# Web Services and Cloud Sample Exam (April 2015) – Messages

Design and implement **RESTful Web Services** based on ASP.NET Web API, Entity Framework Code First and MS SQL Server for a messaging system. The messaging system hold **channels**, **users**, **channel messages** and **user messages**. **Anonymous visitors** can list, create, edit and delete channels, list channel messages, send messages to existing channels and users, register and login. Registered users can list their personal messages and can send messages to existing channels and users.

### Compile and Run the Messages Project

You are given a Visual Studio project "Messages" holding a **data layer** (holding 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, Channel, ChannelMessage and UserMessage, but these classes are unfinished. You will need to **finish the data layer** along with these classes.
* 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 Channels Service

Write a REST service to list all channels alphabetically.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/channels | |
| Response | 200 OK  [{"Id":1,"Name":"Bulgaria"},{"Id":3,"Name":"SoftUni"},[{"Id":4,"Name":"Varna"},…] | Returns the list of channels ordered alphabetically by name as JSON array. Each channel holds Id and Name. |

5 score

### Get Channel by ID

Write a REST service to get channel details by ID.

|  |  |  |  |
| --- | --- | --- | --- |
| Request | GET /api/channels/*{id}* | Example | GET /api/channels/1 |
| Response | 200 OK  {"Id":1,"Name":"Bulgaria"} | Returned when the requested channel exists. Holds the channel Id and Name as JSON object. | |
| Error Response | 404 Not Found | Returned when the requested channel does not exists (invalid channel id). | |

5 score

### Create a New Channel

Write a REST service to create new channels. Channel **name** should be **unique** in the database. The channel name is posted in the HTTP body as form data (URL-encoded).

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/channels  Content-Type: application/x-www-form-urlencoded  name=*{channel-name}* | | POST /api/channels  name=Web+Sevices |
| Response | 201 Created  Location: http://localhost:7777/api/channels/8  {"Id":8,"Name":"Web Services"} | On success, the service returns 201 (Created) + a header "Location" holding the URL of the created channel + the channel data as JSON object in the response body (Id and Name). |
| Error Response | 400 Bad Request | Returned in case of missing or invalid channel data (e.g. empty channel name). |
| Error Response | 409 Conflict | Returned in case of duplicated channel name (when a channel with the specified name already exists in the DB). |

10 score

### Edit Existing Channel

Write a REST service to edit existing channel. Channel id should be passed as part of the URI. The channel name is posted in the HTTP body as form data (URL-encoded). Channel name should be **unique** in the database.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| PUT /api/channels/*{id}*  Content-Type: application/x-www-form-urlencoded  name=*{channel-name}* | | PUT /api/channels/8  name=New+Name |
| Response | 200 OK  {"Message":"Channel #8 edited successfully."} | On success, the service returns 200 (OK) + optional human-readable message, explaining that the channel was edited. |
| Error Response | 404 Not Found | Returned when the requested channel does not exists (invalid channel id). |
| Error Response | 400 Bad Request | Returned in case of missing or invalid channel data (e.g. empty channel name). |
| Error Response | 409 Conflict | Returned in case of duplicated channel name (when a channel with the specified name already exists in the DB). |

10 score

### Delete Channel by ID

Write a REST service to delete a channel by ID. A channel can be deleted only when it is empty (it does not hold any messages).

|  |  |  |  |
| --- | --- | --- | --- |
| Request | DELETE /api/channels/*{id}* | Example | DELETE /api/channels/8 |
| Response | 200 OK  {"Message":"Channel #8 deleted."} | On success, the service returns 200 (OK) + optional human-readable message, explaining that the channel was deleted. | |
| Error Response | 404 Not Found | Returned when the requested channel does not exists (invalid channel id). | |
| Error Response | 409 Conflict  {"Message":"Cannot delete channel #1 because it is not empty."} | Returned when a non-empty channel failed to delete. Optionally return a human-readable message in the response body. | |

5 score

### Get Channel Messages

Write a REST service to list all channels alphabetically.

|  |  |  |
| --- | --- | --- |
| Request | GET /api/channel-messages/*{channelName}* | Example |
| GET /api/channel-messages/Bulgaria |
| Response | 200 OK  [{"Id":3,"Text":"Hi","DateSent":"2015-04-15T15:24:50.363","Sender":null}, {"Id":2,"Text":"hi from Peter","DateSent":"2015-04-15T15:24:14.917","Sender":"peter"},…] | Returns the list of channel messages ordered descending by date as JSON array. Each message holds Id, Text, DateSent and Sender. Anonymous messages hold null as sender. All dates are given in ISO 8601 format. |
| Error Response | 404 Not Found | Returned when the requested channel does not exists (invalid channel name). |

5 score

### Get Channel Messages with Limit

Extend the previous REST service to list all the first few channels alphabetically by a specified **limit**. The limit should be an integer number in the range [1…1000] inclusively.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| GET /api/channel-messages/*{channel}*?limit=*{limit}* | | GET /api/channel-messages/Bulgaria?limit=1 |
| Response | 200 OK  [{"Id":3,"Text":"Hi","DateSent":"2015-04-15T15:24:50.363","Sender":null}] | Returns the first limit channel messages (Id, Text, DateSend and Sender) ordered descending by date as JSON array. |
| Error Response | 404 Not Found | Returned when the requested channel does not exists (invalid channel name). |
| Error Response | 400 Bad Request | Returned when the requested limit is invalid or out of range. |

5 score

### Send Anonymous Message to Existing Channel

Write a REST service to send anonymous message to an existing channel.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/channel-messages/*{channel-name}*  Content-Type: application/x-www-form-urlencoded  text=*Some+Anonymous+Message* | | POST /api/channel-messages/Bulgaria  text=*Some+Anonymous+Message* |
| Response | 200 OK  {"Id":4, "Message":"Anonymous message sent to channel Bulgaria."} | On success, the service returns 200 (OK) + human-readable message, explaining that the message was sent. The response body should also hold the message Id. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid channel data (e.g. empty channel name). |
| Error Response | 404 Not Found | Returned when the requested channel does not exists (invalid channel name). |

5 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/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 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/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 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). |

### Send User Message to Existing Channel (after Login)

Modify your REST service to send **non-anonymous** message from logged-in user to an existing channel. Use **bearer authorization** with the access\_token from the login service to identify the **sender user**.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/channel-messages/*{channel-name}*  Authorization: Bearer *{access\_token}*  Content-Type: application/x-www-form-urlencoded  text=*Some+Message* | | POST /api/channel-messages/Bulgaria  Authorization: Bearer VccMrKjEWki…  text=*Some+Message* |
| Response | 200 OK  {"Id":5, "Sender":"peter", "Message": "Message sent to channel Bulgaria."} | On success, the service returns 200 (OK) + human-readable message, explaining that the message was sent. The response body should also hold the message Id and Sender username. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid channel data (e.g. empty channel name). |
| Error Response | 404 Not Found | Returned when the requested channel does not exists (invalid channel name). |

5 score

### Get User's Personal Messages (after Login)

Write a REST service to list the personal messages sent to an existing logged-in user. Use **bearer authorization** with the access\_token from the login service to identify the **user**.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| GET /api/user/personal-messages  Authorization: Bearer *{access\_token}* | | GET /api/user/personal-messages  Authorization: Bearer VccMrKjEWki… |
| Response | 200 OK  [{"Id":8,"Text":"hi","DateSent":"2015-04-15T20:54:17.077","Sender":null}, {"Id":2,"Text":"Hi Nakov, I am Peter from Sofia.","DateSent":"2015-04-15T15:24:30.033","Sender":"peter"}, …] | On success, the service returns 200 (OK) + a list of all messages successfully sent to the user identified by the bearer token, sorted in descending order by date. Each message should hold Id, Text, DateSent, Sender. |
| Error Response | 401 Unauthorized | Returned in case of missing or invalid bearer authorization token or invalid username. |

5 score

### Send Anonymous Personal Message

Write a REST service to send anonymous personal message to an existing user.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/user/personal-messages  Content-Type: application/x-www-form-urlencoded  recipient=*username*&text=*Some+Message* | | POST /api/user/personal-messages  recipient=*maria*&text=*Some+Message* |
| Response | 200 OK  {"Id":9, "Message":"Anonymous message sent successfully to user maria."} | On success, the service returns 200 (OK) + human-readable message, explaining that the message was sent. The response body should also hold the message Id. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid message data (e.g. empty message text or invalid recipient username). |

5 score

### Send User Personal Message (after Login)

Write a REST service to send **non-anonymous** personal message from logged-in user to an existing user. Use **bearer authorization** with the access\_token from the login service to identify the **sender user**.

|  |  |  |
| --- | --- | --- |
| Request | | Example |
| POST /api/user/personal-messages  Content-Type: application/x-www-form-urlencoded  Authorization: Bearer *{access\_token}*  recipient=*username*&text=*Some+Message* | | POST /api/user/personal-messages  Authorization: Bearer VccMrKjEWki…  recipient=*maria*&text=*Some+Message* |
| Response | 200 OK  {"Id":10, "Sender":"peter", "Message": "Message sent to user maria."} | On success, the service returns 200 (OK) + human-readable message, explaining that the message was sent. The response body should also hold the message Id and the Sender username. |
| Error Response | 400 Bad Request | Returned in case of missing or invalid message data (e.g. empty message text or invalid recipient username). |

5 score

### Write Integration Tests for "Delete Channel" Service

Write **integration tests** for the "**Delete Channel**" REST service. Ensure you cover all interesting cases. Put your tests in a new class called "ChannelIntegrationTests".

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.

10 score

### Write Unit Tests for "Get Channel by ID" Service

Write **unit tests** with **mocking** for the "**Get Channel by ID**" 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 "ChannelUnitTestsWithMocking".

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**.