



UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

DIPARTIMENTO DI SCIENZE TEORICHE E APPLICATE

CORSO DI STUDIO TRIENNALE IN
INFORMATICA

Laboratorio B

Manuale Tecnico Climate Monitoring

Sviluppato da:

Fusè Lorenzo 753168

Ciminella Alessandro 753369

Dragan Cosmin 754427

Anno accademico:

2024/2025

Indice

1	Introduzione	1
2	Caratteristiche Generali	2
2.1	Panoramica	2
2.2	Tecnologie Utilizzate	2
2.3	Architettura Client-Server	3
3	Librerie	4
3.1	JavaFX	4
3.2	PostgreSQL JDBC Driver	4
3.3	Java RMI	4
3.4	Time	5
3.5	NET	5
4	Funzionalità	6
5	Architettura del Progetto	7
5.1	Model	7
5.2	View	9
5.3	Controller	10
5.4	Singleton	12
6	Progettazione	13
6.1	Use Case	13
6.2	Struttura statica	14
6.3	climate-monitoring-client	14
6.4	climate-monitoring-common	15
6.5	climate-monitoring-server	16
6.6	Struttura dinamica	18
7	Database	21
7.1	Analisi dei requisiti	21
7.2	Progettazione del database	22
7.3	Fase di ristrutturazione	22
7.4	Scelte progettuali	23
7.5	Vincoli utilizzati	24
7.6	Integrità referenziale	25

7.7	Implementazione Database - Traduzione schema ER	26
7.8	Query	30
8	Scelte algoritmiche	32
8.1	Architettura e comunicazione	32
8.2	Database	32
8.3	Algoritmi di ricerca	33
8.4	Ordinamento dei risultati	33
8.5	Aggregazione dei dati climatici	34
8.6	Validazione degli input	34
8.7	Strutture dati utilizzati	35
8.8	Cicli Utilizzati	35
8.9	for-each	36
9	Limiti del sistema	37
10	Sitografia	38

Introduzione

Climate Monitoring è un progetto accademico sviluppato durante i corsi di 'Laboratorio A' e 'Laboratorio B', finalizzato a dimostrare le competenze acquisite nei primi due anni di studi.

Il progetto prevede un sistema di monitoraggio e salvataggio di parametri climatici forniti da centri di monitoraggio, gestiti da Operatori abilitati e fruibili anche dagli utenti comuni.

1. Durante la progettazione del software, per l'interfaccia grafica è stato scelto JavaFX per la somiglianza con il codice Kotlin utilizzato nella programmazione di dispositivi mobili.

2. Il progetto è stato suddiviso in più package per migliorare la distinzione tra le varie parti.

3. Grazie all'implementazione della tecnologia RMI, il progetto è stato diviso in 'Server' e 'Client', con il server configurabile per i parametri di collegamento a PostgreSQL da parte dell'utente, permettendo così la distribuzione su più macchine, anche remote. Il progetto è stato sviluppato in Java 17, utilizzando l'interfaccia grafica JavaFX. Per lo sviluppo è stato impiegato l'IDE IntelliJ IDEA Community Edition 2024.1, con la creazione del progetto Maven per utilizzare le librerie integrate in IntelliJ. Come database è stato scelto PostgreSQL (versione > 17.x). Il progetto è stato strutturato per funzionare su diverse piattaforme ed è stato testato direttamente su:

- Windows 11 (Architettura x64)
- macOS (Architettura x64 e ARM)

Caratteristiche Generali

2.1 Panoramica

ClimateMonitoring è un'applicazione Java, sviluppata utilizzando Maven, progettata per monitorare e analizzare dati climatici. La sua architettura distribuita sfrutta un modello client-server che consente di suddividere le funzionalità tra diverse componenti comunicanti attraverso una rete. Questa struttura garantisce flessibilità, scalabilità e manutenibilità, requisiti fondamentali nei sistemi software moderni.

2.2 Tecnologie Utilizzate

Java e Maven: L'applicazione è sviluppata in Java, un linguaggio orientato agli oggetti noto per la sua robustezza e scalabilità. Maven, un tool di gestione dei progetti, semplifica il processo di costruzione, testing e distribuzione.

RMI (Remote Method Invocation): La comunicazione tra le componenti avviene tramite RMI, che consente a oggetti distribuiti su macchine diverse di interagire come fossero locali. Questo approccio astratto nasconde le complessità della rete, rendendo l'accesso ai servizi remoto intuitivo e fluido.

JavaFX: Per la progettazione dell'interfaccia utente, è utilizzata JavaFX, una libreria moderna che consente di creare applicazioni interattive e visivamente accattivanti. JavaFX integra funzionalità come:

- **FXML:** Un linguaggio dichiarativo che separa la logica di presentazione dalla logica applicativa.
- **Grafica vettoriale:** Creazione di elementi grafici scalabili e di alta qualità.
- **Animazioni ed effetti speciali:** Strumenti per animazioni fluide e interazioni dinamiche.
- **Controllo avanzato degli eventi:** Gestione efficace degli eventi e dell'interazione utente.

In **ClimateMonitoring**, JavaFX è essenziale per sviluppare interfacce utente intuitive e personalizzabili, migliorando l'esperienza dell'utente finale.

2.3 Architettura Client-Server

Server: Il server gestisce il database PostgreSQL e implementa la logica di business. I servizi sono esposti come oggetti remoti tramite RMI, offrendo ai client accesso semplificato alla funzionalità applicativa.

Client: I client sono responsabili dell'interazione con l'utente tramite un'interfaccia realizzata con JavaFX. Essi inviano richieste al server tramite RMI e visualizzano i risultati in modo chiaro e intuitivo.

Questa architettura, che separa nettamente le responsabilità tra server e client, garantisce modularità e scalabilità, facilitando l'integrazione di nuove funzionalità e la gestione dei carichi di lavoro.

Librerie

3.1 JavaFX

JavaFX Libreria per lo sviluppo di interfacce grafiche (GUI) in Java. Fornisce controlli, layout e supporto per FXML (un formato XML per definire l'interfaccia utente).

- `javafx.application.Application`
Punto di ingresso per le applicazioni JavaFX.
- `javafx.fxml.FXMLLoader`
Per caricare layout definiti in file `.fxml`
- `javafx.scene`
Per costruire scene e componenti grafici.
- `javafx.scene.control.Alert`
Per mostrare notifiche e messaggi all'utente.

3.2 PostgreSQL JDBC Driver

JDBC: Driver JDBC per PostgreSQL, che consente alle applicazioni Java di connettersi a un database PostgreSQL. Il driver fornisce le classi necessarie per stabilire una connessione, inviare query e gestire i risultati.

- `org.postgresql.Driver`
Caricato per gestire connessioni al database..
- `java.sql.Connection`
- `java.sql.PreparedStatement`
- `java.sql.ResultSet`
Utilizzati per eseguire query SQL.

3.3 Java RMI

Remote Method Invocation: RMI è una libreria Java che consente di invocare metodi su oggetti distribuiti in rete. Permette alle applicazioni Java di comunicare tra loro su una rete, rendendo possibile l'esecuzione di metodi su oggetti remoti

- `java.rmi.Naming`

- `java.rmi.registry.LocateRegistry`
Per pubblicare e recuperare oggetti remoti.
- `java.rmi.server.UnicastRemoteObject`
Per esportare oggetti remoti.

3.4 Time

LocalDate: Rappresenta una data senza informazioni relative all'ora o al fuso orario. È particolarmente utile per gestire date in modo semplice e intuitivo.

- `import java.time.LocalDate`

3.5 NET

URL: classe della libreria `java.net` che rappresenta un Uniform Resource Locator, ovvero un indirizzo web. Questa classe consente di manipolare URL e fornisce metodi per ottenere informazioni come il protocollo, l'host, il percorso e i parametri di query.

- `import java.net.URL`

Funzionalità

L'applicazione **Climate Monitoring** consente agli utenti di interagire tramite un'interfaccia client per accedere a diverse funzionalità, supportate da un server RMI che gestisce i dati climatici e le operazioni sui centri di monitoraggio. Di seguito sono riportate le funzionalità principali, classificate in base ai diritti di accesso

Ricerca Aree Geografiche

- Per Nome e Stato
- Per Denominazione
- Per Coordinate

I risultati includono informazioni come nome della città, stato, paese e coordinate (latitudine e longitudine).

Visualizzazione dei Parametri Climatici associati

È possibile visualizzare i dati climatici medi e dettagliati associati alle aree di interesse ricercata. I parametri includono: velocità del vento, umidità, pressione, temperatura, precipitazioni, altitudine e massa dei ghiacciai ed eventuali note dell'operatore.

Funzionalità Riservate agli Operatori Autorizzati

Gli operatori possono registrarsi fornendo dati personali e di accesso, come nome, cognome, email e password, oppure autenticarsi.

L'operatore, entrando nella sua area personale, può creare il suo Centro di Monitoraggio, all'interno del quale può creare delle Aree. Infine l'operatore può scegliere se inserire dei parametri climatici all'interno di aree già presenti a sistema, oppure dentro alle aree che precedentemente ha creato

Architettura del Progetto

5.1 Model

Il Model è responsabile della gestione dei dati e della logica di business. Nel progetto, include le seguenti classi principali:

- OperatoriRegistrati

Rappresenta gli operatori registrati nel sistema.

```
public class OperatoriRegistrati implements Serializable {  
    private int id;  
    private String nome;  
    private String cognome;  
    private String codice_fiscale;  
    private String email;  
    private String userid;  
    private String password;  
    // Getter e setter...  
}
```

- CoordinateMonitoraggio

Rappresenta una posizione geografica con latitudine, longitudine e altre informazioni.

```
public class CoordinateMonitoraggio implements Serializable  
↪ {  
    private int id;  
    private String nomeCitta;  
    private String stato;  
    private String paese;  
    private double latitudine;  
    private double longitudine;  
  
    @Override  
    public String toString() {  
        return String.format("%s, %s (Lat: %.2f, Long:  
↪ %.2f)",  
            nomeCitta, stato, latitudine, longitudine);  
    }  
}
```

```

    }
}

```

- CentroMonitoraggio
Modello per i centri di monitoraggio.

```

public class CentroMonitoraggio implements Serializable {
    private int id;
    private String nome;
    private String indirizzo;
    private String cap;
    private String comune;
    private String provincia;

    // Getter e setter...
}

```

- ParametriClimatici
Contiene i dati climatici raccolti.

```

public class ParametriClimatici implements Serializable {
    private int id;
    private Date data_rilevazione;
    private int vento;
    private int umidita;
    private int pressione;
    private int temperatura;
    private int precipitazioni;
    private int altitudine;
    private int massa_ghiacciai;
    private String note;

    // Getter e setter...
}

```

5.2 View

La View si occupa della presentazione visiva e dell'interazione con l'utente. Nel progetto si utilizza JavaFX per costruire l'interfaccia grafica.

File FXML

- RootLayout.fxml
- LoginView.fxml
- MainView.fxml
- ServerLogin.fxml

Caricamento delle View

Il caricamento avviene tramite FXMLLoader, come nell'esempio seguente:

```
FXMLLoader loader = new  
    ↪ FXMLLoader(getClass().getResource("/fxml/LoginView.fxml"));  
BorderPane loginView = loader.load();  
rootLayout.setCenter(loginView);
```

Scene e Stage

La gestione della finestra principale (Stage) è centralizzata nelle classi ClientCM e ServerCM.

```
@Override  
public void start(Stage stage) {  
    this.primaryStage = stage;  
    this.primaryStage.setTitle("ClimateMonitoring");  
  
    try {  
        initRootLayout();  
        loginView();  
    } catch (Exception e) {  
        showError("Errore di Inizializzazione", "Impossibile  
            ↪ avviare l'applicazione", e.getMessage());  
        System.exit(1);  
    }  
}
```

Gestione delle scene aggiuntive

Passaggio da una schermata di login alla schermata principale

```
public void showMainView() {  
    try {  
        FXMLLoader loader = new  
            ↪ FXMLLoader(getClass().getResource("/fxml/MainView.fxml"));
```

```

        BorderPane mainView = loader.load();
        rootLayout.setCenter(mainView);
    } catch (IOException e) {
        showError("Errore di caricamento", "Impossibile
        ↪ caricare la vista principale", e.getMessage());
    }
}

```

5.3 Controller

Il Controller media tra il Model e la View, gestendo la logica applicativa e le interazioni dell'utente.

ServerLogin

Gestisce l'interfaccia di login lato server.

- handleConnessioni()
- handleDisconnessione()

```

private void handleConnessioni() {
    String host = hostField.getText().trim();
    String username = usernameField.getText().trim();
    String password = passwordField.getText().trim();

    if (host.isEmpty() || username.isEmpty()) {
        appendLog("Errore: Tutti i campi sono obbligatori");
        return;
    }

    try {
        DatabaseManager dbManager =
            ↪ DatabaseManager.initialize(host, username,
            ↪ password);
        if (!dbManager.testConnection()) {
            appendLog("Errore: Impossibile connettersi al
            ↪ database.");
            return;
        }
        appendLog("Connessione stabilita con successo!");
    } catch (Exception e) {
        appendLog("Errore durante la connessione: " +
            ↪ e.getMessage());
    }
}

```

```

    }
}

MainController
Gestisce le interazioni lato client.


- handlecercaAreaGeograficaNome()
- handleVisualizzaDatiClim()
- handleCreaCentroMonitoraggio()
- handleCreaArea()


@FXML
private void handlecercaAreaGeograficaNome() {
    String cityName = searchField.getText().trim();
    String stateName = stateField.getText().trim();

    if (cityName.isEmpty() || stateName.isEmpty()) {
        showAlert(Alert.AlertType.ERROR, "Errore di
        ↳ Ricerca", "Campi vuoti", "Inserisci sia la
        ↳ citt\u00e0 che lo stato.");
        return;
    }

    try {
        List<CoordinateMonitoraggio> results =
            ↳ service.cercaAreaGeograficaNome(cityName,
            ↳ stateName);
        displayResults(results);
    } catch (RemoteException e) {
        showAlert(Alert.AlertType.ERROR, "Errore di
        ↳ Connessione", "Errore del Server",
        ↳ e.getMessage());
    }
}
}

```

5.4 Singleton

Il pattern Singleton è utilizzato per garantire che una classe abbia una sola istanza e fornire un punto di accesso globale a essa.

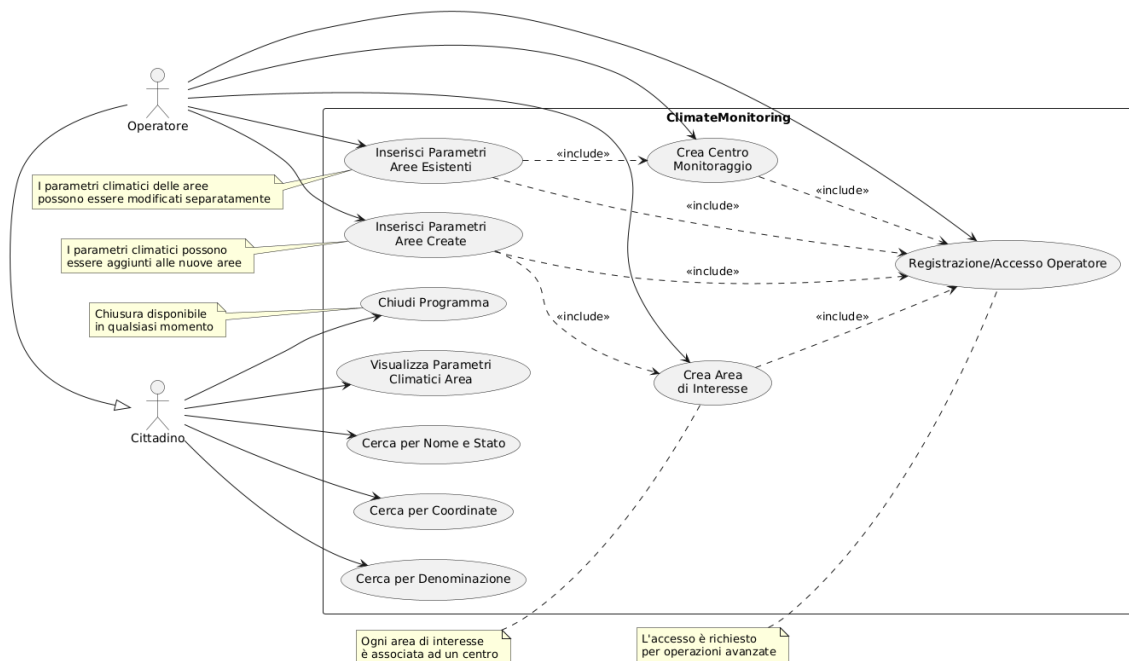
DatabaseManager implementa questo pattern:

- Il metodo `initialize()` crea un'istanza univoca.
 - Il metodo `getInstance()` fornisce l'accesso all'istanza esistente.
- ```
public static synchronized DatabaseManager getInstance() {
 if (dbManager == null) {
 throw new IllegalStateException("DatabaseManager
 ↪ deve essere inizializzato prima dell'uso");
 }
 return dbManager;
}
```

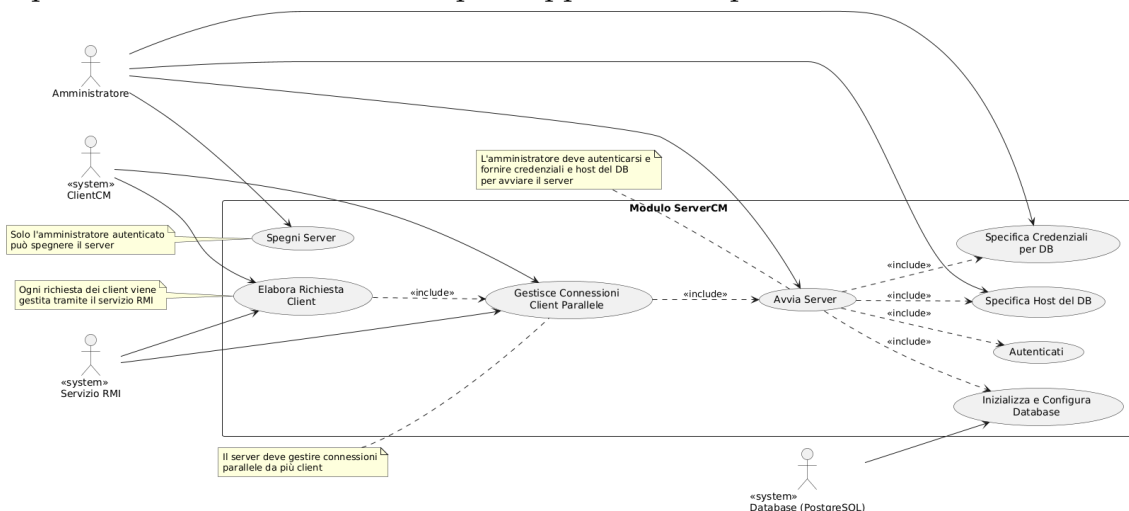
# Progettazione

## 6.1 Use Case

**Use Case lato Client:** Questo diagramma illustra le interazioni tra gli utenti e il sistema tramite il client. In particolare, evidenzia le funzionalità accessibili agli utenti generici (come la ricerca delle aree geografiche e la visualizzazione dei parametri climatici) e agli operatori autorizzati (come la registrazione, il login e la gestione delle aree di interesse e dei centri di monitoraggio).



**Use Case lato Server:** Questo diagramma rappresenta i servizi principali esposti dal server tramite RMI per supportare le operazioni richieste dal client





## 6.2 Struttura statica

Il sistema è stato sviluppato utilizzando **Maven** come sistema di build e gestione delle dipendenze, organizzato in tre moduli principali. Per la gestione dei dati, si utilizza un database **PostgreSQL**, a cui si accede tramite il modulo server.

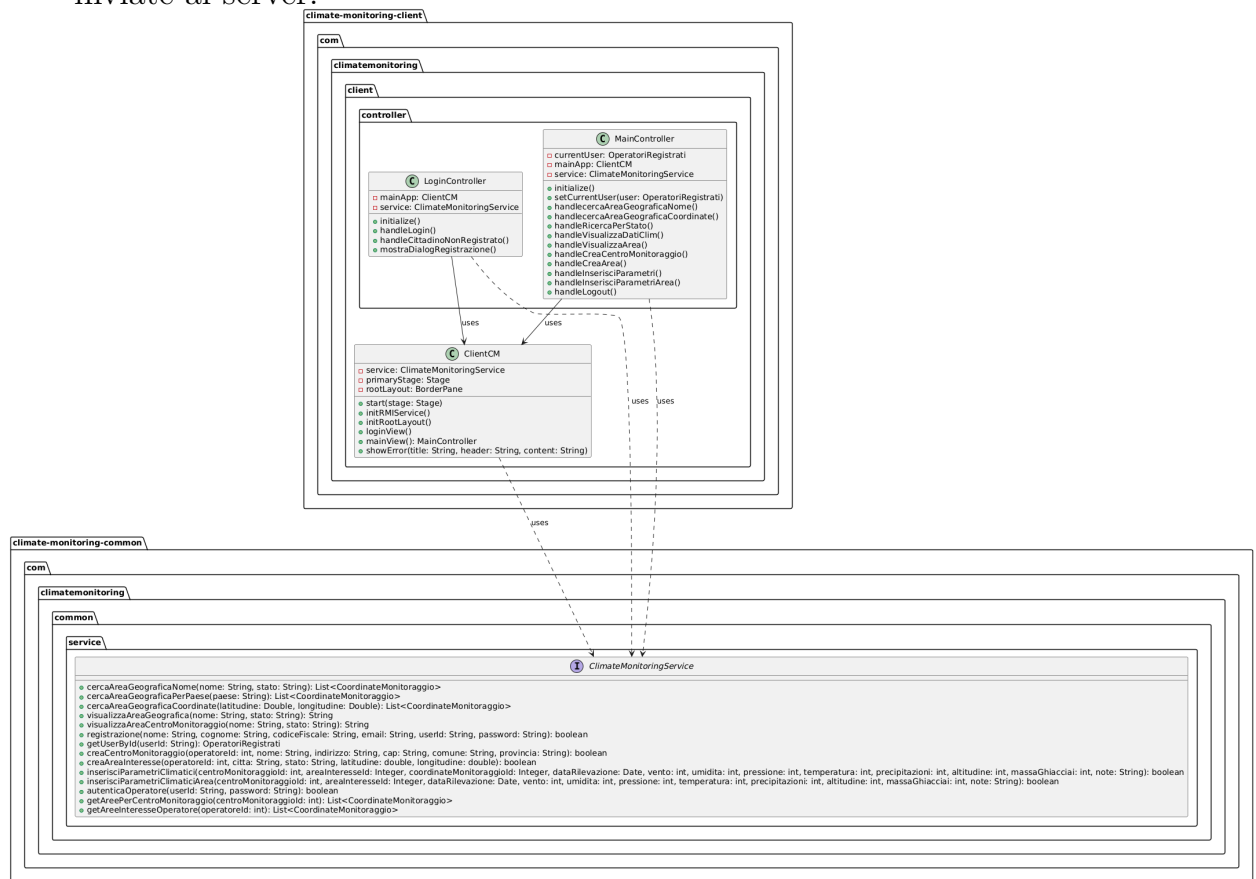
- **climate-monitoring-client**  
Gestisce l'interfaccia utente e le comunicazioni con il server.
- **climate-monitoring-common**  
Contiene i modelli condivisi tra client e server e i servizi esposti tramite RMI.
- **climate-monitoring-server**  
Implementa la logica applicativa, il collegamento al database e i servizi offerti tramite RMI.

## 6.3 climate-monitoring-client

Il modulo client è il punto di accesso per l'utente finale. È qui che vengono gestite le interazioni con l'utente tramite un'interfaccia grafica creata con JavaFX. Questo modulo permette agli utenti di autenticarsi, visualizzare e gestire informazioni legate ai dati climatici e ai centri di monitoraggio. Il client comunica con il server tramite RMI, sfruttando l'interfaccia comune definita nel modulo condiviso.

- **ClientCM**  
Entry point del modulo client.  
Avvia l'applicazione JavaFX, configura la connessione RMI e gestisce il caricamento delle viste (login e schermata principale).
- **LoginController**  
Gestisce la schermata di login. Quando un utente inserisce username e password, questo controller invia le credenziali al server per la verifica. Se tutto va a buon fine, l'utente viene reindirizzato alla schermata principale. In caso di errori, come credenziali non valide o server non raggiungibile, il controller si occupa di gestire i messaggi di errore.
- **MainController**  
Gestisce la schermata principale dell'applicazione. Questa classe consente agli utenti di effettuare operazioni come cercare dati climatici, gestire le aree di monitoraggio e registrare nuovi operatori. È il cuore della logica client, poiché coordina tutte le interazioni con l'utente e le richieste

inviare al server.



## 6.4 climate-monitoring-common

Il modulo "common" è il ponte tra il client e il server. Contiene tutti gli elementi condivisi, come i modelli dati e le interfacce dei servizi. È grazie a questo modulo che i dati possono fluire tra il client e il server in modo coerente e standardizzato.

- **CentroMonitoraggio**

Rappresenta un centro di monitoraggio climatico, con dettagli come il nome, l'indirizzo e la località. Questo modello è utilizzato sia dal client per visualizzare i centri, sia dal server per salvarli o aggiornarli nel database.

- **CoordinateMonitoraggio**

Rappresenta un'area geografica. Contiene attributi come latitudine e longitudine, ed è fondamentale per individuare le zone in cui vengono raccolti i dati climatici.

- **OperatoriRegistrati**

Modello per gli operatori registrati al sistema.

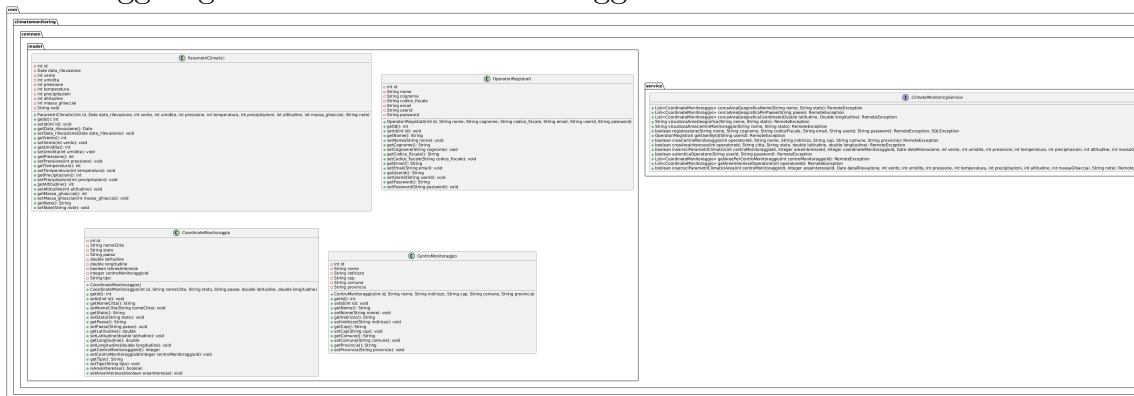
Memorizza informazioni come nome, cognome, email, userId e password.

- **ParametriClimatici**

Rappresenta i dati climatici rilevati, come temperatura, vento, pressione e umidità ecc. È grazie a questa classe che le informazioni vengono inviate dal server al client e mostrate agli utenti

- **ClimateMonitoringService**

Questa interfaccia definisce tutti i metodi remoti che il client può invocare sul server, come recuperare dati climatici, registrare un nuovo operatore o aggiungere un centro di monitoraggio



## 6.5 climate-monitoring-server

Il modulo server è il "cervello" del sistema. Gestisce tutte le operazioni richieste dai client, come la verifica delle credenziali, la registrazione di nuovi operatori e il recupero dei dati climatici. Inoltre, si occupa di interagire con il database per memorizzare e recuperare informazioni.

- **ServerLogin**

Gestisce la configurazione iniziale del server. Quando il server viene avviato, questa classe permette di configurare le credenziali per accedere al database e di avviare il servizio RMI.

- **ClimateMonitoringServiceImpl**

Classe JDBC che implementa i metodi definiti nell'interfaccia ClimateMonitoringService. Questa classe riceve le richieste dal client e si occupa di soddisfarle. Ad esempio, se il client richiede dati climatici, questa classe si collega al database tramite DatabaseManager per recuperarli e poi li restituisce al client

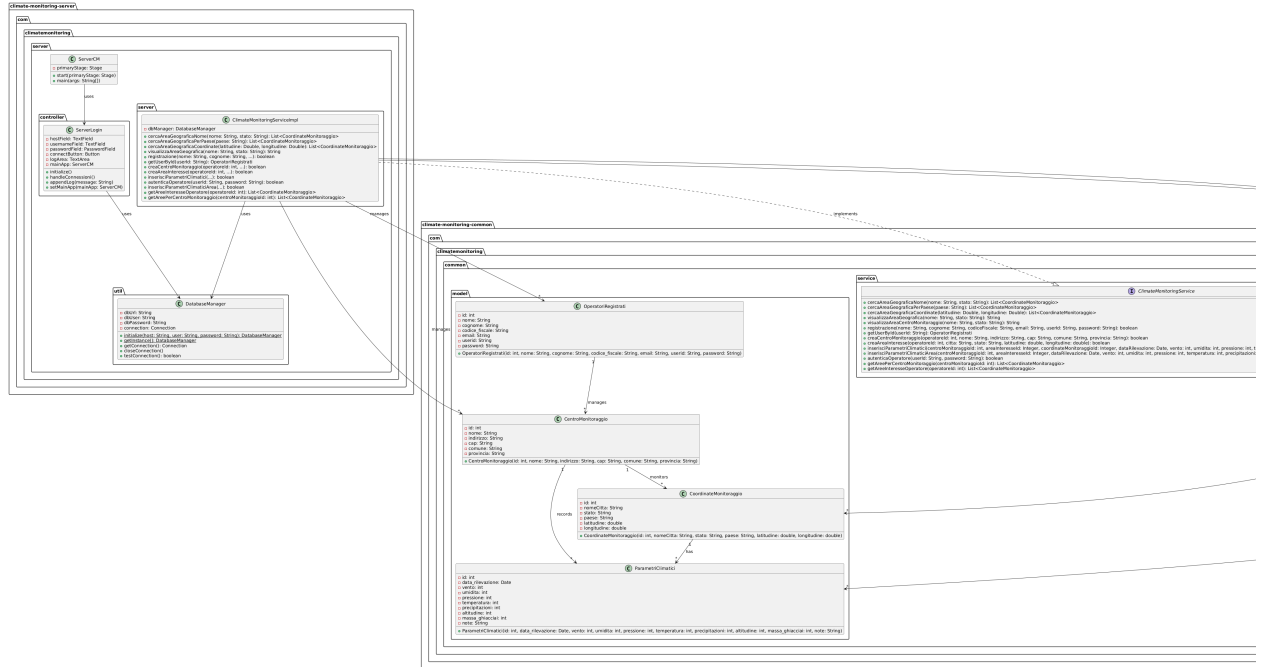
- **DatabaseManager**

Per gestire le operazioni sul database, si utilizza questa classe per stabi-

lire connessioni sicure al database PostgreSQL. Questa classe utilizza il pattern Singleton per garantire che ci sia un'unica connessione attiva al database, ottimizzando le risorse.

- **ServerCM**

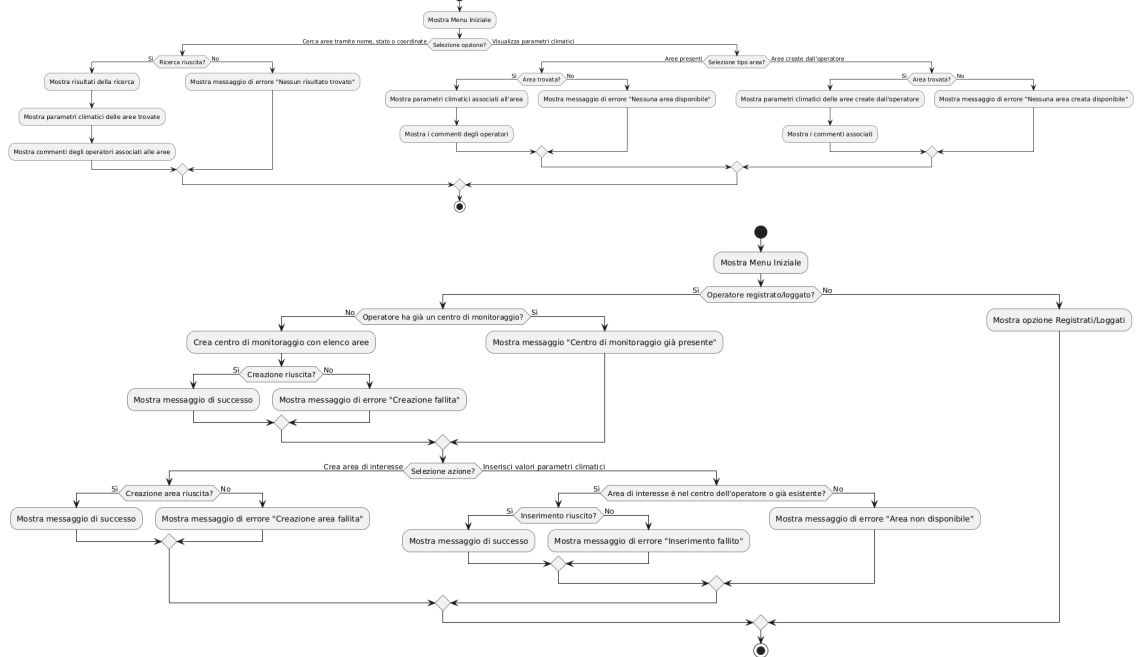
Entry point del server. Questa classe avvia il servizio RMI e si assicura che tutto sia configurato correttamente, inclusa la connessione al database.



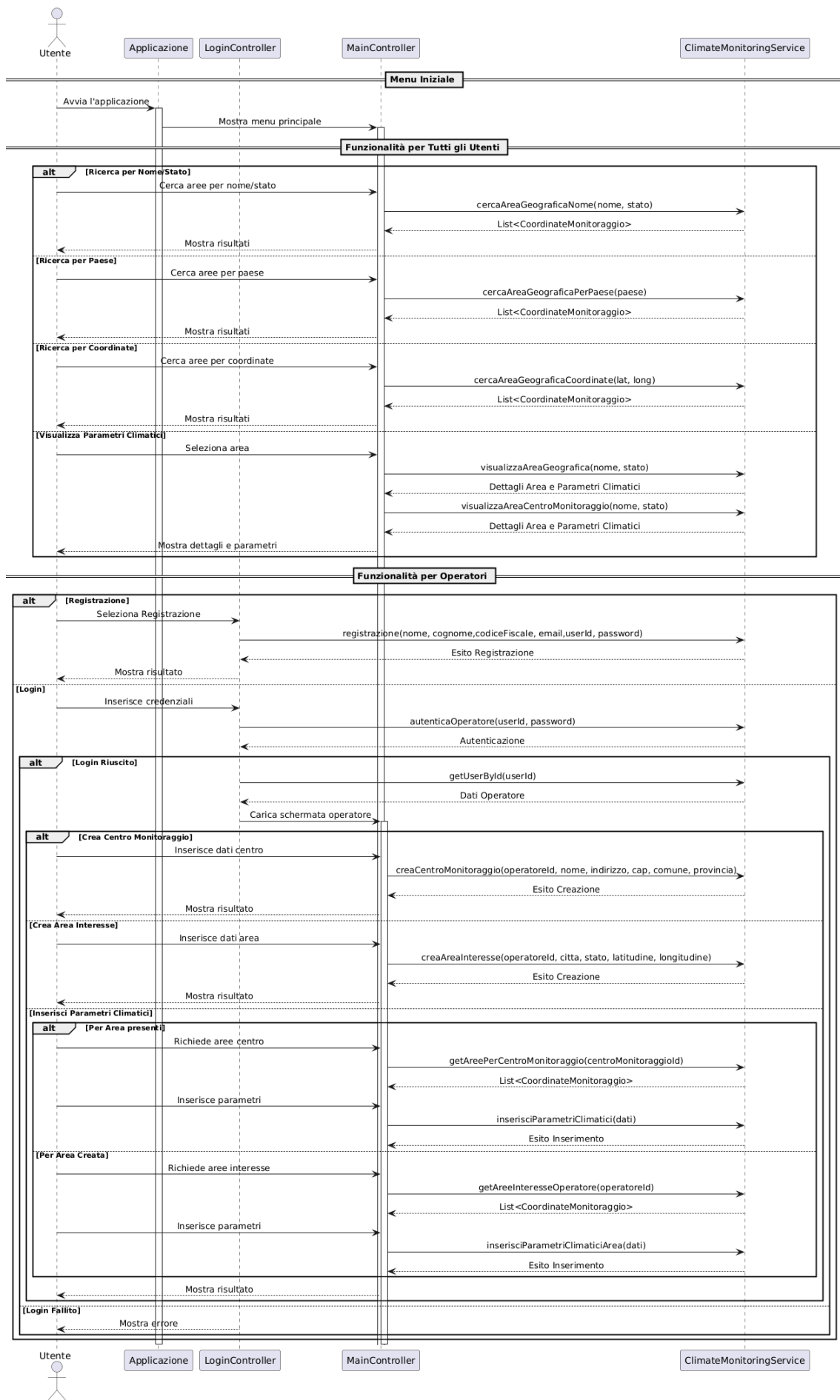
## 6.6 Struttura dinamica

La struttura dinamica descrive il comportamento del sistema durante l'esecuzione, evidenziando l'interazione tra le diverse componenti e il flusso delle operazioni, analizzando come queste entità si comportano e collaborano nel tempo per raggiungere specifici obiettivi.

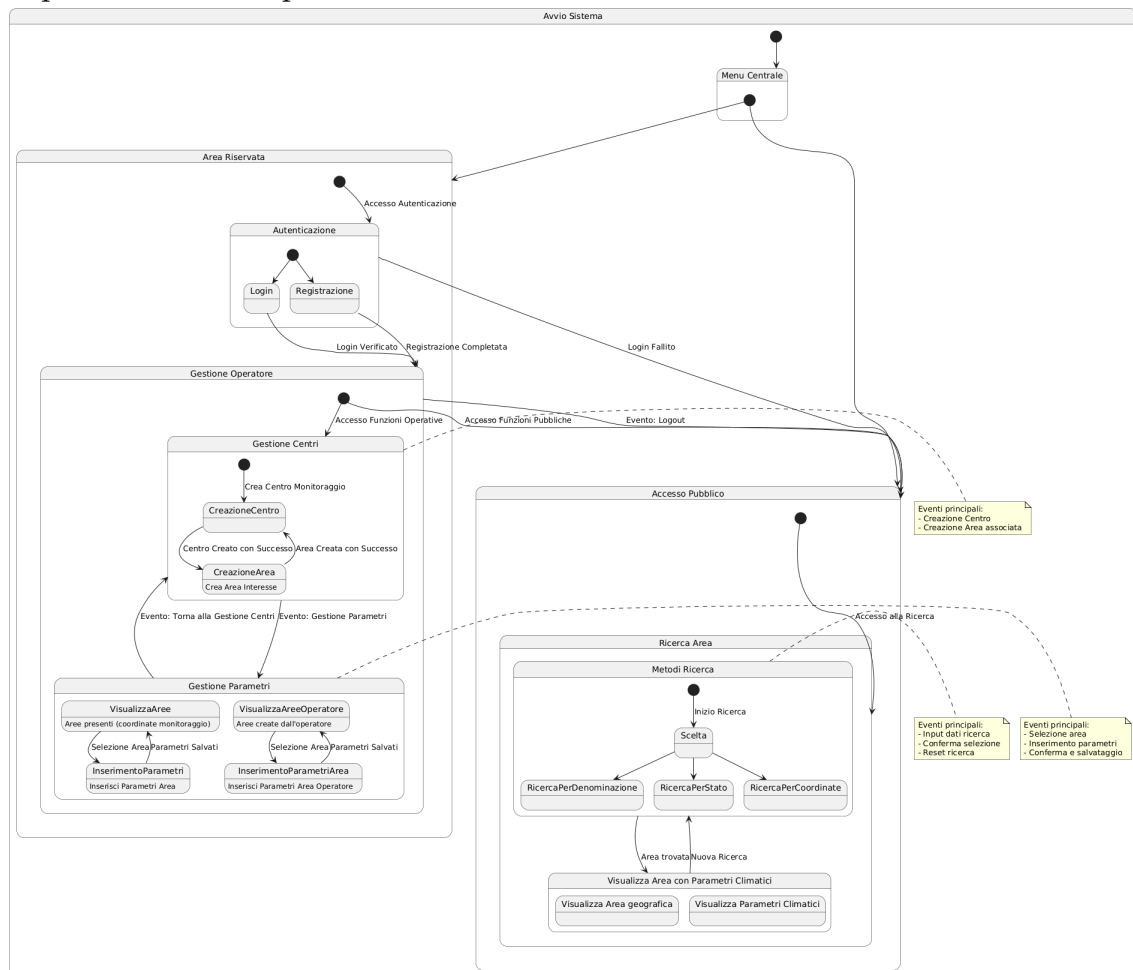
- L'**Activity Diagram** si concentra sul flusso di attività o operazioni nel sistema. A differenza degli altri diagrammi, si concentra sul processo e non sui singoli oggetti o stati, mostrando decisioni, condizioni e ramificazioni nel flusso di lavoro.



- Il **Sequence Diagram** illustra il flusso di messaggi tra gli oggetti o componenti del sistema in ordine cronologico. Si concentra sull'interazione tra le entità, mostrando chiaramente chi comunica con chi e in quale sequenza avviene lo scambio.



- Il **State Diagram** rappresenta i diversi stati in cui un oggetto o un componente può trovarsi durante il suo ciclo di vita e descrive come gli eventi esterni o interni causano transizioni tra questi stati. È utile per analizzare il comportamento di un'entità in contesti in cui il sistema risponde a eventi specifici.



# Database

In questa sezione viene spiegato come sia stato implementato il database ClimateMonitoring utilizzando PostgreSQL come database relazionale e pgAdmin per la sua gestione.

PostgreSQL è stato scelto per la sua affidabilità, scalabilità e per le sue funzionalità avanzate nella gestione dei dati relazionali.

pgAdmin è un'interfaccia grafica che permette agli sviluppatori e agli amministratori di gestire il database PostgreSQL in modo efficace. Tramite pgAdmin, è possibile:

- Creare, modificare ed eliminare tabelle applicative.
- Gestire sequenze, viste, procedure e trigger.
- Monitorare le connessioni e le performance del database.

Il sistema richiede la configurazione delle credenziali per accedere al database PostgreSQL. Le informazioni necessarie includono:

- **URL**: Specifica l'indirizzo del server PostgreSQL.
- **Username** dell'utente con cui si è registrato.
- **Password** associata all'utente

## 7.1 Analisi dei requisiti

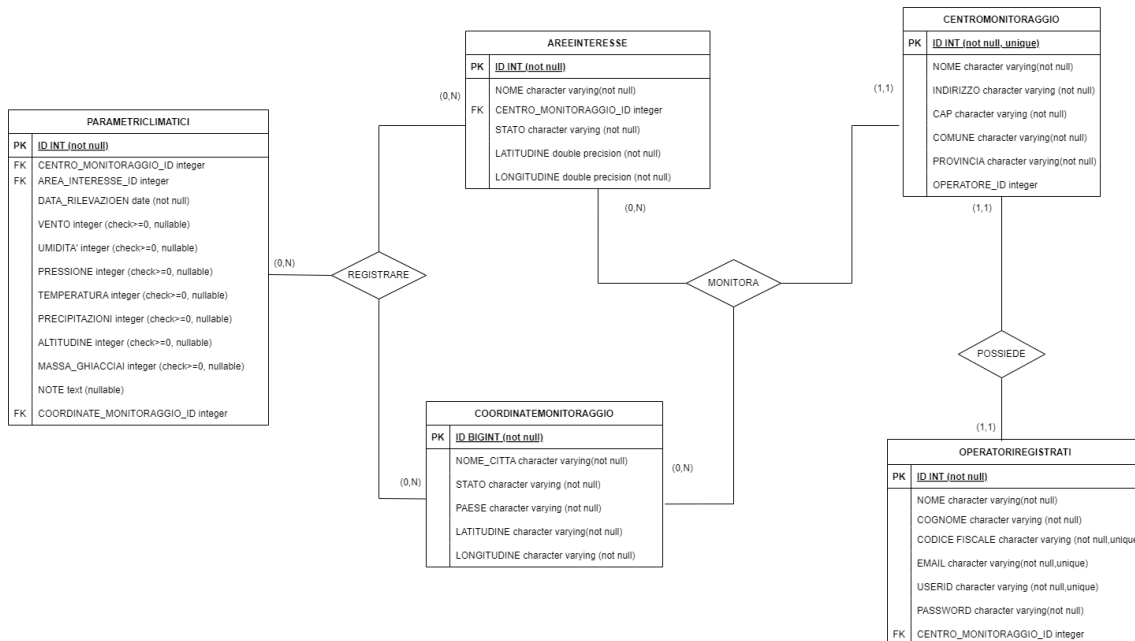
- **Monitoraggio Aree** Il sistema deve permettere di registrare e monitorare i parametri climatici relativi a diverse aree di interesse, preesistenti oppure create dall'operatore
- **Registrazione degli operatori** Gli utenti non registrati possono registrarsi come operatori e opzionalmente, se desiderano contribuire al funzionamento dell'applicazione, creare il proprio centro di monitoraggio.
- **Gestione dei centri di monitoraggio** Ogni centro di monitoraggio sarà responsabile della gestione di specifiche aree geografiche. Questo include la raccolta e la registrazione dei dati relativi ai parametri climatici rilevati in tali aree preesistenti oppure in quelle che l'operatore ha creato
- **Rilevazione dei parametri climatici** I parametri da monitorare includono temperatura, umidità, pressione atmosferica, velocità del vento e, nel caso di aree montane o glaciali, altezza e massa dei ghiacciai. Sarà a discrezione dell'operatore inserire i parametri rilevati all'interno delle



aree con dei valori numerici e commenti

## 7.2 Progettazione del database

### SCHEMA CONCETTUALE



## 7.3 Fase di ristrutturazione

Lo schema ER di partenza non necessita alcuna ristrutturazione per i seguenti motivi:

- **Assenza di attributi composti e multi-valore** : Viene rispettato il principio di atomicità degli attributi. Tutti i campi sono definiti in forma semplice e atomica. Non sono presenti attributi composti, come un unico campo "indirizzo" che richiederebbe una scomposizione in "via", "città", "CAP", né attributi multi-valore
- **Gestione delle gerarchie di generalizzazione** : Non sono presenti gerarchie di generalizzazione o specializzazione che richiederebbero l'introduzione di ulteriori entità o relazioni per rappresentare classi più specifiche o sovra-classi. Tutte le entità e le loro relazioni sono già ben definite e complete.
- **Integrità referenziale garantita** : Lo schema utilizza correttamente chiavi primarie e chiavi esterne per mantenere l'integrità referenziale tra le entità. I vincoli di integrità sono chiaramente definiti, garantendo che

i dati rispettino le relazioni logiche tra le tabelle. Ad esempio, le chiavi esterne che collegano entità come areeinteresse, centrimonitoraggio, e parametriclimatici sono già implementate con vincoli ON UPDATE CASCADE e ON DELETE SET NULL, assicurando la consistenza anche in caso di aggiornamenti o eliminazioni.

- **Normalizzazione avanzata** : Le tabelle rispettano già i principi della terza forma normale (3NF). Ogni attributo dipende unicamente dalla chiave primaria della propria entità, eliminando ridondanze e anomalie di aggiornamento, inserimento e cancellazione

## 7.4 Scelte progettuali

### Entità

- **COORDINATEMONITORAGGIO**

La tabella raccoglie i dettagli relativi a specifiche coordinate geografiche che fungono da punti di rilevazione già preesistenti a sistema

- **id** chiave primaria
- **Nome della città, stato, e paese** di appartenenza.
- **Latitudine e longitudine** per identificare con precisione la posizione geografica.

- **OPERATORIREGISTERATI**

La tabella gestisce gli utenti registrati nel sistema. Ogni operatore ha:

- **id** chiave primaria
- **nome, cognome, codice fiscale, email, userid, password** Campi obbligatori che descrivono l'operatore e consentono la sua registrazione e autenticazione.
- **centro monitoraggio id** Collegamento opzionale a un centro di monitoraggio

- **AREEINTERESSE** Questa tabella rappresenta le diverse aree geografiche di interesse che l'operatore può creare nella sezione dedicata, e automaticamente associate al proprio centro

- **id** chiave primaria
- **nome, stato, latitudine, longitudine** Campi obbligatori da inserire per l'operatore durante la creazione
- **centro monitoraggio id** Collegamento opzionale a un centro di monitoraggio
- **centro monitoraggio id** Collegamento opzionale a un centro di monitoraggio

- **CENTRIMONITORAGGIO** Questa tabella rappresenta i centri di monitoraggio che supervisionano le aree e raccolgono i dati.
  - **id** chiave primaria
  - **nome, indirizzo, cap, comune, provincia** Campi obbligatori da inserire per l'operatore durante la creazione del centro
  - **operatore id** Collegamento opzionale ad un operatore
- **PARAMETRICLIMATICI** Questa tabella vengono salvati i parametri che l'operatore ha inserito per l'area, con gli opportuni riferimenti
  - **id** chiave primaria
  - **data rilevazione** Data in cui i parametri climatici sono stati registrati, obbligatoria
  - **vento, umidita, pressione, temperatura, precipitazioni, altitudine, massa ghiacciai, note** I parametri che l'utente può inserire

## 7.5 Vincoli utilizzati

- **NOT NULL** Utilizzato per campi fondamentali che devono sempre avere un valore, come chiavi primarie, nomi e dati climatici. Questo impedisce l'inserimento di record incompleti.
- **NULLABLE** Consente di lasciare vuoti i campi opzionali come **centro monitoraggio id** o **operatore id**, permettendo flessibilità nel gestire i dati.
- **CHECK** Garantisce che i valori rispettino regole logiche e coerenza. Ad esempio, evita l'inserimento di valori negativi nei parametri climatici.
- **Relazioni con chiavi esterne**
  - **ON DELETE SET NULL** Elimina i riferimenti senza rimuovere i dati collegati, ad esempio nelle aree di interesse quando un centro viene eliminato.
  - **ON DELETE CASCADE** Elimina automaticamente i dati associati, garantendo coerenza nel database (es. eliminazione di operatori legati a un centro)
- **UNIQUE** Evita duplicati in campi come codice fiscale, email e userid, fondamentali per identificare univocamente gli operatori registrati.

## 7.6 Integrità referenziale

Sono state definite le 3 relazioni che permettono una circolazione e salvataggio dei dati in modo adeguato

- **POSSIEDE** L'operatore si crea il proprio e unico centro di monitoraggio
- **MONITORA** L'operatore che ha creato il centro di monitoraggio, ha la possibilità di associare ad esso delle aree già presenti oppure di crearsene nuove
- **REGISTRARE** Selezionando l'area, l'operatore inserisce i parametri climatici a sua discrezione

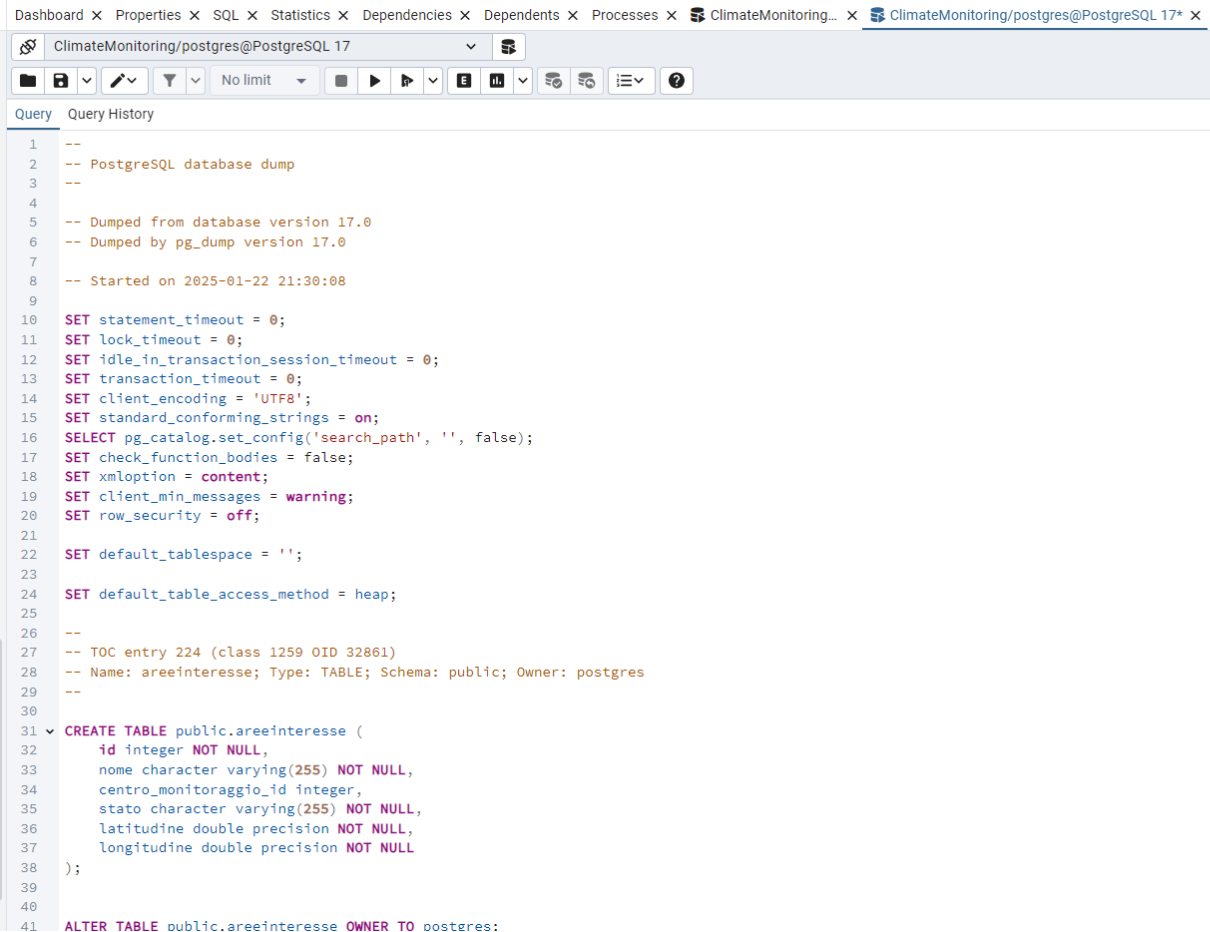
## 7.7 Implementazione Database - Traduzione schema ER

Per ricreare in locale una versione pulita del database è consigliabile scaricare il file di backup del db originale situato all'interno della directory data.

Creare il database chiamandolo : **ClimateMonitoring**

Successivamente :

Aprire pgAdmin > Query Tool > Open File o trascinare il file sql ClimMon-backup.sql > Execute Script



```
Dashboard x Properties x SQL x Statistics x Dependencies x Dependents x Processes x ClimateMonitoring... x ClimateMonitoring/postgres@PostgreSQL 17* x
ClimateMonitoring/postgres@PostgreSQL 17
No limit
Query Query History
1 --
2 -- PostgreSQL database dump
3 --
4 --
5 -- Dumped from database version 17.0
6 -- Dumped by pg_dump version 17.0
7 --
8 -- Started on 2025-01-22 21:30:08
9 --
10 SET statement_timeout = 0;
11 SET lock_timeout = 0;
12 SET idle_in_transaction_session_timeout = 0;
13 SET transaction_timeout = 0;
14 SET client_encoding = 'UTF8';
15 SET standard_conforming_strings = on;
16 SELECT pg_catalog.set_config('search_path', '', false);
17 SET check_function_bodies = false;
18 SET xmloption = content;
19 SET client_min_messages = warning;
20 SET row_security = off;
21
22 SET default_tablespace = '';
23
24 SET default_table_access_method = heap;
25
26 --
27 -- TOC entry 224 (class 1259 OID 32861)
28 -- Name: areeinteresse; Type: TABLE; Schema: public; Owner: postgres
29 --
30
31 CREATE TABLE public.aaeeinteresse (
32 id integer NOT NULL,
33 nome character varying(255) NOT NULL,
34 centro_monitoraggio_id integer,
35 stato character varying(255) NOT NULL,
36 latitudine double precision NOT NULL,
37 longitudine double precision NOT NULL
38);
39
40
41 ALTER TABLE public.aaeeinteresse OWNER TO postgres;
```

Altrimenti si potrebbe procedere con l'esecuzione delle query

## Tabelle

--Creazione Tabelle

```
CREATE TABLE areeinteresse (
 id integer NOT NULL,
 nome character varying(255) NOT NULL,
 centro_monitoraggio_id integer,
 stato character varying(255) NOT NULL,
 latitudine double precision NOT NULL,
 longitudine double precision NOT NULL
);

CREATE TABLE centrimonitoraggio (
 id integer NOT NULL,
 nome character varying(255) NOT NULL,
 indirizzo character varying(255) NOT NULL,
 cap character varying(10) NOT NULL,
 comune character varying(100) NOT NULL,
 provincia character varying(100) NOT NULL,
 operatore_id integer
);

CREATE TABLE coordinatemonitoraggio (
 id bigint NOT NULL,
 nome_citta character varying(255),
 stato character varying(100),
 paese character varying(100),
 latitudine numeric(10,8) NOT NULL,
 longitudine numeric(11,8) NOT NULL
);

CREATE TABLE operatoriregistrati (
 id integer NOT NULL,
 nome character varying(100) NOT NULL,
 cognome character varying(100) NOT NULL,
 codice_fiscale character varying(16) NOT NULL,
 email character varying(255) NOT NULL,
 userid character varying(50) NOT NULL,
 password character varying(255) NOT NULL,
 centro_monitoraggio_id integer
);

CREATE TABLE parametriclimatici (
 id integer NOT NULL,
 centro_monitoraggio_id integer NOT NULL,
 area_interesse_id integer,
 data_rilevazione date NOT NULL,
 vento integer,
 umidita integer,
 pressione integer,
 temperatura integer,
 precipitazioni integer,
 altitudine integer,
 massa_ghiacciai integer,
 note text,
 coordinate_monitoraggio_id integer,
 CONSTRAINT parametriclimatici_altitudine_check CHECK ((altitudine >= 0)),
 CONSTRAINT parametriclimatici_massa_ghiacciai_check CHECK ((massa_ghiacciai >= 0)),
 CONSTRAINT parametriclimatici_precipitazioni_check CHECK ((precipitazioni >= 0)),
 CONSTRAINT parametriclimatici_pressione_check CHECK ((pressione >= 0)),
 CONSTRAINT parametriclimatici_temperatura_check CHECK ((temperatura >= 0)),
 CONSTRAINT parametriclimatici_umidita_check CHECK ((umidita >= 0)),
 CONSTRAINT parametriclimatici_vento_check CHECK ((vento >= 0))
);
```

## Sequenze

```
CREATE SEQUENCE areeinteresse_id_seq
AS integer
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;

CREATE SEQUENCE centrimonitoraggio_id_seq
AS integer
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;

CREATE SEQUENCE parametriclimatici_id_seq
AS integer
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;
```

## Generated by default as identity

```
ALTER TABLE areeinteresse ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
SEQUENCE NAME areeinteresse_id_seq
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1
);

ALTER TABLE centrimonitoraggio ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
SEQUENCE NAME centrimonitoraggio_id_seq
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1
);

ALTER TABLE parametriclimatici ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
SEQUENCE NAME parametriclimatici_id_seq
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1
);
```

## Primary Key

```
ALTER TABLE ONLY areeinteresse
ADD CONSTRAINT areeinteresse_pkey PRIMARY KEY (id);

ALTER TABLE ONLY centrimonitoraggio
ADD CONSTRAINT centrimonitoraggio_pkey PRIMARY KEY (id);

ALTER TABLE ONLY coordinatemonitoraggio
ADD CONSTRAINT coordinatemonitoraggio_pkey PRIMARY KEY (id);

ALTER TABLE ONLY operatoriregistrati
ADD CONSTRAINT operatoriregistrati_pkey PRIMARY KEY (id);

ALTER TABLE ONLY parametriclimatici
ADD CONSTRAINT parametriclimatici_pkey PRIMARY KEY (id);
```

## Unique

```
ALTER TABLE ONLY operatoriregistrati
ADD CONSTRAINT operatoriregistrati_codice_fiscale_key UNIQUE (codice_fiscale);

ALTER TABLE ONLY operatoriregistrati
ADD CONSTRAINT operatoriregistrati_email_key UNIQUE (email);

ALTER TABLE ONLY operatoriregistrati
ADD CONSTRAINT operatoriregistrati_userid_key UNIQUE (userid);
```

## Foreign Key

```
ALTER TABLE ONLY areeinteresse
 ADD CONSTRAINT fk_areeinteresse_centro FOREIGN KEY (centro_monitoraggio_id) REFERENCES centrimonitoraggio(id) ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE ONLY operatoriregistrati
 ADD CONSTRAINT fk_operatoriregistrati_centro FOREIGN KEY (centro_monitoraggio_id) REFERENCES centrimonitoraggio(id) ON UPDATE CASCADE;

ALTER TABLE ONLY parametriclimatici
 ADD CONSTRAINT fk_parametriclimatici_area FOREIGN KEY (area_interesse_id) REFERENCES areeinteresse(id) ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE ONLY parametriclimatici
 ADD CONSTRAINT fk_parametriclimatici_centro FOREIGN KEY (centro_monitoraggio_id) REFERENCES centrimonitoraggio(id) ON UPDATE CASCADE;

ALTER TABLE ONLY parametriclimatici
 ADD CONSTRAINT fk_parametriclimatici_coordinate FOREIGN KEY (coordinate_monitoraggio_id) REFERENCES coordinatemonitoraggio(id) ON UPDATE CASCADE ON DELETE SET NULL;
```



## 7.8 Query

```
--Cerca area geografica per nome, ad es Onset, US
SELECT * FROM coordinatemonitoraggio WHERE nome_citta LIKE ? AND stato = ?

--Cerca area geografica per paese, ad es United States
SELECT * FROM coordinatemonitoraggio WHERE paese LIKE ?

--Cerca area per coordinate
SELECT * FROM coordinatemonitoraggio WHERE latitudine BETWEEN ? - ? AND ? + ? AND longitudine BETWEEN ? - ? AND ? + ?

--Visualizza area geografica
SELECT * FROM coordinatemonitoraggio WHERE nome_citta = ? AND stato = ?

--Allega parametri climatici
--medie
SELECT COUNT(*) AS num_rilevazioni, " +
 "AVG(vento) AS avg_vento, " +
 "AVG(umidita) AS avg_umidita, " +
 "AVG(pressione) AS avg_pressione, " +
 "AVG(temperatura) AS avg_temperatura, " +
 "AVG(precipitazioni) AS avg_precipitazioni, " +
 "AVG(altitudine) AS avg_altitudine, " +
 "AVG(massa_ghiacciai) AS avg_massa_ghiacciai " +
 "FROM parametriclimatici " +
 "WHERE " + idColumnType + " = ?

--dettaglio
SELECT p.*, p.data_rilevazione, " +
 "p.vento, p.umidita, p.pressione, p.temperatura, " +
 "p.precipitazioni, p.altitudine, p.massa_ghiacciai, p.note " +
 "FROM parametriclimatici p " +
 "WHERE p." + idColumnType + " = ? " +
 "ORDER BY p.data_rilevazione DESC
SELECT p.*, op.nome AS nome_operatore, op.cognome AS cognome_operatore, " +
 "p.data_rilevazione, p.vento, p.umidita, p.pressione, p.temperatura, " +
 "p.precipitazioni, p.altitudine, p.massa_ghiacciai, p.note " +
 "FROM parametriclimatici p " +
 "JOIN centrimonitoraggio cm ON p.centro_monitoraggio_id = cm.id " +
 "JOIN operatoriregistrati op ON cm.operatore_id = op.id " +
 "WHERE p." + idColumnType + " = ? " +
 "ORDER BY p.data_rilevazione DESC

--Allega commenti operatori
SELECT note, data_rilevazione FROM parametriclimatici " +
 "WHERE " + idColonna + " = ? " +
 "AND note IS NOT NULL AND note != '' " +
 "ORDER BY data_rilevazione DESC LIMIT 5

--Visualizza Aree create dall'operatore per centro di monitoraggio
SELECT ai.id, ai.nome, ai.stato, ai.latitudine, ai.longitudine, " +
 "ai.centro_monitoraggio_id, cm.nome AS centro_nome " +
 "FROM areeinteresse ai " +
 "JOIN centrimonitoraggio cm ON ai.centro_monitoraggio_id = cm.id " +
 "WHERE ai.nome = ? AND ai.stato = ?

--Registrazione operatore
INSERT INTO operatoriregistrati (nome, cognome, codice_fiscale, email, userid, password) VALUES (?, ?, ?, ?, ?, ?)

--Autentica operatore
SELECT * FROM operatoriregistrati WHERE userid = ? AND password = ?

--Ottieni User id operatori
SELECT * FROM operatoriregistrati WHERE userid = ?

--Crea centro di monitoraggio
INSERT INTO centrimonitoraggio (operatore_id, nome, indirizzo, cap, comune, provincia) VALUES (?, ?, ?, ?, ?, ?)

--Crea area di interesse
INSERT INTO areeinteresse (nome, stato, centro_monitoraggio_id, latitudine, longitudine) VALUES (?, ?, ?, ?, ?)

--Ottieni centro di monitoraggio
SELECT id FROM centrimonitoraggio WHERE operatore_id = ?
|
--Recupera aree esistenti per centro di monitoraggio
SELECT * FROM coordinatemonitoraggio

--Recupera aree create dall'operatore per centro di monitoraggio
SELECT a.*
 FROM areeinteresse a
 WHERE a.centro_monitoraggio_id = ?
```

```

--Inserisci parametri climatici per aree già esistenti
INSERT INTO parametriclimatici (centro_monitoraggio_id, area_interesse_id, " +
 "coordinate_monitoraggio_id, data_rilevazione, vento, umidita, pressione, " +
 "temperatura, precipitazioni, altitudine, massa_ghiacciai, note) " +
 "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

|
--Inserisci parametri climatici per aree create dall'operatore
INSERT INTO parametriclimatici "
 + "(centro_monitoraggio_id, area_interesse_id, data_rilevazione, "
 + "vento, umidita, pressione, temperatura, precipitazioni, "
 + "altitudine, massa_ghiacciai, note) "
 + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

```

# Scelte algoritmiche

## 8.1 Architettura e comunicazione

Il progetto si basa su RMI per la comunicazione tra il client e il server. Questa scelta consente al client di invocare metodi remoti sul server come se fossero locali, centralizzando la logica applicativa sul server.

```
LocateRegistry.createRegistry(1099);
ClimateMonitoringServiceImpl climateService = new
 ↪ ClimateMonitoringServiceImpl(dbManager);
Naming.rebind("rmi://localhost/ClimateMonitoringService",
 ↪ climateService);

Registry registry = LocateRegistry.getRegistry("localhost",
 ↪ 1099);
service = (ClimateMonitoringService)
 ↪ registry.lookup("ClimateMonitoringService");
```

## 8.2 Database

Il database PostgreSQL è integrato attraverso la classe DatabaseManager, che gestisce la connessione con un approccio singleton.

```
public Connection getConnection() throws SQLException {
 if (connection == null || connection.isClosed()) {
 connection = DriverManager.getConnection(dbUrl, dbUser,
 ↪ dbPassword);
 }
 return connection;
}
```

## 8.3 Algoritmi di ricerca

### Ricerca per Nome e Stato

```
String sql = "SELECT * FROM coordinatemonitoraggio WHERE
↪ nome_citta LIKE ? AND stato = ?";
```

### Ricerca per coordinate

```
private double calcolaDistanzaKm(double lat1, double lon1,
↪ double lat2, double lon2) {
 final int R = 6371;

 double latDistance = Math.toRadians(lat2 - lat1);
 double lonDistance = Math.toRadians(lon2 - lon1);

 double a = Math.sin(latDistance / 2) *
 ↪ Math.sin(latDistance / 2)
 + Math.cos(Math.toRadians(lat1)) *
 ↪ Math.cos(Math.toRadians(lat2))
 * Math.sin(lonDistance / 2) *
 ↪ Math.sin(lonDistance / 2);

 double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 -
 ↪ a));

 return R * c;
}
```

## 8.4 Ordinamento dei risultati

I risultati delle ricerche sono ordinati utilizzando la funzione sort e un comparatore personalizzato basato sulla distanza calcolata.

```
aree.sort((a1, a2) -> Double.compare(
 calcolaDistanzaKm(latitudine, longitudine,
 ↪ a1.getLatitude(), a1.getLongitude()),
 calcolaDistanzaKm(latitudine, longitudine,
 ↪ a2.getLatitude(), a2.getLongitude())
));
```

## 8.5 Aggregazione dei dati climatici

I risultati delle ricerche sono ordinati utilizzando la funzione sort e un comparatore personalizzato basato sulla distanza calcolata.

```
String sql = "SELECT COUNT(*) AS num_rilevazioni, AVG(vento) AS
↪ avg_vento FROM parametriclimatici WHERE
↪ coordinate_monitoraggio_id = ?";
```

## 8.6 Validazione degli input

Ogni metodo che interagisce con il database o con i servizi remoti include controlli di validazione per garantire robustezza e sicurezza.

```
if (nome == null || nome.trim().isEmpty()) {
 throw new IllegalArgumentException("Il nome non può essere
↪ nullo o vuoto");
}
```

**Chiusura del server** Il server esegue le operazioni per arrestare il servizio rmi dal registro e chiude le connessioni al database

```
Registry registry = LocateRegistry.getRegistry(1099);
registry.unbind("ClimateMonitoringService");
dbManager.closeConnection();

public void closeConnection() {
 try {
 if (connection != null && !connection.isClosed()) {
 connection.close();
 connection = null;
 System.out.println("Connessione al database chiusa
↪ con successo");
 }
 } catch (SQLException e) {
 System.err.println("Errore durante la chiusura della
↪ connessione: " + e.getMessage());
 }
}
```

## 8.7 Strutture dati utilizzati

Liste

```
List<CoordinateMonitoraggio>
```

Memorizzazione dei risultati di ricerca geografica.

```
List<CoordinateMonitoraggio> aree = new ArrayList<>();
```

StringBuilder

```
StringBuilder result = new StringBuilder();
```

```
result.append("=== Informazioni Area Geografica ===\n\n");
```

ResultSet Per recuperare i dati delle query SQL

```
while (rs.next()) {
 CoordinateMonitoraggio area = new CoordinateMonitoraggio(
 rs.getInt("id"),
 rs.getString("nome_citta"),
 rs.getString("stato"),
 rs.getDouble("latitudine"),
 rs.getDouble("longitudine")
);
 aree.add(area);
}
```

## 8.8 Cicli Utilizzati

While Per iterare sui risultati delle query

```
while (rs.next()) {
 CoordinateMonitoraggio area = new CoordinateMonitoraggio(
 rs.getInt("id"),
 rs.getString("nome_citta"),
 rs.getString("stato"),
 rs.getDouble("latitudine"),
 rs.getDouble("longitudine")
);
 aree.add(area);
}
```

## 8.9 for-each

Utilizzati per iterare su liste

```
for (CoordinateMonitoraggio area : results) {
 sb.append("Città: ").append(area.getNomeCitta())
 .append("\nCoordinate: ").append(area.getLatitudine())
 .append(", ").append(area.getLongitude());
}
```

# Limiti del sistema

I possibili limiti dell'applicazione riguardano diversi aspetti critici:

- **Sicurezza**

L'attuale implementazione presenta significative vulnerabilità di sicurezza. Le comunicazioni tra client e server non sono crittografate. È necessario sviluppare un meccanismo più robusto per proteggere informazioni sensibili, in particolare le credenziali di accesso in quanto vengono salvate in chiaro nel database senza alcun algoritmo di hashing.

- **Compatibilità Tecnica**

Sussistono limitazioni di compatibilità con sistemi meno recenti. Le versioni obsolete di Java potrebbero non supportare correttamente l'applicazione, riducendo di fatto l'accessibilità del software per utenti con hardware o ambienti di sviluppo datati. Le versioni di Java e Postgre da utilizzare devono essere compatibili a quelle mostrate nel Manuale Utente. E' necessaria l'installazione delle sdk di javafx 23 in quanto non compatibili con l'utilizzo di Java 17 da specifiche, vedere dal manuale utente il processo di installazione.

- **Funzionamento**

Il corretto funzionamento del client dipende strettamente dall'avvio e configurazione preliminare del server. Il database è esclusivamente locale, e richiede una configurazione manuale.

- **Opzioni di ricerca**

I risultati delle ricerche non vengono ordinati secondo una metodologia, ma dipende dalla lettura che viene eseguita all'interno del database

- **Autenticazione e Operatore**

L'autenticazione può avvenire solamente tramite User Id e password, non è possibile accedere tramite email o codice fiscale.

Una volta che l'operatore ha creato il proprio centro di monitoraggio, esso non può cancellarlo, così come le aree che esso decide di creare all'interno del proprio centro.



# Sitografia

- JRE, JDK e JAVA

<https://www.oracle.com/it/java/>

<https://docs.oracle.com/en/java/>

<https://gluonhq.com/products/javafx/>

<https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html>

<https://www.jetbrains.com/help/idea/javadocs.html>

- Jar

<https://stackoverflow.com/questions/1238145/how-to-run-a-jar-file>

<https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html>

<https://gluonhq.com/products/javafx/>

- UML

<https://www.plantuml.com/plantuml/uml/SyfFKj2rKt3CoKnELR1Io4ZDoSa700003>

- ER

<https://www.drawio.com/>