Web Services and Cloud – Practical Exam – September 2014

Task description

Bulls and Cows game description

Implement the server logic of the Bulls and Cows game. The rules of the game are:

1. Both players think up a **4 digits**, consisting of the digits 0-9, with no repeating digits
2. We will name one of the players "**red player**" and the other one "**blue player**"
3. The game is **played in turns**, a player is **randomly selected** to have the first turn
4. The player who is in turn to play **tries to guess the opponent's number**:
    - He receives information about his guess in the forms of "count of bulls" and "count of cows"
    - A "bull" is a **correct digit** at the **correct position**
    - A "cow" is a **correct digit** at the **wrong position**
    - E.g. if the opponent's number is 1234 and the current player guesses "4631", then he has 1 bull (3) and 2 cows (1 and 4)
5. It becomes the other player's turn.
6. Steps 4 and 5 are repeated until someone guesses the opponents number (i.e. has 4 "bulls")

The game always ends with one player winning

Game logic

The game logic is performed as follows:

1. A **user registers** an account, providing credentials (username and password)
2. The **user can login** into their account using their credentials (username and password)
    - S/he receives an access token that is used to authenticated theirself in the system
3. An authenticated user **creates a game**
    - The game is created and is marked as **available for joining**
    - The creator user is **marked as red player**
    - S/he **provides a four non-repeating digits** that are their number (called *user-number*)
4. Another authenticated user **joins the game**
    - S/he **provides a four non-repeating digits**, that are their number (called *user-number*)
    - He is **marked as blue player**
5. When a blue player joins a game:
    - The server **randomly decides** which player starts first
        - For explanation purposes, let's assume the **red** is chosen
    - The **creator of the game receives a notification** that their game now has a blue player
    - The **player in turn receives a notification** that it is their turn in the game
    - The game can be played
6. The player in turn (in our case - the red player) **can make a guess against the number** of their opponent (in our case - the blue player)
    - They **send four non-repeating digits** (called *guess-number*)

- The server compares the *guess-number* against their opponents *user-number* and returns the corresponding bulls and cows
- If the *guess-number* is the same as the opponent's *user-number* (i.e. it has 4 bulls), the game is **finished**, and **scores are applied**
  - The winner gets a score "**won**"
  - The loser gets a score "**lost**"
- If the *guess-number* is not the same as the user-number, the server marks that the other player is in turn (in our case the blue)
- The **other player receives a notification** that their opponent have finished their turn

7. **Repeat step 6** while one of the players guesses right *the user-number* of their opponent
8. **When a player guesses the number** of their opponent:
   - The **winner** is given **a score "won"** and **receives a notification** that s/he has won the game
   - The **loser** is given **a score "lost"** and **receives a notification** that s/he has lost the game

Every user can play many games simultaneously.

Data tasks

Problem 1: Data layer (8 points)

Create a database that can be used with the Bulls and Cows game. You must use MS SQL server and Entity framework with either code-first or database-first approaches.

Problem 2: Repositories (5 points)

Create repositories using the *Repository pattern* to abstract the usage of the data layer. The usage of the *unit-of-work pattern* is not obligatory.

Web API tasks

All services have a full description in the file "Web-Services-and-Cloud-Services-Description.docx"

Problem 3: Login and register services (5 points)

Create a Login/Register RESTful services following the format:

HTTP Method	Url endpoint	Description
POST	/api/account/register	Registers a new user
POST	/token	Logs in an existing user

Problem 4: Games services (20 points)

Implement functionality for **listing/creating/joining** games.

Not authenticated users can see all games that are created, but still don't have two players (i.e. a blue player has not joined yet)

Authenticated players can:

- **See all games** that are created, but still don't have two players (i.e. a blue player has not joined yet)

- **See all games** that s/he is part of and are still not finished (i.e. they are still played)
- **Create a new game**, proving a name for the game and a *user-number* for the blue player to guess
- **Join a created game**, that is still available (i.e. a blue player has not joined yet), providing a user-number for the red player to guess
- **View the details for a game**, that s/he is in

HTTP Method	Url endpoint	Description
GET	/api/games	**Does not require authenticatin** Gets games that are in state available for joining. Return only the top 10 games, after sorting: • By game state • Then by the name of the game • Then by the date of creation • Then by the name of the red player
GET	/api/games?*page=P*	**Does not require authentication** **The same as *api/games***, but returns only the games on the page P. Each page has a size of 10 games
GET	/api/games	**Requires an authentication** Get games that either: • The authenticated user is part of and are not finished yet • Are available for joining, meaning that a blue player has yet to join the game Return only the top 10 games, after sorting: • By game state • Then by the name of the game • Then by the date of creation • The by the name of the red player
GET	/api/games?*page=P*	**Requires an authentication**. **The same as *api/games***, but returns only the games on the page P. Each page has a size of 10 games
GET	/api/games/*ID*	**Requires an authentication**. Can be called only from a player that is either red or blue player in this game. Gets the details for a started game: • Names of red and blue players • Red player's guesses and results • Blue player's guesses and results • User's user-number in the corresponding game
POST	/api/games	**Requires an authentication**. Creates a new game, proving a *name of the game* and *user-number*
PUT	/api/games	**Requires an authentication**.

		Joins a game that is available for joining, proving a *user-number*

Problem 5: Guess services (7 points)

Implement functionality for playing a game. Follow the criteria:

HTTP Method	Url endpoint	Description
POST	/api/games/*GAME_ID*/guess	**Requires an authentication**. The player in turn makes a guess for a game with ID = GAME_ID

Problem 6: Notifications services (15 points)

Implement functionality for notifications.

Notification is created when:

- A game is joined by a blue player, the red player receives a notification
- A player plays their turn, the other player receives a notification that it is their turn
- A player guesses the number of their opponent. Both players receive notifications

All returned UNREAD notifications are marked as READ

HTTP Method	Url endpoint	Description
GET	/api/notifications/	**Requires an authentication**. Returns the notifications of the authenticated user. The notifications are ordered by date and only the most recent 10 are returned
GET	/api/nofications?page=P	**Requires an authentication**. **The same as the above**, but returns the notifications from p*10 to p*11-1, i.e. if P=6, then returns notifications from 60 to 69
GET	/api/notifications/next	**Requires an authentication**. Returns a single notification - the oldest unread notification of the authenticated user

Problem 7: Scores services (5 points)

Implement web services for the high score board of the application

HTTP Method	Url endpoint	Description
GET	/scores	**Does not require an authentication**. Returns the top 10 players with highest ranks. The rank is calculated using the following formula: Rank = winsCount * 100 + lossesCount*15

WCF tasks

Problem 8: Users info services (10 points)

Implement the following services using WCF:

HTTP Method	Url endpoint	Description
GET	/users	**Does not require an authentication**. Returns the first 10 users, sorted by username
GET	/users*?page=P*	**Does not require an authentication**. **The same as with /users**, but returns the P-th page with users. i.e. if P=6, then the returned users must be from 60 to 69, when sorted alphabetically by username
GET	/users/*ID*	**Does not require an authentication**. Returns a detailed information about a user with the provided ID. The information contains the user's id, username, number of won games, number of lost games and rank

Testing tasks

Problem 9: Unit testing (7 points)

Write unit tests for the following functionality that uses the endpoints:

- /scores – gets the high score board
- /api/games – gets the available for joining games (the one without authentication)

Problem 10: Integration testing (8 points)

Write integration tests for the endpoints:

- /scores – get the high score board
- /api/games – get the available for joining games (the one without authentication)

High quality code and validation tasks

Problem 11: High-quality code (10 points)

- Provide validation on all needed services
- In case of error return the appropriate HTTP Status codes
- Write high-quality, abstract code that is easy to maintain and extend

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.

Other Terms

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

The Telerik Academy Anti-cheat client should be turned on during the entire exam.

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.