WYŻSZA SZKOŁA INFORMATYKI i ZARZĄDZANIA z siedzibą w Rzeszowie

Faculty of Applied Information Technology

Field of Study: INFORMATION TECHNOLOGY

Specialty: Programming

AIKYN ARYN ,Rustem Kuntuganov

No. of student’s record book: 65387, 66256

programming languages

Supervisor: mgr inż. Marcin Jagieła

Rzeszów 2024

Family Feud Console Application Documentation

Project Overview

Aim

The aim of this project is to analyze the selected system, develop the project, and implement it in a client-server architecture.

Scope

- Preparation of analysis and design of the selected system, presented in the form of system documentation.
- Development of the server part of the system (business part of the application), providing the REST interface.
- Development of the system's client application.

System Analysis and Design

Description of the Analyzed System

The Family Feud game is a popular trivia game where players guess the most common answers to survey questions. The objective of the game is to match the top answers provided by a survey of 100 people. Points are awarded based on the popularity of the answers.

The system will consist of a server that handles game logic and scoring, and a client application that allows users to play the game by interacting with the server. The server will expose a RESTful API for the client to communicate with it.

Key Components:

- **Server**: Handles game logic, scoring, and provides a RESTful API.
- **Client**: Console application that interacts with the server via the API.

Functionalities of the Analyzed System

1. **Start a New Game**: As a user, I want to start a new game so that I can play Family Feud.
2. **Submit an Answer**: As a user, I want to submit my answer to a question so that I can earn points.
3. **Get Question**: As a user, I want to get the current question so that I know what to answer.
4. **View Score**: As a user, I want to view my current score so that I can track my progress.
5. **End Game**: As a user, I want to end the game so that I can see my final score.
6. **View Top Answers**: As a user, I want to view the top answers for a question after I submit my answer.
7. **Retry Answer**: As a user, I want to retry answering a question if my previous answer was incorrect.
8. **Track Attempts**: As a user, I want the system to track my attempts for each question.
9. **Get Game Status**: As a user, I want to get the current status of the game so that I know how many questions are left.
10. **Provide Feedback**: As a user, I want to provide feedback on the game so that I can share my experience.

Use Case Diagram

(placeholder for actual diagram)

Selected Use Cases

Use Case 1: Start a New Game

- **Success Conditions**: The game starts, and the first question is presented.
- **Failure Conditions**: The game fails to start due to a server error.
- **Basic Processing Path**:
    1. User clicks "Start Game".

    2. Server initializes game state.
    3. First question is retrieved.
    4. Question is displayed to the user.
- **Alternate Processing Path**: None.

Use Case 2: Submit an Answer

- **Success Conditions**: The answer is accepted, and points are awarded if it matches a top answer.
- **Failure Conditions**: The answer is invalid or not among the top answers.
- **Basic Processing Path**:
    1. User submits an answer.
    2. Server validates the answer.
    3. Points are awarded if correct.
    4. Feedback is given to the user.
- **Alternate Processing Path**:
    1. User submits an invalid answer.
    2. Error message is displayed.
    3. User retries.

Use Case 3: View Score

- **Success Conditions**: The current score is displayed to the user.
- **Failure Conditions**: The score fails to load due to a server error.
- **Basic Processing Path**:
    1. User requests current score.
    2. Server retrieves score.
    3. Score is displayed to the user.
- **Alternate Processing Path**: None.

Use Case 4: End Game

- **Success Conditions**: The game ends, and the final score is displayed.
- **Failure Conditions**: The game fails to end due to a server error.
- **Basic Processing Path**:
    1. User clicks "End Game".
    2. Server finalizes game state.
    3. Final score is retrieved.
    4. Final score is displayed to the user.
- **Alternate Processing Path**: None.

Test Cases

1. **Test Case 1**: Start a new game and verify the first question is displayed.
2. **Test Case 2**: Submit a correct answer and verify points are awarded.
3. **Test Case 3**: Submit an incorrect answer and verify no points are awarded.
4. **Test Case 4**: View the current score and verify it matches the expected score.
5. **Test Case 5**: End the game and verify the final score is displayed.

Activity Diagram

(placeholder for actual diagram)

Description: This diagram illustrates the main business function of submitting an answer, including the paths for correct and incorrect answers.

State Diagram

(placeholder for actual diagram)

Description: This diagram shows the states of a reactive object representing a game session, including states like "New Game", "Question Displayed", "Answer Submitted", and "Game Ended".

Component Diagram

(placeholder for actual diagram)

Description: This diagram outlines the components of the system, including the client application, the REST API, and the server-side logic.

Implementation Diagram

(placeholder for actual diagram)

Description: This diagram shows the deployment of the system, including the client application, server, and database.

Programming Technologies

- **Server**: ASP.NET Core for building the REST API.
- **Client**: .NET Console Application.
- **Database**: SQLite for storing questions and answers.
- **Additional Tools**: Entity Framework Core for database access, Swagger for API documentation.

Server Part of the System

API Project

The server exposes the following endpoints:

1. **POST /api/game/start**: Starts a new game.
2. **POST /api/game/answer**: Submits an answer.
3. **GET /api/game/question**: Retrieves the current question.
4. **GET /api/game/score**: Retrieves the current score.
5. **POST /api/game/end**: Ends the game.

System Resources

- **Game**: Represents the game session.
- **Question**: Represents a question in the game.
- **Answer**: Represents an answer to a question.
- **Score**: Represents the score of the game.

General Description of API Functionality

The API allows the client to start a new game, submit answers, retrieve the current question and score, and end the game. The API follows REST architecture principles.

Client Application

Description of the Implemented Application – User’s Manual

1. **Starting the Game**: Run the client application and select "Start Game" to begin.
2. **Answering Questions**: Type your answer and press Enter to submit.
3. **Viewing Score**: Select "View Score" to see your current score.
4. **Ending the Game**: Select "End Game" to finish and see your final score.

Technologies Used

- **.NET Console Application**: For the client-side implementation.
- **HTTP Client**: For communication with the REST API.