link"}, {"Id":8, "Text":"confirmed, I will fix this", "Author":"peter", "DateCreated":"2015-01-27T18:00:00", "BugId":3, "BugTitle":"Login broken"}, {"Id":6, "Text":"it works", "Author":null, "DateCreated":"2015-01-27T18:00:00", "BugId":2, "BugTitle":"Another bug"}, …] | Returns the list of comments ordered descending by date as JSON array. Each comment holds Id, Text, Author, DateCreated, BugId and BugTitle. Anonymous comments hold null as author. All dates are given in ISO 8601 format. |

5 score

### Get Comments for Given Bug

Write a REST service to list all comments for given bug.

|  |  |  |  |
| --- | --- | --- | --- |
| Request | GET /api/bugs/*{id}*/comments | Example | GET /api/bugs/1/comments |
| Response | 200 OK  [{"Id":20, "Text":"this works, it is not a bug", "Author":null, "DateCreated":"2015-04-18T10:03:07.11"}, {"Id":19, "Text":"I found it. Will be fixed in the next release", "Author":"peter", …] | Returns the list of comments for the specified bug ordered descending by date as JSON array. Each comment holds Id, Text, Author and DateCreated. Anonymous comments hold null as author. All dates are given in ISO 8601 format. | |
| Error Response | 404 Not Found | Returned when the requested bug does not exists (invalid id). | |

10 score

### Add a Comment for Given Bug

Write a REST service to add anonymous comment for specified bug.

If a valid user's bearer authorization access\_token is given in the request headers, the comment author should be the logged-in user. Otherwise, the comment should be anonymous (author is null).

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/bugs/*{id}*/comments  Authorization: Bearer *{access\_token}*  Content-Type: application/x-www-form-urlencoded  text=*Some+Comment* | | POST /api/bugs/1/comments Authorization: Bearer iyA1OenE4aX…  text=this+works%2C+it+is+not+a+bug |
| Response | 200 OK  {"Id":21,"Message":"Added anonymous comment for bug #1"} | On success, when **anonymous comment** is added, the service returns 200 (OK) + human-readable message + comment Id in the response body. |
| Response | 200 OK  {"Id":23, "Author":"nakov", "Message": "User comment added for bug #1"} | On success, when **user's comment** is added, the service returns 200 (OK) + human-readable message + comment Id + Author username in the response body. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid comment data (e.g. empty text). |
| Error Response | 404 Not Found | Returned when the specified bug does not exists (invalid id). |

15 score

### Write Integration Tests for "Get Bug Comments" Service

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

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 "Edit Bug" Service

Write **unit tests** with **mocking** for the "**Edit Bug**" 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 "EditBugUnitTestsWithMocking".

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**.