Entities: Players, Categories, Words, Games

Player’s fields:

PlayerID int

UserName navchar(20) unique

Password navchar(20)

Categories:

CategoryID int

CategoryName navchar(20)

Words:

WordID int

CategoryID int

Word navchar(100)

Description navchar(100)

Games:

GameID int

PlayerID int

WordID int

WrongChars int

Success bool



**Data Base Operations:**

Because of simplicity of the problem no ORM and LINQ will be used. Instead database operation will be via separate layer using hard coded SQL statements and Sql Command. (or just because of personal preference …). Methods are exposed via IHangmanRepository interface (following Repository pattern).

Players:

bool AddPlayer(string username, string password); //fails if user exist

int GetPlayerID(string username); //returns 0 if username is invalid

bool VerifyUser(int id, string username, string password); //returns false if incorrect username + password is provided

bool UpdatePlayerByID(int id, string username, string password); // returns false if failed

void RemoveUserByID(int id); // exits silently if id is not correct

Categories:

String[] GetCategories();

Words:

int[] GetWords(string category);

string GetWordByID(int id);

Games:

RecordGame(int playerID, int wordID, int wrongChars, int correctChars, bool success);

Int[] GetGamesForPlayer(int playerID);

Struct GamesResult

NumberOfGames int

NumberOfSuccess int

NumberOfFails int

NumberOfCharGuesses int

WholeWordGuesses int

GamesResult GetGamesResultForPlayer(int playerID)

**Data Access Implementation Details**

As implementation of abstraction the interface is provided by ***HangmanRepositoryFactory*** class:

***IHangmanRepository HangmanRepositoryFactory.CreateRepository()***

It creates the repository based on a key in the configuration file. While in the current implementation it connects to a local database, in further versions it may upscale to a remote service, etc. /*Dependency Inversion Principle*/. The actual implementation is in ***LocalHangmanRepository*** class. It is implemented in the assembl***y Hangman.Repository.dll***.

**UI Implementation**

UI implementation is by means of MVC project.

Controllers :

*AccountController -*  handles the user logging and registration

*HomeController -* handles selection of category and the game

*GamesResultController –* handles the games result request

Models:

*AccountViewModel and IdentityModel -*  handle the user logging view (those are auto generated by VS scaffolding)

*Game –* represent the data for a game per user.

The Game object is preserved between requests in the Session object. Alternatively all current games could be preserved in a DB table like CurrentGames …

**Files and Solutions**

*Create.sql -*  sql script for creation of the data base tables and constraints

*DBIntegrity* - sql script to add triggers

*Populate* - sql script to populate DB with sample data

*Бесеница.docx* – this document

*HangmanRepository.sln* – VS solution for HangmanRepository project with unit test project

*HangmanWeb* - VS Solution for the Web UI Implementation