|  |  |  |  |
| --- | --- | --- | --- |
| Brioschi Kilian | Frattini Gaia | Puglisi Mirko | Ramelli Riccardo |
| MT. 748918 | MT. 736610 | MT. 748198 | MT. 748711 |

Versione: 1.1

Data di rilascio: 05/01/2024

Climate Monitoring

MANUALE TECNICO



Sommario

[Sommario - 2 -](#_Toc158218854)

[Introduzione - 4 -](#_Toc158218855)

[Requisiti e problematiche di funzionamento - 5 -](#_Toc158218856)

[Librerie esterne - 5 -](#_Toc158218857)

[Dipendenze Maven - 5 -](#_Toc158218858)

[Progettazione - 6 -](#_Toc158218859)

[Use case diagram - 6 -](#_Toc158218860)

[Class diagram - 7 -](#_Toc158218861)

[Class Diagram del modulo Client - 7 -](#_Toc158218862)

[Class Diagram del modulo Server - 7 -](#_Toc158218863)

[Sequence diagram - 8 -](#_Toc158218864)

[Registrazione operatore - 8 -](#_Toc158218865)

[Inserimento parametri climatici - 9 -](#_Toc158218866)

[Visualizzazione parametri climatici - 10 -](#_Toc158218867)

[Struttura dell’applicazione - 11 -](#_Toc158218868)

[Moduli dell’applicazione - 11 -](#_Toc158218869)

[Client-Server - 13 -](#_Toc158218870)

[RMI (Remote Method Invocation) - 13 -](#_Toc158218871)

[DataStructures - 14 -](#_Toc158218872)

[ClientCM - 15 -](#_Toc158218873)

[ServerCM - 15 -](#_Toc158218874)

[Uso del pattern MVC (Model-View-Controller) - 15 -](#_Toc158218875)

[Uso del pattern Singleton - 15 -](#_Toc158218876)

[Database - 16 -](#_Toc158218877)

[Diagramma ER - 16 -](#_Toc158218878)

[Fase di traduzione - 17 -](#_Toc158218879)

[Schema logico - 17 -](#_Toc158218880)

[Interfaccia grafica - 19 -](#_Toc158218881)

[SceneController - 19 -](#_Toc158218882)

[Riferimenti - 20 -](#_Toc158218883)

# Introduzione

L’applicazione Climate Monitoring permette il monitoraggio dei parametri climatici in diverse aree del pianeta. Sfruttando i dati raccolti e inseriti da operatori affiliati ai centri di monitoraggio ubicati esclusivamente in territorio italiano.

L’applicazione permette agli operatori registrati al sistema l’inserimento di rilevazioni di una data area geografica.

I parametri registrabili sono:

* Velocità del vento
* Percentuale di umidità
* Pressione atmosferica
* Temperatura
* Precipitazioni
* Altitudine dei ghiacciai
* Massa dei ghiacciai

la cui criticità di ogni parametro è indicata in una scala da 1 a 5. dove 1 è “Critico” e 5 è “Ottimale”. Attualmente, l’applicazione permette di effettuare la rilevazione dei parametrici relativi a un’area geografica una sola volta al giorno per operatore, ciò comporta l’impossibilità di catturare un’ulteriore indagine da parte del medesimo operatore riguardante la stessa area.

Gli utenti, non necessitano di registrarsi al sistema, possono accedere ai dati registrati dagli operatori e visionare in forma aggregata i parametri climatici delle aree di interesse da loro ricercate.

Tale analisi dei dati avviene per mezzo di più tipi di ricerca cui l’utente può usufruire:

* Ricerca per nome
  + Ricerca per denominazione
  + Ricerca per denominazione e stato (lo stato agisce da filtro di ricerca)
* Ricerca per coordinate
  + Ricerca per coordinate, specificando un raggio di ricerca
  + Ricerca per coordinate senza limiti, trovando le prime corrispondenze più vicine

# Requisiti e problematiche di funzionamento

L’applicazione è stata sviluppata in Java, utilizzando la JDK versione 17, lo sviluppo dell’interfaccia grafica è stato reso possibile dall’utilizzo di JavaFX una libreria per la creazione di rich client applications con Java

Questa fornisce un'API per la progettazione di applicazioni GUI che vengono eseguite su quasi tutti i dispositivi con supporto Java.

L’applicazione è stata testata sia su sistemi Windows 11 sia su MAC OS 13.4 Ventura. Il corretto funzionamento su altri sistemi operativi non è garantito.

# Librerie esterne

* JavaFX11+: come già accennato, l’utilizzo di tale libreria è stato necessario per la costruzione dell’interfaccia grafica, tramite l’utilizzo di Scene Builder, un framework che permette di creare GUI con metodologia drag-n-drop.
* Java.sql: per la lettura e scrittura su database.

# Dipendenze Maven

* org.PostgreSQL
  + postgreSQL versione 42.7.0
* org.OpenJFx:
  + Javafx-control
  + Javafx-fxml

# Progettazione

Per la progettazione si è vista necessaria la costruzione di diversi tipi di diagrammi UML che descrivessero diversi livelli dell’applicazione.

## Use case diagram

Il diagramma dei casi d’uso è stato utilizzato per descrivere ad alto livello l’applicazione generale, andando a rappresentare l’intero sistema e le sue interazioni.

|  |
| --- |
| Immagine che contiene schermata, cerchio  Descrizione generata automaticamente |
| *Use Case Diagram* |

Osservando il diagramma è possibile notare che:

* L’utente è in grado solamente di ricercare le aree di interesse e leggerne i dati.
* L’operatore interagisce maggiormente con l’applicazione:
  + Registrandosi e creando eventualmente un nuovo centro di monitoraggio e aggiungendo le aree ad esso associate,
  + Effettuando il login,
  + Visualizzando le aree associate,
  + Inserendo i parametri climatici inerenti all’area ed eventualmente aggiungendo un commento relativo alla rilevazione.

## Class diagram

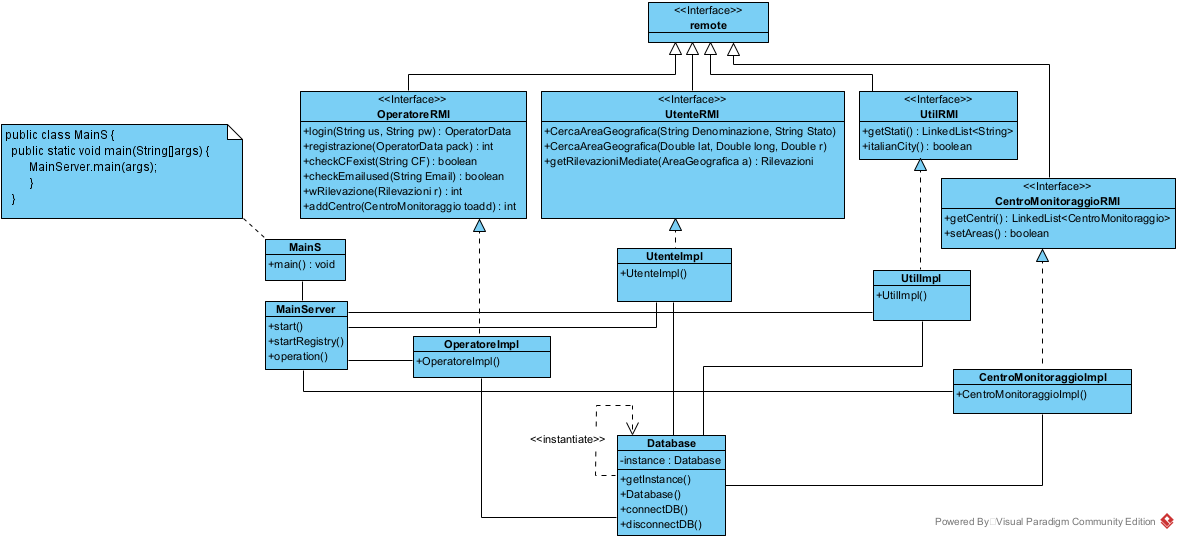
Il diagramma delle classi è stato utilizzato per descrivere le singole classi presenti nell’applicazioni e le loro relazioni. Questo è diviso principalmente in due moduli: client e server.

### Class Diagram del modulo Client

Immagine che contiene testo, schermata, Carattere, design

Descrizione generata automaticamente

### Class Diagram del modulo Server



## Sequence diagram

Il sequence diagram, utilizzato principalmente per scenari precisi, è stato utilizzato più volte per descrivere le azioni principali eseguibili da coloro che interagiscono con l’applicazione.

### Registrazione operatore

La registrazione all’interno dell’applicazione è richiesta solamente agli operatori. Essa prevede diversi passaggi, dei quali alcuni possono essere opzionali, rappresentati nel sequence diagram di seguito.

|  |
| --- |
| Immagine che contiene diagramma, schermata, Piano, design  Descrizione generata automaticamente |
| *Sequence Diagram – Registrazione operatore* |

Avviando il processo di registrazione, l’operatore può aggiungere, nel caso non fosse disponibile, il centro di monitoraggio a cui è affiliato e le aree di interesse associate.

I dati inseriti vengono controllati e, se rispettano i vincoli di integrità e correttezza, vengono salvati all’interno del database.

L’operatore risulta quindi registrato e può accedere a tutte le funzionalità garantite dall’applicazione.

### Inserimento parametri climatici

Una delle più importanti operazioni disponibili ad un operatore all’interno dell’applicazione è sicuramente l’inserimento di parametri climatici inerenti a un’area di interesse associata al proprio centro di monitoraggio.

|  |
| --- |
| Immagine che contiene schermata, diagramma, testo, design  Descrizione generata automaticamente |
| *Sequence Diagram – Inserimento parametri* |

Siccome i parametri si riferiscono a una specifica area di interesse e le aree sono associate ai centri di monitoraggio, l’inserimento dei parametri climatici implica il riconoscimento, mediante login o registrazione dell’operatore.

Una volta eseguito l’accesso, l’operatore può selezionare l’area di interesse tra quelle gestite dal centro di monitoraggio di appartenenza (queste aree sono elencate all’interno del suo profilo)

Una volta selezionata l’area sarà possibile inserire la propria rivelazione.

La rivelazione verrà successivamente salvata all’interno del database **dbCM**, così da poter essere visualizzabile all’interno di una una successiva ricerca dell’area.

### Visualizzazione parametri climatici

Un’ importante operazione eseguibile dagli utenti è quella di visualizzare i parametri climatici delle aree di interesse.

|  |
| --- |
| Immagine che contiene schermata, diagramma, design  Descrizione generata automaticamente |
| *Sequence Diagram – Visualizzazione parametri* |

A differenza dell’operatore, l’utente può accedere alle funzionalità di ricerca dell’area senza il bisogno di registrarsi.

Dopo essere entrato nella parte dell’applicazione adibita alla ricerca, egli potrà ricerca l’area di interesse desiderata attraverso i campi di ricerca.

Se non ci saranno violazioni nell’inserimento dei dati di ricerca, verrà eseguita una richiesta dei dati al database, il quale mostrerà a schermo i dati inerenti se presenti, altrimenti egli verrà notificato dello scorretto inserimento mediante una finestra di alert.

# Struttura dell’applicazione

### Moduli dell’applicazione

Il sistema è diviso principalmente in tre moduli:

* ClientCM: che comprende le classi per la gestione del client, composto da due package e due classi:
  + Package “controller” contenente le classi controller dell’interfaccia grafica
  + Package “model” contenente:
    - le classi che gestiscono la logica del client
    - un package RMI che contiene le classi utilizzate per RMI.
  + ClimateMonitoring.java (da dove parte l’applicazione client)
  + Main.java (è necessario per la creazione di shaded-jar)
* ServerCM: contenente le classi per la gestione del server e del database, composto anch’esso da due package e due classi:
  + Package “controller” contenente le classi controller dell’interfaccia grafica
  + Package “model” contenente:
    - le classi che gestiscono la logica del server
    - un package RMI che contiene le classi utilizzate per RMI e la loro relativa implementazione.
  + MainServer.java (da dove parte l’applicazione server)
  + Main.java (è necessario per la creazione di shaded-jar)
* DataStructures, un modulo condiviso da client e server, contenente le strutture dati condivise tra client e server, cui classi sono:
  + AreaGeografica.java
  + CentroMonitoraggio.java
  + OperatorData.java
  + Rilevazioni.java

**ClientCM e ServerCM** possiedono una directory “resources” contenente le risorse dei due moduli, ovvero:

* Per il client:
  + Files contenente i file di supporto alle funzioni di controllo del codice fiscale (tabellacomuni.txt, tabpari.txt, …)
  + images che comprende tutte le immagini utilizzati all’interno della grafica del client
  + objects in cui è contenuto il file “commentBox.fxml” che serve per rappresentare graficamente un commento.
  + Style in cui sono definiti i font utilizzati nell’applicazione e il foglio di style “style.css”
  + Views all’interno della quale sono definite le viste.
* Per il server:
  + Views in cui all’interno è presente la vista del login lato server.

Inoltre, per il server è specificata una directory “sql” contenente lo script scritto in SQL per costruire il database dbCM popolarlo con le coordinate delle aree geografiche, per mezzo di un plugin Maven.

## Client-Server

### RMI (Remote Method Invocation)

Per l’implementazione della comunicazione client-server, si è scelto di utilizzare RMI (Remote Method Invocation.

RMI è una tecnologia Java, più precisamente un middleware che permette la “location transparency” ovvero a un processo client di eseguire chiamate di metodi a oggetti remoti contenuti in un processo server come se questi fossero presenti sulla stessa macchina, mantenendo un codice più pulito e leggibile.

Il server crea oggetti remoti, rendendoli disponibili e aspetta che i client invochino metodi su questi.

Se il client non possiede nessuna informazione del riferimento a un oggetto remoto, lo può cercare mediante il servizio di Registry.

Nel nostro caso tale funzionalità è esplicitata nel metodo “startRegistry” della classe “MainServer”, come indicato sottostante:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Per ogni oggetto remoto ci sono due processi:

* un processo **Stub**, lato client, che funge da surrogato locale dell’oggetto remoto, questo invoca il metodo, il cui risultato è la creazione di un messaggio contenente il nome del metodo e gli argomenti impacchettati (marshalling) che viene inviato al server.
* un processo **Skeleton**, lato server, che riceve il messaggio, ricostruisce i parametri (unmarshalling) e chiama il metodo.

### DataStructures

All’interno di “DataStructures”, ogni classe rappresenta un POJO (ovvero una classe semplice senza riferimenti a strutture particolari) per i tipi definiti per l’applicazione.

Tutti i data model implementano l’interfaccia “Serializable” vincolo imposto da RMI il quale permette passare un oggetto di tipo non primitivo o di riceverlo.

#### AreaGeografica.java

La classe “AreaGeografica” rappresenta le aree geografiche, ovvero le aree di interesse presenti nell’applicazione.

Sono presenti due costruttori, uno che utilizza i parametri principali (idArea, denomination, state, latitude e longitude) e un secondo che include anche il parametro della distanza dai punti di riferimento.

La classe implementa i metodi di accesso getter e setter per i dati relativi ai campi.

#### CentroMonitoraggio.java

La classe “CentroMonitoraggio” permette lo scambio dei dati relativi ai centri di monitoraggio presenti nel sistema client/server.

Implementa i metodi di accesso getter e setter per i dati relativi ai campi.

#### OperatorData.java

La classe “OperatorData” contiene al suo interno i dati dell’operatore, e alcuni flag di controllo per permettere il corretto il login al sistema. Questa classe funge da “pacchetto” per trasmettere i dati di un operatore tra stub e skeleton di RMI.

Contiene i metodi di accesso getter e setter per l’acquisizione dei dati

#### Rilevazioni.java

La classe “Rilevazioni” rappresenta i dati relativi alle rilevazioni effettuate dagli operatori tramite l’utilizzo dell’applicazione.

Anch’esso contiene i metodi di accesso getter e setter per l’acquisizione dei dati.

#### CommentBoxData.java

Contiene al suo interno i dati relativi ai commenti effettuati dagli operatori.

### ClientCM

Esso rappresenta il lato client del sistema, al suo interno sono contenute:

* La classe “ClimateMonitoring.java” che rappresenta il punto di ingresso all’applicazione.
* La classe “Main.java” che andando a richiamare il metodo “main” della classe “ClimateMonitoring.java” permette di creare un file shaded-JAR funzionante, in quanto questo non funzionerebbe se si indicasse come MainClass del file JAR poiché estende un’altra classe (javafx.Application).
* Il package "Controller” contenente le classi controller necessarie per il funzionamento dell’interfaccia grafica.
* Il package “Model” contenente le interfacce per la comunicazione RMI e le classi di “business logic”.

### ServerCM

ServerCM rappresenta il lato Server del sistema. Al suo interno sono contenute:

* La classe “MainServer.java”, che rappresenta il main del server; esso esegue l’avvio del server, permette il login al database e carica sul registry i metodi delle interfacce implementate.
* La classe “MainS”, che analogamente alla classe contenuta nel modulo client va richiamare il metodo “main” della classe “MainServer.java”, questo come spiegato in precedenza permette di creare un file shaded-JAR funzionante, in quanto questo non funzionerebbe se si indicasse come MainClass del file JAR poiché estende un’altra classe (javafx.Application).
* La classe “MainServerController.java”, contenuta nel package controller, che gestisce il controllo dell’interfaccia grafica.
* La directory Model: contenente interfacce e le loro implementazioni per la comunicazione RMI e la classe che gestisce la connessione con il database.

#### MainServer

Questa classe in ServerCM rappresenta il main del server. Estende javafx.application.Application per la gestione dell’interfaccia grafica sviluppata con javaFX.

Implementa il metodo start() che permette l’avvio del server caricando il file fxml per l’interfaccia e il metodo startRegistry() per avviare il registry RMI rendendo gli oggetti remoti accessibili ai client.

#### Model

“Model” contiene al suo interno le interfacce necessarie per la comunicazione RMI da lato Server.

Le interfacce RMI contenute in Model sono:

* CentroMonitoraggioRMI.java
* OperatoreRMI.java
* UtenteRMI.java
* UtilRMI.java

Esse sono progettate per essere utilizzate nella comunicazione RMI, ciò implica che gli oggetti che implementano le interfacce possono essere chiamati dal client remoto tramite le invocazioni dei metodi presenti in ciascuna interfaccia.

Le classi riguardanti le entità che interagiscono con il client e i metodi che implementano le operazioni da loro eseguite sono:

* Operatore: classe relativa all’operatore, contiene i metodi necessari per la registrazione dei dati dell’operatore, la registrazione di un eventuale centro di monitoraggio, il controllo della correttezza della e-mail e del codice fiscale, il login, la rilevazione dei parametri e infine i metodi getter e setter. Essa implementa il design pattern singleton
* Utente: classe relativa all’utente, contiene i metodi necessari per la ricerca delle aree geografiche.
* checkCodiceFiscale: classe aggiuntiva inerente al controllo della correttezza del codice fiscale. Genera un codice fiscale italiano basandosi sui dati personali inseriti in fase di registrazione dall’operatore.

Come predisposto dall’utilizzo di RMI, per ogni classe rappresentante un’entità remota, è stata definita un’interfaccia che permette la creazione degli oggetti remoti. Le interfacce hanno denominazione “xRMI[[1]](#footnote-1)” e sono implementate dalle classi con denominazione “xImpl”.

* CentroMonitoraggioImpl
* OperatoreImpl
* UtenteImpl
* UtilImpl
* CentroMonitoraggioRMI
* OperatoreRMI
* UtenteRMI
* UtilRMI

### Uso del pattern MVC (Model-View-Controller)

Per garantire una maggiore leggibilità del sorgente, una struttura più ordinata e facilitare la manutebilità del sistema si è optato per l’utilizzo del pattern **Model-View-Controller**, separando la parte di business logic dalla GUI e unendole per mezzo di classi Controller.

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Model

Controller

View

A titolo esemplificativo, come mostrato nell’immagine sulla destra, la pagina di ricerca “SearchView.fxml” viene gestita dal controller “SearchAreaController.java” il quale sovraintende al funzionamento delle componenti grafiche della vista (textfield, bottoni, ecc..) e si interfaccia alla classe “Utente.java” la quale fornisce servizi di Business Logic come, ad esempio, la ricerca delle aree geografiche richiamando il metodo “cercaAreaGeografica()”.

### Uso del pattern Singleton

Il pattern singleton è un modello di progettazione creazionale, questo consente di garantire che una classe disponga di una sola istanza, fornendo al contempo un punto di accesso globale a tale istanza.

Questo è stato utilizzato lato client per permettere di accedere globalmente all’istanza dell’operatore contenuta in “Operatore.java” e garantire che al più di un’istanza di questa classe sia istanziata all’interno del processo, attraverso il metodo “getOperatore”.

Analogamente, il concetto è il medesimo con la classe “DataBase.java”, esiste un punto di accesso globale all’interno del server per accedere all’istanza e una sola istanza al massimo è garantita. Ogni qualvolta si debba fare riferimento al collegamento tra Server e database basta richiamare il metodo statico “getConnection” della classe “DataBase.java”.

#### Interfaccia grafica

L’interfaccia grafica per il server prevede l’utilizzo di una sola risorsa, HomeServer.fxml, contenuta nel percorso ServerCM\src\main\resources.

La risorsa permette l’implementazione di una semplice interfaccia che consente il login, la connessione e la disconnessione dal database.

## Database

Il database si interfaccia a lato server utilizzando la classe Database.java, che implementa il design pattern singleton, presente in ServerCM.

Al fine dello sviluppo è stato utilizzato Postgresql, un sistema di gestione di un database relazionale open-source. La connessione al database avviene tramite un URL a tale sistema.

### Diagramma ER

Per la progettazione concettuale del database è stato necessario la costruzione del seguente diagramma ER:

|  |
| --- |
| Immagine che contiene schermata, testo, diagramma, design  Descrizione generata automaticamente |
| *Diagramma ER* |

Nella modellazione del diagramma le entità sono rappresentate da “Operatore”, “Rilevazione”, “Centro di monitoraggio” e “Coordinate geografiche”.

Oltre ai vincoli che si possono osservare dal diagramma E-R devono essere soddisfatti anche i seguenti vincoli di integrità:

* Il commento di un operatore non può eccedere i 256 caratteri
* L’indirizzo della città deve appartenere ad un territorio italiano
* I parametri (Vento, …) devono esser degli interi e appartenere all’intervallo [1,5]

### Fase di traduzione

Per tradurre il diagramma ER in un equivalente schema relazionale, non ci sono state molte complicanze, si è seguito il modello di traduzione consueto.

Le scelte effettuate durante la traduzione sono state le seguenti.

#### Traduzione delle entità

Traduzione dell’entità “Operatore”:

Per tradurre l’entità operatore si è scelta come chiave primaria l’identificatore “id” poiché utilizzare come chiave primaria il binomio (“Codice Fiscale”, “Email”) avrebbe portato ad una chiave primaria molto grande, con relative problematiche di efficienza.

### Schema logico

Sulla base della ristrutturazione, si è creato quindi il seguente schema logico:

CoordinateMonitoraggio(ID, LATITUDINE, LONGITUDINE, DENOMINAZIONE, STATO)

CentriMonitoraggio(ID, NOME, INDIRIZZO)

Controllare(ID\_AREACoordinateMonitoraggio, ID\_CENTROCentriMonitoraggio)

OperatoriRegistrati(ID\_OP, CODICE\_FISCALE, NOME, COGNOME, PASSWORD, ID\_CENTROCentriMonitoraggio)

Rilevazioni(ID, VENTO, UMIDITA, PRESSIONE, TEMPERATURA, PRECIPITAZIONI, ALTITUDINE\_GHIACCIAIo, MASSA\_GHACCIAIo, COMMENTOo, DATA\_RILEVAZIONE, ID\_AREACoordinateMonitoraggio, ID\_CENTROCentriMonitoraggio, ID\_OPOperatoriRegistrati)

Cui traduzione in linguaggio DDL di SQL è:

CREATE TABLE CoordinateMonitoraggio (

ID integer PRIMARY KEY,

LATITUDINE real NOT NULL,

LONGITUDINE real NOT NULL,

DENOMINAZIONE VARCHAR(80) NOT NULL,

STATO VARCHAR(50) NOT NULL);

CREATE TABLE CentriMonitoraggio (

ID SERIAL PRIMARY KEY,

NOME VARCHAR(50) UNIQUE NOT NULL,

INDIRIZZO VARCHAR(100) NOT NULL);

CREATE TABLE Controllare (

ID\_AREA integer REFERENCES CoordinateMonitoraggio(ID) ON UPDATE CASCADE NOT NULL,

ID\_CENTRO SERIAL REFERENCES CentriMonitoraggio(ID) ON UPDATE CASCADE NOT NULL,

PRIMARY KEY (ID\_AREA, ID\_CENTRO));

CREATE TABLE OperatoriRegistrati (

ID\_OP SERIAL PRIMARY KEY,

CODICE\_FISCALE CHAR(16) UNIQUE NOT NULL,

NOME VARCHAR(50) NOT NULL,

COGNOME VARCHAR(50) NOT NULL,

EMAIL VARCHAR(254) UNIQUE NOT NULL,

PASSWORD VARCHAR(100) NOT NULL,

ID\_CENTRO integer REFERENCES CentriMonitoraggio(ID) ON UPDATE CASCADE NOT NULL

);

CREATE TABLE Rilevazioni (

ID SERIAL PRIMARY KEY,

VENTO integer NOT NULL,

UMIDITA integer NOT NULL,

PRESSIONE integer NOT NULL,

TEMPERATURA integer NOT NULL,

PRECIPITAZIONI integer NOT NULL,

ALTITUDINE\_GHIACCIAI integer,

MASSA\_GHIACCIAI integer,

COMMENTO VARCHAR(256),

DATA\_RILEVAZIONE Date NOT NULL,

ID\_AREA integer REFERENCES CoordinateMonitoraggio(ID) ON UPDATE CASCADE NOT NULL,

ID\_CENTRO integer REFERENCES CentriMonitoraggio(ID) ON UPDATE CASCADE NOT NULL,

ID\_OP integer REFERENCES OperatoriRegistrati(ID\_OP) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL

);

Non sono stati inseriti vincoli o asserzioni all’interno dello script SQL, in quanto l’integrità del database è garantito dall’applicativo.

## Interfaccia grafica

L’interfaccia grafica lato client è stata sviluppata mediante l’utilizzo di SceneBuilder, un toolkit grafico che permette la creazione di interfacce grafiche in formato FXML e l’associazione di determinati metodi a degli eventi che accadono nella GUI (per mezzo di attributi nei tag FXML).

Le interfacce grafiche sono elencate di seguito:

* AreaInfoView
* AreaView
* CenterView
* HomePageView
* OperatorView
* SignInView
* SearchView

Ogni file “.fxml” corrisponde a una schermata dell’applicazione.

Per garantire l’interazione con gli utenti, tali risorse interagiscono con i rispettivi controller, grazie all’implementazione del pattern architetturale Model-View-Controller sopracitato, dove il controller funge da intermediario tra la vista dell’interfaccia e il model della struttura dati.

I metodi dei controller, annotati con @FXML, vengono invocati come risposta alle azioni dell’utente sull’interfaccia, quelli per la gestione delle transizioni di scena vengono richiamati dal “SceneController”.

### SceneController

Al fine di ottenere un codice più pulito si è visto necessario l’inserimento di un controller aggiuntivo, non associato a nessuna schermata in particolare, contenente i metodi per il cambio di scena all’interno dell’applicazione.

I metodi contenuti in questo controller vengono richiamati dagli altri controller per le istruzioni che prevedono il cambio di schermata.

# Riferimenti

La scala e misura dei venti: la classificazione Beaufort. (s.d.). *http://www.centrometeo.com/articoli-reportage-approfondimenti/fisica-atmosferica/5864-scala-misura-venti-classificazione-beaufort*.

1. x corrisponde al nome della classe [↑](#footnote-ref-1)