RELAZIONE TECNICA REALIZZAZIONE APPLICAZIONE CLIENT-SERVER

Creazione di un programma java che interroga il file csv

<https://stackoverflow.com/questions/55084846/fastest-way-to-read-a-csv-file-java>

Lettura file csv da java

**Relazione sul Progetto: Applicazione Client-Server per la Consultazione Remota di un File CSV**

**1. Introduzione**

L'obiettivo di questo progetto è la realizzazione di un'applicazione client-server in linguaggio Java, che consenta agli utenti di consultare in remoto un file CSV contenente informazioni sulle **strutture ricettive** della Regione Piemonte. Il file CSV fornito contiene dati strutturati relativi a hotel, agriturismi, B&B e altre strutture di ospitalità.

L’applicazione è strutturata in modo tale che:

* Il **server** carichi il file CSV, lo interpreti e renda disponibili i dati ai client.
* Il **client** invii richieste al server per ottenere informazioni specifiche.
* La comunicazione avvenga attraverso **socket TCP**.

**2. Analisi del File CSV**

Il file CSV fornito, Regione-Piemonte---Elenco-delle-strutture-ricettive.csv, contiene i seguenti campi principali:

* **ID struttura**: Identificativo univoco della struttura ricettiva.
* **Nome struttura**: Nome ufficiale della struttura.
* **Tipologia**: Hotel, B&B, Agriturismo, etc.
* **Indirizzo**: Via e numero civico.
* **Comune**: Comune in cui si trova la struttura.
* **Provincia**: Provincia di appartenenza.
* **Capacità**: Numero massimo di ospiti.

Il file CSV viene interpretato e memorizzato in una struttura dati interna per un accesso efficiente.

**3. Progettazione del Sistema**

Il sistema segue l'architettura **Client-Server** con le seguenti caratteristiche:

* **Server**:
  + Carica e interpreta il file CSV.
  + Gestisce richieste di consultazione dai client.
  + Restituisce i dati richiesti o un messaggio di errore.
  + Gestisce più client contemporaneamente con i thread.
* **Client**:
  + Invia richieste al server specificando ID struttura o altri criteri.
  + Riceve e visualizza i dati richiesti.

**4. Implementazione**

**4.1 Lettura del File CSV nel Server**

Il server utilizza la classe BufferedReader per leggere il file CSV e memorizzare i dati in una **lista di oggetti**.

import java.io.\*;

import java.util.\*;

public class CSVReader {

private List<String[]> data = new ArrayList<>();

public CSVReader(String filePath) throws IOException {

BufferedReader br = new BufferedReader(new FileReader(filePath));

String line;

while ((line = br.readLine()) != null) {

String[] values = line.split(",");

data.add(values);

}

br.close();

}

public String[] getRow(int index) {

if (index >= 0 && index < data.size()) {

return data.get(index);

}

return null;

}

}

**4.2 Implementazione del Server**

Il server attende connessioni client e gestisce richieste di dati.

import java.io.\*;

import java.net.\*;

import java.util.List;

public class Server {

private static List<String[]> data;

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(12345)) {

System.out.println("Server in ascolto sulla porta 12345...");

data = new CSVReader("Regione-Piemonte---Elenco-delle-strutture-ricettive.csv").getData();

while (true) {

Socket clientSocket = serverSocket.accept();

new ClientHandler(clientSocket, data).start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**4.3 Implementazione del Client**

Il client invia richieste e visualizza la risposta del server.

import java.io.\*;

import java.net.\*;

public class Client {

public static void main(String[] args) {

try (Socket socket = new Socket("localhost", 12345);

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Inserisci ID struttura: ");

String command = userInput.readLine();

out.println("GET\_ID " + command);

String response;

while ((response = in.readLine()) != null) {

System.out.println(response);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**5. Protocollo di Comunicazione**

**5.1 Richieste del Client**

* GET\_ID <ID> → Restituisce le informazioni della struttura con ID specificato.
* GET\_ALL → Restituisce tutte le strutture ricettive presenti nel file.

**5.2 Risposte del Server**

* Se la richiesta è valida: Dati della struttura.
* Se l'ID non esiste: ERROR: Invalid ID.

**6. Estensioni**

Possibili miglioramenti futuri:

* Implementare **filtri avanzati** (es. ricerca per provincia o tipologia di struttura).
* Creare un **client con interfaccia grafica**.
* Supportare **protocollo UDP** per migliorare le prestazioni.

**7. Conclusioni**

Questo progetto ha permesso di approfondire l'uso dei **socket in Java**, la gestione di **file CSV** e la gestione di **thread per connessioni multiple**. L'applicazione sviluppata fornisce un sistema efficiente per la consultazione remota di informazioni sulle strutture ricettive in Piemonte.

a quale di queste strutture che mi hai fatto creare prima ti riferisci?

-Classe StrutturaRicettiva

-Classe GestoreCSV

-Classe ServerStrutture

-Classe GestioneClient

-Classe ClientStrutture

## ****1. Introduzione****

Nell'era della digitalizzazione, la gestione e l'accesso ai dati in modo rapido ed efficiente rappresentano una sfida fondamentale per numerose applicazioni. Questo progetto si propone di realizzare un sistema **client-server** in linguaggio **Java**, capace di permettere la consultazione remota di un file **CSV** contenente dati strutturati sulle **strutture ricettive** della Regione Piemonte. L'obiettivo principale è consentire agli utenti di ottenere informazioni aggiornate sulle strutture di loro interesse senza dover accedere direttamente al file CSV, migliorando così l'usabilità e l'efficienza nell'elaborazione delle richieste.

L'architettura del sistema prevede due componenti principali:

* Un **server**, che si occupa di leggere il file CSV, organizzare i dati in una struttura interna e rispondere alle richieste dei client.
* Un **client**, che si connette al server tramite una comunicazione basata su **socket TCP**, invia richieste specifiche e visualizza le informazioni ricevute.

Uno degli aspetti più rilevanti di questa implementazione riguarda la gestione delle richieste multiple da parte di diversi client in simultanea, sfruttando la programmazione multithreading per garantire un servizio scalabile e performante. Inoltre, il protocollo di comunicazione tra client e server è stato progettato in modo semplice ed efficace, utilizzando comandi testuali per facilitare l'interazione e ridurre la complessità del parsing delle richieste.

Dal punto di vista pratico, questo progetto offre un'opportunità per approfondire concetti fondamentali della programmazione di rete in Java, tra cui:

* **Gestione dei socket** per la comunicazione tra processi remoti;
* **Elaborazione di file CSV**, con particolare attenzione alla robustezza del parsing e alla gestione di eventuali errori;
* **Programmazione multithreading**, necessaria per gestire più client simultaneamente;
* **Gestione degli errori e delle eccezioni**, per garantire un'esperienza utente fluida e affidabile.

Il sistema sviluppato può essere facilmente esteso con funzionalità aggiuntive, come la possibilità di filtrare i risultati per categoria, ordinare le strutture in base a specifici criteri o persino integrare un'interfaccia grafica per migliorare l'usabilità. Infine, la stessa architettura potrebbe essere adattata per supportare altri formati di dati o differenti protocolli di comunicazione, come UDP o REST API.

In questa relazione verranno dettagliate tutte le fasi di progettazione e implementazione del sistema, analizzando nel dettaglio le scelte tecniche effettuate e i risultati ottenuti.