

# Manuale Tecnico Climate Monitoring

## Indice

<b>Informazioni Generali.....</b>	<b>4</b>
Strumenti di sviluppo.....	4
API e Librerie Utilizzate.....	4
<b>Requisiti Sistema.....</b>	<b>4</b>
<b>Github Repository.....</b>	<b>5</b>
<b>Funzionalità dell'Applicazione.....</b>	<b>5</b>
Client Climate Monitoring.....	5
Server Climate Monitoring.....	10
<b>Struttura Applicazione.....</b>	<b>10</b>
Moduli Principali.....	10
Modulo clientCM.....	11
Classe ClimateMonitoring(Client Main).....	12
Package views.....	13
ClientHomeGUI.....	14
PoiSearchResultGUI.....	14
PoiDataGUI.....	16
CommentViewGUI.....	17
OperatorHomeGUI.....	18
OperatorRegistrationGUI.....	19
CenterSelectionGUI.....	20
CenterRegistrationGUI.....	21
PoiSearchForRegistrationGUI.....	22
Package controllers.....	24
ClientHomeGUIController.....	25
PoiSearchResultGUIController.....	25
PoiDataGUIController.....	26
CommentViewGUIController.....	26
OperatorHomeGUIController.....	27
OperatorRegistrationGUIController.....	28
CenterSelectionGUIController.....	28
CenterRegistrationGUIController.....	29

PoiSearchForRegistrationGUIController.....	30
Package network.....	30
ClientManager.....	31
Package utils.....	32
FieldFormatException.....	32
Modulo serverCM.....	32
Classe ServerMain.....	33
Package database.....	35
DbManager.....	35
PredefinedQuery.....	36
Package network.....	37
RemoteDatabaseService.....	37
Modulo shared.....	39
RemoteDatabaseServiceInterface.....	39
Package models.....	41
PointOfInterest.....	42
User.....	42
Monitoring Center.....	43
Survey.....	44
SurveyAggregate.....	45
<b>Struttura Database.....</b>	<b>46</b>
Analisi e considerazioni in linguaggio naturale.....	46
Progettazione Concettuale.....	48
Progettazione Logica.....	49
Vincoli d'integrità.....	51
Progettazione Pratica.....	51
Creazione Database.....	51
Creazione Tabelle.....	52
SELECT Queries.....	53
User (Operatore Registrato).....	53
User Login Info.....	53
User existence.....	53
Email existence.....	53
Monitoring Centers.....	53
Monitoring Center da ID.....	53
Monitoring Center da Dati.....	54
Point Of Interest (Area Geografica).....	54
Point Of Interest da dati.....	54
Point Of Interest da denominazione.....	54
Point Of Interest da coordinate.....	54
Survey Aggregate.....	54
Point Of Interest da ID Centro.....	55
Survey per ID Point Of Interest (Area Geografica).....	55
Database.....	56

Table da database.....	56
INSERT Queries.....	56
User (Operatore Registrato).....	56
Monitoring Center.....	56
Point Of Interest.....	56
Survey.....	56
Point Of Interest per Centro.....	56
<b>Sitografia/Bibliografia.....</b>	<b>57</b>

# Informazioni Generali

## Strumenti di sviluppo

Strumenti sviluppo java :

- JDK 17.0.9
- IntelliJ 2023.3.2
- Apache Maven 3.9.5 (integrazione IntelliJ)
- Git Version Control (integrazione IntelliJ)

Strumenti database :

- PostgreSQL 16.0
- pgAdmin 4
- Linguaggio SQL

Strumenti UML e ERD(Entity Relation Diagrams :

- StarUML 6.0.1
- Draw.io (web app)
- Dia 0.97.2

## API e Librerie Utilizzate

Tutte le API e Librerie esterne a Java utilizzate per il progetto sono indicate nel file di Maven pom.xml che contiene l'intera configurazione del progetto Maven.

Nello specifico le API/Librerie e dependencies utilizzate sono contenute tra le tag di dependencies del pom.xml.

Tra quelle utilizzate le più significative sono :

- **Apache Poi Common e Apache POI OPC and OOXML** : utilizzate per avere accesso a strumenti per la lettura di documenti Microsoft Office, in particolare è utilizzata per recuperare dal file “[geonames-and-coordinates.xlsx](#)” i dati per inizializzare il database.
- **PostgreSQL JDBC Driver** : che è utilizzato per poter connettersi al database PostgreSQL.
- **Apache Commons DBCP** : che fornisce le classi e librerie per implementare connection pooling per connessioni a database.
- **Varie librerie IntelliJ** : per garantire che non ci siano errori in fase di build dovuti al codice autogenerato dai GUI form di IntelliJ, che sono stati utilizzati per lo sviluppo.

## Requisiti Sistema

Per eseguire il client di Climate Monitoring, è necessario aver installato Java sul proprio Sistema tramite l'installazione del JRE (Java Runtime Environment).

Si consiglia di utilizzare una versione oltre la 8 e non superiore alla 17.

Viene garantito il funzionamento dell'applicazione Climate Monitoring su sistemi Windows (preferibilmente 10 o 11) e sulle più popolari distribuzioni Linux Debian-based (preferibilmente Ubuntu e Linux Mint).

Il programma dovrebbe funzionare anche su sistemi macOS ma il funzionamento non viene garantito in quanto non testato.

E' inoltre necessaria una connessione ad internet in tutti gli ambienti che prevedono che l'applicazione venga lanciata da un sistema che non è all'interno della rete in cui è in esecuzione il server dell'applicazione Climate Monitoring.

**Si ricorda comunque che in questa versione dell'applicazione il server prevede l'esecuzione sullo stesso sistema.**

## Github Repository

Lo sviluppo del progetto può essere attivamente seguito a questo link :

[https://github.com/Arcii/Climate\\_Monitoring](https://github.com/Arcii/Climate_Monitoring)

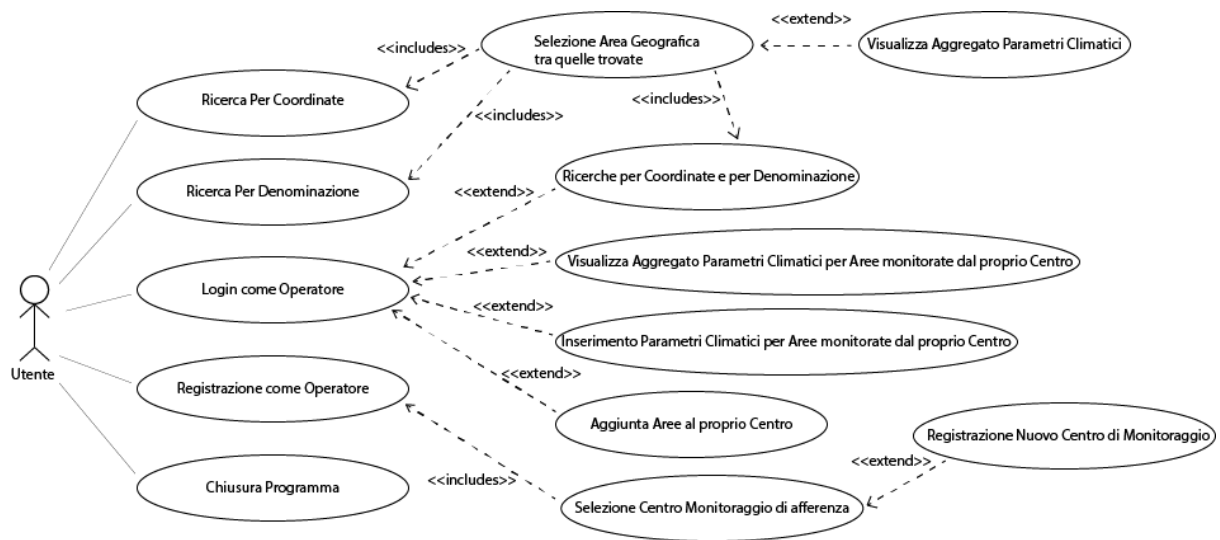
Per segnalare bug/errori si consiglia di creare un issue post a questo link con più dettagli possibili per riprodurlo :

[https://github.com/Arcii/Climate\\_Monitoring/issues](https://github.com/Arcii/Climate_Monitoring/issues)

## Funzionalità dell'Applicazione

### Client Climate Monitoring

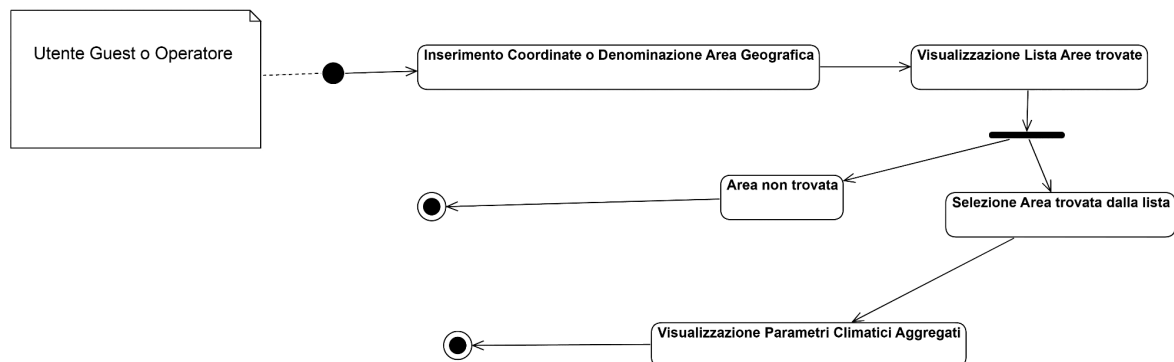
Il client dell'applicazione permette ad un utente di accedere a queste funzionalità a seconda che sia un utente "Guest" o un Operatore registrato e connesso :



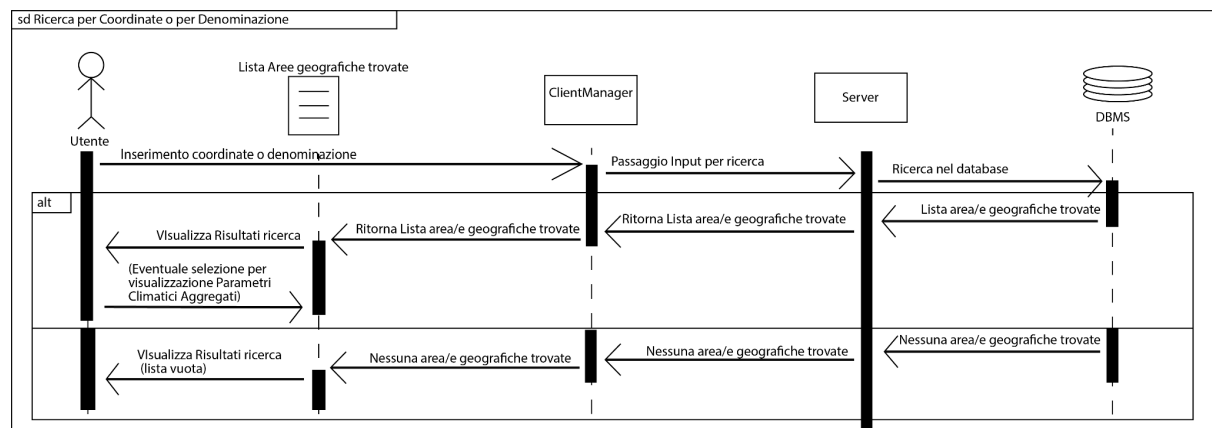
Vediamo ora ogni funzionalità nello specifico.

1. La funzione di Ricerca segue la stessa logica sia per la Ricerca per Coordinate che per la ricerca per Denominazione:

## Activity Diagram

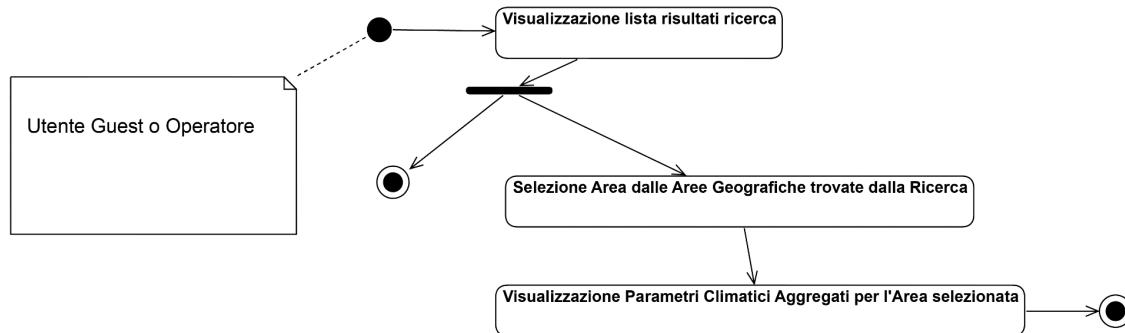


## Sequence Diagram

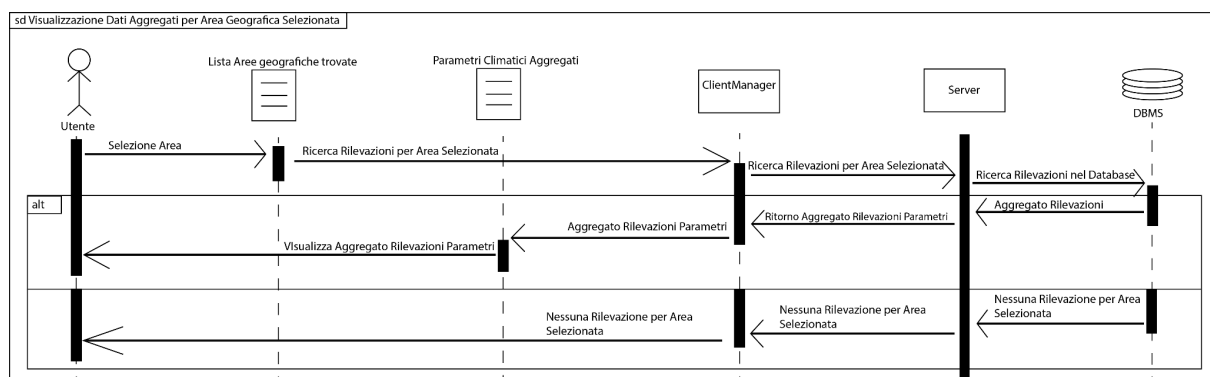


2. La visualizzazione dei dati aggregati dei Parametri Climatici per un'Area Geografica trovata e selezionata è riassunta dai seguenti diagram (da considerare successivi ad una ricerca):

### Activity Diagram

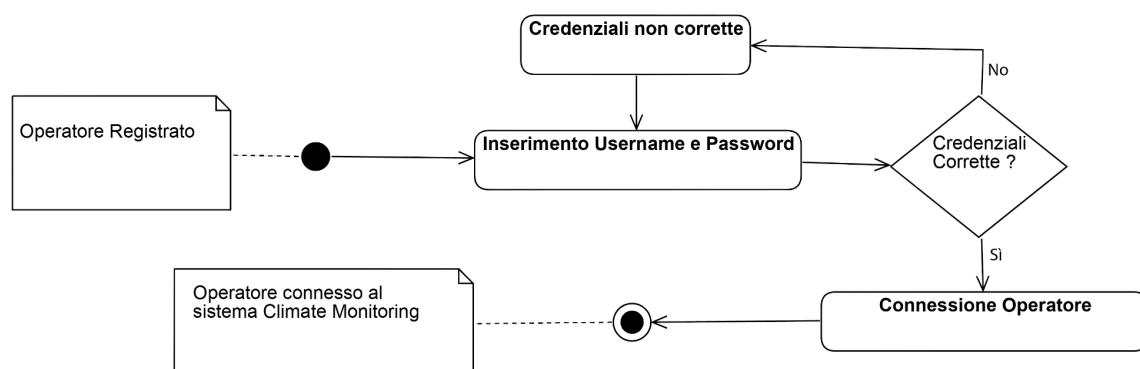


### Sequence Diagram

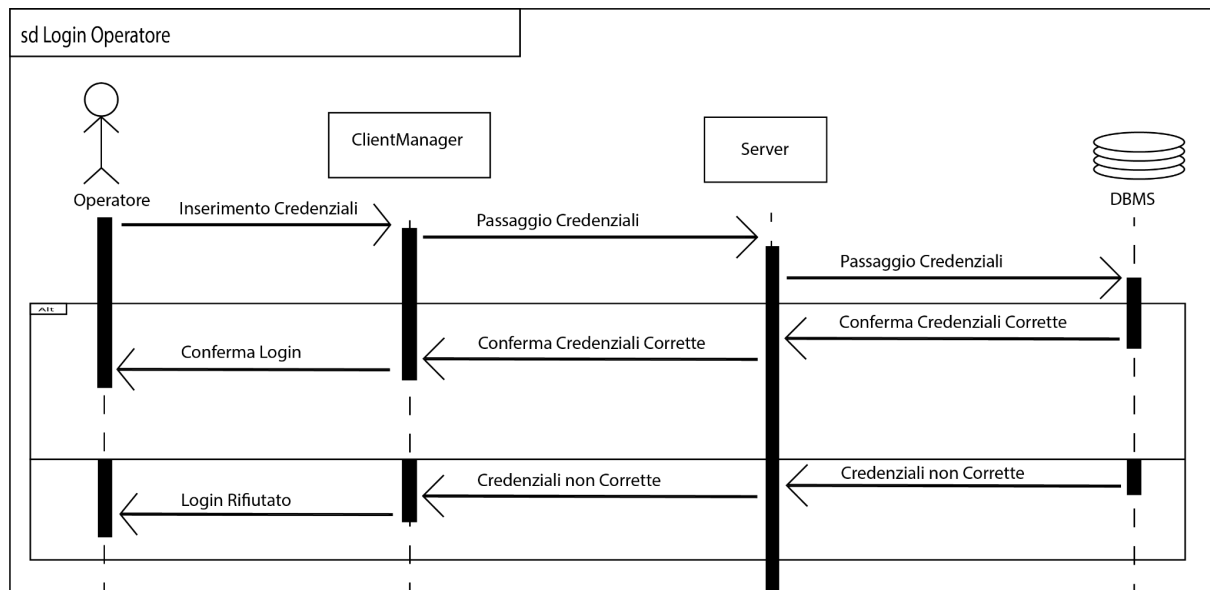


3. La funzione di Login è riassunta da questi diagrams:

### Activity Diagram

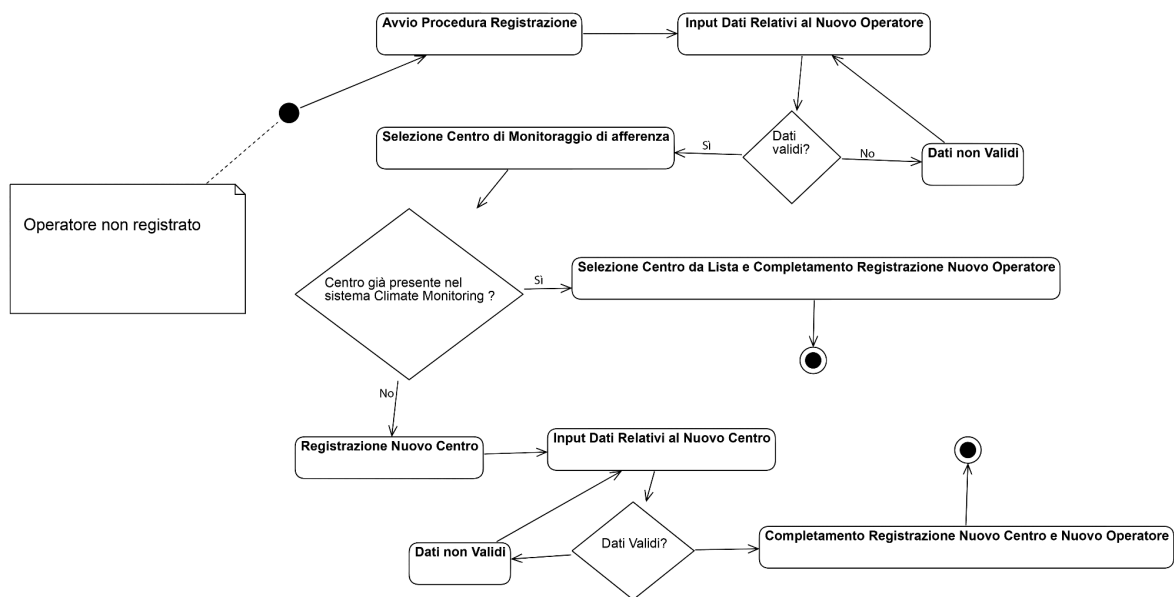


### Sequence Diagram



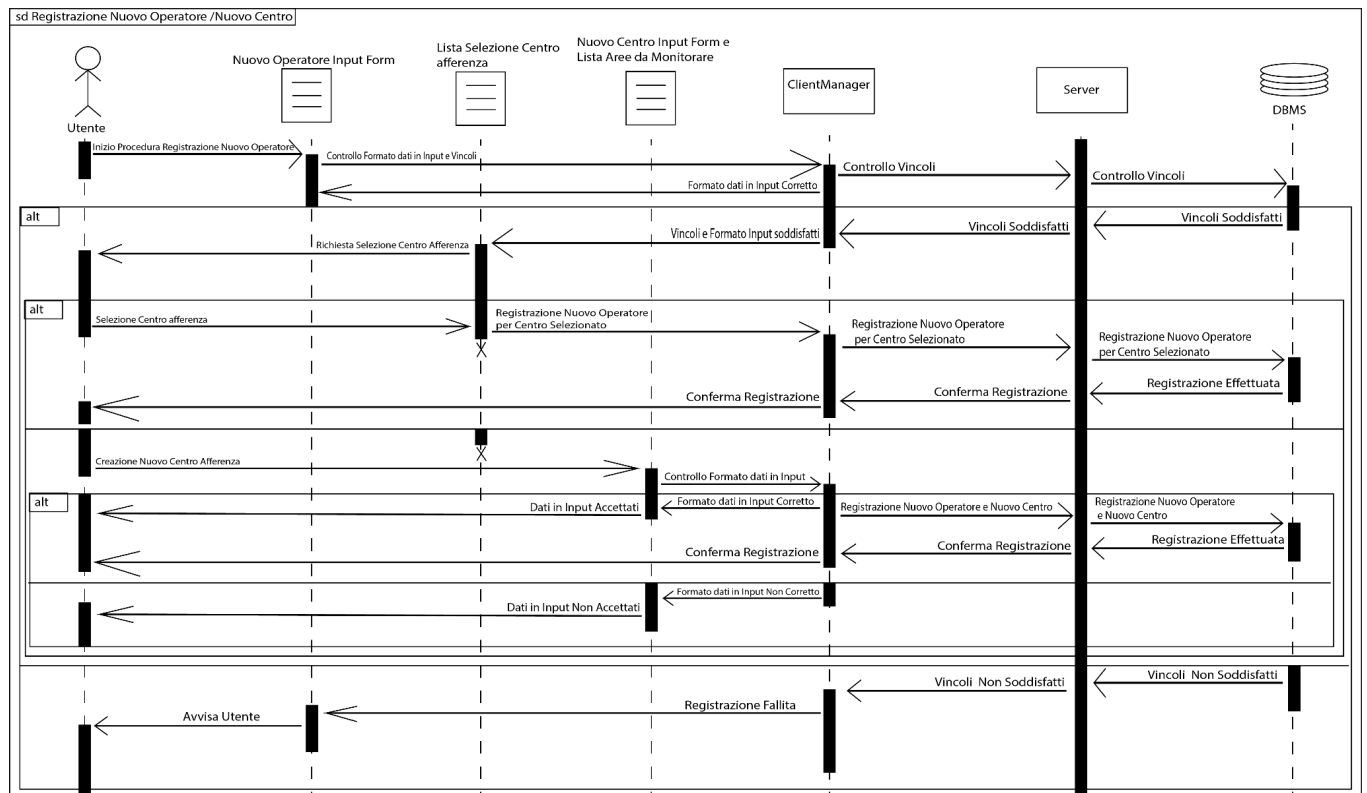
4. La funzione di registrazione di un nuovo Operatore per il sistema Climate Monitoring è riassunta nei seguenti diagrams:

#### Activity Diagram



#### Sequence Diagram

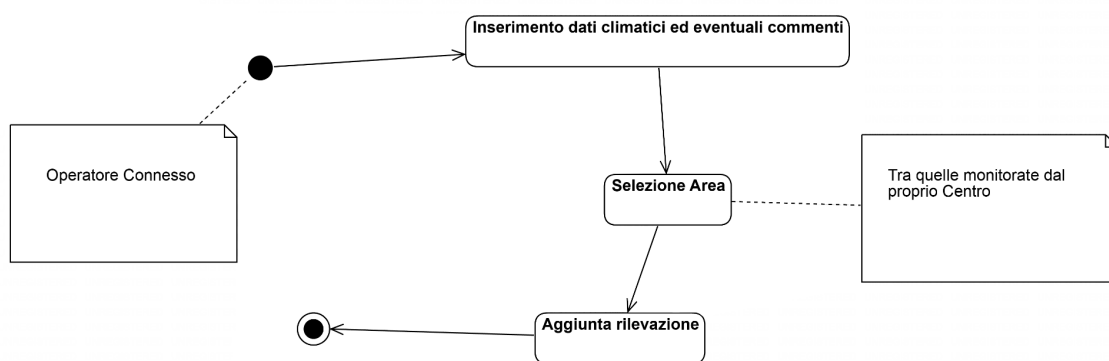




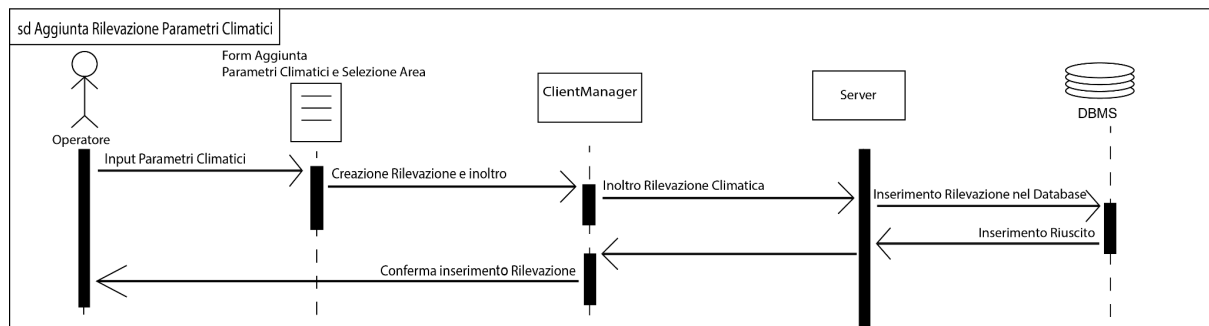
**Nota** : Per evitare un ulteriore sezione “alt” si assume che nel primo form l’operatore inserisca i dati in input con formato corretto.

5. La funzionalità di inserimento di una rilevazione di Parametri Climatici (accessibile solo ad Operatori connessi) può essere riassunta dai seguenti diagrams:

Activity Diagram



Sequence Diagram



6. Per la chiusura non si ritengono necessari diagrams.

## Server Climate Monitoring

Mentre per il server dell'applicazione l'interazione è limitata in quanto l'unico input richiesto all'admin del server sono le eventuali credenziali PostgreSQL:



## Struttura Applicazione

### Moduli Principali

L'applicazione "Climate Monitoring" è una soluzione software strutturata su tre moduli distinti: **clientCM**, **serverCM** e **shared**. Questa divisione in package è stata adottata per organizzare e separare le responsabilità funzionali dell'applicazione, fornendo una progettazione modulare e scalabile. Di seguito, si fornisce una breve descrizione di ciascun modulo (per la struttura e il funzionamento dei singoli moduli fare riferimento alle opportune voci nell'indice):

- **clientCM** : Il modulo **clientCM** rappresenta la componente client dell'applicazione.

Questo modulo è responsabile dell'interfaccia utente e dell'interazione diretta con gli utenti finali e realizza il "lato client" o "lato utente" dell'applicazione Climate Monitoring (front-end).

Contiene la logica per la gestione delle operazioni di login, visualizzazione dei dati climatici e interazione con il server di Climate Monitoring tramite un'apposita classe manager (classe [ClientManager](#)) per ottenere e/o aggiornare le informazioni dal database. Include componenti grafiche, controller e servizi per fornire un'esperienza utente intuitiva.

- **serverCM** : Il modulo serverCM è il nucleo del sistema e gestisce la logica e la comunicazione con il database.

Si occupa dell'autenticazione degli utenti, delle richieste provenienti dai client e delle operazioni di interrogazione e aggiornamento del database climatico.

Questo modulo implementa protocolli di comunicazione sicuri tra client e server, garantendo la protezione delle informazioni sensibili.

Gestisce inoltre le connessioni al database “[dbcm](#)” che contiene tutti i dati relativi a Aree Geografiche (modellati con la classe [PointOfInterest](#) nel package [shared](#)), Parametri Climatici (modellati con la classe [Survey](#) nel package [shared](#)), Operatori Registrati (modellati con la classe [User](#) nel package [shared](#)) e Centri di Monitoraggio (modellati con la classe [MonitoringCenter](#) nel package [shared](#)).

Tali connessioni sono gestite di modo da evitare problemi di concorrenza.

- **Shared** : Il modulo shared contiene classi condivise tra il [clientCM](#) e il [serverCM](#).

Queste risorse condivise includono classi per la definizione dei dati scambiati tra client e server (modelli) e l'interfaccia [RemoteDatabaseService](#) utilizzata per la comunicazione tra client e server dell'applicazione Climate Monitoring.

Questo approccio contribuisce a ridurre la duplicazione del codice, migliorando la manutenibilità e la coerenza del sistema.

La suddivisione in moduli facilita lo sviluppo, la manutenzione e l'evoluzione dell'applicazione, consentendo una maggiore flessibilità e scalabilità nel tempo.

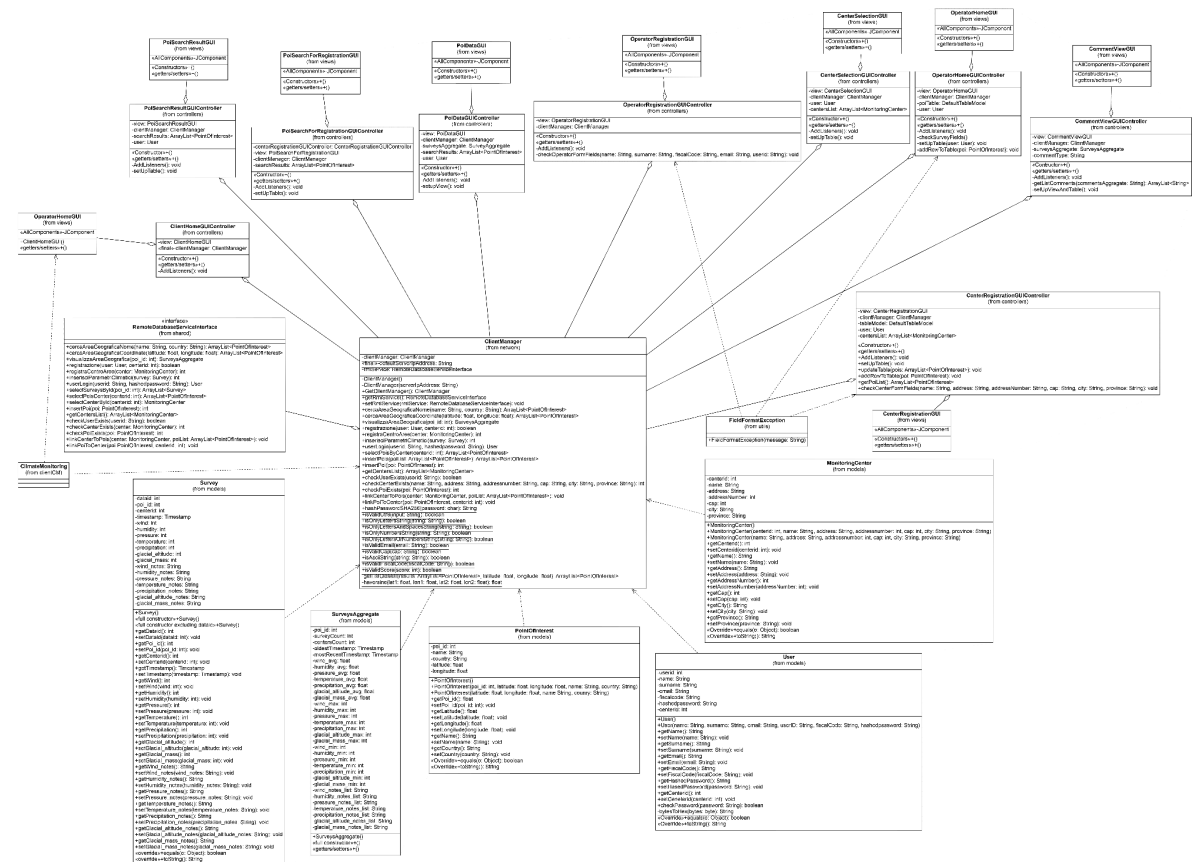
## Modulo clientCM

Il modulo [clientCM](#) contiene tutti i package e classi per implementare il client dell'applicazione Climate Monitoring.

La struttura adottata per gestire la GUI del client dell'applicazione Climate Monitoring segue parzialmente il pattern Model-View-Controller (MVC). Questo approccio organizza il codice in tre componenti principali, ciascuna con responsabilità specifiche.

In particolare all'interno del modulo **clientCM** troviamo la seguente struttura :

- Classe ClimateMonitoring
- Package views
- Package controllers
- Package network
- Package utils



Nota: per problemi di spazio e chiarezza le associazioni tra le classi controllers e i modelli non sono state raffigurate nell'immagine.

## Classe ClimateMonitoring(Client Main)

La classe `ClimateMonitoring` all'interno del package `clientCM` rappresenta il punto di ingresso principale (main) dell'applicazione client. Questa classe svolge un ruolo centrale nell'inizializzazione e nell'avvio dell'interfaccia utente, e nell'inizializzazione della classe `ClientManager` del package `network`.

## Package views

I package views e controllers (assieme alle classi del package shared.models) contengono le classi necessarie per implementare il Client di Climate Monitoring seguendo il design pattern MVC (Model View Controller).

Il package `views` all'interno del modulo `clientCM` contiene tutte le classi che definiscono le diverse finestre (view) dell'applicazione Climate Monitoring.

Ogni classe view estende `JFrame` e gestisce l'interfaccia utente associata a una specifica funzionalità o area di interesse dell'applicazione ed è composta dai soli metodi getters e setters dei componenti in essa presenti.

L'implementazione è stata effettuata utilizzando lo Swing UI Designer fornito da IntelliJ (versione 2023) che creando un GUI form permette di creare queste classi di view con del codice autogenerato da IntelliJ per l'inizializzazione.

Le classi contenute in questo package sono :

- ClientHomeGUI (finestra visualizzata al lancio dell'applicazione)
- CenterRegistrationGUI
- CenterSelectionGUI
- CommentViewGUI
- OperatorHomeGUI
- OperatorRegistrationGUI
- PoiDataGUI
- PoiSearchForRegistrationGUI
- PoiSearchResultGUI

Ogni classe view è responsabile di presentare i dati agli utenti e di raccogliere input da loro.

Queste classi sono progettate per essere separate dal resto della logica dell'applicazione, garantendo una gestione pulita delle interfacce utente e una facile manutenzione.

Nei casi in cui l'input utente richiede un cambio di finestra il controller della finestra aperta chiude la finestra invocando su di essa il `dispose()` e gestisce la creazione della nuova view con opportuni parametri che a sua volta crea il corrispondente controller a cui delega eventuali dati (necessari nel caso l'utente voglia ritornare alla finestra precedente).

Le finestre sono state progettate di modo da fornire un minimo di istruzioni all'utente con l'utilizzo di alcune JLabel contenenti testo di aiuto o una breve descrizione delle funzionalità.

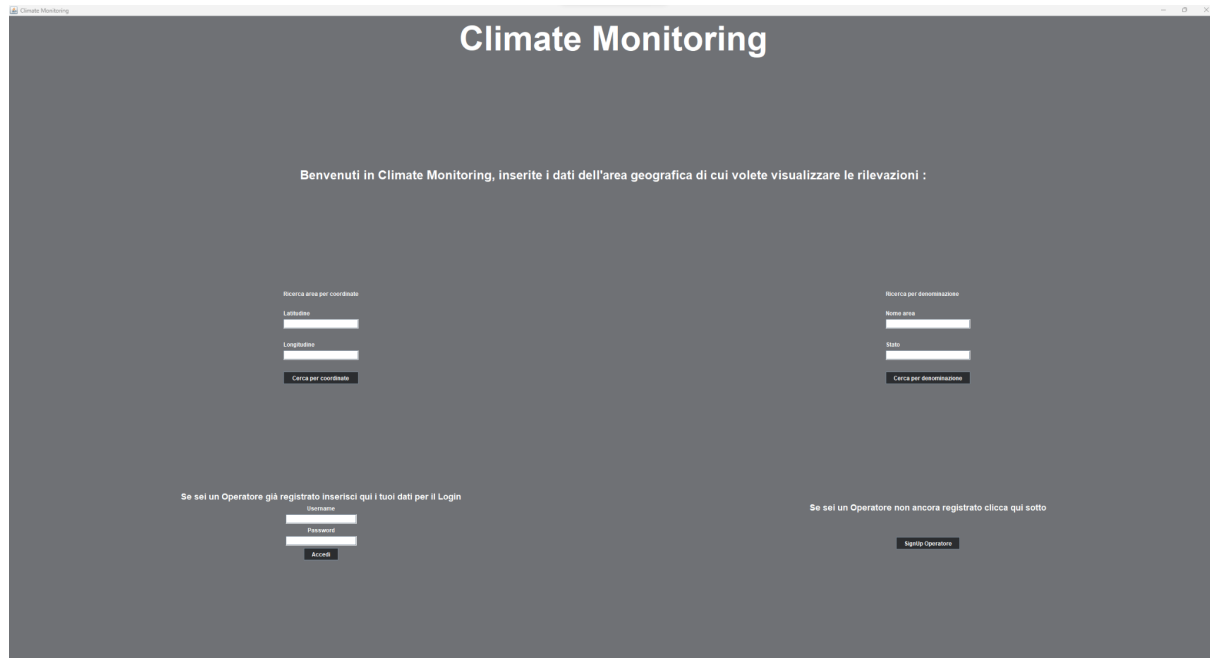
Come già detto il controllo degli input è delegato ai controllers e le view sono legate esclusivamente all'aspetto grafico della finestra che rappresentano.

Oltre alle view altri elementi grafici dell'applicazione possono essere eventuali JOptionPane per visualizzare messaggi di dialogo all'utente, come ad esempio il fatto che un campo fornito in input non ha un formato corretto.

Questi JOptionPane sono direttamente “lanciati” come “alert” dai controller dove necessario.

## ClientHomeGUI

Questa finestra è la finestra visualizzata all’avvio dell’applicazione Climate Monitoring.



Tramite questa finestra a seconda degli input forniti dall’utente è possibile raggiungere le view(finestre) :

- **PoiSearchResultGUI** : se si effettua una ricerca per coordinate o per denominazione.
- **OperatorHomeGUI** : se si effettua il login inserendo i dati corretti di un Operatore registrato.
- **OperatorRegistrationGUI** : se si effettua inizia la procedura di registrazione di un Operatore.

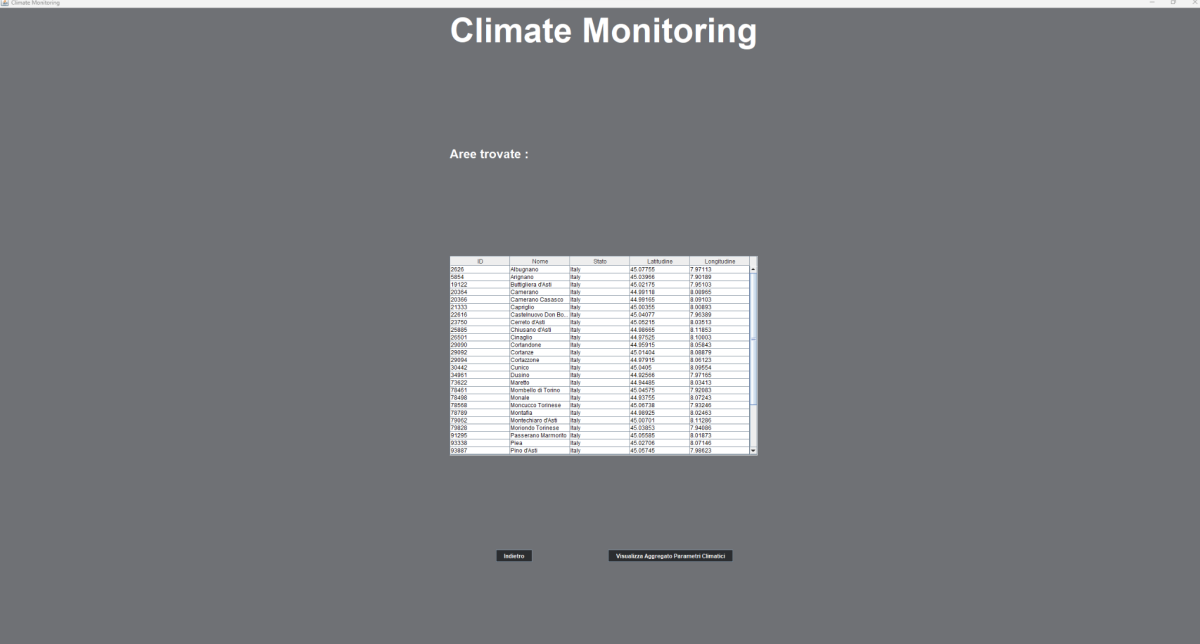
Questa view comprende componenti Java Swing di questo tipo :

- JLabel
- JTextField
- JButton
- JPasswordField
- JPanel

## PoiSearchResultGUI

Questa view(finestra) permette di visualizzare le Aree Geografiche ([PointOfInterest](#)) che sono risultato della ricerca in una JTable, gli attributi di un [PointOfInterest](#) sono visualizzati come colonne della tabella e ogni ID identifica univocamente l’Area e corrisponde alla chiave

primaria per ogni Area Geografica salvata nell'apposita tabella del database dal lato server dell'applicazione.



ID	Nome	Stato	Latitudine	Longitudine
2026	Milagrano	Ita	45.07758	7.87113
2024	Milgrano	Ita	45.07845	7.87188
19132	Bellignara d'Ad	Ita	45.02178	7.95193
20284	Camerano	Ita	44.99118	8.08959
20386	Camerano Casasco	Ita	44.99168	8.09103
21333	Cenigole	Ita	45.10365	8.08851
20116	Castelmuro Con Bi	Ita	45.04277	7.95389
22790	Ceneto d'Ad	Ita	45.05516	8.05151
20889	Chiesano d'Ad	Ita	44.88868	8.11803
20301	Chignolo	Ita	44.91502	8.10921
20090	Colfandara	Ita	44.93116	8.09242
20092	Colfandara	Ita	44.91434	8.08879
20094	Colfandara	Ita	44.91916	8.09129
30442	Corona	Ita	45.0435	8.09554
30481	Corona	Ita	44.92658	8.07188
78622	Marotta	Ita	44.94485	8.03451
78441	Montebello di Tene	Ita	45.04575	7.93803
78488	Montella	Ita	44.93755	8.07243
78558	Montesano Tene	Ita	45.04738	7.93249
78789	Montella	Ita	44.98825	8.02483
78802	Montebello d'Ad	Ita	45.05711	8.11209
78828	Montebello Tene	Ita	45.03853	7.94889
91495	Passerano Marotta	Ita	45.05685	8.01971
91138	Paia	Ita	45.02708	8.07148
10887	Pizzo d'Ad	Ita	45.05748	7.98233

Questa finestra viene visualizzata sia a seguito di una ricerca (per coordinate) da Operatore connesso a Climate Monitoring (che la effettua dalla view [OperatorHomeGUI](#)) sia a seguito di una ricerca di un Utente non Operatore (che la effettua dalla view [ClientHomeGUI](#)).

La differenziazione tra questi due casi è riconosciuta a seconda del costruttore utilizzato tra quelli disponibili a [PoiSerchResultGUI](#).

Infatti nel caso questa finestra sia stata raggiunta da un operatore connesso si utilizza un costruttore senza il parametro “user” che è di tipo [User](#) (che modella un utente registrato) mentre in caso contrario si utilizza il costruttore che non richiede di fornire il parametro “user”.

L'unica differenza tra i due costruttori è che quando creano il controller per la view inoltrano il parametro “user” in modi diversi, nel caso di un non operatore lo inoltrano come **null** e nel caso di operatore connesso inoltrano il parametro “user” ricevuto.

Questa differenziazione è utile per utilizzare la stessa view sia per ricerche effettuate da operatori e non operatori ed è necessaria per implementare la possibilità degli utenti di tornare alla corretta pagina precedente premendo su “[Indietro](#)”

Le finestre accessibili da questa finestra sono :

- **PoiDataGUI** : se si seleziona un'area nella tabella e si clicca su “[Visualizza Aggregato Parametri Climatici](#)”
- **ClientHomeGUI** : se si clicca sul tasto “[Indietro](#)” e non si è effettuato la ricerca da Operatore connesso, ovvero dalla view [OperatorHomeGUI](#)

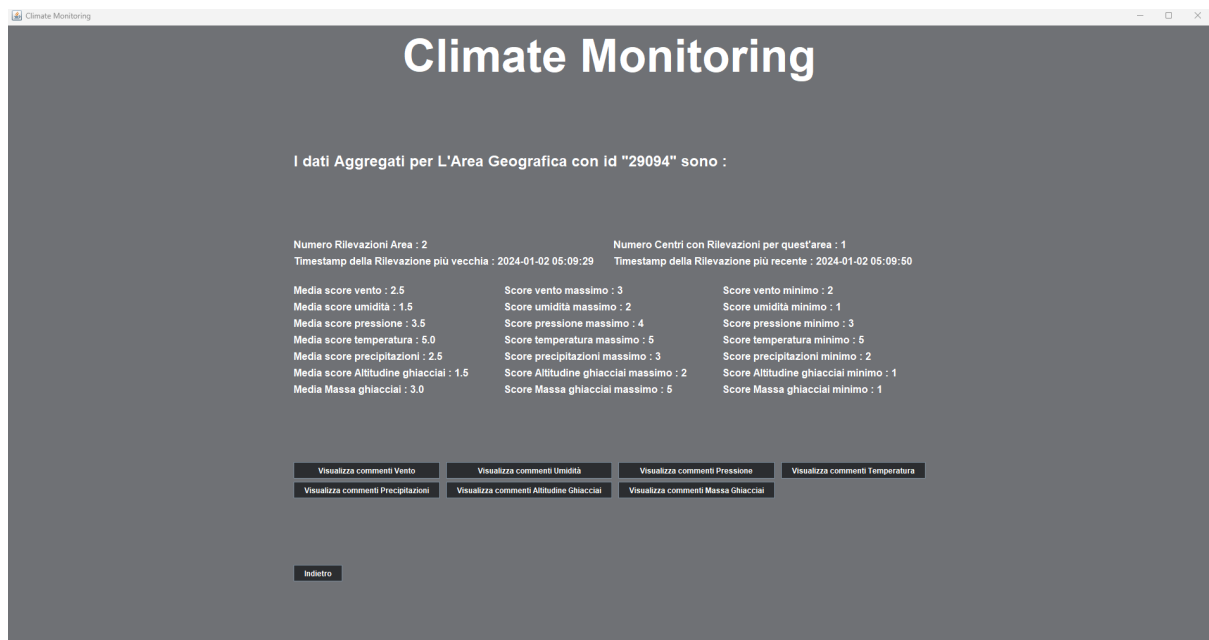
- **OperatorHomeGUI** : se si clicca sul tasto “[Indietro](#)” e si è effettuato la ricerca da Operatore connesso, ovvero dalla view [OperatorHomeGUI](#)

Questa view comprende componenti Java Swing di questo tipo :

- JLabel
- JTable
- JButton
- JPanel
- JScrollPane

## PoiDataGUI

La finestra [PoiDataGUI](#) è la finestra utilizzata per la visualizzazione dei dati aggregati ([SurveysAggregate](#)) dei Parametri Climatici su varie Rilevazioni ([Survey](#)) inserite dagli operatori di Climate Monitoring.



Questa view è raggiungibile sia da [PoiSearchResultGUI](#) sia dalla Home di un operatore ([OperatorHomeGUI](#)).

Siccome quindi anche questa finestra è comune a utenti non operatori e operatori registrati la differenziazione viene implementata seguendo la stessa logica di [PoiSearchResultGUI](#).

Da questa finestra si può navigare alle view :

- **CommentViewGUI** : se si clicca sui pulsanti per visualizzare i commenti relativi a un parametro climatico specifico (questa verrà aperta senza chiudere la view [PoiDataGUI](#)).
- **PoiSearchResultGUI** : se si clicca sul tasto “[Indietro](#)” e si è raggiunto la view a seguito di una ricerca.



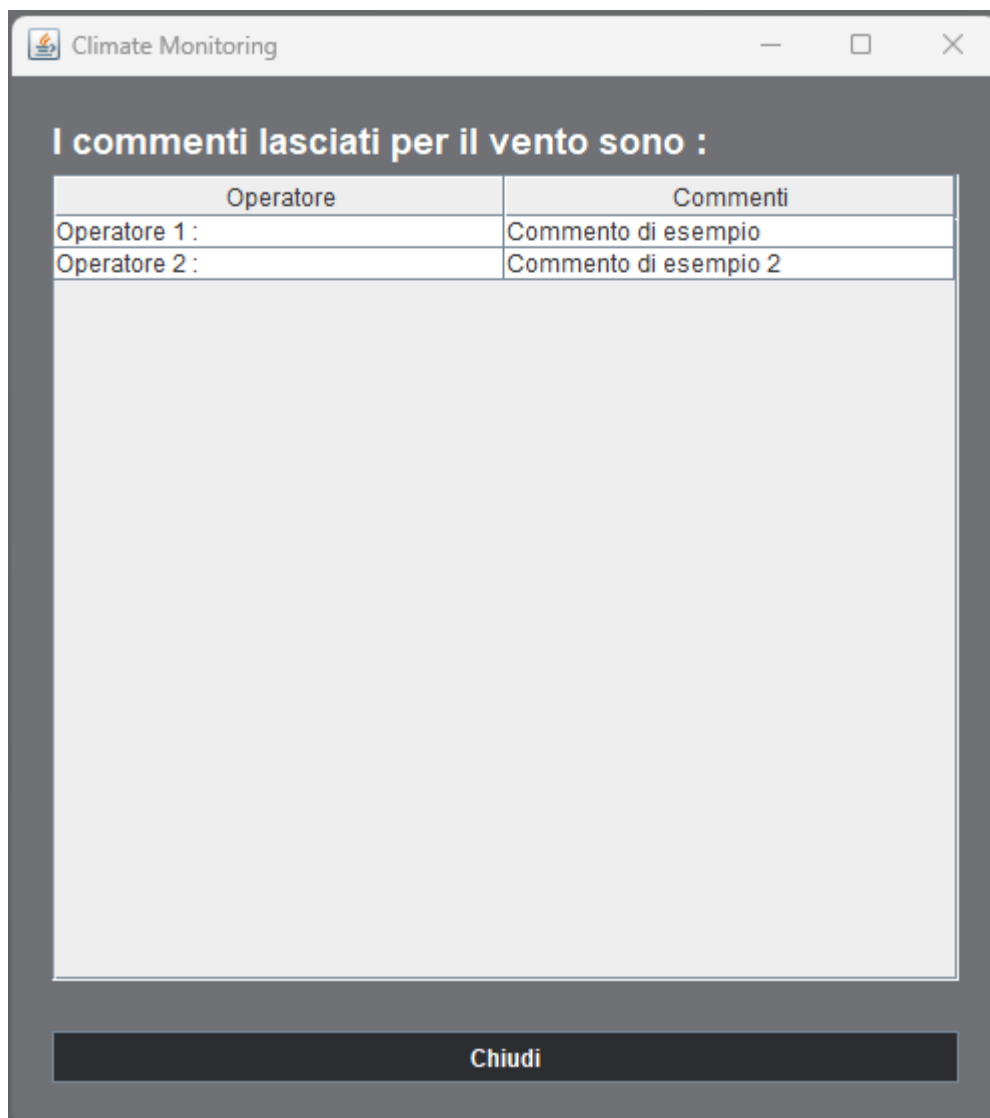
- **OperatorHomeGUI** : se si clicca sul tasto “[Indietro](#)” e si è raggiunto la view a seguito di una richiesta di visualizzazione dei dati di un'Area Geografica (monitorata dal Centro di Monitoraggio di afferenza dell’operatore) di un operatore dalla [OperatorHomeGUI](#).

Questa view comprende componenti Java Swing di questo tipo :

- JLabel
- JPanel
- JButton

### CommentViewGUI

In questa finestra l’utente può visualizzare la lista di commenti per uno specifico parametro di un'Area Geografica specifica.



I commenti vengono visualizzati in una JTable.

Cliccando su “[Chiudi](#)” la [CommentViewGUI](#) viene chiusa.

Questa view comprende componenti Java Swing di questo tipo :

- JLabel
- JTable
- JButton
- JPanel
- JScrollPane

## OperatorHomeGUI

Questa finestra è la finestra visualizzata a seguito del Login di un operatore.

Climate Monitoring

Benvenuto Lorenzo del centro con id "1" inserisci i dati dell'area geografica di cui vuoi visualizzare le rilevazioni :

Ricerca area per coordinate

Latitudine

Longitudine

Cerca per coordinate

Ricerca per denominazione

Nome area

Stato

Cerca per denominazione

Se desideri inserire i Parametri Climatici di una rilevazione completa il form sottostante, seleziona l'Area Geografica di appartenenza e clicca su "Aggiungi Parametri".

Categorie Climatiche

Specie

Score (da 1 critico a 5 ottimo)

Nome Area

Stato

Latitudine

Longitudine

Aggiungi Parametri

Visualizza Aggiunti Parametri Climatici

Aggiungi Area

Logout

ID	Categoria	Nome	Stato	Latitudine	Longitudine
1004	Categoria	Nome	Stato	44.37115	9.36123

Alla creazione di questa view vengono passati al costruttore di questa view (che crea il suo controller e li inoltra ad esso) i dati relativi all'utente che ha effettuato il Login rappresentati da un'istanza della classe [User](#) (package shared.models).

Grazie a questi dati è possibile popolare la JTable con le Aree Geografiche del Centro di Monitoraggio di appartenenza permettendo così l'aggiunta dei parametri compilando il form, selezionando una delle aree nella JTable e cliccando su “[Aggiungi Parametri](#)”.

Tramite questa finestra a seconda degli input forniti dell'utente è possibile raggiungere le view(finestre) :

- **PoiSearchResultGUI** : se si utilizzano le funzionalita di Ricerca per Coordinate e Ricerca per Denominazione.
- **ClientHomeGUI** : se si l'operatore preme il tasto per disconnettersi [“Logout”](#).
- **PoiDataGUI** : se l'operatore seleziona dalla JTable un'Area e clicca su [“Visualizza Aggregato Parametri Climatici”](#).

Questa view comprende componenti Java Swing di questo tipo :

- JLabel
- JButton
- JPanel
- JTable
- JScrollPane
- JTextField

## OperatorRegistrationGUI

In questa view si inizia la procedura di registrazione di un nuovo Operatore.

Tramite questa finestra a seconda degli input forniti dell'utente è possibile raggiungere le view seguenti :

- **CenterSelectionGUI** : se si inseriscono i dati richiesti e si clicca su [“Continua”](#).
- **ClientHomeGUI** : se si preme il tasto [“Indietro”](#)

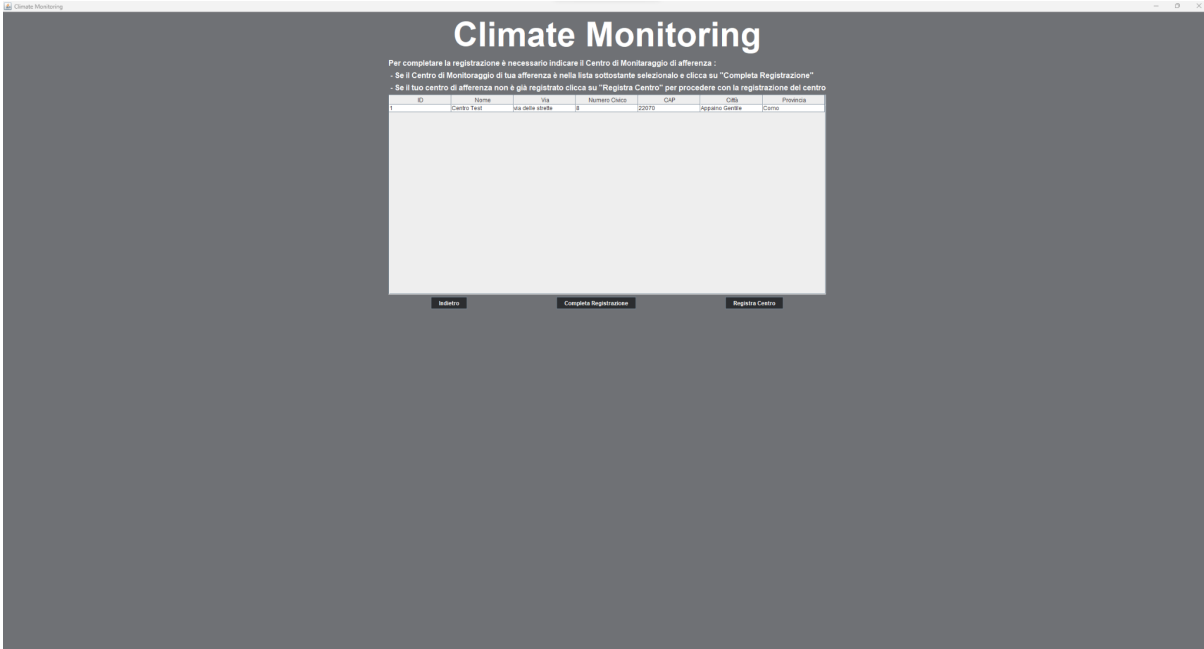
In questa finestra si raccolgono gli input necessari alla prima fase di registrazione in cui vengono inseriti i dati relativi all'utente che verranno, alla conclusione della registrazione, inseriti nel database.

Questa view comprende componenti Java Swing di questo tipo :

- JPanel
- JLabel
- JTextField
- JPasswordField
- JButton

## CenterSelectionGUI

Questa view ha lo scopo di determinare se l'operatore che si sta registrando fa' afferenza a un Centro di Monitoraggio ([MonitoringCenter](#)) già presente nel database o se vuole creare un nuovo Centro di Monitoraggio.



The screenshot shows a Java Swing window titled "Climate Monitoring". Inside the window, there is a registration form with the following structure:

**Climate Monitoring**

Per completare la registrazione è necessario indicare il Centro di Monitoraggio di afferenza :

- Se il Centro di Monitoraggio di tua afferenza è nella lista sottostante selezionalo e clicca su "Completa Registrazione"
- Se il tuo centro di afferenza non è già registrato clicca su "Registra Centro" per procedere con la registrazione del centro

ID	Nome	Via	Numero Civico	CAP	Città	Provincia
1	Centro Test	Via della scuola	1	20010	Appiano Gentile	Como

At the bottom of the form, there are three buttons: "Indietro", "Completa Registrazione", and "Registra Centro".

Questo step nella registrazione di un nuovo operatore è necessario perché per come si è progettato il database dell'applicazione Climate Monitoring ogni operatore deve avere un Centro di afferenza.

Non deve essere quindi possibile creare operatori senza un Centro.

Da questa finestra è possibile essere reindirizzati alle view :

- **ClientHomeGUI:** sia in caso si preme su "Indietro", sia in caso di registrazione completata con successo (in questo caso un opportuno JOptionPane avvisa l'utente della registrazione e alla chiusura manda a [ClientHomeGUI](#)).
- **CenterRegistrationGUI:** se l'operatore vuole registrare un nuovo Centro e quindi preme su "[Registra Centro](#)".

Questa view comprende componenti Java Swing di questo tipo :

- JPanel
- JLabel
- JButton
- JTable
- JScrollPane

## CenterRegistrationGUI

Questa view contiene tutti i campi necessari per inserire i dati necessari per la registrazione di un nuovo Centro.

The screenshot shows a Java Swing window titled "Climate Monitoring". The main content area has a dark gray background. At the top, the title "Climate Monitoring" is displayed in white. Below the title, there is a section for registering a new center. It starts with the instruction "Inserisci qui sotto i dati necessari per la registrazione del centro di monitoraggio". This is followed by a form with four input fields: "Nome Centro:", "Numero Centro:", "Città:", "Indirizzo:", "CAP:", and "Provincia:". Below the form, there is a line of text: "Cerca le aree di interesse che il tuo Centro di Monitoraggio monitora, se non presenti sarà possibile crearle". Underneath, there is a section for searching areas by coordinates. It includes a "Ricerca aree per coordinate" label, two input fields for "Latitudine" and "Longitudine", and a "Cerca per coordinate" button. To the right of the search section, there is a table titled "Aree aggiunte al Centro di Monitoraggio". The table has five columns: "ID", "Nome", "Stato", "Latitudine", and "Longitudine". Below the table, there is a line of text: "Quando la lista visualizzata comprende tutte le aree necessarie clicca su 'Completa Registrazione'". At the bottom of the window, there are two buttons: "Indietro" and "Completa Registrazione".

Nella finestra si è disposto un form per l'inserimento dei dati relativi al Centro che si vuole registrare, una JTable contenente la lista delle Aree Geografiche ([PointOfInterest](#)) che si vuole aggiungere al Centro e la funzionalità di ricerca per coordinate che l'utente deve utilizzare per aggiungerle.

Utilizzando la ricerca si aprirà (senza chiudere la view [CenterRegistrationGUI](#)) la view di [PoiSearchResultForRegistrationGUI](#) tramite la quale sarà possibile popolare la JTable con Aree trovate o create dalla stessa finestra [PoiSearchResultForRegistrationGUI](#).

Le finestre raggiungibili dalla view sono :

- **CenterSelectionGUI** : se l'utente preme il tasto "[Indietro](#)"
- **ClientHomeGUI**: se l'utente completa la registrazione del Centro e sua con successo(anche in questo caso un opportuno JOptionPane avvisa l'utente della registrazione e alla chiusura manda a [ClientHomeGUI](#)).

- **PoiSearchResultForRegistrationGUI** : questa view viene visualizzata come “popup” e non comporta la chiusura della view [CenterRegistrationGUI](#).

**Nota:** Si noti che la soluzione proposta non prevede la possibilità di creare Centri di Monitoraggio senza almeno un Area Geografica di monitoraggio, si faccia riferimento alla sezione riguardante il database per più informazioni.

Questa view comprende componenti Java Swing di questo tipo :

- JPanel
- JLabel
- JTextField
- JButton
- JTable
- JScrollPane

### PoiSearchForRegistrationGUI

Questa view viene aperta per permettere all'utente in fase di registrazione di un centro di visualizzare i risultati di una ricerca effettuata ed eventualmente permettergli di scegliere l'Area Geografica o crearla e associarla al Centro che si sta creando.

Climate Monitoring

Se l'area che volete aggiungere è stata trovata, selezionatela nella tabella sottostante e cliccate su "Aggiungi Area"

ID	Nome	Stato	Latitudine	Longitudine
2626	Albugnano	Italy	45.07755	7.97113
5854	Arignano	Italy	45.03966	7.90189
19122	Buttiglieria d'Asti	Italy	45.02175	7.95103
20364	Camerano	Italy	44.99118	8.08965
20366	Camerano Casasco	Italy	44.99165	8.09103
21333	Capriglio	Italy	45.00355	8.00893
22616	Castelnuovo Don Bo...	Italy	45.04077	7.96389
23750	Cerreto d'Asti	Italy	45.05215	8.03513
25885	Chiusano d'Asti	Italy	44.98665	8.11853
26501	Cinaglio	Italy	44.97525	8.10003
29090	Cortandone	Italy	44.95915	8.05843
29092	Cortanze	Italy	45.01404	8.08879
29094	Cortazzone	Italy	44.97915	8.06123
30442	Cunico	Italy	45.0405	8.09554
34961	Dusino	Italy	44.92566	7.97165
73622	Maretto	Italy	44.94485	8.03413
78461	Mombello di Torino	Italy	45.04575	7.92083
78498	Monale	Italy	44.93755	8.07243
78568	Moncucco Torinese	Italy	45.06738	7.93246
78789	Montafia	Italy	44.98925	8.02463
79062	Montechiaro d'Asti	Italy	45.00701	8.11286
79828	Monfondo Torinese	Italy	45.03853	7.94086
91295	Passerano Marmorito	Italy	45.05585	8.01873
93338	Piea	Italy	45.02706	8.07146
93887	Pino d'Asti	Italy	45.05745	7.98623

Aggiungi Area

Se l'area che volete aggiungere non è presente compilate i campi sottostanti e cliccate su "Crea e Aggiungi Area"

Nome

Stato

Latitudine

Longitudine

Annulla

Crea e Aggiungi Area

Inserisci qui sotto i dati necessari

Nome Centro :

Numero Civico :

Città :

Cerca le aree di interesse che il

Ricerca area per coordinate

Latitudine

45

Longitudine

8

Cerca per coordinate

La JTable viene popolata con il risultato della ricerca che viene passato alla creazione della [PoiSearchForRegistrationGUI](#), ed è possibile selezionare un'Area dalla tabella e aggiungerla al Centro.

In alternativa l'utente può popolare i campi per la creazione di una nuova Area Geografica e crearla.

In entrambi i casi questa view viene chiusa e il [PointOfInterest](#) selezionato o creato viene passato al controller della view [CenterRegistrationGUI](#) che la inserisce nella JTable opportuna.

Questa finestra viene visualizzata senza chiudere la view [CenterRegistrationGUI](#). Questa view comprende componenti Java Swing di questo tipo :

- JPanel
- JLabel
- JButton
- JTable
- JScrollPane

## Package controllers

Il package [controllers](#) contiene le classi controller che fungono da intermediari tra il modello e la vista.

Questi controller ricevono gli input dagli utenti attraverso le viste, elaborano le richieste, interagiscono con i modelli e il [ClientManager](#) per ottenere o aggiornare i dati e quindi aggiornano la vista di conseguenza.

La separazione del controller consente di isolare la logica dell'applicazione, facilitando la gestione e il testing.

Le classi contenute in questo package sono :

- ClientHomeGUIController
- CenterRegistrationGUIController
- CenterSelectionGUIController
- CommentViewGUIController
- OperatorHomeGUIController
- OperatorRegistrationGUIController
- PoiDataGUIController
- PoiSearchForRegistrationGUIController
- PoiSearchResultGUIController

Come intuibile dal nome, ogni controller gestisce la corrispondente view, aggiornandola e modificandola a seconda degli input dati dall'utente sulla view.

Alla loro creazione si occupano di registrare gli opportuni Listeners ai componenti delle view chiamando il metodo privato [AddListeners\(\)](#) (presente in tutti i controllers), che una volta chiamato da un controller associa ad ogni componente della view del controller una logica che verrà eseguita alla ricezione dell'input sul componente.

Questi Listeners quindi eseguiranno il codice necessario ad elaborare l'input a cui rispondono eventualmente richiamando le opportune procedure del [ClientManager](#).

Nella maggior parte dei casi il codice eseguito ad un input è composto da tre fasi:

1. Recupero dei dati inseriti e rilevanti all'input dato dalla view tramite i getter forniti dalle classi view.
2. Elaborazione e verifica della correttezza di tali dati (localmente o richiamando opportuni metodi statici della classe [ClientManager](#)), gestendo tutti quei casi in cui i dati non sono stati inseriti correttamente informando l'utente tramite l'utilizzo di JOptionPane con opportuni messaggi.
3. Esecuzione del codice che implementa la funzionalità associata all'input ricevuto.
4. Eventuale esecuzione del codice per passare ad un'altra view e chiusura della finestra obsoleta.



In particolare il punto 3 comporta l'interazione con il singleton del [ClientManager](#) per tutte quelle operazioni che richiedono la comunicazione con il lato server dell'applicazione per recuperare dati dal database o interrogarlo.

Si occupano inoltre di gestire le situazioni in cui l'input dell'utente provoca uno "switch" ad un'altra finestra(view), creandola e passandogli gli opportuni dati da delegare al controller che la gestirà e chiudendo la propria view ormai obsoleta tramite [dispose\(\)](#).

### ClientHomeController

Questa classe funge da controller per la view [ClientHomeGUI](#), alla sua creazione inizializza i suoi attributi con: il riferimento al singleton del [ClientManager](#) e il riferimento alla view [ClientHomeGUI](#).

Le funzionalità che gestisce sono :

- **Ricerca per Coordinate**
- **Ricerca per Denominazione**
- **Login**
- **SignUp**

Ad eccezione del SignUp che effettua solo un "switch" di view sono tutte gestite elaborando i dati inseriti rilevanti, verificandone la correttezza e richiamando gli opportuni metodi del [ClientManager](#) che gestisce la comunicazione con il server e restituisce il risultato opportuno al [ClientHomeController](#).

Il [ClientHomeController](#) si occupa poi di lanciare la view opportuna e chiudere la [ClientHomeGUI](#).

### PoiSearchResultGUIController

Il [PoiSearchResultGUIController](#) è il controller della view [PoiSearchResultGUI](#), alla sua creazione riceve il riferimento alla view [PoiSearchResultGUI](#) a cui si deve associare, una lista di [PointOfInterest](#) (Aree geografiche risultato della ricerca) e un riferimento di tipo [User](#) (**null** se creato da un utente non connesso come operatore).

Inoltre, sempre alla sua creazione, popola la JTable della sua view con la lista di Aree Geografiche ricevute e chiama il metodo [AddListeners\(\)](#).

Il riferimento di tipo User ricevuto alla creazione viene utilizzato per capire a quale view tornare nel caso l'utente decida di tornare alla view precedente.

Le uniche funzionalità che deve gestire sono :

- **Visualizzazione dei dati aggregati di un'Area Geografica**
- **Ritorno alla view precedente**

La **Visualizzazione dei dati aggregati di un'Area Geografica** verifica che un area sia selezionata nella JTable e chiamando il metodo `visualizzaAreaGeografica()` del `ClientManager` riceve il risultato che poi invia alla view `PoiDataGUI` che crea dove verranno visualizzati i dati.

Inoltre dopo la creazione della `PoiDataGUI` procede a fare il `dispose()` della sua `PoiSearchResultGUI`.

Il **Ritorno alla view precedente** avviene andando a determinare la view precedente a seconda del riferimento `User` ricevuto in creazione, creando tale view e chiudendo quella a cui il controller è associato.

### PoiDataGUIController

Questo è il controller della view `PoiDataGUI` e alla sua creazione riceve come parametri la view a cui si deve associare e i dati aggregati delle rilevazioni che devono essere visualizzati in forma di un'istanza del modello `SurveysAggregate`.

Inoltre vengono ricevuti come parametri una lista di `PointOfInterest` e un'istanza di `User` che a seconda di come si è raggiunto la `PoiDataGUI` view sono o non sono `null`.

Controllando questi parametri il controller riconosce come comportarsi nel caso l'utente voglia tornare alla pagina precedente.

Alla creazione viene inoltre modificato il testo delle JLabel significative alla visualizzazione dei dati contenuti nell'istanza di `SurveysAggregate`, chiamando un metodo privato interno al controller `setupView()`.

Successivamente registra i Listeners ai componenti della view con il metodo `AddListeners()`.

Le funzionalità che deve gestire sulla view sono :

- **Visualizzazione dei commenti**
- **Ritorno alla view precedente**

La **visualizzazione dei commenti** è relativa ad ogni singolo parametro climatico ma in ogni caso viene semplicemente creata e aperta una `CommentViewGUI` nella quale visualizzarli, passando anche una stringa per identificare per quale parametro climatico si vogliono visualizzare.

Il **Ritorno alla view precedente** avviene andando a determinare la view precedente a seconda del riferimento `User` e la lista di `PointOfInterest` ricevuti in creazione , creando tale view e chiudendo quella a cui il controller è associato.

### CommentViewGUIController

E' il controller della view `CommentViewGUI` per la visualizzazione dei commenti di uno specifico parametro.

Alla creazione riceve la view a cui associarsi , il `SurveyAggregate` da cui prelevare i commenti di interesse e una stringa `commentType` che identifica di quale parametro climatico si vogliono visualizzare i commenti.

Il controller procede poi a inizializzare la view in modo corretto chiamando il metodo privato `setUpViewAndTable()` al suo interno e il metodo `AddListeners()` per la registrazione degli Action Listener ai componenti della finestra.

Questo controller deve controllare solo la funzionalità di chiusura della sua view tramite tasto "`Chiudi`".

### OperatorHomeGUIController

Il controller `OperatorHomeGUIController` funge da controller per la view `OperatorHomeGUI`.

Alla sua creazione riceve la view a cui associarsi e un riferimento ad un'istanza di `User` che rappresenta tramite il modello l'operatore che ha effettuato il Login.

Questo riferimento è essenziale per identificare a quale Centro di Monitoraggio afferisce l'operatore.

Il controller inizializza la view chiamando il metodo `setUpTable()` passandogli l'oggetto `User`, questo metodo popola la `JTable` presente nella `OperatorHomeGUI` con le Aree Geografiche del Centro di Monitoraggio di appartenenza.

Le funzionalità che deve gestire per questa view sono :

- **Ricerca per coordinate e Ricerca per denominazione**
- **Aggiunta Parametri Climatici**
- **Aggiunta di un'Area Geografica al proprio centro**
- **Visualizzazione dei dati aggregati di un'Area Geografica del proprio centro**
- **Logout**

La **ricerca per coordinate** e **ricerca per denominazione** vengono gestite analogamente a come il `ClientHomeGUIController` le gestisce.

Per l'**aggiunta parametri climatici** il controller verifica che sia selezionata l'Area Geografica alla quale si vogliono aggiungere i parametri, verifica la correttezza del formato dei dati inseriti e se tutto corretto procede a creare un'istanza del modello `Survey` (passandogli i parametri inseriti e gli id dell'Area e del Centro) che viene poi inviata al `ClientManager` con una chiamata al metodo `inserisciParametriClimatici()` che gestisce la comunicazione con il server per l'inserimento.

L' **aggiunta di un'Area Geografica al proprio centro** viene gestita controllando sempre la correttezza del formato dei dati inseriti, la creazione di un [PointOfInterest](#) che rappresenta l'Area che si vuole creare e aggiungere al database e al Centro e la successiva call al metodo [insertPoi\(\)](#) di [ClientManager](#) passandogli il [PointOfInterest](#) creato.

In questo metodo il [ClientManager](#) comunicherà con il server l'intenzione di inserire una nuova Area chiamando l'opportuno metodo RMI.

La **Visualizzazione dei dati aggregati di un'Area Geografica del proprio centro** segue la stessa logica vista per il [PoiSearchResultGUIController](#) con la differenza che l'utente deve avere selezionato un'area dalla JTable contenente la lista delle aree monitorate dal proprio centro.

Il **Logout** viene gestito semplicemente tornando alla [ClientHomeGUI](#) e chiamando il [dispose\(\)](#) per la [OperatorHomeGUI](#).

### OperatorRegistrationGUIController

Costituisce il controller per la view [OperatorRegistrationGUI](#) e alla creazione si associa ad essa e chiama il metodo [AddListeners\(\)](#) per registrare i Listeners necessari.

Essendo il controller della view corrispondente alla prima fase della registrazione di un nuovo operatore ha poche funzionalità da gestire :

- **Controllo dati inseriti per l'operatore e procedere in caso corretti**
- **Ritorno alla view precedente**

Per il **controllo dati inseriti per l'operatore e procedere in caso siano corretti** vengono semplicemente elaborati gli input forniti per i campi rilevanti alla creazione di un nuovo operatore e in caso siano corretti si recupera tramite [ClientManager](#) la lista dei centri esistenti (il [ClientManager](#) recupera dal database comunicando con il server). Successivamente viene creata la view della fase di registrazione successiva ([CenterSelectionGUI](#)) a cui viene passata la lista dei centri e un'istanza User con i dati inseriti. Infine avviene il [dispose\(\)](#) della view [OperatorRegistrationGUI](#).

Il **Ritorno alla view precedente** avviene tornando sulla view [ClientHomeGUI](#).

### CenterSelectionGUIController

Questo controller è associato a view di tipo [CenterSelectionGUI](#) e riceve alla creazione oltre alla view a cui si deve associare due parametri.

Il primo è di tipo User e contiene le informazioni necessarie per registrare l'operatore in caso di successo della procedura di registrazione.

Il secondo è la lista di Centri di Monitoraggio ([MonitoringCenter](#)) con cui popolare la JTable della [CenterSelectionGUI](#), di modo da permettere all'utente di poter selezionare come centro di afferenza il suo centro se presente nella lista. Questo è gestito tramite [setUpTable\(\)](#).

Infine il controller chiama il metodo `AddListeners()`.

Le funzionalità che il controller gestisce sulla view sono:

- **Completamento Registrazione**
- **Inizio Creazione Nuovo Centro**
- **Ritorno alla view precedente**

Per gestire il **Completamento Registrazione** il controller controlla che sia stato selezionato un centro dalla `JTable` ne recupera l'id e chiama il metodo `registrazione()` del `ClientManager` passandogli l'`User` e l'id del centro selezionato, il `ClientManager` passa questi valori al server che provvede a registrare l'utente come operatore.

A questo punto se il controller procede ad informare l'utente, e in caso di registrazione con successo a riportarlo sulla view `ClientHomeGUI`.

L' **Inizio Creazione Nuovo Centro** viene gestito dal controller creando la view `CenterRegistrationGUI` e inoltrando comunque l'istanza di `User` e la lista centri (utile solo nel caso l'utente decida poi di tornare su questa view), successivamente procede al `dispose()` della sua view.

Il **Ritorno alla view precedente** è gestito tornando alla view `OperatorRegistrationGUI`.

#### CenterRegistrationGUIController

Questo controller si associa a istanze di `CenterRegistrationGUI` e alla sua creazione riceve la view a cui associarsi, l'istanza di `User` e la lista dei centri (utilizzata in caso sia necessario tornare alla finestra di `CenterSelectionGUI`).

Inoltre nel suo costruttore inizializza la `JTable` della `CenterRegistrationGUI`, dove verranno aggiunte le Aree Geografiche da associare al centro che si sta creando, tramite il metodo privato interno al controller `setUpTable()`.

Infine aggiunge i listeners a `CenterRegistrationGUI` chiamando `AddListeners()`.

Le funzionalità fornite dalla view che deve gestire sono:

- **Ricerca per coordinate**
- **Completamento della registrazione del Centro e dell'Operatore**
- **Ritorno alla view precedente**

La **Ricerca per coordinate** viene gestita sempre richiamando il metodo `cercaAreaGeograficaCoordinate()` (dopo un controllo della correttezza delle coordinate fornite) del `ClientManager` che si occupa di contattare il server e ritornare il risultato come una lista di `PointOfInterest`.

La differenza in questo caso è che il risultato viene visualizzato in una view specifica per questa ricerca durante la procedura di registrazione, ovvero la view `PoiSearchForRegistrationGUI`, che il controller avvia passando la lista risultato della ricerca.

In questo caso inoltre la view [CenterRegistrationGUI](#) non viene chiusa. In questa view sarà possibile scegliere l'Area Geografica da inserire o crearne una per l'inserimento.

Per il **Completamento della registrazione del Centro e dell'Operatore** il controller controlla come primo passo che tutti i dati per la registrazione del centro siano stati forniti in un formato corretto, controlla che la JTable contenente le Aree da aggiungere al nuovo Centro non sia vuota e procede a richiamare i metodi del [ClientManager](#) per la registrazione del Centro e dell'operatore.

Procede infine ad informare l'utente con un messaggio di successo o insuccesso della registrazione. Nel caso di successo riporta su la view [ClientHomeGUI](#).

Il **Ritorno alla view precedente** è gestito tornando alla view [CenterSelectionGUI](#).

All'interno del controller sono anche presenti alcuni metodi privati che vengono utilizzati per l'aggiornamento della JTable delle Aree del Centro con i [PointOfInterest](#) che l'utente decide di aggiungere.

## PoiSearchForRegistrationGUIController

Questa classe funge da controller per le view [PoiSearchForRegistrationGUI](#), alla creazione sicve i risultati della ricerca con cui popola la JTable della view chiamando [setUpTable\(\)](#) e in seguito chiama il metodo [AddListeners\(\)](#).

Le funzionalità della view che il controller deve gestire sono :

- **Aggiunta dell'Area Geografica selezionata**
- **Creazione e aggiunta di un'Area Geografica**
- **Chiusura finestra**

L'**Aggiunta dell'Area Geografica selezionata** viene gestita dal controller andando a controllare che l'utente abbia selezionato un'area dalla JTable e passando al [CenterRegistrationGUIController](#) l'area selezionata come istanza di [PointOfInterest](#).

La **Creazione e aggiunta di un'Area Geografica** viene gestita facendo un controllo sui dati inseriti dall'utente e creando una corrispondente istanza di [PointOfInterest](#) da passare al [CenterRegistrationGUIController](#)

**Nota** : In questo caso l'Area creata viene solo aggiunta alla lista di quelle che l'utente vuole aggiungere al centro, che è visualizzata nella JTable della view [CenterRegistrationGUI](#), e non nel database. Solo al completamento della registrazione del centro verrà effettivamente inserita nel database e gli verrà assegnato un id.

La **Chiusura finestra** vede il controller richiamare semplicemente il `dispose()` sulla view.

## Package network

All'interno di questo package è presente la sola classe [ClientManager](#).

## ClientManager

La classe [ClientManager](#) costituisce il nucleo del lato client dell'applicazione "Climate Monitoring".

La sua responsabilità principale è gestire tutte le comunicazioni tra il lato client e il lato server mediante l'utilizzo di Java RMI (Remote Method Invocation).

La comunicazione avviene attraverso l'interfaccia [RemoteDatabaseServiceInterface](#) contenuta nel package [shared](#), la quale definisce i metodi che il lato client può invocare in remoto sul lato server.

In particolare le funzionalità principali di questa classe sono:

**Comunicazione RMI:** è responsabile di stabilire e gestire la comunicazione RMI(Remote Method Invocation) con il lato server.

Tramite [RemoteDatabaseServiceInterface](#), consente ai controller del lato client di invocare le opportune operazioni sul lato server, fornendo un'interfaccia pulita e orientata agli oggetti per le richieste e le risposte.

**Elaborazioni Locali dei Dati:** Gestisce elaborazioni locali dei dati che non richiedono l'intervento del lato server, come ad esempio l'hashing delle password utilizzando l'algoritmo SHA-256 fornito dalla classe [java.security.MessageDigest](#).

Questo approccio permette di diminuire il carico sul lato server, ottimizzando le prestazioni complessive dell'applicazione.

**Metodi di Utility:** fornisce metodi di utility per i controller del lato client, inclusi calcoli come la distanza tra due coordinate di Aree Geografiche e il test dei formati degli input tramite espressioni regolari.

In particolare tra questi metodi il metodo [getListClosest\(\)](#) che utilizza il metodo [haversine\(\)](#) è utilizzato per trovare tutte le aree di interesse ritenute abbastanza vicine (entro i 10Km) in caso le coordinate non corrispondano ad un'area specifica nel database.

Queste funzionalità permettono una gestione locale e efficiente delle operazioni non direttamente connesse alla comunicazione col server ma richieste da vari controller.

Permettono inoltre una gestione centralizzata di operazioni comuni a tutti i controller, contribuendo a diminuire la duplicazione di parti di codice.

**Implementazione del Pattern Singleton:** La classe [ClientManager](#) implementa il pattern Singleton, garantendo che esista una sola istanza della classe nell'intera applicazione client.

Questo approccio è stato ritenuto adeguato per una gestione delle comunicazioni centralizzata.

L'utilizzo di un'unica istanza riduce la complessità dell'applicazione e facilita il controllo delle risorse semplificando la coerenza delle operazioni e facilitando la manutenzione.

Alla creazione dell'unica istanza di [ClienteManager](#) che avviene al lancio dell'applicazione, esso crea procede a recuperare un riferimento ad un oggetto remoto (che implementa l'interfaccia [RemoteDatabaseServiceInterface](#)) dal Registry, salvandolo tra i suoi attributi.

Tale riferimento funge quindi da stub RMI nella comunicazione tra il [ClientManager](#) e lo skeleton lato server (istanza di [RemoteDatabaseService](#)).

All'interno del codice del costruttore privato di [ClientManager](#) troviamo infatti le seguenti linee di codice:

```
Registry registry =  
LocateRegistry.getRegistry(defaultServerIpAddress, 1099);  
  
this.rmiService = (RemoteDatabaseServiceInterface)  
registry.lookup("RemoteDatabaseService");
```

**NOTA** : il valore di [defaultServerIpAddress](#) è stato lasciato "localhost". La classe [ClientManager](#) prevede che l'hostname del server sia codificato in modo definitivo all'interno di questa variabile se si desidera lanciare client da macchine diverse da quelle su cui è in esecuzione il server di Climate Monitoring. Infatti in un vero deployment dell'applicazione l'hostname del server sarebbe statico e invariabile.

## Package utils

Questo package contiene solo una classe per modellare un tipo specifico di Exception chiamata [FieldFormatException](#).

### FieldFormatException

Questa classe estende la classe Exception e viene utilizzata nel lato client laddove è necessario sollevare un errore dovuto a inserimenti di formati non corretti in Field di input (JTextField).

Viene quindi utilizzata nei controller nelle fasi di verifica degli input testuali.

## Modulo serverCM

Il modulo serverCM costituisce la componente server dell'applicazione "Climate Monitoring", fornendo le funzionalità necessarie per la gestione e l'accesso ai dati nel database PostgreSQL.

Progettato seguendo principi di modularità e scalabilità, questo modulo si compone di diverse classi e package, ognuno svolgendo un ruolo specifico all'interno del sistema.





Una volta che l'utente inserisce i propri dati o preme "Invio" senza inserire dati, il [ServerMain](#) procede a creare un'istanza di [RemoteDatabaseService](#) fornendogli le credenziali ottenute.

Creando l'istanza di [RemoteDatabaseService](#) viene inizializzato il singleton del [DbManager](#).

Nel terminale in cui è stato lanciato il server viene visualizzato se il collegamento al database è riuscito e se quindi il server è pronto ad offrire i servizi di Climate Monitoring.

```
Insert your postgresql username, or press Enter to log with the default credentials (the default credentials can be found in the DBconfig.config file inside the jar launched) :  
  
Connecting to database ...  
Connection to db postgres successful.  
Checking if database dbcm already exists ...  
Database dbcm already exists, reconnecting ...  
Reconnecting to database ...  
Reconnection successful, connection ready.  
DATABASE READY.  
SERVER IS READY.
```

In caso DbManager abbia verificato che il database non esiste ancora l'output è il seguente:

```
Insert your postgresql username, or press Enter to log with the default credentials (the default credentials can be found in the DBconfig.config file inside the jar launched) :  
  
Connecting to database ...  
Connection to db postgres successful.  
Checking if database dbcm already exists ...  
Database dbcm does not exist, starting the create process ...  
Creating database ...  
Database created successfully.  
Switching to database dbcm ...  
Reconnecting to database ...  
Reconnection successful, connection ready.  
Creating Tables ...  
Tables created successfully.  
Initializing Database ...  
Populating table "coordinatemonitoraggio" ...  
Data inserted successfully.  
Database ready.  
DATABASE READY.  
SERVER IS READY.
```

**Nota** : il processo di inizializzazione del database potrebbe richiedere alcuni minuti, data la grande mole di dati da inserire.

Il [ServerMain](#) si occupa anche di creare un shutdown hook al metodo pubblico statico [closeDataSource\(\)](#) del [DbManager](#) per chiudere la pool di connessioni che esso mantiene al database, permettendo così di assicurare che in caso il server venga chiuso o fermato le connessioni al database vengano chiuse.

```
Runtime.getRuntime().addShutdownHook(new Thread(() -> {  
    System.out.println("Shutting down...");  
    closeDataSource();  
    System.out.println("Shutdown complete.");  
}));
```

Dopodiché rende disponibile sul Registry RMI l'istanza di [RemoteDatabaseService](#) alla porta di default 1099 e conclude la sua esecuzione.

```
Registry registry = LocateRegistry.createRegistry(1099);  
registry.rebind("RemoteDatabaseService", remoteDatabaseService);
```

## Package database

Il package database all'interno del modulo serverCM è responsabile della gestione dell'accesso al database PostgreSQL utilizzato nell'applicazione "Climate Monitoring". Contiene due classi principali, [DbManager](#) e [PredefinedQuery](#), ciascuna svolge ruoli specifici nell'interazione con il database.

In particolare il [DbManager](#) gestisce le connessioni al database e se necessario lo inizializza nel caso non sia già stato creato, mentre la classe [PredefinedQuery](#) contiene costanti statiche e metodi statici che rappresentano tutte le query sql che verranno utilizzate dall'applicazione.

### DbManager

La classe [DbManager](#) è la componente fondamentale del package database. Essa implementa il pattern Singleton per garantire un'unica istanza nell'applicazione e gestisce una pool di connessioni al database PostgreSQL utilizzando JDBC come driver di connessione e Apache DBCP2 per la gestione della pool.

Nel caso in cui il database "dbcm" non esista, [DbManager](#) gestisce la sua creazione automatica e l'inizializzazione con i dati presenti nel file draft di excel "[geonames-and-coordinates.xlsx](#)" presente nella cartella config della root del jar o nella cartella [main/resources/config](#) dei source file.

Questo semplifica il processo di installazione e configurazione dell'applicazione, eliminando la necessità di interventi manuali sul database.

Grazie ad Apache DBCP2 crea e mantiene una pool di connessioni che verrà utilizzata man mano che al [RemoteDatabaseService](#) arriveranno richieste di interazione con il database da parte di client.

Questa pool è rappresentata da un oggetto di tipo [BasicDataSource](#) di DBCP2 e viene inizializzata con il seguente codice.

```
// Initialize DBCP DataSource
static {
    dataSource = new BasicDataSource();
    dataSource.setUrl(JDBC_URL);
    dataSource.setUsername(jdbcUser);
    dataSource.setPassword(jdbcPassword);
    dataSource.setMinIdle(10);
    dataSource.setMaxIdle(20);
    dataSource.setMaxTotal(200);
    dataSource.setMaxOpenPreparedStatements(400);
}
```

I valore di [MinIdle](#), [MaxIdle](#), [MaxTotal](#) e [MaxOpenPreparedStatements](#) sono stati scelti secondo quello che è stato valutato essere il carico possibile del database.

E' dunque prevista un cambiamento attivo di tali valori a seconda del rate di adozione dell'applicazione in versioni successive.

Per maggiori informazioni a riguardo si consiglia di visitare il seguente link:

<https://commons.apache.org/proper/commons-dbcp/apidocs/index.html>

Questi valori sono pensati per essere modificati a seconda del carico effettivo dell'applicazione con uno sviluppo attivo dell'applicazione.

Si è scelto l'utilizzo di una connection pool con Apache DBCP2 per i seguenti motivi:

- **Miglioramento delle Prestazioni:** La connection pool mantiene aperte e pronte per l'uso un certo numero di connessioni al database. Ciò elimina la necessità di aprire e chiudere una connessione ogni volta che un'operazione di database deve essere eseguita. L'apertura e la chiusura di connessioni possono essere costose in termini di tempo e risorse, quindi l'utilizzo di una pool riduce significativamente l'overhead associato.
- **Riduzione del Carico sul Database:** La pool di connessioni può gestire in modo efficiente le connessioni attive e inattive. Mantenendo le connessioni inattive nel pool, è possibile evitare il sovraccarico del database causato da un'elevata frequenza di apertura e chiusura delle connessioni.
- **Gestione delle Risorse:** La connection pool gestisce in modo automatico e ottimale le risorse di connessione. Ad esempio, può ridurre il numero di connessioni aperte durante periodi di basso carico e aumentare automaticamente il numero di connessioni disponibili durante periodi di picco.
- **Riuso delle Connessioni:** Una volta che una connessione è stata aperta e utilizzata, viene restituita alla pool e può essere riutilizzata da altre parti dell'applicazione. Questo promuove il riuso delle connessioni, contribuendo a una gestione più efficiente delle risorse.
- **Controllo di multiple Connessioni:** in situazioni in cui le richieste di accesso al database sono simultanee tra client la pool di connessioni può gestire in modo efficiente le richieste in arrivo, distribuendo le connessioni disponibili tra le richieste concorrenti, diminuendo sostanzialmente problemi relativi alla concorrenza.
- **Facilità d'Uso e Configurabilità:** L'API di DBCP è progettata per essere user-friendly e facile da utilizzare con una vasta gamma di opzioni di configurazione per adattarsi alle esigenze specifiche dell'applicazione.

In conclusione questa classe costituisce l'interfaccia tra il database "dbcm" e il [RemoteDatabaseService](#), che richiede connessioni al database dalla pool gestita dal [DbManager](#) per soddisfare le richieste dei client.

### PredefinedQuery

Come detto precedentemente questa classe contiene costanti statiche e metodi statici che rappresentano tutte le query sql che verranno utilizzate dal [RemoteDatabaseService](#) per soddisfare le richieste dei client o dal DbManager per creare il database nel caso non esista.

In particolare abbiamo tre sezioni di costanti statiche:

- Insert: sono tutti i comandi sql utili per inserire i dati del database
- Select: sono tutti i comandi sql utili per interrogare il database e ottenere informazioni
- Create: sono tutti i comandi sql utili alla creazione del database

Tramite la definizione di tre classi enumerative all'interno di [PredefinedQuery](#) (una per ogni tipo di operazione), si è raggruppato e dato un nome ad ogni query specifica che richiama il codice sql grazie a tre [Hashtable](#) statiche la cui chiave è un'istanza del tipo enumerativo che ha come valore il codice corrispondente.

Con questo approccio si centralizza in una sola classe tutte le informazioni relative alle query utilizzate nell'applicazione e si facilita l'accesso al codice delle stesse da parte del [RemoteDatabaseService](#) o del [DbManager](#).

## Package network

Il package network contiene la sola classe [RemoteDatabaseService](#) che implementando l'interfaccia Java RMI (Remote Method Invocation) [RemoteDatabaseServiceInterface](#) garantisce l'implementazione dei metodi chiamati in remoto dai client al lato server per eseguire le operazioni necessarie sul database.

### RemoteDatabaseService

La classe [RemoteDatabaseService](#), implementando l'interfaccia RMI [RemoteDatabaseServiceInterface](#), offre una serie di metodi RMI per consentire ai client di interagire con il sistema di gestione del database che è diviso nella classe [RemoteDatabaseService](#) che determina ed esegue su connessioni le query opportune e il singleton [DbManager](#) che gestisce le connessioni al database.

La classe [RemoteDatabaseService](#) implementa tutti i metodi definiti nell'interfaccia [RemoteDatabaseServiceInterface](#).

Questi metodi coprono diverse funzionalità, tra cui la ricerca di aree geografiche, la visualizzazione di informazioni aggregate, la registrazione di utenti, l'inserimento di parametri climatici, il login degli utenti, e altro ancora.

Ogni metodo RMI implementato nella classe è progettato per gestire specifiche operazioni richieste dai client (che vi accedono dal loro [ClientManager](#)), fornendo un'interfaccia chiara e distinta per ciascuna funzionalità.

Ad esempio, il metodo [cercaAreaGeograficaNome\(\)](#) consente la ricerca di aree geografiche basate su nome e paese, mentre registrazione gestisce la registrazione degli utenti.

Durante l'esecuzione di ciascun metodo, la classe [RemoteDatabaseService](#) utilizza l'istanza singleton di [DbManager](#) per ottenere una connessione dal pool di connessioni gestito da [DbManager](#) con l'istruzione [DbManager.getDataSource\(\)](#).

Questa connessione viene utilizzata per eseguire le query specifiche per ciascun metodo, garantendo l'accesso sicuro e ottimale al database.

La classe [RemoteDatabaseService](#) preleva la query corrispondente alla funzionalità che deve soddisfare dalla classe [PredefinedQuery](#), prelevandole dall'[Hashtable](#) opportuno.

Questo approccio centralizzato semplifica la manutenzione delle query e la gestione di eventuali modifiche.

Dopo l'esecuzione delle query, la classe elabora il [ResultSet](#) ottenuto in modelli dati specifici come [MonitoringCenter](#), [User](#), [Survey](#), [SurveysAggregate](#), e [PointOfInterest](#).

Questi modelli a differenza del [ResultSet](#) implementano l'interfaccia [Serializable](#) ed è quindi possibile ritornarli in remoto, inoltre permettono una più facile comprensione di cosa contiene il risultato per il lato client.

L'implementazione modulare dei metodi RMI favorisce l'espandibilità del sistema, consentendo l'aggiunta di nuove funzionalità senza impattare sulle funzioni esistenti.

All'interno della classe, oltre all'implementazione dei metodi di [RemoteDatabaseServiceInterface](#), sono presenti alcuni metodi privati utilizzati principalmente allo scopo di estrarre le informazioni necessarie dai [ResultSet](#) ed incapsularli in apposite istanze dei modelli dati.

Le funzionalità che potrebbero creare problemi di concorrenza vengono eseguite come transazione.

In tutti i metodi che svolgono operazioni sensibili per la concorrenza viene utilizzata la seguente logica :

- La transazione è avviata con [connection.setAutoCommit\(false\)](#), indicando che una sequenza di operazioni costituirà un'unità transazionale.
- Il metodo [executeUpdate](#) viene utilizzato per eseguire l'operazione effettiva nel database. Se l'operazione ha successo la transazione viene confermata con [connection.commit\(\)](#). In caso di fallimento, viene eseguito il rollback con [connection.rollback\(\)](#).
- Infine in entrambi i casi viene eseguita [connection.setAutoCommit\(true\)](#) per ripristinare il comportamento di commit automatico, garantendo la coerenza delle future operazioni.

La gestione delle eccezioni assicura che qualsiasi errore durante la transazione venga gestito correttamente, con rollback del database in caso di fallimento.

In sintesi, l'utilizzo di transazioni insieme alla gestione della connection pool garantisce la coerenza e l'integrità delle operazioni eseguite da [RemoteDatabaseService](#), mitigando i potenziali problemi di concorrenza che possono derivare dalle operazioni sensibili sul database.

Un'ulteriore protezione per la concorrenza che si è deciso di non implementare perchè si ritiene esagerata è la dichiarazione dei metodi sensibili come metodi [synchronized](#).

Questa ulteriore protezione verrà valutata in fase di sviluppo attivo dell'applicazione Climate Monitoring.

## Modulo shared

Il package shared è fondamentale per l'integrazione tra i moduli [clientCM](#) e [serverCM](#) di [ClimateMonitoring](#).

Esso contiene le classi e le interfacce condivise da entrambi i moduli, facilitando la comunicazione e la condivisione di dati essenziali.

Le principali componenti di questo package sono l'interfaccia [RemoteDatabaseServiceInterface](#) e il package models.

Il package models contiene le classi che modellano i dati trattati dall'applicazione Climate Monitoring.

## RemoteDatabaseServiceInterface

L'interfaccia [RemoteDatabaseServiceInterface](#) rappresenta il contratto di comunicazione RMI tra il modulo client e il modulo server.

Essa definisce tutti i metodi remoti necessari per l'interazione con il database, permettendo al lato client di inviare richieste al lato server e ricevere le risposte.

Viene implementata nel lato server dalla classe [RemoteDatabaseService](#).

I metodi che definisce costituiscono il cuore dell'interazione tra un client e il server.

Il contratto che definisce promette l'implementazione lato server di questi metodi :

- [cercaAreaGeograficaNome\(String name, String country\)](#)
  - **Descrizione:** Cerca e restituisce un elenco di aree geografiche basate sul nome e sul paese specificati.
  - **Parametri:** [name](#) - Nome dell'area, [country](#) - Paese dell'area.
  - **Risultato:** [ArrayList](#) di oggetti [PointOfInterest](#).
- [cercaAreaGeograficaCoordinate\(float latitude, float longitude\)](#)



- **Descrizione:** Cerca e restituisce un elenco di aree geografiche basate sulle coordinate specificate.
- **Parametri:** [latitude](#) - Latitudine dell'area, [longitude](#) - Longitudine dell'area.
- **Risultato:** [ArrayList](#) di oggetti [PointOfInterest](#).
- [visualizzaAreaGeografica\(int poi\\_id\)](#)
  - **Descrizione:** Restituisce un oggetto [SurveysAggregate](#) contenente informazioni aggregate per un'area geografica specifica.
  - **Parametri:** [poi\\_id](#) - ID dell'area geografica.
  - **Risultato:** Oggetto [SurveysAggregate](#).
- [registrazione\(User user, int centerid\)](#)
  - **Descrizione:** Registra un nuovo utente associandolo a un centro di monitoraggio specifico.
  - **Parametri:** [user](#) - Oggetto [User](#) da registrare, [centerid](#) - ID del centro di monitoraggio.
  - **Risultato:** Booleano che indica il successo dell'operazione.
- [registraCentroAree\(MonitoringCenter center\)](#)
  - **Descrizione:** Registra un nuovo centro di monitoraggio.
  - **Parametri:** [center](#) - Oggetto [MonitoringCenter](#) da registrare.
  - **Risultato:** ID assegnato al centro di monitoraggio registrato.
- [inserisciParametriClimatici\(Survey survey\)](#)
  - **Descrizione:** Inserisce i parametri climatici di un'area geografica nel database.
  - **Parametri:** [survey](#) - Oggetto [Survey](#) contenente i parametri climatici.
  - **Risultato:** ID assegnato al sondaggio inserito.
- [userLogin\(String userid, String hashedpassword\)](#)
  - **Descrizione:** Esegue il login di un utente verificando le credenziali.
  - **Parametri:** [userid](#) - ID dell'utente, [hashedpassword](#) - Password hashed dell'utente.
  - **Risultato:** Oggetto [User](#) rappresentante l'utente loggato o [null](#) in caso di fallimento.
- [selectSurveysById\(int poi\\_id\)](#)
  - **Descrizione:** Recupera le rilevazioni climatiche per un'area geografica specifica.
  - **Parametri:** [poi\\_id](#) - ID dell'area geografica.
  - **Risultato:** [ArrayList](#) di oggetti [Survey](#).
- [selectPoisByCenter\(int centerid\)](#)
  - **Descrizione:** Recupera le aree geografiche associate a un centro di monitoraggio.
  - **Parametri:** [centerid](#) - ID del centro di monitoraggio.
  - **Risultato:** [ArrayList](#) di oggetti [PointOfInterest](#).
- [selectCenterById\(int centerid\)](#)
  - **Descrizione:** Recupera le informazioni del centro di monitoraggio in base all'ID.
  - **Parametri:** [centerid](#) - ID del centro di monitoraggio.
  - **Risultato:** Oggetto [MonitoringCenter](#).
- [insertPoi\(PointOfInterest poi\)](#)
  - **Descrizione:** Inserisce una nuova area geografica nel database.
  - **Parametri:** [poi](#) - Oggetto [PointOfInterest](#) da inserire.



- **Risultato:** ID assegnato all'area geografica inserita.
- **getCentersList()**
  - **Descrizione:** Recupera la lista dei centri di monitoraggio registrati.
  - **Risultato:** `ArrayList` di oggetti `MonitoringCenter`.
- **checkUserExists(String userid)**
  - **Descrizione:** Verifica se un utente con l'ID specificato esiste nel database.
  - **Parametri:** `userid` - ID dell'utente.
  - **Risultato:** Booleano che indica l'esistenza dell'utente.
- **checkCenterExists(MonitoringCenter center)**
  - **Descrizione:** Verifica se un centro di monitoraggio con le informazioni specificate esiste nel database.
  - **Parametri:** `center` - Oggetto `MonitoringCenter` da verificare.
  - **Risultato:** ID del centro di monitoraggio se esiste, altrimenti -1.
- **checkPoiExists(PointOfInterest poi)**
  - **Descrizione:** Verifica se un'area geografica con le informazioni specificate esiste nel database.
  - **Parametri:** `poi` - Oggetto `PointOfInterest` da verificare.
  - **Risultato:** ID dell'area geografica se esiste, altrimenti -1.
- **linkCenterToPois(MonitoringCenter center, ArrayList<PointOfInterest> poiList)**
  - **Descrizione:** Collega un centro di monitoraggio a una lista di aree geografiche.
  - **Parametri:** `center` - Oggetto `MonitoringCenter` da collegare, `poiList` - Lista di oggetti `PointOfInterest` da collegare.
- **linkPoiToCenter(PointOfInterest poi, int centerid)**
  - **Descrizione:** Collega un'area geografica a un centro di monitoraggio specifico.
  - **Parametri:** `poi` - Oggetto `PointOfInterest` da collegare, `centerid` - ID del centro di monitoraggio.

## Package models

Il package models contiene le classi che modellano i dati fondamentali trattati dall'applicazione Climate Monitoring.

Questi modelli rappresentano entità chiave nel contesto di un sistema di monitoraggio climatico.

Raggruppando i modelli in un package dedicato, si promuove l'incapsulamento e l'organizzazione del codice.

I modelli sono progettati per essere condivisi tra i moduli clientCM e serverCM (da qui il motivo per cui sono contenuti nel modulo shared).

La presenza di un package models nel modulo shared garantisce che entrambi i moduli possano accedere alle stesse definizioni di oggetti.

I modelli di questo pacchetto sono definiti dalle classi :

- PointOfInterest (Area Geografica)
- User (Operatore)
- MonitoringCenter (Centro di Monitoraggio)
- Survey (Parametri Climatici)
- SurveyAggregate (Aggregato di Parametri Climatici)

## PointOfInterest

La classe [PointOfInterest](#) rappresenta un'Area Geografica monitorata da Centri di Monitoraggio Climatico dell'applicazione Climate Monitoring.

La classe implementa l'interfaccia Serializable, consentendo la serializzazione degli oggetti [PointOfInterest](#).

Ciò è utile per la trasmissione di oggetti attraverso la rete, come nel caso dell'architettura RMI (Remote Method Invocation) utilizzata nell'applicazione.

Attributi :

- [poi\\_id](#): Identificatore univoco del punto di interesse, è la chiave primaria nella tabella delle aree geografiche nel database.
- [name](#): Nome dell'Area Geografica.
- [country](#): Paese in cui si trova.
- [latitude](#): Latitudine della posizione del punto di interesse in gradi decimali.
- [longitude](#): Longitudine della posizione del punto di interesse in gradi decimali.

La classe mette a disposizione tre costruttori :

- Costruttore vuoto (mai utilizzato nell'applicazione)
- Costruttore con [poi\\_id](#) fornito
- Costruttore senza [poi\\_id](#) fornito

I due costruttori differenti servono per permettere di creare [PointOfInterest](#) che non sono ancora stati inseriti nel database e che quindi non hanno ancora ricevuto un id univoco.

Ogni attributo principale è associato a un metodo getter e setter che consente di accedere e modificare i valori degli attributi.

Sono anche stati riscritti metodi [equals](#) e [toString](#) ad hoc, mai utilizzati ma utili per il debugging.

## User

La classe User è progettata per rappresentare un operatore registrato nell'applicazione Climate Monitoring.

Gli operatori in fase di registrazione devono associarsi a un Centro di Monitoraggio Climatico, e la classe gestisce informazioni come nome, cognome, email, identificatore unico, codice fiscale, password hashata, e l'ID del centro al quale l'utente è associato.

La classe implementa l'interfaccia `Serializable`, consentendo la serializzazione degli oggetti `User`.

Attributi :

- `userid`: Identificatore unico dell'utente, funge da chiave primaria nella tabella degli operatori registrati nel database.
- `name`: Nome dell'utente.
- `surname`: Cognome dell'utente.
- `email`: Indirizzo email associato all'utente.
- `fiscalCode`: Codice fiscale, identificativo unico per scopi fiscali, associato all'utente.
- `hashedPassword`: Hash della password dell'utente (SHA-256).
- `centerid`: Identificatore del Centro di Monitoraggio Climatico a cui l'utente è associato.

La classe mette a disposizione due costruttori :

- Costruttore vuoto (mai utilizzato nell'applicazione)
- Costruttore standard

Ogni attributo principale è associato a un metodo getter e setter che consente di accedere e modificare i valori degli attributi.

Anche qui sono stati riscritti i metodi `equals` e `toString` ad hoc, mai utilizzati ma utili per il debugging.

La classe contiene anche un metodo mai utilizzato `checkPassword` (che utilizza il metodo privato `bytesToHex` ) che verifica se la password fornita corrisponde alla password hashata memorizzata.

Il metodo non è stato rimosso per possibili futuri utilizzi.

## Monitoring Center

La classe `MonitoringCenter` è progettata per rappresentare un Centro di Monitoraggio Climatico nell'applicazione Climate Monitoring.

Questo centro è associato a un identificatore unico (`centerid`) che gli viene fornito in automatico dal database al momento della registrazione e quindi dell'inserimento e include informazioni come il nome del centro, indirizzo, numero civico, CAP, città e provincia.

Come gli altri modelli la classe implementa l'interfaccia [Serializable](#), consentendo la serializzazione degli oggetti [MonitoringCenter](#) per supportare l'architettura RMI dell'applicazione.

Attributi :

- [centerid](#): Identificatore unico del Centro di Monitoraggio Climatico.
- [name](#): Nome del Centro di Monitoraggio Climatico.
- [address](#): Indirizzo del Centro di Monitoraggio Climatico.
- [addressNumber](#): Numero civico del Centro di Monitoraggio Climatico.
- [cap](#): CAP (Codice di Avviamento Postale) dell'indirizzo del Centro di Monitoraggio Climatico.
- [city](#): Città in cui si trova il Centro di Monitoraggio Climatico.
- [province](#): Provincia in cui si trova il Centro di Monitoraggio Climatico.

La classe mette a disposizione tre costruttori :

- Costruttore vuoto (mai utilizzato nell'applicazione)
- Costruttore con [centerid](#) fornito
- Costruttore senza [centerid](#) fornito

I due costruttori differenti servono per permettere di creare [MonitoringCenter](#) che non sono ancora stati inseriti nel database e che quindi non hanno ancora ricevuto un id univoco.

Anche per questa classe ogni attributo principale è associato a un metodo getter e setter che consente di accedere e modificare i valori degli attributi, inoltre sono sempre stati riscritti i metodi [equals](#) e [toString](#) ad hoc, mai utilizzati ma utili per il debugging.

## Survey

La classe [Survey](#) è progettata per modellare un set di parametri climatici inseriti dagli utenti ([User](#)) di un centro climatico ([MonitoringCenter](#)) per un'area geografica ([PointOfInterest](#)) specifico monitorato dal centro dell'utente.

Questo centro è associato a un identificatore unico ([dataid](#)) che gli viene fornito in automatico dal database al momento dell'inserimento.

La classe implementa l'interfaccia [Serializable](#) per supportare la serializzazione degli oggetti [Survey](#).

Attributi :

- [dataid](#): Identificatore unico per i dati del sondaggio.
- [poi\\_id](#): Identificatore unico per il punto di interesse associato al sondaggio.
- [centerid](#): Identificatore unico per il centro di monitoraggio associato al sondaggio.
- [timestamp](#): Timestamp quando i dati del sondaggio sono stati registrati.

- Dati climatici registrati: [wind](#), [humidity](#), [pressure](#), [temperature](#), [precipitation](#), [glacial\\_altitude](#), [glacial\\_mass](#). (sotto forma di score, ovvero int da 1 a 5)
- Note aggiuntive per ciascun tipo di dato climatico: [wind\\_notes](#), [humidity\\_notes](#), [pressure\\_notes](#), [temperature\\_notes](#), [precipitation\\_notes](#), [glacial\\_altitude\\_notes](#), [glacial\\_mass\\_notes](#). (rappresentano i commenti facoltativi che l'operatore può fornire)

La classe mette a disposizione tre costruttori :

- Costruttore vuoto (mai utilizzato nell'applicazione)
- Costruttore con [dataid](#) fornito
- Costruttore senza [dataid](#) fornito

Anche qui i due costruttori differenti servono per permettere di creare [Survey](#) che non sono ancora stati inseriti nel database e che quindi non hanno ancora ricevuto un id univoco.

Sono forniti getters e setters e metodi [equals](#) e [toString](#) come per gli altri modelli.

## SurveyAggregate

La classe [SurveysAggregate](#) rappresenta le informazioni aggregate su una o più set di parametri climatici ([Survey](#)) presenti per un'area geografica specifica nell'applicazione Climate Monitoring.

La classe implementa l'interfaccia [Serializable](#), consentendo la serializzazione degli oggetti [SurveyAggregate](#), così da rendere possibile la loro trasmissione tra lato client e lato server.

Attributi :

- [poi\\_id](#): L'ID del punto di interesse associato a questo [SurveysAggregate](#).
- [surveysCount](#): Il conteggio totale dei set di parametri climatici inseriti per la specifica area geografica (ovvero corrispondente al [poi\\_id](#)).
- [centersCount](#): Il conteggio dei centri unici che hanno contribuito a inserire i parametri climatici per la specifica area geografica (ovvero corrispondente al [poi\\_id](#)).
- [oldestTimestamp](#): Il timestamp della più vecchia rilevazione tra tutte quelle presenti per questa specifica area geografica (ovvero corrispondente al [poi\\_id](#)).
- [mostRecentTimestamp](#): Il timestamp della più recente rilevazione tra tutte quelle presenti per questa specifica area geografica (ovvero corrispondente al [poi\\_id](#)).
- [wind\\_avg](#), [humidity\\_avg](#), [pressure\\_avg](#), [temperature\\_avg](#), [precipitation\\_avg](#), [glacial\\_altitude\\_avg](#), [glacial\\_mass\\_avg](#): Valori medi tra i vari parametri climatici inseriti.
- [wind\\_max](#), [humidity\\_max](#), [pressure\\_max](#), [temperature\\_max](#), [precipitation\\_max](#), [glacial\\_altitude\\_max](#), [glacial\\_mass\\_max](#): Valori massimi tra i vari parametri climatici inseriti.

- [wind\\_min](#), [humidity\\_min](#), [pressure\\_min](#), [temperature\\_min](#), [precipitation\\_min](#), [glacial\\_altitude\\_min](#), [glacial\\_mass\\_min](#): Valori minimi tra i vari parametri climatici inseriti.
- [wind\\_notes\\_list](#), [humidity\\_notes\\_list](#), [pressure\\_notes\\_list](#), [temperature\\_notes\\_list](#), [precipitation\\_notes\\_list](#), [glacial\\_altitude\\_notes\\_list](#), [glacial\\_mass\\_notes\\_list](#): Liste dei commenti per diverse osservazioni climatiche.

La classe mette a disposizione due costruttori :

- Costruttore vuoto (mai utilizzato nell'applicazione)
- Costruttore standard

Questo modello è utilizzato nell'unica funzionalità che permette di visualizzare i dati inseriti dagli operatori per un'area geografica in modo aggregato.

In particolare il metodo [visualizzaAreaGeografica\(\)](#) del [RemoteDatabaseService](#) crea istanze di questa classe quando un client richiede di visualizzare i dati aggregati di un'area geografica (fornendo come parametro il `poi_id` che la identifica).

Una volta creato lo inizializza con i dati estratti dal [ResultSet](#) della query che permette di raccogliarli dal database e invia questa istanza al client.

## Struttura Database

### Analisi e considerazioni in linguaggio naturale

Il database dell'applicazione Climate Monitoring è stato pensato per soddisfare i requisiti per un sistema di monitoraggio di parametri climatici fornito da centri di monitoraggio sul territorio italiano, in grado di rendere disponibili, ad operatori e "guest users" diverse funzionalità.

In particolare il database è stato pensato per memorizzare i dati necessari e mantenere i vincoli sui dati necessari per rendere possibili le seguenti operazioni :

- Visualizzazione dei dati in delle rilevazioni su un'area geografica in forma aggregata (possibile sia operatori che "guest users").
- Ricerca di un'area geografica per coordinate o per denominazione (possibile sia operatori che "guest users").
- Registrazione di un nuovo operatore a Climate Monitoring.
- Registrazione di un nuovo centro a Climate Monitoring.
- Inserimento di rilevazioni di parametri climatici.

Per quanto riguarda il carico di lavoro si ipotizza un rilascio regione per regione dell'applicazione Climate Monitoring sul territorio italiano e uno sviluppo attivo della stessa a seconda del feedback ricevuto.

La prima versione è pensata con l'idea di dover gestire un numero di centri nell'ordine delle migliaia, un numero di operatori nelle migliaia e massimo di 10000 e un numero di utenti "guest" più elevato.

E' quindi necessario memorizzare le seguenti informazioni nel database :

- Aree Geografiche (Point Of Interest)
- Operatori Registrati (User)
- Centri di Monitoraggio (Monitoring Center)
- Rilevazioni dei parametri climatici (Survey)

Queste sono quindi le Entità nello schema concettuale del database.

Per le Aree Geografiche è necessario memorizzate all'interno del database almeno le seguenti informazioni :

- Nome : denominazione dell'area
- Latitudine : latitudine in formato decimale
- Longitudine : longitudine in formato decimale
- Stato : lo stato in cui è situata

Per gli Operatori Registrati è necessario memorizzate all'interno del database almeno le seguenti informazioni :

- Userid : un identificativo univoco dell'operatore
- Nome : nome dell'operatore
- Cognome : cognome dell'operatore
- Email : email dell'operatore
- Codice Fiscale : codice fiscale dell'operatore
- Password : password dell'operatore
- Centro di Monitoraggio di appartenenza : l'id univoco del centro di appartenenza

Per i Centri di Monitoraggio è necessario memorizzate all'interno del database almeno le seguenti informazioni :

- Nome Centro Monitoraggio : il nome del centro
- Indirizzo fisico : via, numero civico, cap, comune e provincia
- Elenco aree di interesse : elenco delle aree che il centro monitora

Per le rilevazioni dei parametri climatici è necessario memorizzate all'interno del database almeno le seguenti informazioni :

- Area di appartenenza : area geografica a cui la rilevazione fa riferimento
- Centro o Operatore di appartenenza : il centro o l'operatore che ha inserito la rilevazione

- Scores (vento, umidità, pressione, temperatura, precipitazioni, altitudine ghiacciai, massa ghiacciai)
- Commenti Opzionali (vento, umidità, pressione, temperatura, precipitazioni, altitudine ghiacciai, massa ghiacciai)

Ogni operatore ha un unico centro di monitoraggio di afferenza e per ogni centro ci sono da 1 a più operatori che gli afferiscono.

Ogni rilevazione di parametri climatici è inserita da un operatore che appartiene di sicuro ad un centro, in questo caso la rilevazione salva solo il centro da cui è stata inserita e non l'operatore (non si è ritenuto necessario salvare anche l'operatore che l'ha inserita).

Per ogni rilevazione viene anche salvato il timestamp dell'inserimento.

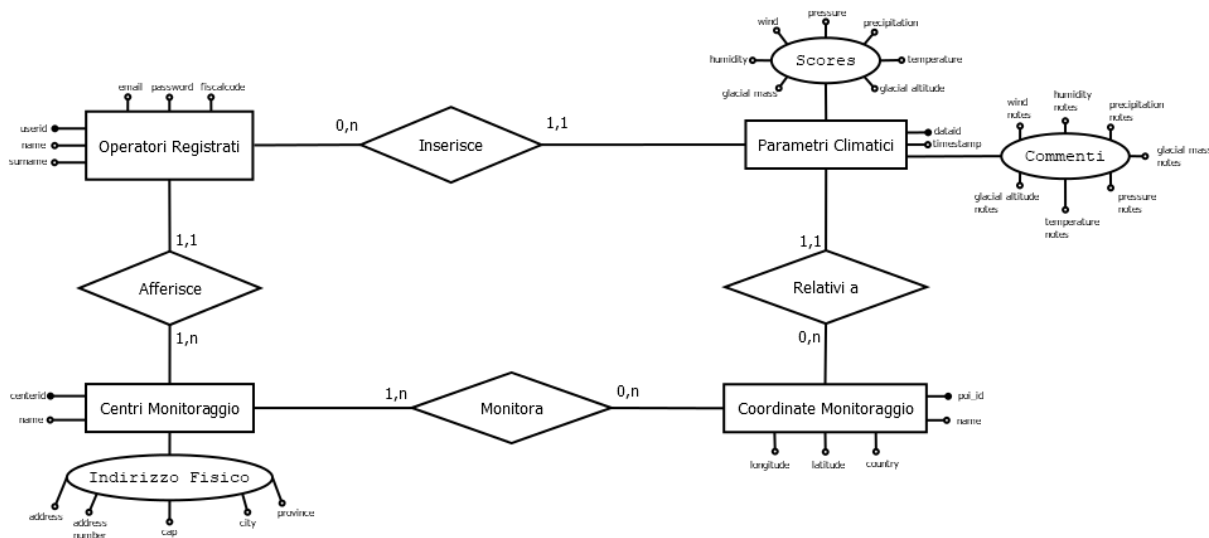
Una importante considerazione è che ogni Area Geografica può essere monitorata da più di un centro e quindi è necessario memorizzare queste relazioni in una apposita tabella.

Per ognuno dei dati da memorizzare (eccetto gli operatori registrati) si è deciso di utilizzare come identificativo univoco un int autogenerated sfruttando la funzionalità SERIAL di PostgreSQL.

Per gli operatori registrati la chiave primaria è l'userid inserito dall'operatore in fase di registrazione e utilizzato per il login con la password.

## Progettazione Concettuale

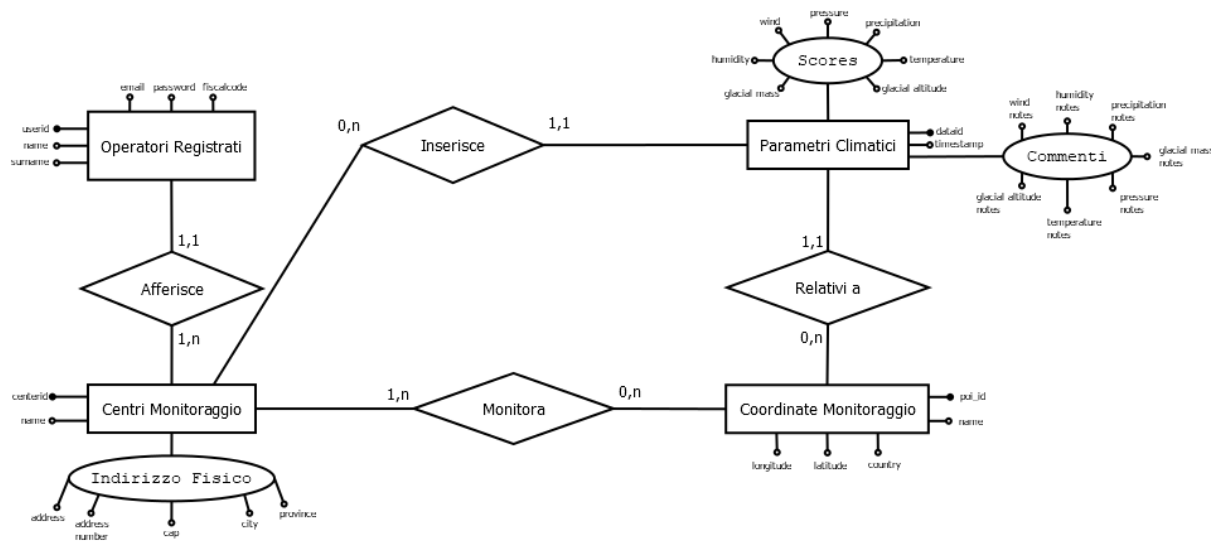
Lo schema concettuale di partenza è quindi il seguente.



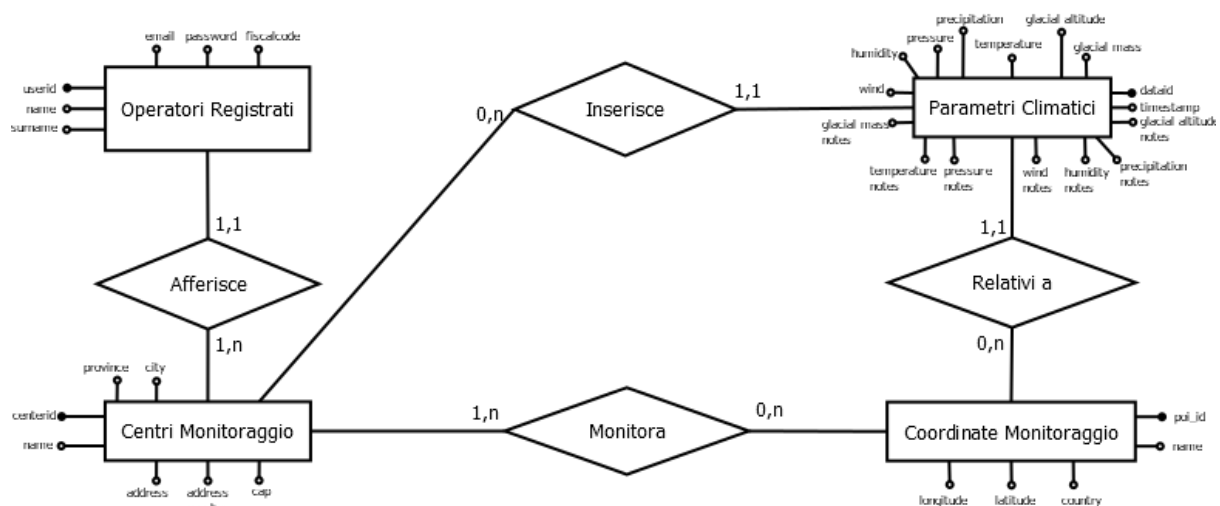
Inoltre come specificato nell'analisi precedente si è deciso di non tenere traccia di quale operatore inserisce una rilevazione di parametri climatici ma semplicemente a quale centro di monitoraggio afferisce l'operatore stesso.

Quindi si è modificato lo schema concettuale come segue.





La ristrutturazione di questo schema concettuale porta poi al seguente risultato che viene utilizzato per la traduzione in schema logico nella fase successiva.



## Progettazione Logica

Gli Operatori Registrati (User) sono salvati in una tabella così strutturata :

```
OperatoriRegistrati(
    userid, varchar(30)
    centeridcentrimonitoraggio , integer
    name, varchar(30)
    surname, varchar(30)
    email, varchar(80)
    fiscalcode, varchar(16)
    hashedpassword, varchar(64)
```

)

**Nota** : considerando che la password non viene salvata in chiaro ma viene salvato l'hash il nome dell'attributo è cambiato in "hashedpassword".

I Centri di Monitoraggio (MonitoringCenter) sono salvati in una tabella così strutturata :

```
CentriMonitoraggio(  
    centerid, SERIAL  
    name, varchar(50)  
    address, varchar(50)  
    addressnumber, integer  
    cap, integer  
    city, varchar(30)  
    province, varchar(30)  
)
```

**Nota** : In PostgreSQL, il tipo di dato SERIAL viene comunemente utilizzato per creare colonne di tipo intero auto-incrementanti. Questo è particolarmente utile per definire colonne chiave primarie in cui si desidera che il database assegni automaticamente un identificatore unico a ogni nuova riga in una tabella.

Le Aree Geografiche e le loro coordinate (PointOfInterest) sono salvati in una tabella così strutturata :

```
CoordinateMonitoraggio(  
    poi_id, SERIAL  
    name, varchar(100)  
    country, varchar(100)  
    latitude, numeric(8,5)  
    longitude, numeric(8,5)  
)
```

Le Rilevazioni dei parametri climatici (Survey) sono salvati in una tabella così strutturata:

```
ParametriClimatici(  
    dataid, SERIAL  
    poi_idcoordinatemonitoraggio, integer  
    center_idcentrimonitoraggio, integer  
    timestamp, TIMESTAMP (without time zone)  
    wind, integer  
    humidity, integer  
    pressure, integer  
    temperature, integer  
    precipitation, integer  
    glacial_altitude, integer  
    glacial_mass, integer  
    wind_notes, varchar(256)  
    humidity_notes, varchar(256)  
    pressure_notes, varchar(256)
```

```

        temperature_notes, varchar(256)
        precipitation_notes, varchar(256)
        glacial_altitude_notes, varchar(256)
        glacial_mass_notes, varchar(256)
    )

```

La traduzione della relazione “Monitora” tra i Centri di Monitoraggio e le Coordinate è rappresentata da questo schema logico

```

Coordinate_Centri(
    centeridcentrimonitoraggio, integer
    poi_idcoordinatemonitoraggio, integer
)

```

## Vincoli d'integrità

Si sono ritenuti necessari questi ulteriori vincoli:

- Tutti i dati eccetto i commenti per le rilevazioni dei parametri climatici devono essere NOT NULL
- La email deve essere unica.
- Un codice fiscale non può essere di più di 16 caratteri.
- Una password hashata con SHA-256 sarà sempre di 64 caratteri di cifre esadecimali (0-9 , a-f).
- Un cap è sempre di 5 cifre.
- Gli score per i parametri climatici sono sempre compresi tra 1 e 5.
- Se un centro o un'area geografica sono rimossi dalle corrispettive tabelle anche tutte le associazioni relative nella tabella Coordinate\_Centri vengono rimosse (si ricorda però che nella presente versione di Climate Monitoring non esiste la funzione che permette di cancellare un centro registrato o un'area inserita, dato che non è stato specificato nelle specifiche di progetto)

## Progettazione Pratica

Di seguito si fornisce tutto il codice SQL utilizzato dall'applicazione per creare ed interagire con il database.

### Creazione Database

```
CREATE DATABASE dbcm;
```

## Creazione Tabelle

```
CREATE TABLE operatoriregistrati(  
    userid VARCHAR(30) PRIMARY KEY NOT NULL,  
    name VARCHAR(30) NOT NULL,  
    surname VARCHAR(30) NOT NULL,  
    email VARCHAR(80) UNIQUE NOT NULL,  
    fiscalcode VARCHAR(16) NOT NULL CHECK (LENGTH(fiscalcode) = 16),  
    hashedpassword VARCHAR(64) NOT NULL CHECK (LENGTH(hashedpassword) =  
64),  
    centerid INTEGER NOT NULL REFERENCES CentriMonitoraggio(centerid)  
);
```

```
CREATE TABLE centrimonitoraggio(  
    centerid SERIAL PRIMARY KEY,  
    name VARCHAR(50) NOT NULL,  
    address VARCHAR(50) NOT NULL,  
    addressnumber INTEGER NOT NULL,  
    cap INTEGER NOT NULL CHECK (LENGTH(cap::TEXT) = 5),  
    city VARCHAR(30) NOT NULL,  
    province VARCHAR(30) NOT NULL  
);
```

```
CREATE TABLE coordinatemonitoraggio(  
    poi_id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    country VARCHAR(100) NOT NULL,  
    latitude NUMERIC(8,5) NOT NULL,  
    longitude NUMERIC(8,5) NOT NULL  
);
```

```
CREATE TABLE parametriclimatici(  
    dataid SERIAL PRIMARY KEY,  
    poi_id INTEGER NOT NULL REFERENCES CoordinateMonitoraggio(poi_id),  
    centerid INTEGER NOT NULL REFERENCES CentriMonitoraggio(centerid),  
    timestamp TIMESTAMP WITHOUT TIME ZONE NOT NULL,  
    wind INTEGER NOT NULL CHECK (wind IN (1,2,3,4,5)),  
    humidity INTEGER NOT NULL CHECK (humidity IN (1,2,3,4,5)),  
    pressure INTEGER NOT NULL CHECK (pressure IN (1,2,3,4,5)),  
    temperature INTEGER NOT NULL CHECK (temperature IN (1,2,3,4,5)),  
    precipitation INTEGER NOT NULL CHECK (precipitation IN (1,2,3,4,5)),  
    glacial_altitude INTEGER NOT NULL CHECK (glacial_altitude IN (1,2,3,4,5)),  
    glacial_mass INTEGER NOT NULL CHECK (glacial_mass IN (1,2,3,4,5)),  
    wind_notes VARCHAR(256),  
    humidity_notes VARCHAR(256),  
    pressure_notes VARCHAR(256),  
    temperature_notes VARCHAR(256),
```

```

        precipitation_notes VARCHAR(256),
        glacial_altitude_notes VARCHAR(256),
        glacial_mass_notes VARCHAR(256)
    );

CREATE TABLE coordinate_centri (
    centerid INTEGER REFERENCES CentriMonitoraggio(centerid) ON DELETE
    CASCADE,
    poi_id INTEGER REFERENCES CoordinateMonitoraggio(poi_id) ON DELETE
    CASCADE,
    PRIMARY KEY (centerid, poi_id)
);

```

## SELECT Queries

### User (Operatore Registrato)

```

SELECT *
FROM operatoriregistrati
WHERE userid = ? ;

```

### User Login Info

```

SELECT *
FROM operatoriregistrati
WHERE userid = ? AND hashedpassword = ?;

```

### User existence

```

SELECT EXISTS
    (SELECT 1 FROM operatoriregistrati WHERE userid = ?);

```

### Email existence

```

SELECT EXISTS
    (SELECT 1 FROM operatoriregistrati WHERE email = ?);

```

### Monitoring Centers

```

SELECT *
FROM centrimonitoraggio;

```

### Monitoring Center da ID

```
SELECT *  
FROM centrimonitoraggio  
WHERE centerid = ?;
```

#### Monitoring Center da Dati

```
SELECT *  
FROM centrimonitoraggio  
WHERE name = ? AND address = ? AND addressnumber = ? AND cap = ? AND city = ?  
AND province = ?;
```

#### Point Of Interest (Area Geografica)

```
SELECT *  
FROM coordinatemonitoraggio
```

#### Point Of Interest da dati

```
SELECT *  
FROM coordinatemonitoraggio  
WHERE name = ? AND country = ? AND latitude = ? AND longitude = ?;
```

#### Point Of Interest da denominazione

```
SELECT *  
FROM coordinatemonitoraggio  
WHERE name ILIKE lower(?) AND country ILIKE lower(?);
```

#### Point Of Interest da coordinate

```
SELECT *  
FROM coordinatemonitoraggio  
WHERE latitude > ? AND latitude < ? AND longitude > ? AND longitude < ?;
```

#### Survey Aggregate

```
SELECT  
    poi_id,  
    COUNT(*) AS survey_count,  
    COUNT(DISTINCT centerid) AS number_of_centers,  
    MIN(DATE_TRUNC('second', timestamp)) AS oldest_timestamp,  
    MAX(DATE_TRUNC('second', timestamp)) AS most_recent_timestamp,  
    ROUND(AVG(wind), 2) AS avg_wind,
```

```

ROUND(AVG(humidity), 2) AS avg_humidity,
ROUND(AVG(pressure), 2) AS avg_pressure,
ROUND(AVG(temperature), 2) AS avg_temperature,
ROUND(AVG(precipitation), 2) AS avg_precipitation,
ROUND(AVG(glacial_altitude), 2) AS avg_glacial_altitude,
ROUND(AVG(glacial_mass), 2) AS avg_glacial_mass,
MAX(wind) AS max_wind,
MAX(humidity) AS max_humidity,
MAX(pressure) AS max_pressure,
MAX(temperature) AS max_temperature,
MAX(precipitation) AS max_precipitation,
MAX(glacial_altitude) AS max_glacial_altitude,
MAX(glacial_mass) AS max_glacial_mass,
MIN(wind) AS min_wind,
MIN(humidity) AS min_humidity,
MIN(pressure) AS min_pressure,
MIN(temperature) AS min_temperature,
MIN(precipitation) AS min_precipitation,
MIN(glacial_altitude) AS min_glacial_altitude,
MIN(glacial_mass) AS min_glacial_mass,
STRING_AGG(CASE WHEN wind_notes IS NOT NULL AND wind_notes <> " THEN
wind_notes ELSE NULL END, '|' ) AS wind_notes_list,
STRING_AGG(CASE WHEN humidity_notes IS NOT NULL AND humidity_notes <> " THEN
humidity_notes ELSE NULL END, '|') AS humidity_notes_list,
STRING_AGG(CASE WHEN pressure_notes IS NOT NULL AND pressure_notes <> " THEN
pressure_notes ELSE NULL END, '|') AS pressure_notes_list,
STRING_AGG(CASE WHEN temperature_notes IS NOT NULL AND temperature_notes <> "
THEN temperature_notes ELSE NULL END, '|') AS temperature_notes_list,
STRING_AGG(CASE WHEN precipitation_notes IS NOT NULL AND precipitation_notes <> "
THEN precipitation_notes ELSE NULL END, '|') AS precipitation_notes_list,
STRING_AGG(CASE WHEN glacial_altitude_notes IS NOT NULL AND
glacial_altitude_notes <> " THEN glacial_altitude_notes ELSE NULL END, '|') AS
glacial_altitude_notes_list,
STRING_AGG(CASE WHEN glacial_mass_notes IS NOT NULL AND glacial_mass_notes <>
" THEN glacial_mass_notes ELSE NULL END, '|') AS glacial_mass_notes_list
FROM parametriclimatici
WHERE poi_id = ?
GROUP BY poi_id ; "

```

### Point Of Interest da ID Centro

```

SELECT *
FROM coordinatemonitoraggio JOIN coordinate_centri ON coordinatemonitoraggio.poi_id =
coordinate_centri.poi_id
WHERE coordinate_centri.centerid = ? ;

```

### Survey per ID Point Of Interest (Area Geografica)

```

SELECT *
FROM parametriclimatici

```

WHERE poi\_id = ? ;

## Database

```
SELECT datname
FROM pg_database
WHERE datname = ?;
```

## Table da database

```
SELECT table_name
FROM information_schema.tables
WHERE table_name = ?;
```

## INSERT Queries

### User (Operatore Registrato)

```
INSERT INTO operatoriregistrati (userid, name, surname, email, fiscalcode,
hashedpassword, centerid)
VALUES (?, ?, ?, ?, ?, ?, ?);
```

### Monitoring Center

```
INSERT INTO centrimonitoraggio (name, address, addressnumber, cap, city, province)
VALUES (?, ?, ?, ?, ?, ?);
```

### Point Of Interest

```
INSERT INTO coordinatemonitoraggio (name, country, latitude, longitude)
VALUES (?, ?, ?, ?);
```

### Survey

```
INSERT INTO parametriclimatici (poi_id, centerid, timestamp, wind, humidity, pressure,
temperature, precipitation, glacial_altitude, glacial_mass, wind_notes, humidity_notes,
pressure_notes, temperature_notes, precipitation_notes, glacial_altitude_notes,
glacial_mass_notes)
VALUES (?, ?, CURRENT_TIMESTAMP, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
RETURNING dataid;
```



Point Of Interest per Centro

```
INSERT INTO coordinate_centri (centerid, poi_id)
VALUES (?,?);
```

## Sitografia/Bibliografia

Apache Maven:

<https://maven.apache.org/index.html>  
<https://mvnrepository.com/>

Apache Commons DBCP 2:

<https://commons.apache.org/proper/commons-dbc/apidocs/index.html>  
<https://commons.apache.org/proper/commons-dbc/index.html>

JDBC Driver:

<https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>

Java RMI:

<https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html>  
[https://en.wikipedia.org/wiki/Java\\_remote\\_method\\_invocation](https://en.wikipedia.org/wiki/Java_remote_method_invocation)

GitHub Repository and bug reporting:

[https://github.com/Arcii/Climate\\_Monitoring](https://github.com/Arcii/Climate_Monitoring)  
[https://github.com/Arcii/Climate\\_Monitoring/issues](https://github.com/Arcii/Climate_Monitoring/issues)