MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE TUNIS EL MANAR

INSTITUT SUPERIEUR D'INFORMATIQUE

RAPPORT DE PROJET

Matière : Architecture Client-Serveur & Programmation Middleware

Présenté Par

Bey Mehrez – 2 ING 01

Implémentation d'un Serveur de Jeux en Ligne avec Java RMI : PlayZone

Encadrant académique **Madame Amen Ben Hadj Ali**

Année universitaire 2024/2025

1. Présentation du projet

Dans le cadre de ce projet, j'ai décidé d'implémenter un serveur de jeux en ligne appelé **PlayZone**, basé sur Java RMI. Ce serveur permet à plusieurs utilisateurs de jouer à une variété de. L'objectif est de fournir une plateforme interactive et distribuée pour des jeux simples et amusants, tout en exploitant les concepts de programmation distribuée.

1.1. Fonctionnalités du Serveur

Pour un seul joueur, PlayZone propose les jeux suivants :

- **Number Guessing Game** : Le joueur tente de deviner un nombre aléatoire choisi par le serveur, avec des indications comme "plus haut" ou "plus bas".
- **Trivia Quiz** : Le joueur répond à des questions à choix multiple sur des thématiques précises.
- **Word Scramble** : Le joueur essaie de décoder un mot dont les lettres sont mélangées.
- **Guess the Word** : Le joueur devine un mot caché en utilisant des indices.

1.2. Déroulement d'une Partie

1. L'utilisateur entre son **nom d'utilisateur**, qui doit être unique.
2. Il sélectionne l'un des jeux proposés.
3. Il commence à jouer directement en interagissant avec le serveur.

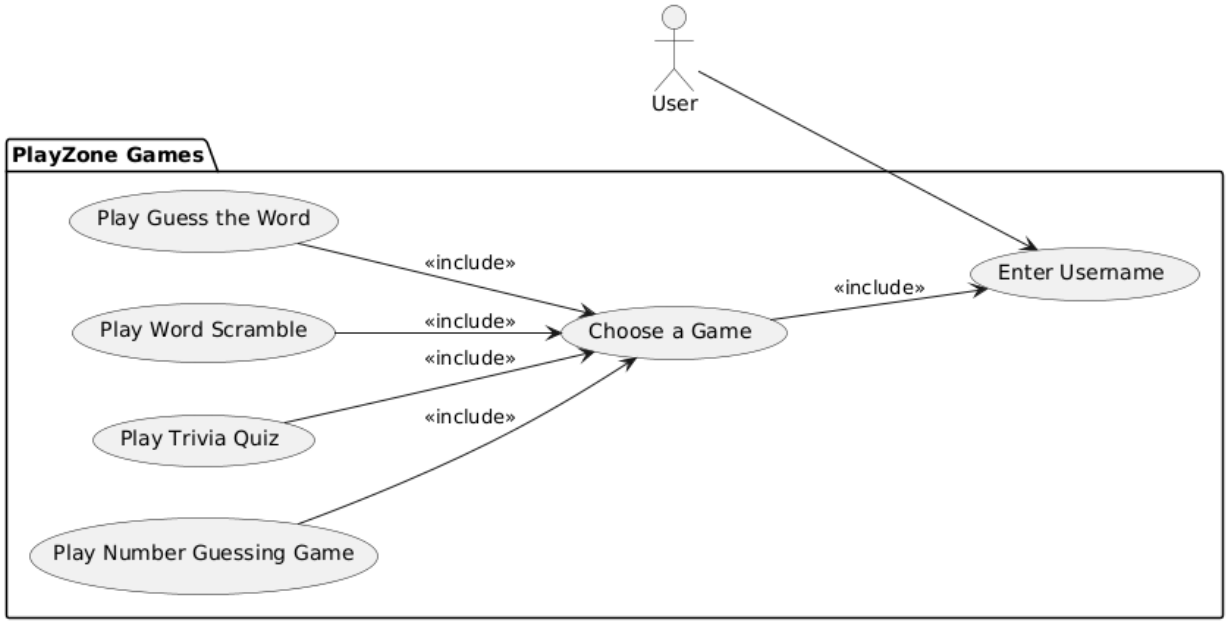
1.3 Aspects Techniques

- **Java RMI** : Le serveur utilise Java RMI (Remote Method Invocation) pour permettre la communication distante entre les clients et le serveur. Cela permet de répartir les différents composants du système sur plusieurs machines, rendant ainsi l'application réellement distribuée.
- **Multithreading (Implémentation des Threads Explicites)** : Le serveur est conçu pour être multithreadé afin de gérer simultanément plusieurs connexions de clients. Chaque client est traité dans un thread indépendant, assurant une expérience fluide sans blocage entre les utilisateurs.

- **Gestion des noms uniques** : Le serveur vérifie l'unicité des noms d'utilisateur pour éviter les conflits entre les différentes connexions. Un système d'enregistrement avec une structure de données efficace est mis en place pour stocker et valider les noms.
- **Synchronisation entre les threads** : Les threads sont gérés de manière indépendante pour s'assurer que chaque client dispose d'une session isolée (Contexte). Une synchronisation explicite est implémentée pour éviter les problèmes de concurrence dans les accès partagés. Par exemple, on utilise **ConcurrentHashMap** a la place des **hashmap** car elle est thread safe.
- **Extensibilité** : La conception du serveur est modulaire, facilitant l'ajout de nouveaux jeux ou de nouvelles fonctionnalités sans perturber le fonctionnement existant.

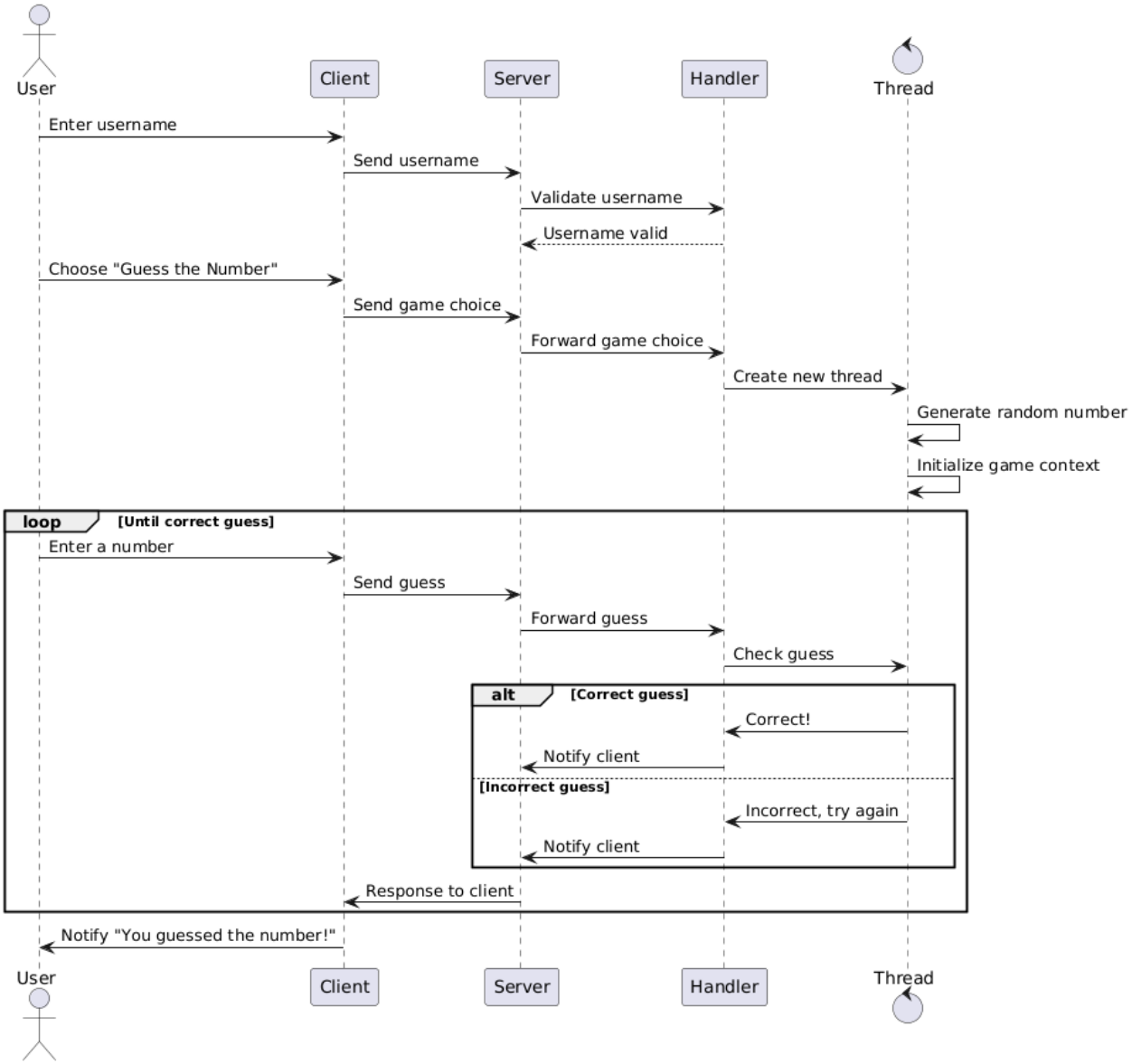
2. Architecture et conception :

2.1. Diagramme de cas d'utilisation :



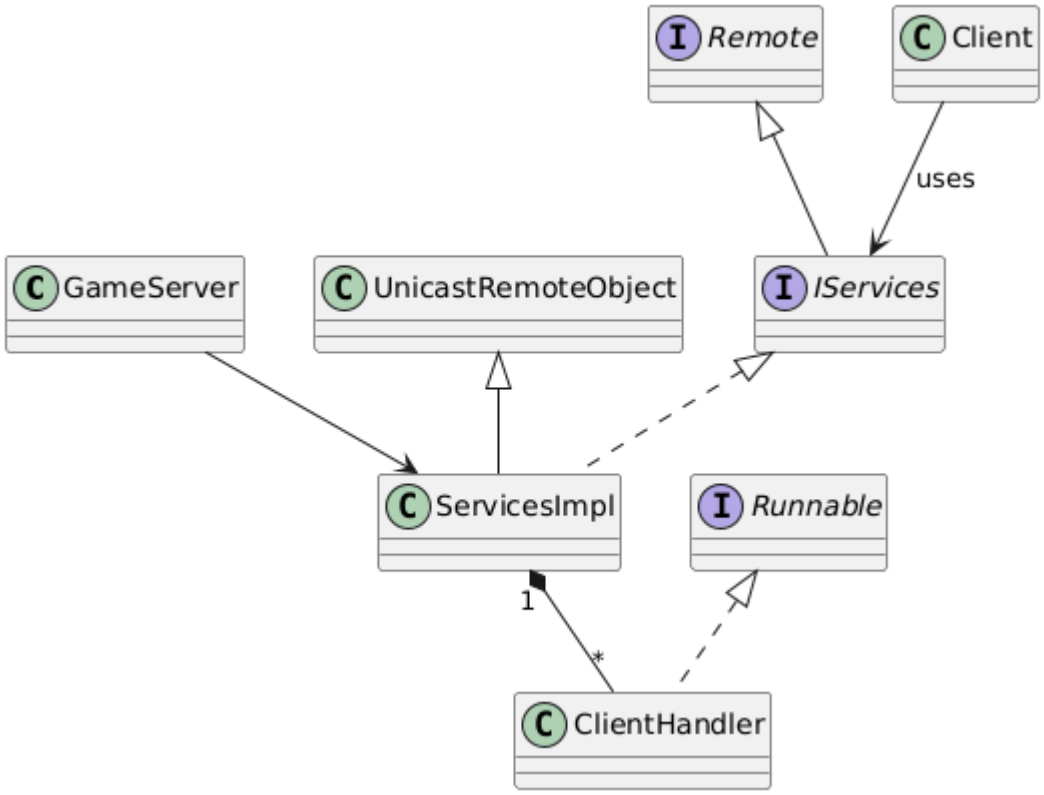
2.2. Diagramme de séquence :

Cas d'utilisation : Guess The Number

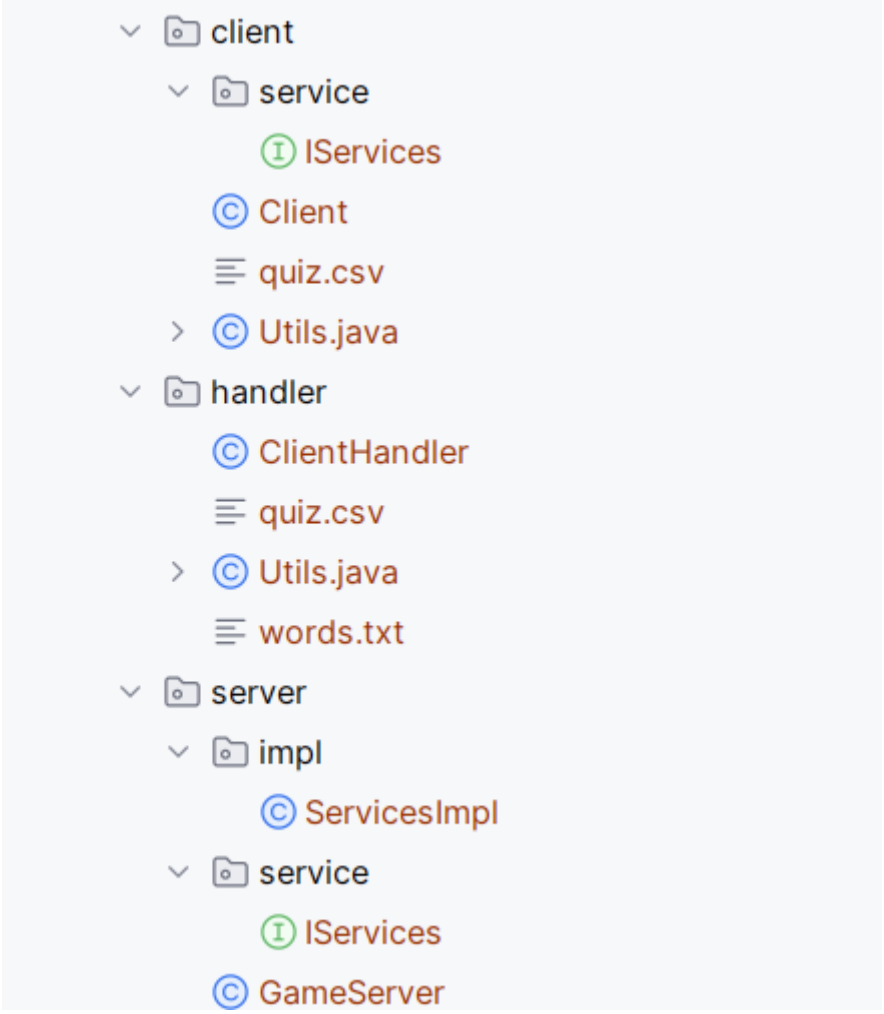


2.3. Diagramme de classe :

Comme les methodes et les attributs sont trop nombreux, on va représenter juste les classes pour que la cabture soit claire.



2.4. Organisation des Packages :



Le projet est organisé en trois principaux packages :

Package client

Ce package contient les classes et interfaces nécessaires pour le fonctionnement du client.

- service.IServices : Interface partagée entre le client et le serveur pour définir les méthodes RMI accessibles à distance(interface requise).
- Client : Classe principale qui gère la connexion au serveur et permet à l'utilisateur d'interagir avec le système.
- Utils : Classe utilitaire permettant de gérer les interactions avec les fichiers locaux (par exemple lire les fichier les questions des quizs).

Package server

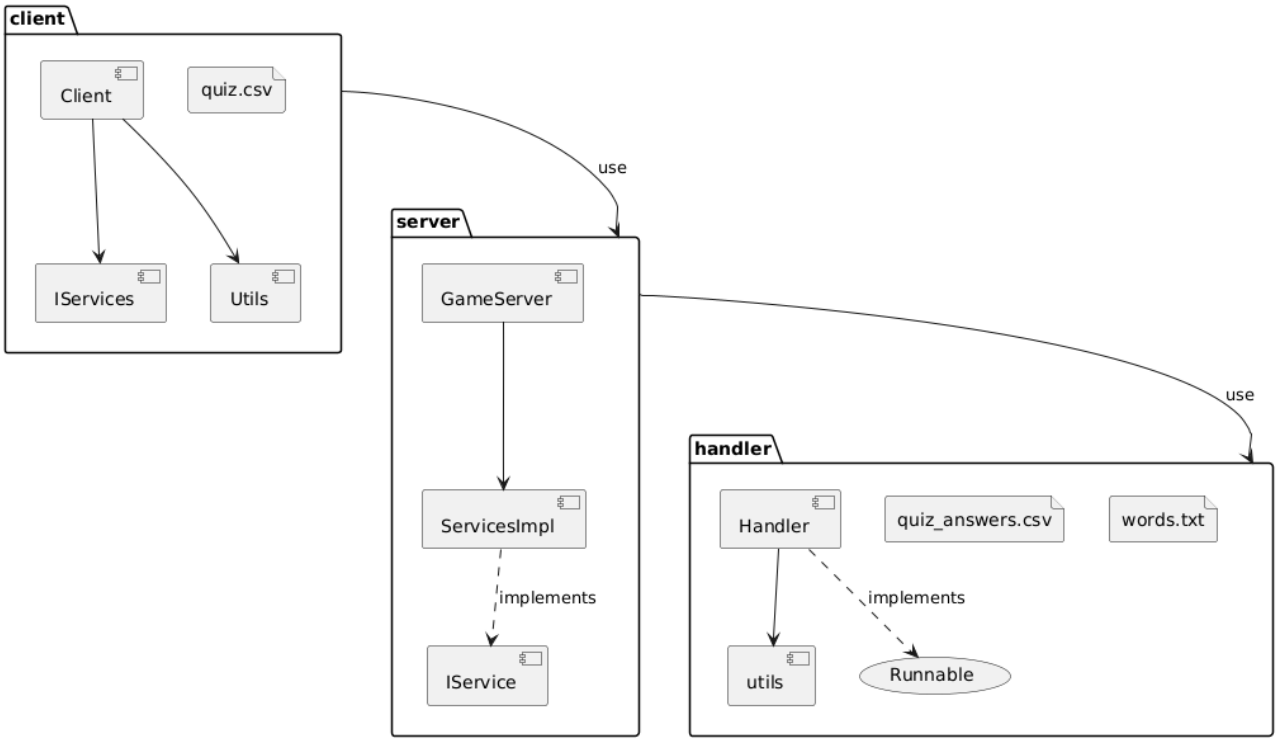
Ce package contient les classes et interfaces pour le côté serveur.

- service.IServices : Interface définissant les méthodes accessibles via RMI(interface fournie).
- impl. ServicesImpl : Implémentation de l'interface IService qui contient la logique des jeux et la gestion des utilisateurs.
- GameServer : Classe principale pour démarrer le serveur et enregistrer les services RMI.

Package handler

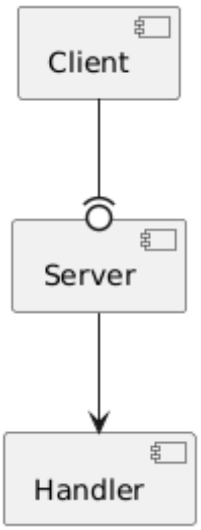
Ce package contient des classes pour la gestion des clients et des fichiers.

- Handler : Classe pour gérer les connexions clients, créer des threads pour chaque client, sauvegarder le contexte, et assurer une exécution concurrente et la synchronisation.
- Utils : Classe utilitaire pour manipuler les fichiers et les configurations côté serveur (exemple : Chosir le mot a deviner).



2.4. Diagramme de déploiement :

Client : Envoie des requêtes pour appeler les méthodes distantes.
Server : Fournit les implémentations des.
Handler : contient la logique pour interagir avec le client, traiter les requêtes et renvoyer les réponses.



3. Implémentation :

3.1. Package Client :

```
package client.service;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IServices extends Remote {
   void startGame(String username, int gameNumber) throws RemoteException;
   String guessNumber(String username, int guess) throws RemoteException;
   String wordScramble(String username, String word) throws  RemoteException;
   String getShuffleWord(String username) throws  RemoteException;
   int verifyAnswerQuiz(String username, int index, int answer) throws RemoteException;
   String getWordToGuess(String username)  throws  RemoteException;
   String wordGuess(String username, String guess) throws  RemoteException;
   void endGame(String username) throws RemoteException;

}
```

```
package client;

import server.service.IServices;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.List;
import java.util.Scanner;

public class Client {

   public static void menuDisplay(){
      System.out.println("\n================================================");
      System.out.println("       Please choose a game to play:          ");
      System.out.println("================================================");
      System.out.println("  1. Number Guessing Game");
      System.out.println("  2. Trivia Quiz");
      System.out.println("  3. Word Scramble");
      System.out.println("  4. Guess The Word");
      System.out.println("  5. Quit");
      System.out.print("> ");
   }


   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      boolean playing = true;

      try {
         // Connect to RMI registry and lookup server
         Registry registry = LocateRegistry.getRegistry("localhost", 2301);
         IServices games = (IServices) registry.lookup("Games");


         System.out.println("========================================");
         System.out.println("     WELCOME TO PLAYZONE!          ");
```

```
        System.out.println("========================================");
        System.out.println();
        System.out.println("Please enter your username to get started:");
        System.out.print("> ");
        String username = scanner.nextLine();

        if (username.isEmpty()) {
            System.out.println("\n⚠ Username cannot be empty! Please restart the game.");
            return;
        }

        System.out.println("\nHello, " + username + "! Are you ready to have some fun?");
        System.out.println();
        menuDisplay();

        int gameChoice = scanner.nextInt();
        while (gameChoice>0){
            switch (gameChoice) {
                case 1:
                    System.out.println("\n🎯 You chose: Number Guessing Game");
                    games.startGame(username,0);
                    System.out.println("Game started! Guess a number between 1 and 100.");
                    while (playing) {
                        System.out.print("Enter your guess: ");
                        int guess = scanner.nextInt();
                        String response = games.guessNumber(username, guess);
                        System.out.println(response);
                        if (response.contains("Congratulations")) {
                            playing = false;
                        }
                    }
                    games.endGame(username);
                    System.out.println("Game ended. Thanks for playing!");
                    break;
                case 2:
                    System.out.println("\n🧠 You chose: Trivia Quiz");
                    games.startGame(username,1);
                    System.out.println("Game started!");
                    List<Question> questions = Utils.readQuestionsFromFile("C:\\Users\\Mehrez\\projects\\play-
zone\\play-zone\\src\\client\\quiz.csv");
                    int score = 0;
                    int index= 0;
                    while (playing) {
                        for (Question q : questions) {
                            System.out.println(" - "+q.getQuestion());
                            for (int i = 0; i < q.getAnswers().length; i++) {
                                System.out.println((i + 1) + ". " + q.getAnswers()[i]);
                            }
                            System.out.print("Enter your answer (1-4):");
                            int answer = scanner.nextInt();
                            score += games.verifyAnswerQuiz(username,index,answer-1);
                            index++;
                        }
                        playing = false;
                    }
                    games.endGame(username);
                    System.out.println("Game ended. Thanks for playing! Your score = "+score);
                    break;
```

```
            case 3:
                System.out.println("\n🔤 You chose: Word Scramble");
                games.startGame(username,2);
                System.out.println("Game started! Guess the word : "+ games.getShuffleWord(username));
                while (playing) {
                    System.out.println();
                    System.out.print("Enter your guess: ");
                    String guess = scanner.next();
                    String response = games.wordScramble(username, guess);
                    System.out.println(response);
                    if (response.contains("Congratulations")) {
                        playing = false;
                    }
                }
                games.endGame(username);
                System.out.println("Game ended. Thanks for playing!");
                break;
            case 4:
                System.out.println("\n🔤 You chose: Guess The Word");
                games.startGame(username,3);
                String wordToGuess = games.getWordToGuess(username);
                System.out.println("Game started! Guess the word : "+ wordToGuess);
                while (playing) {
                    System.out.println();
                    System.out.print("Enter your guess: ");
                    String guess = scanner.next();
                    String response = games.wordGuess(username, guess);
                    System.out.println(response);
                    if (response.contains("Congratulations")) {
                        playing = false;
                    }
                    else{
                        if((wordToGuess.chars()
                                .filter(ch -> ch == '-')
                                .count())==1){
                            System.out.println("The word was : "+ games.getWordToGuess(username));
                            break;
                        }
                        wordToGuess = games.getWordToGuess(username);
                        System.out.println("The word : "+ wordToGuess);
                    }
                }
                games.endGame(username);
                System.out.println("Game ended. Thanks for playing!");
                break;
            case 5:
                return;
            default:
                System.out.println("\n⚠ Invalid choice. Please restart the game.");
        }
        menuDisplay();
        System.out.print("> ");
        gameChoice = scanner.nextInt();
        playing = true;
    }

    System.out.println("\n=======================================");
    System.out.println("       THANK YOU FOR PLAYING!        ");
```

```
            System.out.println("========================================");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
package client;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

public class Utils {

    public static List<Question> readQuestionsFromFile(String fileName) {
        List<Question> questions = new ArrayList<>();
        try {
            List<String> lines = Files.readAllLines(Paths.get(fileName));
            for (String line : lines) {
                if (!line.startsWith("Question")) { // Skip header line
                    String[] parts = line.split(",");
                    String questionText = parts[0];
                    String[] answers = {parts[1], parts[2], parts[3], parts[4]};
                    String correctAnswer = parts[5];
                    questions.add(new Question(questionText, answers, correctAnswer));
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return questions;
    }
}

class Question {
    private String question;
    private String[] answers;
    private String correctAnswer;

    public Question(String question, String[] answers, String correctAnswer) {
        this.question = question;
        this.answers = answers;
        this.correctAnswer = correctAnswer;
    }

    public String getQuestion() {
        return question;
    }

    public String[] getAnswers() {
        return answers;
```

```
    }
}
```

3.2. Package handler :

```
package handler;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;

public class ClientHandler implements Runnable {
    private String username;
    private int gameNumber;
    private ConcurrentHashMap<String, String> gameDetails;
    private List<Question> questions;

    private boolean running;


    public ClientHandler(String username,int gameNumber) {
        this.username = username;
        this.gameNumber = gameNumber;
        this.running = true;
        this.gameDetails = new ConcurrentHashMap<>();
        if(gameNumber == 0){
            gameDetails.put("targetNumber", String.valueOf(new Random().nextInt(100) + 1));// Random
number between 1 and 100
        } else if (gameNumber == 1) {
            questions = Utils.readQuestionsFromFile("C:\\Users\\Mehrez\\projects\\play-zone\\play-
zone\\src\\handler\\quiz.csv");

        } else if (gameNumber == 2) {
            // Path to the file
            String filePath = "C:\\Users\\Mehrez\\projects\\play-zone\\play-zone\\src\\handler\\words.txt";
            try {
                // Read all lines from the file into a List
                List<String> words = Files.readAllLines(Paths.get(filePath));
                // Check if the list is not empty
                if (words.isEmpty()) {
                    System.out.println("The file is empty. Please add words to the file.");
                    return;
                }
                // Generate a random index and get the word
                Random random = new Random();
                String randomWord = words.get(random.nextInt(words.size()));
                gameDetails.put("shuffledWord",shuffleString(randomWord));
                gameDetails.put("targetWord",randomWord);

            } catch (IOException e) {
                System.out.println("An error occurred while reading the file: " + e.getMessage());
            }
        } else if (gameNumber==3) {
            String filePath = "C:\\Users\\Mehrez\\projects\\play-zone\\play-zone\\src\\handler\\words.txt";
            try {
                // Read all lines from the file into a List
```

```
            List<String> words = Files.readAllLines(Paths.get(filePath));
            // Check if the list is not empty
            if (words.isEmpty()) {
                System.out.println("The file is empty. Please add words to the file.");
                return;
            }
            // Generate a random index and get the word
            Random random = new Random();
            String randomWord = words.get(random.nextInt(words.size()));
            gameDetails.put("wordToGuess",randomWord);
            gameDetails.put("WordToDisplay",new StringBuilder("-
".repeat(gameDetails.get("wordToGuess").length())).toString());
        } catch (IOException e) {
            System.out.println("An error occurred while reading the file: " + e.getMessage());
        }
    }
}

@Override
public void run() {
    while (running) {
        try {
            Thread.sleep(1000); // Simulate server-side waiting or processing
        } catch (InterruptedException e) {
            System.out.println("Game thread interrupted for user: " + username);
            return;
        }
    }
}

public String processGuess(int guess) {
    if (guess < Integer.valueOf(gameDetails.get("targetNumber"))) {
        return "Too low!";
    } else if (guess > Integer.valueOf(gameDetails.get("targetNumber"))) {
        return "Too high!";
    } else {
        running = false; // Stop the game if guessed correctly
        return "Congratulations! You guessed the number!";
    }
}

public String processScramble(String word) {
    if(word.equalsIgnoreCase(gameDetails.get("targetWord"))){
        running = true;
        return "Congratulations! You guessed the word!";
    }
    else{
        return "Try again!";
    }
}
public String getShuffleWord() {
    return gameDetails.get("shuffledWord");
}


public void stopGame() {
    running = false;
}
```

```
    private static String shuffleString(String input) {

        // Convert String to a list of Characters
        input = input.toLowerCase();
        List<Character> characters = new ArrayList<>();
        for (char c : input.toCharArray()) {
            characters.add(c);
        }

        // Shuffle the list
        Collections.shuffle(characters);

        // Convert the list back to String
        StringBuilder shuffledString = new StringBuilder();

        for (char c : characters) {
            shuffledString.append(c);
        }

        return shuffledString.toString();
    }


    public int verifyAnswerQuiz(String username,int index, int answer) {
        return (Integer.valueOf(questions.get(index).getCorrectAnswer().trim()) == answer)?1:0;
    }




    public String getWordToGuess() {
        String wordToGuess = gameDetails.get("wordToGuess");
        String wordToDisplay = gameDetails.get("WordToDisplay");
        int pos = new Random().nextInt(wordToGuess.length());
        while (wordToDisplay.charAt(pos)!='-'){
            pos = new Random().nextInt(wordToGuess.length());
        }
        gameDetails.put("WordToDisplay",(new
StringBuilder(wordToDisplay).replace(pos,pos+1,""+wordToGuess.charAt(pos))).toString());
        return gameDetails.get("WordToDisplay");
    }

    public String processWordGuess(String guess) {
        if(guess.equalsIgnoreCase(gameDetails.get("wordToGuess"))){
            running = true;
            return "Congratulations! You guessed the word!";
        }
        else{
            return "Try again!";
        }
    }
}
```

```
package handler;

import java.io.IOException;
import java.nio.file.Files;
```

```
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

public class Utils {

    public static List<Question> readQuestionsFromFile(String fileName) {
        List<Question> questions = new ArrayList<>();
        try {
            List<String> lines = Files.readAllLines(Paths.get(fileName));
            for (String line : lines) {
                if (!line.startsWith("Question")) { // Skip header line
                    String[] parts = line.split(",");
                    String questionText = parts[0];
                    String[] answers = {parts[1], parts[2], parts[3], parts[4]};
                    String correctAnswer = parts[5];
                    questions.add(new Question(questionText, answers, correctAnswer));
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return questions;
    }
}

class Question {
    private String question;
    private String[] answers;
    private String correctAnswer;

    public Question(String question, String[] answers, String correctAnswer) {
        this.question = question;
        this.answers = answers;
        this.correctAnswer = correctAnswer;
    }

    public String getQuestion() {
        return question;
    }

    public String[] getAnswers() {
        return answers;
    }

    public String getCorrectAnswer() {
        return correctAnswer;
    }

}
```

3.3. Package server :

```
package server.impl;

import handler.ClientHandler;
import server.service.IServices;
```

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.concurrent.ConcurrentHashMap;

public class ServicesImpl extends UnicastRemoteObject implements IServices {
    private ConcurrentHashMap<String, Thread> activeThreads; // Track active threads
    private ConcurrentHashMap<String, ClientHandler> clientHandlers; // Client-specific handlers

    public ServicesImpl() throws RemoteException {
        super();
        activeThreads = new ConcurrentHashMap<>();
        clientHandlers = new ConcurrentHashMap<>();
    }

    @Override
    public synchronized void startGame(String username, int gameNumber) throws RemoteException {
        if (activeThreads.containsKey(username)) {
            throw new RemoteException("A game is already active for user: " + username);
        }
        ClientHandler handler = new ClientHandler(username,gameNumber);
        Thread clientThread = new Thread(handler);
        activeThreads.put(username, clientThread);
        clientHandlers.put(username, handler);
        clientThread.start();
        System.out.println("Game started for user: " + username);
    }

    @Override
    public synchronized String guessNumber(String username, int guess) throws RemoteException {
        ClientHandler handler = clientHandlers.get(username);
        if (handler == null) {
            return "No active game found for user: " + username;
        }
        return handler.processGuess(guess);
    }

    @Override
    public String wordScramble(String username, String word) throws RemoteException {
        ClientHandler handler = clientHandlers.get(username);
        if (handler == null) {
            return "No active game found for user: " + username;
        }
        return handler.processScramble(word);
    }

    @Override
    public String getShuffleWord(String username) throws RemoteException {
        ClientHandler handler = clientHandlers.get(username);
        if (handler == null) {
            return "No active game found for user: " + username;
        }
        return handler.getShuffleWord();
    }

    @Override
    public int verifyAnswerQuiz(String username, int index, int answer) throws RemoteException{
        ClientHandler handler = clientHandlers.get(username);
        if (handler == null) {
```

```
            return 0;
        }
        return handler.verifyAnswerQuiz(username,index,answer);
    }

    @Override
    public String getWordToGuess(String username) throws RemoteException {
        ClientHandler handler = clientHandlers.get(username);
        if (handler == null) {
            return "No active game found for user: " + username;
        }
        return handler.getWordToGuess();
    }

    @Override
    public String wordGuess(String username, String guess) throws RemoteException {
        ClientHandler handler = clientHandlers.get(username);
        if (handler == null) {
            return "No active game found for user: " + username;
        }
        return handler.processWordGuess(guess);
    }


    @Override
    public synchronized void endGame(String username) throws RemoteException {
        Thread clientThread = activeThreads.remove(username);
        ClientHandler handler = clientHandlers.remove(username);

        if (clientThread != null && handler != null) {
            handler.stopGame(); // Stop the handler logic
            clientThread.interrupt(); // Interrupt the thread
            System.out.println("Game ended for user: " + username);
        } else {
            System.out.println("No active game found for user: " + username);
        }
    }
}
```

```
package server.service;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IServices extends Remote {
    void startGame(String username, int gameNumber) throws RemoteException;
    String guessNumber(String username, int guess) throws RemoteException;
```

```
    String wordScramble(String username, String word) throws  RemoteException;
    String getShuffleWord(String username) throws  RemoteException;
    int verifyAnswerQuiz(String username, int index, int answer) throws RemoteException;
    String getWordToGuess(String username)  throws  RemoteException;
    String wordGuess(String username, String guess) throws  RemoteException;
    void endGame(String username) throws RemoteException;

}
```

```
package server;

import server.impl.ServicesImpl;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class GameServer {
    public static void main(String[] args) {
        try {
            ServicesImpl server = new ServicesImpl();
            Registry registry = LocateRegistry.createRegistry(2301); // Start RMI registry on port 1099
            registry.rebind("Games", server); // Bind server instance to name
            System.out.println("Game server is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4. Conclusion :

Ce projet démontre l'utilisation des concepts de programmation distribuée avec Java RMI et la gestion des threads pour un système de jeux en ligne. L'architecture modulaire et évolutive permet de rajouter facilement de nouvelles fonctionnalités ou de nouveaux jeux dans le futur.