**Relazione del Progetto**

**STUDENTE:**  
Fava Alvi (alvise.favaro@itiszuccante.edu.it)

**Obiettivo del Progetto**

L'obiettivo del progetto è la realizzazione di un'applicazione **Client-Server** utilizzando il linguaggio **Java**, che consenta agli utenti di consultare in remoto le informazioni relative agli ospedali presenti in un file CSV. Il sistema è strutturato in due componenti principali: un server che gestisce i dati e risponde alle richieste dei client, e un client che permette all'utente di inviare richieste e visualizzare i dati ricevuti dal server.

Il file CSV, che contiene i dati relativi agli ospedali in Italia, viene utilizzato per popolare una struttura dati nel server. Gli utenti possono quindi effettuare ricerche specifiche sugli ospedali, ad esempio per comune, provincia, regione, e visualizzare informazioni come il nome, la latitudine, la longitudine e altre caratteristiche degli ospedali. L'applicazione è progettata per gestire anche richieste complesse e permettere una ricerca mirata in base a vari criteri.

**Descrizione dell'Esercizio**

L'esercizio richiedeva di progettare e implementare un'applicazione **Client-Server** in Java che consenta agli utenti di consultare un file CSV contenente informazioni sugli ospedali. I principali compiti erano:

1. **Progettare un Server:**
   * Caricare il file CSV e memorizzare i dati in una struttura interna.
   * Offrire un'interfaccia di comunicazione tramite socket per ricevere richieste dal client e inviare risposte.
   * Gestire richieste per visualizzare i dati di uno o più ospedali, cercare in base a vari criteri e restituire le informazioni richieste.
2. **Progettare un Client:**
   * Permettere all'utente di inviare richieste al server, come la ricerca per comune, nome, provincia, ecc.
   * Visualizzare i dati ricevuti dal server in modo chiaro e leggibile.
3. **Protocollo di Comunicazione:**
   * Stabilire un protocollo semplice per la comunicazione tra client e server. Ad esempio:
     + Comando: GET\_ROW n per ottenere la riga n del CSV.
     + Risposta: i dati della riga n o un messaggio di errore.
4. **Gestione degli Errori:**
   * Implementare una gestione robusta degli errori, come richieste malformate o dati non trovati.

**Architettura e Progettazione**

**1. Progettazione del Server**

Il **Server** è responsabile di:

* **Caricare e interpretare il file CSV** contenente i dati sugli ospedali.
* **Memorizzare i dati** in una struttura dati appropriata, una lista di oggetti Ospedale, dove ogni oggetto rappresenta un ospedale e contiene le informazioni rilevanti (comune, provincia, nome, regione, latitudine, longitudine, ecc.).
* **Offrire una comunicazione tramite socket** per ricevere le richieste dei client e inviare risposte.
* **Gestire richieste** che permettono di ottenere i dati sugli ospedali in base a vari criteri di ricerca (comune, nome, anno di inserimento, ecc.).
* **Gestire le richieste simultanee** da parte di più client attraverso l'uso di **thread**.

Le principali funzioni del server includono:

* **Caricamento dei dati** da un file CSV.
* **Ricerca per comune, nome, provincia, regione, anno di inserimento, identificatore OSM, longitudine, latitudine**.
* **Invio di risposte** che contengono i dati richiesti o messaggi di errore.

**2. Progettazione del Client**

Il **Client** è progettato per:

* **Inviare comandi al server** utilizzando una **comunicazione socket**.
* I comandi includono:
  + GET\_ROW n: Restituisce la riga n del file CSV.
  + GET\_ALL: Restituisce tutti gli ospedali.
  + GET\_COMUNE comune: Restituisce tutti gli ospedali in un determinato comune.
  + Altri comandi simili permettono ricerche per provincia, regione, anno di inserimento, ecc.
* **Visualizzare i risultati** ricevuti dal server in modo leggibile.
* Gestire un'interfaccia a **linea di comando** in cui l'utente può interagire con il sistema.

**3. Comunicazione Client-Server**

La comunicazione tra client e server avviene tramite **socket TCP/IP**. Il client invia comandi al server e il server risponde con i dati richiesti o con un errore se la richiesta è malformata o i dati non sono trovati.

**4. Gestione degli Errori**

Il server è progettato per gestire errori comuni come:

* **Righe non valide** nel file CSV (ad esempio, dati mancanti o malformati).
* **Comandi malformati** da parte del client (ad esempio, numeri non validi per GET\_ROW).
* **Ricerche senza risultati**: in caso di una ricerca che non trova dati corrispondenti, il server invia un messaggio di errore appropriato.

**Requisiti Tecnici**

* **Socket TCP/IP**: utilizzati per la comunicazione tra client e server.
* **Threading**: il server gestisce più client simultaneamente grazie all'uso di thread.
* **Gestione di file CSV**: il server legge e analizza il file CSV contenente i dati sugli ospedali.
* **Java**: Il progetto è sviluppato in Java, utilizzando classi come Socket, ServerSocket, BufferedReader, PrintWriter, e altre per la comunicazione e la gestione dei dati.

**Estensioni e Possibili Miglioramenti**

Il sistema è progettato per essere estendibile. Possibili miglioramenti includono:

1. **Implementazione di un'interfaccia grafica (GUI)** per il client, che renderebbe l'interazione più user-friendly.
2. **Supporto per il protocollo UDP**, oltre al TCP, per migliorare l'efficienza della comunicazione in alcune situazioni.
3. **Funzionalità di ordinamento e filtraggio avanzato** dei dati sugli ospedali.
4. **Persistenza dei dati**: memorizzare i dati sugli ospedali in un database anziché in un file CSV per migliorare le performance e la scalabilità.

**Conclusioni**

L'applicazione sviluppata consente di consultare i dati relativi agli ospedali in Italia in modo semplice ed efficiente. Il sistema **Client-Server** è robusto e permette di eseguire ricerche su vari criteri (comune, provincia, anno di inserimento, ecc.). La comunicazione tra client e server è semplice e ben definita, utilizzando il protocollo **TCP/IP**. Grazie all'uso di **thread** sul server, è possibile gestire contemporaneamente più client, migliorando le performance.

Il progetto ha raggiunto gli obiettivi prefissati e offre una base solida per possibili estensioni future, come l'aggiunta di una GUI o l'implementazione del protocollo UDP.

**Codice Sorgente e Istruzioni per l'Esecuzione**

Il codice sorgente completo è allegato e comprende:

* La classe **Ospedale** per la rappresentazione degli ospedali.
* La classe **ServerMain** per la gestione del server.
* La classe **Client** per la gestione del client.
* La classe **ClientHandler** per gestire la comunicazione con ciascun client.

Le istruzioni per l'esecuzione sono contenute nel file **README** allegato, che fornisce dettagli su come compilare ed eseguire il server e il client, nonché informazioni sul formato del file CSV.