Università degli studi dell'Insubria

Climate Monitoring Progam

–

TECHNICAL MANUAL

Autori: Casalini Iacopo, Filice Martina, Radice Samuele.
Versione: September 2024.

Indice

1 Introduzione

Climate Monitoring è un progetto sviluppato nell'ambito del progetto di Laboratorio A/B per il corso di laurea in Informatica dell'Università degli Studi dell'Insubria. Il progetto è sviluppato in Java, è stato sviluppato su sistemi operativi Windows 11 e MacOs BigSur.
Il progetto si sviluppa su due rami fondamentali: i file codice che regolano lo svolgersi delle azioni mostrate all'utente e il database che ospita le credenziali e le informazioni del sistema.

1.1 Struttura della cartella compressa

La cartella `Filice_752916` contiene numerose sottocartelle necessarie per un'organizzazione ordinata dei file, di seguito le elenchiamo e ne spieghiamo il contenuto:

- `README`: file che contiene indicazioni dettagliate sull'installazione e compilazione, specificando i comandi `Maven` da usare.
- `bin`: questa cartella i file necessari per l'esecuzione effettiva del programma, quindi il file `.jar` e quattro file di testo contenenti il Data Set iniziale (che per essere modificati ed acceduti devono risiedere nella stessa directory del programma `.jar`.
- `doc`: cartella nella quale risiede anche questo file, ossia la cartella che contiene la documentazione per la comprensione di tutto quello che riguarda il Climate Monitoring Program, tra cui la Javadoc e il manuale utente.
- `src`: la cartella `src` contiene i file di codice sorgente denominati `.java` che costituiscono il vero e proprio codice del programma.
- `target`: in questa cartella sono contenuti i file `Maven` usati per compilare il progetto, lanciare Client e Server, stilare la documentazione `javadoc` e collegare il database.
- `autori.txt`: semplicemente un file di testo contenente le informazioni degli autori del programma.
- `DueBagIde`: che contiene gli unici due bug che abbiamo riscontrato sul mac dopo che abbiamo installato maven e abbiamo spostato i file Java nella cartella di maven
- `META-INF`: che contiene il file manifest con le indicazione della mainclass quando parte il file jar
- `package climatemonitoring`: che si trova nella cartella src/main/java con dentro tutto le classe .java -
- `main per l'esecuzione`: che nel nostro caso è PaginaInizale e abbiamo spiegato il motivo nel file `DueBugIDE`

2 La Struttura del programma

L'applicazione si struttura su due rami principali: il programma e il database. Attraverso continue operazione di I/O le due parti dialogano per salvare eventuali modifiche riguardo dati immessi a sistema o credenziali degli utenti stessi.

2.1 Frontend

La parte che viene mostrata all'utente attraverso GUI e parte di codice del Client, attraverso operazioni di I/O manda richieste al Server.

2.2 Backend

La parte del programma che gestisce le richieste del Client e attraverso i Socket restituisce i metodi da esso invocati, salvano eventualmente le modifiche sul DB.

2.3 Database

PostgreSQL si occupa di salvare, proteggere e rendere visibili qualora richiesto tutte le informazioni del sistema.

3 Le scelte progettuali

In questa sezione verranno elencate e spiegate le scelte di carattere progettuale usate nell'applicazione:

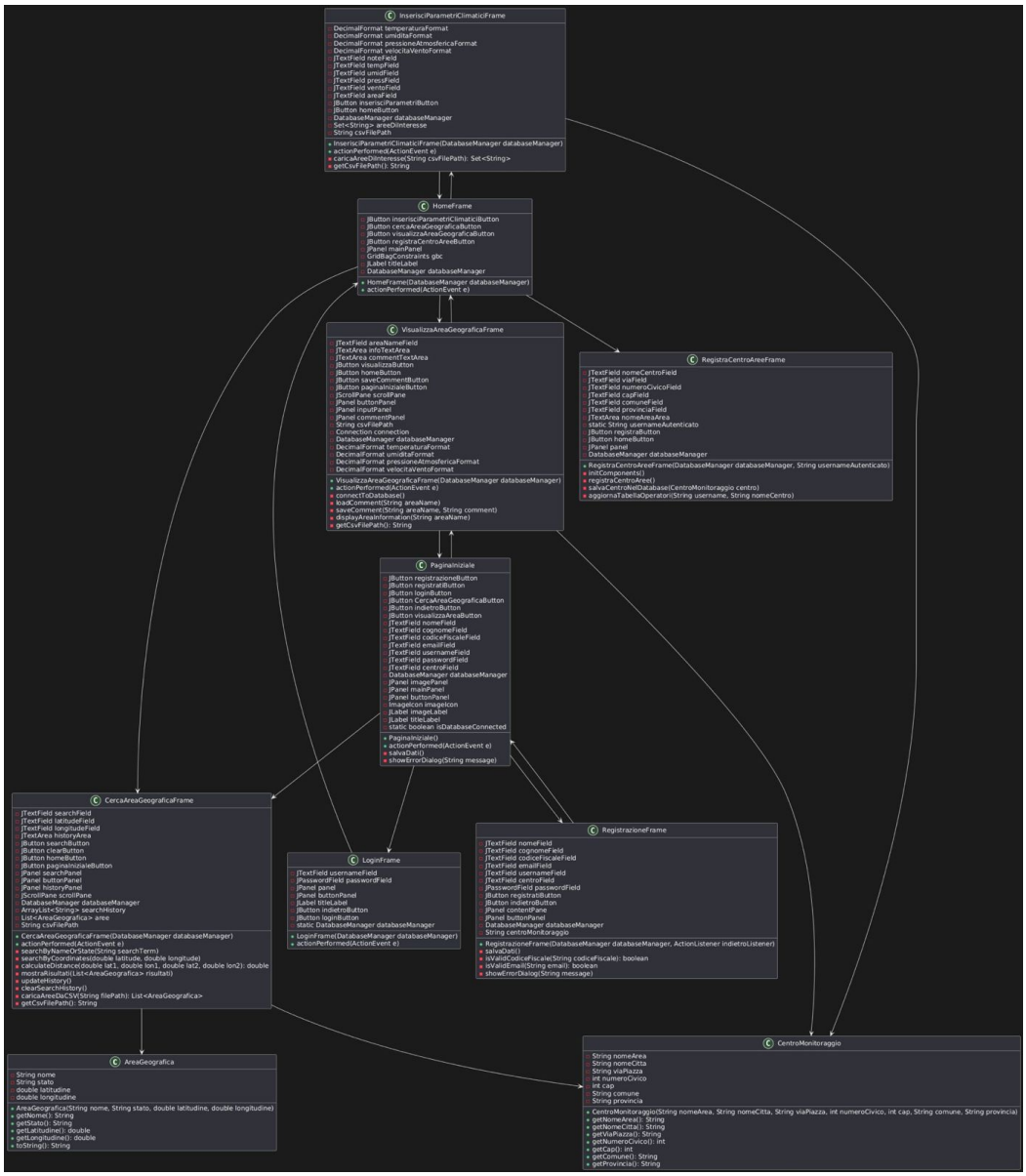
3.1 GUI

Per la Graphic User Interface abbiamo scelto la Swing perché è una libreria che ha tra i suoi pregi l'alto grado di personalizzazione, una compatibilità cross-platform, oltrechè una notevola facilità d'uso.

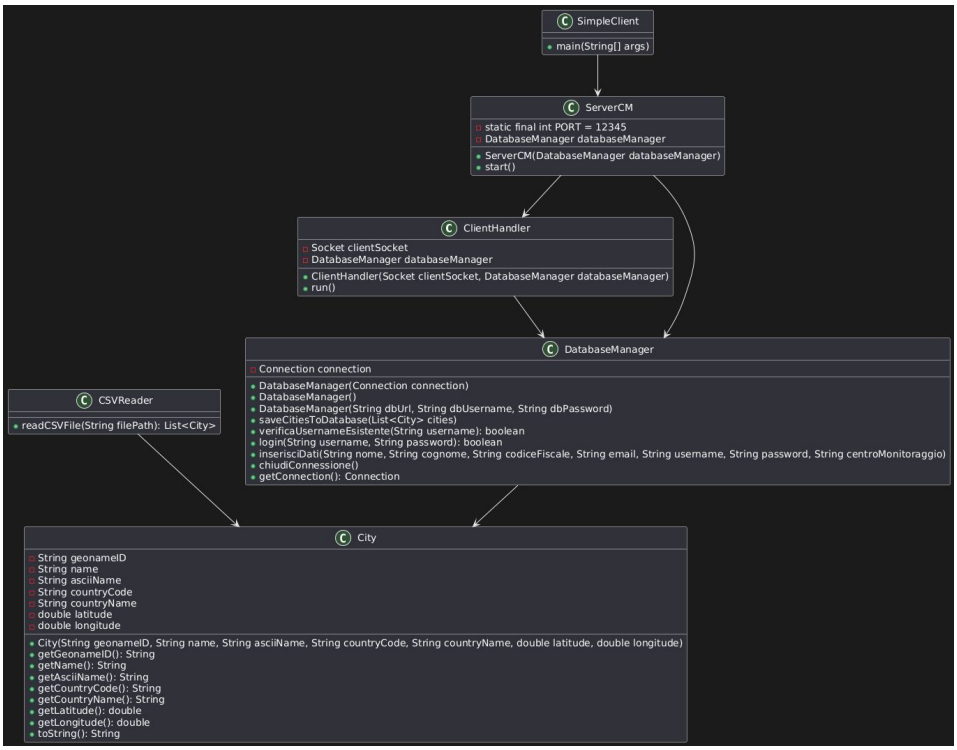
3.2 Software

Per meglio spiegare le scelte progettuali a livello Software di seguito riportiamo i diagrammi `UML` principali.

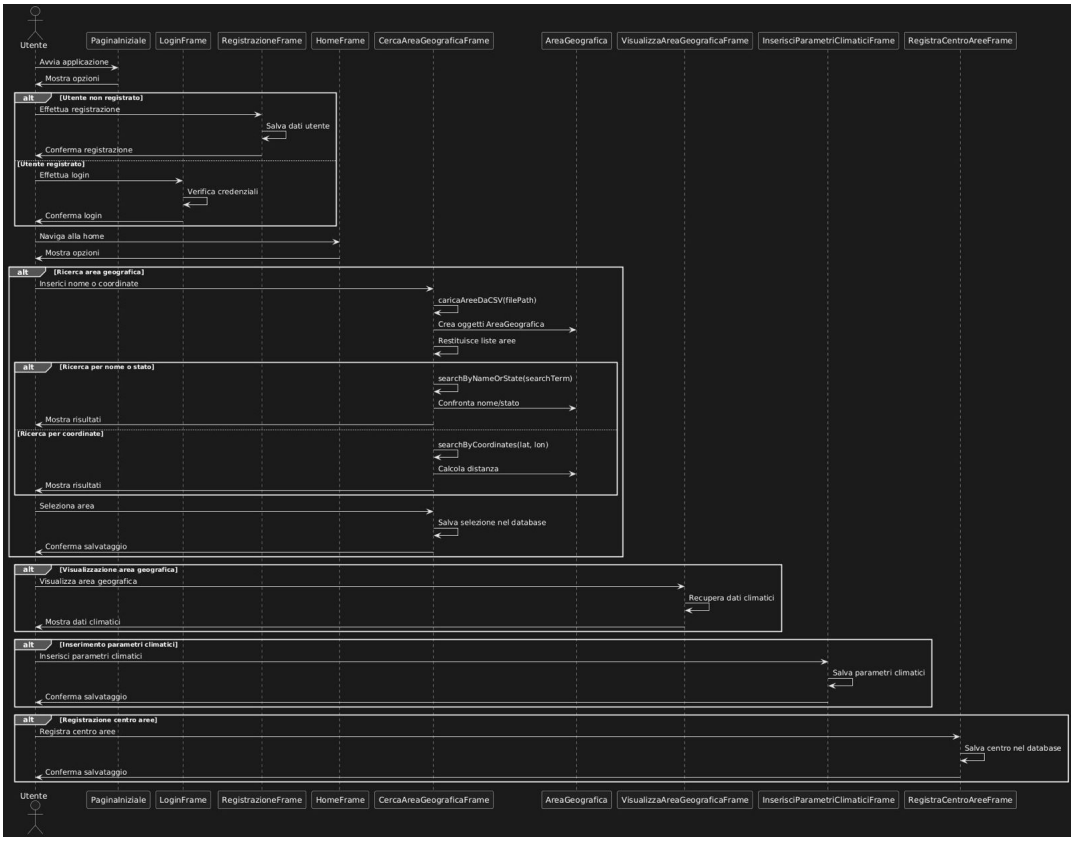
1. Class Diagram



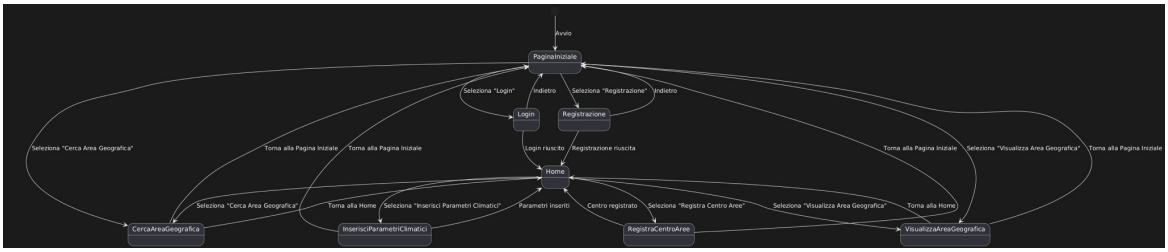
2. Class Diagram (Client/Server)



3. Sequence Diagram



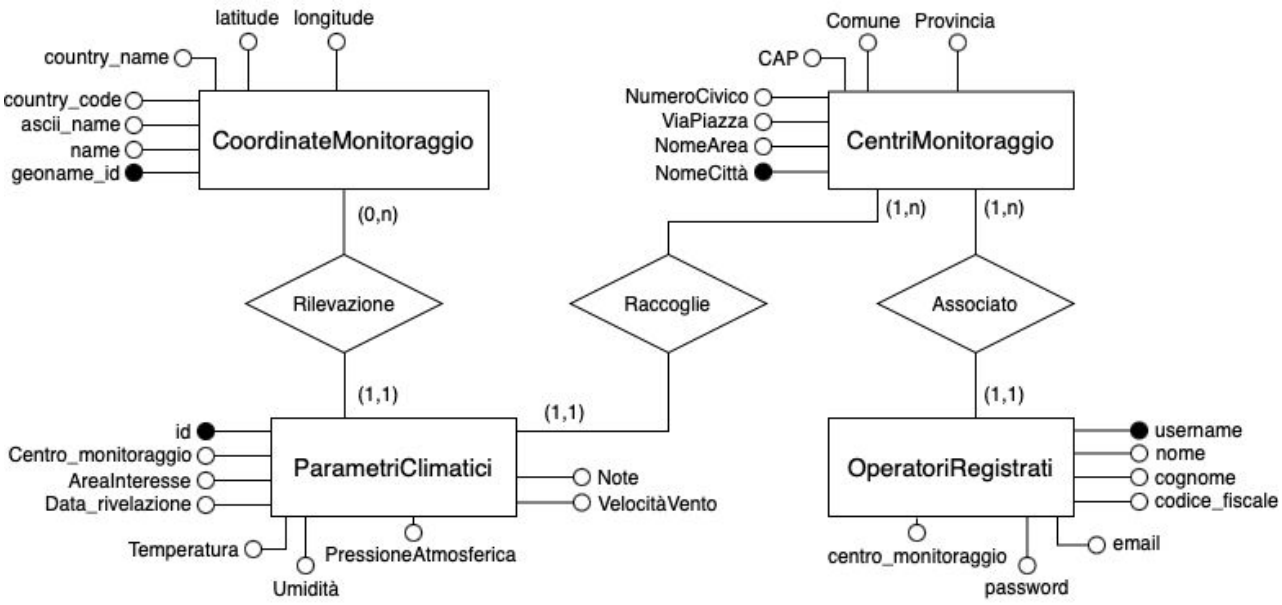
4. State Diagram



3.3 Database

Per il Database la scelta è ricaduta su `PostrgreSQL|` perchè offre un grande livello di personalizzazione, nonchè un alta compatibilità Cross-Platform e la disposizione ad essere integrato in IDE quali Visual-Studio Code o Eclipse.
Per favorire l'eventuale manutenzione, di seguito riportiamo la struttura del DB e delle tabelle inserite nel DBMS:

`Schema E-R:`



`CoordinateMonitoraggio:`

CREATE TABLE "CoordinateMonitoraggio" (
geoname_d VARCHAR(255) PRIMARY KEY,
name VARCHAR(255),
ascii_name VARCHAR(255),
country_code VARCHAR(3),
country_name VARCHAR(255),
latitude DOUBLE PRECISION,
longitude DOUBLE PRECISION);

`CentriMonitoraggio:`

CREATE TABLE "CentriMonitoraggio" (
NomeCittà VARCHAR(255) PRIMARY KEY,
NomeArea VARCHAR(255),
ViaPiazza VARCHAR(255),
NumeroCivico VARCHAR(10),
CAP VARCHAR(10),
Comune VARCHAR(255),
Provincia VARCHAR(255));

`OperatoriRegistrati`

CREATE TABLE "OperatoriRegistrati" (
username VARCHAR(255) PRIMARY KEY,
nome VARCHAR(255),
cognome VARCHAR(255),
codice_fiscale VARCHAR(16),
email VARCHAR(255),
password VARCHAR(255),
centro_monitoraggio VARCHAR(255) REFERENCES "CentriMonitoraggio"(NomeCittà));

`ParametriClimatici`

CREATE TABLE "ParametriClimatici" (
id SERIAL PRIMARY KEY,
Centro_monitoraggio VARCHAR(255) REFERENCES "CentriMonitoraggio"(NomeCittà),
AreaInteresse VARCHAR(255),
Data_rilevazione TIMESTAMP WITH TIME ZONE,
Temperatura DOUBLE PRECISION,
Umidità DOUBLE PRECISION,
PressioneAtmosferica DOUBLE PRECISION,
VelocitàVento DOUBLE PRECISION,
Note TEXT);

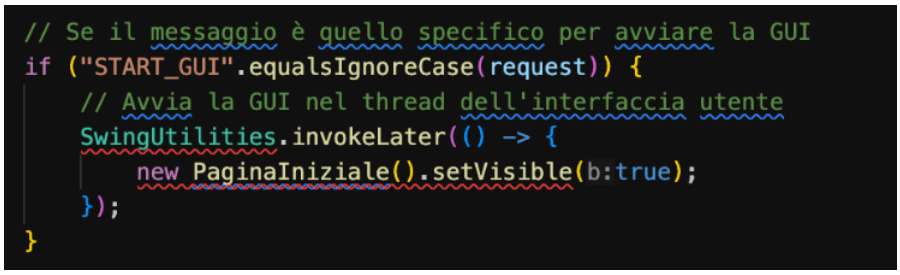
4 Le scelte architetturali

4.1 Design Patterns

Nella progettazione e realizzazione architetturale dell'applicazione sono stati usati diversi Design Pattern per codificare meglio i principali componenti del sistema. Di seguito sono riportati i principali e dove vengono usati.
NB: Informazioni più dettagliate sui Design Pattern nei file `.PDF` nella cartella compressa.

1. Singleton Pattern - crea una singola istanza di una classe per evitarne l'uso inappropriato o addirittura dannoso.
   Compare in: `CercaAreaGeograficaFrame, VisualizzaAreaGeografica, DatabaseManager.`

2. Observer Pattern - automatizza l'osservazione di cambiamenti / aggiunte / rimozioni di componenti e le notifica a tutto il sistema.
   Compare in: `CercaAreaGeograficaFrame, VisualizzaAreaGeografica, Server/Client.`



3. Producer-Consumer Pattern - gestisce il flusso di creazione e fruizione di istanze di una classe per evitare situazioni di lock.
   Compare in: `Server/Client, VisualizzaAreaGeografica.`

5 Le Scelte algoritmiche

5.1 Lettura da file

`BufferedReader`: perché permette di leggere grandi quantità di dati testuali in modo efficiente, riducendo il numero di operazioni I/O attraverso il buffering.

5.2 Strutture dati

`ArrayList`: in quanto rappresentano una struttura dati dinamica che consenta un accesso rapido agli elementi tramite indice e supporti la crescita automatica della dimensione. Offrono prestazioni migliori rispetto agli array tradizionali in scenari che richiedono frequenti operazioni di inserimento e rimozione, soprattutto quando queste operazioni avvengono alla fine della lista.

5.3 Dialogo Server/Client

`Socket`: la scelta è ricaduta sui socket per la capacità di scambiare dati in tempo reale, nonchè per la loro affidabilità e facilità di personalizzazione.

5.4 Cicli

- `For-Each`: Questo ciclo è stato fondamentale per leggere il contenuto delle LinkedList e cercarlo nei file per associare ad una riga del file le credenziali di un singolo utente (contenute appunto nella lista). Ad ogni riga è quindi associata una lista a sua volta associata ad un utente.
  Nella foto un esempio:

```
for(String riga: credenziali){ //ricerca della credenziale in ogni riga del file.
if(riga.contains(user) && riga.contains(pw)){
    trovato = true;
    registrato = true;
    System.out.println("Login avvenuto con successo!");
    break;
}
```

  Il ciclo `For-Each` in questo caso dice che in ogni Stringa contenutta nella lista "*credenziali*" viene cercato l'username e la password (metodo `.contains()`) e nel caso **entrambe le condizioni** fossero vere, il login avviene con successo.

- `Do-While`: Questo metodo è usato ogni qual volta che l'utente inserisce una propria credenziale, è fondamentale in quanto il ciclo viene ripetuto se la condizione che viene verificata **a posteriori** risulta falsa (quindi per esempio quando la credenziale è stata digitata incorrettamente o non rispetta il formato prestabilito).
  Nella foto un esempio:

```
String codice;
do{
    System.out.print("Inserire il proprio codice fiscale: ");
    codice = in.nextLine();

    if(codice.length()!= 16){
        System.out.println("Formato codice errato, riprova! ");
    }

}while(codice.equals("") || codice.length() != 16);
String cf = codice.toUpperCase(); //per averlo in maiuscolo
```

  In questo esempio il codice fiscale non dev'essere nullo e deve essere almeno di 16 caratteri, nel caso una di queste condizioni non si verifichi il ciclo si ripete con un messaggio d'errore coerente. A fine ciclo il codice viene trasformato in maiuscolo per richiamare quella che è la sua forma nella vita reale.

- `While`: Il ciclo `While` infine serve per leggere i file ed il loro contenuto e verificare che esso non sia nullo, in particolare la variabile `br` riferita alla classe `BufferedReader` serve per controllare quanto appena detto..
  Nella foto un esempio:

```
String tmp;
while((tmp = br.readLine()) != null){
    credenziali.add(tmp);
}
```

  In questo caso se il metodo `.readLine()` della classe `BufferedReader` legge il contenuto riga per riga, e quando **la riga è non nulla** la Stringa `tmp` prende il valore della stessa e viene aggiunta alla Lista "*credenziali*" con il metodo `.add()`che quindi ora contiene le credenziali di un nuovo utente.

- `try-catch`: Quando la funzione si basa sulla collaborazione di un file di testo (che si tratti di lettura o scrittura) viene eseguito un ciclo `try-catch` per effettuare tutte le istruzioni del caso e per tentare di afferrare (da qui *catch*) le eccezioni nel caso in cui si verifichino (in questo caso `FileNotFoundException`).