Telerik Academy Blogging System

Task Description

Implement the server-side part of a blogging system application.

Blogging System Description

Using a centralized server the users of the blogging system can **register**, **login** and **logout**. Each registered user can **create posts** and **leave comments on posts**. Registered users can view **all posts**, **their comments** and **leave a comment** on a post. Each post has zero or more tags.

Registered users have **unique username** and **display name**, and **authentication code**.

Posts have a **title**, **text** of the post, **postdate**, a set of **tags** and the user that **created the post**. Each post have a set of **comments**, holding user **responses** to the post.

Blogging System Architecture:

The architecture of the blogging system application consists of two parts: server and client. The server exposes **RESTful API** for the client to consume.

Your task is to implement the RESTful API of the application.

Application Logic

The application works as follows:

- Users register/login in the blogging system
    - They **receive session key** for later authentication
    - **Username** and **Nickname** are **unique and cannot be duplicated**
- Users can view **all posts** and **their comments**
- Users can **create new posts**
    - Tags of a post are **unique and cannot be duplicated**
    - All words from the title are also tags
- Users can **comment on any post**

1. Task 1 – Create a Database

Create a MS SQL database to hold the data about the blogging system application.

Use EntityFramework code-first or database first approach.

Up to 15 points

2. Task 2 – Create the service endpoints

Develop the **RESTful API** of the blogging system application. Use WebAPI.

All services, except Register and Login, need a **sessionKey of a logged user**. The session key can be passed either with a **queryParameter** (as a suffix in the service's url, i.e. '`api/posts&sessionKey=...`') or with a **custom HTTP header**, i.e. '`X-sessionKey:` …' Don't mix and match both, use just one of them. **Up to 65 points**

The services have the following endpoints:

HTTP Method	Web service endpoint	Description	Points
POST	`api/users/register`	Registers a **new user** in the blogging system	5 points
POST	`api/users/login`	Logs in an **existing user** in the blogging system	5 points
PUT	`api/users/logout`	**Logs out** a user from the blogging system	3 points
POST	`api/posts`	Creates a **new post**	13 points
GET	`api/posts`	Gets **all posts**, sorted by their postdate	8 points
GET	`api/posts?page=P&count=C`	Gets the posts **at positions from P*C to (P+1)*C**. The post are sorted by postdate and are at most C.	5 points
GET	`api/posts?keyword=web-services`	Gets all posts **containing "web-service" in their title.** The comparison is case-insensitive, and the result is sorted by postdate.	7 points
GET	`api/posts?tags=web,services`	Gets all posts **that have both tags "web" and "services"**. The comparison is case-insensitive, and the results are sorted by postdate.	7 points
PUT	`api/posts/{postId}/comment`	Leaves a **new comment** on a post with Id = postId	5 points
GET	`api/tags`	Gets **all tags**, sorted alphabetically	3 points
GET	`api/tags/{tagId}/posts`	Gets **all posts that have tag with Id = tagId**, sorted by their postdate.	4 points

For detailed description of the service see **Web-Services-Exam-aug-2013-services-description.doc.**

3. Task 3 – Create Integration Tests

Test the following:

- Correct behavior of the endpoints
    - Do the services work when correct data is passed
- Incorrect behavior of the endpoints
    - Do the services fail with a meaningful response code and message
    - Are the changes deleted from the database, in case of error

Write integration test for the following service endpoints:

Up to 20 points

Web service name	Web service endpoint	Point
Register	`POST api/users/register`	3 points
Logout	`PUT api/users/logout`	3 points
Create post	`POST api/posts`	5 points
Posts by tags	`GET api/posts?tags=web,services`	5 points
Leave a comment	`GET api/posts/{postId}/comment`	4 points

General Requirements

In your application you should keep in mind the following:

- **The endpoints must be exactly like as described in Task 2**
- **The request/response bodies of the HTTP requests must be implemented just like in the examples**
- You are **free to use**, **but not limited to**, the Repository pattern
    - Use repositories or don't use repositories
- **Catch exceptions** in the web services
    - Provide appropriate HTTP status code and message
- Don't test the controllers, instead test the endpoints through HTTP
    - Instead of testing **postsController.GetAll()**, test "**GET api/posts**

Evaluation Criteria

The evaluation criteria are as follows:

- Correct and complete fulfillment of the requirements.
- Good technical design and appropriate use of technologies.
- High-quality programming code – correctness, readability, maintainability.
- Performance – highly-efficient code.

To pass the exam you need 60 score (of 100 scores total).

Other Terms

During the exam you are allowed to use any teaching materials, lectures, books, existing source code, and other paper or Internet resources.

Direct or indirect communication with anybody in class or outside is forbidden. This includes, but does not limit to, technical conversations with other students, using mobile phones, chat software (Skype, ICQ, etc.), email communication, posting in forums, folder synchronization software (like Dropbox), etc.

Exam Duration

Students are allowed to work up to 8 hours.