# Proiect de Programare / C++
# -= Sistem gestiune & interacțiune magazin =-

Student 1: Rareș Gherasă

Student 2: Marius Dînșorean

## *I. Project Description*

**Student 1:** Responsible for game management functionalities in `quiz_app.exe`. Tasks include:

- Loading questions from a file and presenting them to the player.
- Implementing the 50/50 lifeline to eliminate two incorrect options.
- Tracking player scores and updating the leaderboard.
- Handling game flow (question progression, correct/incorrect answers).

**Student 2:** Responsible for administrative functionalities in `manage_app.exe`. Tasks include:

- Managing the question bank (adding, deleting, modifying questions).
- Viewing and managing the leaderboard (displaying scores, clearing entries).
- Viewing player history based on player names.
- Ensuring data persistence through file operations.

The applications communicate via text files (`questions.txt`, `leaderboard.txt`, `history.txt`) to store questions, scores, and player history, enabling seamless interaction between the quiz and management components.

## *2. Data Structures Used by the Team*

The project uses the following C++ classes, designed to support the game's functionality and satisfy the requirement for at least two classes in a relationship (composition and association are used):

- **Question**:
  - **Attributes**:
    - `std::string question`: The question text.
    - `std::string options[4]`: Array of four answer options.
    - `int correctAnswer`: Index of the correct answer (0–3).
    - `bool used`: Flag indicating if the question has been used in a game session.
  - **Purpose**: Represents a single quiz question with its options and metadata.
  - **Relationship**: Used in composition within the `Game` class (see below).
- **Player**:
  - **Attributes**:
    - `std::string name`: The player's name.
    - `float score`: The player's score (number of correct answers).
  - **Purpose**: Stores information about a player's performance.
  - **Relationship**: Associated with the `Game` class, as multiple players' data are stored in the leaderboard.
- **Game**:
  - **Attributes**:
    - `std::vector<Question> questions`: Collection of questions for the game.
    - `Player currentPlayer`: The player currently participating.
    - `bool fiftyUsed`: Flag indicating if the 50/50 lifeline has been used.
  - **Purpose**: Manages the game state, including questions, player data, and lifeline status.
  - **Relationships**:
    - **Composition**: Contains a `std::vector<Question>` to store all questions, as questions are integral to the game's lifecycle.
    - **Association**: References a `Player` object to track the current player's progress.

These classes ensure modularity and support the project's requirements for object-oriented design and class relationships.

## *3. Structure of Files Used for Communication*

The applications communicate through the following text files, which store data persistently and enable interaction between `quiz_app.exe` and `manage_app.exe`:

- **questions.txt**:

**Purpose**: Stores the question bank for the quiz.

**Format**:

<number of questions>
<question1>,<optionA>,<optionB>,<optionC>,<optionD>,<correctAnswerIndex>
<question2>,<optionA>,<optionB>,<optionC>,<optionD>,<correctAnswerIndex>

**Example**:

10
What is the capital of Brazil?,Rio de Janeiro,Brasilia,Sao Paulo,Salvador,1
Which planet is known as the Red Planet?,Venus,Mars,Jupiter,Saturn,1

**Usage**:

`quiz_app.exe` reads this file to load questions.

`manage_app.exe` modifies this file to add, delete, or update questions.

- **leaderboard.txt**:

**Purpose**: Stores player scores for the leaderboard.

**Format**:

<number of entries>
<player_name1> <score1>
<player_name2> <score2>

**Example**:

2
John 10.0
Alice 8.0

**Usage**:

`quiz_app.exe` appends new player scores after a game.

`manage_app.exe` reads this file to display or clear the leaderboard.

- **history.txt**:

**Purpose**: Stores all game sessions for player history lookup.

**Format**:

<number of sessions>
<player_name1> <score1>
<player_name2> <score2>

**Example**:

2
John 10.0
Alice 8.0

**Usage**:

`quiz_app.exe` appends session data after each game.

`manage_app.exe` reads this file to display history for a specific player.

## *4. Commands Implemented by the Applications*

The applications expose the following command-line interfaces to interact with the quiz system, adhering to the requirement of using only command-line arguments:

**Application 1:** `quiz_app.exe` **(Game Management, Rareș Gherasă)**

This application handles the gameplay experience, including question presentation, lifeline usage, and score tracking.

- **Command**: `./quiz_app.exe play <player_name> <max_questions>`
  - **Description**: Starts a new game session for the specified player, presenting up to `max_questions` questions from `questions.txt`.
  - **Example**: `./quiz_app.exe play John 10`
    - Initiates a game for player "John" with a maximum of 10 questions.
  - **Output**: Writes the player's score to `leaderboard.txt` and `history.txt` upon completion.
- **Command**: `./quiz_app.exe use_lifeline <player_name> <question_index>`
  - **Description**: Applies the 50/50 lifeline for the specified player and question, removing two incorrect options for the question at `question_index`.
  - **Example**: `./quiz_app.exe use_lifeline John 5`
    - Uses the 50/50 lifeline for the 5th question in John's game session.
  - **Output**: Updates the game state and displays the remaining two options (via stdout or a temporary file).
- **Command**: `./quiz_app.exe answer <player_name> <question_index> <answer>`
  - **Description**: Submits an answer (`A`, `B`, `C`, or `D`) for the specified question in the player's game session.
  - **Example**: `./quiz_app.exe answer John 5 B`
    - Submits answer "B" for the 5th question in John's game.
  - **Output**: Updates the score and game state, appending to `leaderboard.txt` and `history.txt` if the game ends.

**Application 2:** `manage_app.exe` **(Administrative Functions, Marius Dînșorean)**

This application manages the question bank, leaderboard, and player history.

- **Command**: `./manage_app.exe view_questions`
  - **Description**: Displays all questions stored in `questions.txt`.
  - **Example**: `./manage_app.exe view_questions`
  - **Output**: Prints the question bank to stdout.
- **Command**: `./manage_app.exe add_question "<question>" "<optionA>" "<optionB>" "<optionC>" "<optionD>" <correctAnswerIndex>`

    - **Description**: Adds a new question to `questions.txt`.
    - **Example**: `./manage_app.exe add_question "What is 2+2?" "22" "4" "44" "2" 1`
        - Adds a question with correct answer index 1 (option B: "4").
    - **Output**: Updates `questions.txt`.
- **Command**: `./manage_app.exe delete_question <question_index>`
    - **Description**: Deletes the question at the specified index from `questions.txt`.
    - **Example**: `./manage_app.exe delete_question 5`
        - Removes the 5th question.
    - **Output**: Updates `questions.txt`.
- **Command**: `./manage_app.exe modify_question <question_index> "<question>" "<optionA>" "<optionB>" "<optionC>" "<optionD>" <correctAnswerIndex>`
    - **Description**: Modifies the question at the specified index in `questions.txt`.
    - **Example**: `./manage_app.exe modify_question 5 "What is 3+3?" "33" "6" "66" "3" 1`
        - Updates the 5th question.
    - **Output**: Updates `questions.txt`.
- **Command**: `./manage_app.exe view_leaderboard`
    - **Description**: Displays the leaderboard from `leaderboard.txt`, sorted by score.
    - **Example**: `./manage_app.exe view_leaderboard`
    - **Output**: Prints the leaderboard to stdout.
- **Command**: `./manage_app.exe clear_leaderboard`
    - **Description**: Clears all entries in `leaderboard.txt`.
    - **Example**: `./manage_app.exe clear_leaderboard`
    - **Output**: Empties `leaderboard.txt`.
- **Command**: `./manage_app.exe view_history <player_name>`
    - **Description**: Displays all game sessions for the specified player from `history.txt`.
    - **Example**: `./manage_app.exe view_history John`
        - Shows all scores for player "John".
    - **Output**: Prints the player’s history to stdout.