2023/2024

MANUALE TECNICO

|  |  |
| --- | --- |
| RICCI MARCO  RIZZI EDOARDO  ALBERTO STAGNO  DENIS DI NAPOLI | 755656  754618  755404  755340 |

Climate Monitoring v4.0

UNIVERSITà DEGLI STUDI DELL’INSUBRIA LAUREA TRIENNALE IN INFORMATICA

Sommario

[Introduzione 3](#_Toc168774334)

[Strumenti - linguaggi – SO utilizzati 3](#_Toc168774335)

[Librerie esterne ed API 4](#_Toc168774336)

[Codice fiscale 4](#_Toc168774337)

[JCalendar 4](#_Toc168774338)

[GSON 4](#_Toc168774339)

[POSTGRESQL 5](#_Toc168774340)

[AbsoluteLayout 5](#_Toc168774341)

[Geonames API 5](#_Toc168774342)

[Processo di sviluppo 5](#_Toc168774343)

[Principali milestone 5](#_Toc168774344)

[Suddivisione dei lavori 6](#_Toc168774345)

[Progettazione del database 7](#_Toc168774346)

[Schema concettuale 7](#_Toc168774347)

[Vincoli d’integrità 8](#_Toc168774348)

[Scelte Progettuali 8](#_Toc168774349)

[Progettazione Logica 8](#_Toc168774350)

[Progettazione pratica 10](#_Toc168774351)

[Creazione database 10](#_Toc168774352)

[Creazione tabelle 10](#_Toc168774353)

[Progettazione del software 13](#_Toc168774354)

[Struttura del sistema 13](#_Toc168774355)

[Classi Logiche 13](#_Toc168774356)

[Interfaccia 14](#_Toc168774357)

[Classi Server 14](#_Toc168774358)

[Classi Grafiche 14](#_Toc168774359)

[Class Diagram 15](#_Toc168774360)

[Sequence Diagram 15](#_Toc168774361)

[Use case 16](#_Toc168774362)

[Implementazione del sotfware 16](#_Toc168774363)

[Classi logiche 16](#_Toc168774364)

[JAreaInteresse 16](#_Toc168774365)

[JCoordinate 17](#_Toc168774366)

[JStazione 17](#_Toc168774367)

[JPrevisioni 18](#_Toc168774368)

[JUser 19](#_Toc168774369)

[JNazioni 20](#_Toc168774370)

[DatiCondivisi 20](#_Toc168774371)

[Interfaccia 21](#_Toc168774372)

[DBInterface 21](#_Toc168774373)

[Classi server 22](#_Toc168774374)

[DatabaseConnection 22](#_Toc168774375)

[Server 23](#_Toc168774376)

[Classi grafiche 24](#_Toc168774377)

[admin\_panel 24](#_Toc168774378)

[creaStazione 24](#_Toc168774379)

[homepage 25](#_Toc168774380)

[infoStazione 25](#_Toc168774381)

[login 25](#_Toc168774382)

[mostraPrevisioni 25](#_Toc168774383)

[registrazione 26](#_Toc168774384)

[Implementazione del database 26](#_Toc168774385)

[Operazioni di Insert 26](#_Toc168774386)

[Query 26](#_Toc168774387)

[Delete 26](#_Toc168774388)

[Progettazione Logica 26](#_Toc168774389)

[Errori noti 26](#_Toc168774390)

[Sitografia 26](#_Toc168774391)

[codice-fiscale-java-master - matteocastagnaro 26](#_Toc168774392)

[JCalendar - Kai Tödter 26](#_Toc168774393)

[JDK 17 26](#_Toc168774394)

[PostgresSQL 26](#_Toc168774395)

[PostgresSQL Driver 27](#_Toc168774396)

[pgAdmin 4 27](#_Toc168774397)

[DBeaver 27](#_Toc168774398)

[GSON 27](#_Toc168774399)

[GeoNames 27](#_Toc168774400)

[IntelliJ 27](#_Toc168774401)

[Apache NetBeans 27](#_Toc168774402)

[Clever Cloud 27](#_Toc168774403)

# Introduzione

Il progetto “*Climate Monitoring*” è un sistema di monitoraggio di parametri climatici fornito da centri di monitoraggio sul territorio italiano; utilizzabile sia dal punto di vista di un comune cittadino, che potrà ricercare l’area d’interesse o la stazione metereologica da lui desiderata consultando poi le previsioni inserite, e sia dal punto di vista di un operatore ambientale che potrà, oltre a visionare le previsioni anche modificarle, aggiungerle ed eliminarle.

Quest’ultimo è stato svolto per il corso “Laboratorio interdisciplinare A” nell’anno accademico 2022/2023 per il corso di laurea informatica dell’Università degli Studi dell’Insubria, revisionato e modificato per il corso “Laboratorio interdisciplinare B” dell’anno accademico 2023/2024.

## Strumenti - linguaggi – SO utilizzati

Il progetto sviluppato utilizza i seguenti strumenti con le relative versioni utilizzate:

* JDK 17
* PostgresSQL 16.3
* pgAdmin 4 e DBeaver
* RMI

Linguaggi utilizzati:

* Java per l’applicazione lato client e server
* SQL
* JSON
* XML

Librerie esterne:

* jcalendar-1.4
* gson-2.8.8
* codice-fiscale-java-master
* postgresql-42.7.3
* AbsoluteLayout

API:

* GeoNames

IDE:

* IntelliJ
* Apache Netbeans

Database Management Tool:

* pgAdmin 4
* DBever

SO:

* Windows 10
* Windows 11

ll progetto è sviluppato in Java 17 su sistemi Windows 10 e 11, e testato sugli stessi.

La comunicazione tra client e server avviene tramite la classe *UnicastRemoteObject*, che consente di creare oggetti remoti utilizzando Java RMI (Remote Method Invocation).   
Questo permette al client di invocare metodi sul server, tramite un’interfaccia comune, come se fossero in locale, gestendo in modo trasparente le comunicazioni di rete.

Per la scrittura del codice Java abbiamo utilizzato due IDE differenti:

* Il primo è IntelliJ IDEA, scelto per la stesura delle classi logiche e per interfacciarci al database
* Il secondo ambiente di sviluppo è stato Apache NetBeans, preferito per la gestione delle classi grafiche, grazie alla sua interfaccia intuitiva per il design delle GUI.

Per il database, abbiamo utilizzato PostgreSQL nella versione 16.3, gestito tramite pgAdmin 4 e DBeaver. Abbiamo hostato il database sia in locale, sia online su *Clever Cloud*.

L'applicazione client-server è scritta in Java. Sul lato server, utilizziamo query SQL per l'interazione con il database. Per prendere i dati dal web service di [Geonames](https://www.geonames.org/) abbiamo scelto di usare JSON.

Abbiamo integrato diverse librerie esterne per ampliare le funzionalità dell'applicazione, tra cui jcalendar-1.4 per la gestione delle date, gson-2.8.8 per la manipolazione di JSON, codice-fiscale-java-master per la gestione dei codici fiscali italiani, postgresql-42.7.3 per la connessione al database PostgreSQL, e AbsoluteLayout per la gestione dei layout grafici.

Inoltre, l'applicazione utilizza le API di GeoNames per integrare funzionalità di mappatura e geolocalizzazione.

## Librerie esterne ed API

### Codice fiscale

La libreria “*codice-fiscale-java-master*” ci permette tramite i metodi in essa contenuti di calcolare il codice fiscale di una persona, di cui ci vengono forniti i dati personali necessari per il calcolo. Questa libreria viene utilizzata nella parte di registrazione di un utente.

### JCalendar

JCalendar, è una libreria che abbiamo aggiunto per migliorare l’aspetto grafico del lavoro, la libreria permette di selezionare una data tramite un calendario o dei combobox. Al suo interno troviamo altri oggetti come JDayChooser, JMonthChooser e JYearChooser.

### GSON

Gson è una libreria che semplifica la conversione tra oggetti Java e JSON.   
Utilizzando Gson, è possibile trasformare facilmente un oggetto Java in una stringa JSON, utile per l'archiviazione o l'invio di dati.   
Allo stesso modo, Gson consente di prendere una stringa JSON e convertirla in un oggetto Java, facilitando la manipolazione dei dati ricevuti.

### POSTGRESQL

La libreria esterna postgresql è un driver JDBC che consente alle applicazioni Java di interagire con un database PostgreSQL.   
Questo driver facilita la connessione al database, l'esecuzione di query SQL, l'aggiornamento dei dati e il recupero dei risultati delle query direttamente all'interno di un'applicazione Java.

### AbsoluteLayout

La libreria esterna AbsoluteLayout è un sistema di layout utilizzato nelle interfacce grafiche Java per posizionare i componenti in maniera assoluta all'interno di un contenitore, come ad esempio un JFrame o un JPanel.

### Geonames API

Geonames API è una API che consente di accedere a dati geografici dettagliati. Utilizzando Geonames API, è possibile effettuare ricerche di località geografiche, ottenere informazioni su città, paesi e coordinate GPS

## Processo di sviluppo

Dopo aver letto il documento di specifica dei requisiti lo abbiamo confrontato con il lavoro precedentemente svolto per analizzare le diverse richieste ed iniziare a tracciare i vari discostamenti dei progetti.

Le specifiche più rilevanti che facevano differire i progetti erano sostanzialmente due: come prima era l’utilizzo di un database per salvare i dati dell’applicazione invece che utilizzare dei file; mentre la seconda era quella di trasformare il software in un’architettura di tipo client-server utilizzando java RMI per far comunicare i due lati tramite interfacce.

Assimilato cosa avremmo dovuto modificare abbiamo iniziato la progettazione del database.

Concluso il database siamo passati alla stesura del codice, analizzando prima le parti del codice necessarie assenti, ovvero le classi relative alla connessione con il database e la classe che gestisce il server implementando l’interfaccia che permette al client di comunicare. Individuate queste macroaree abbiamo suddiviso il lavoro tramite delle milestone.

Successivamente ci siamo messi ad implementare il codice e durante tutto questa fase abbiamo regolarmente testato e debuggato il codice.

## Principali milestone

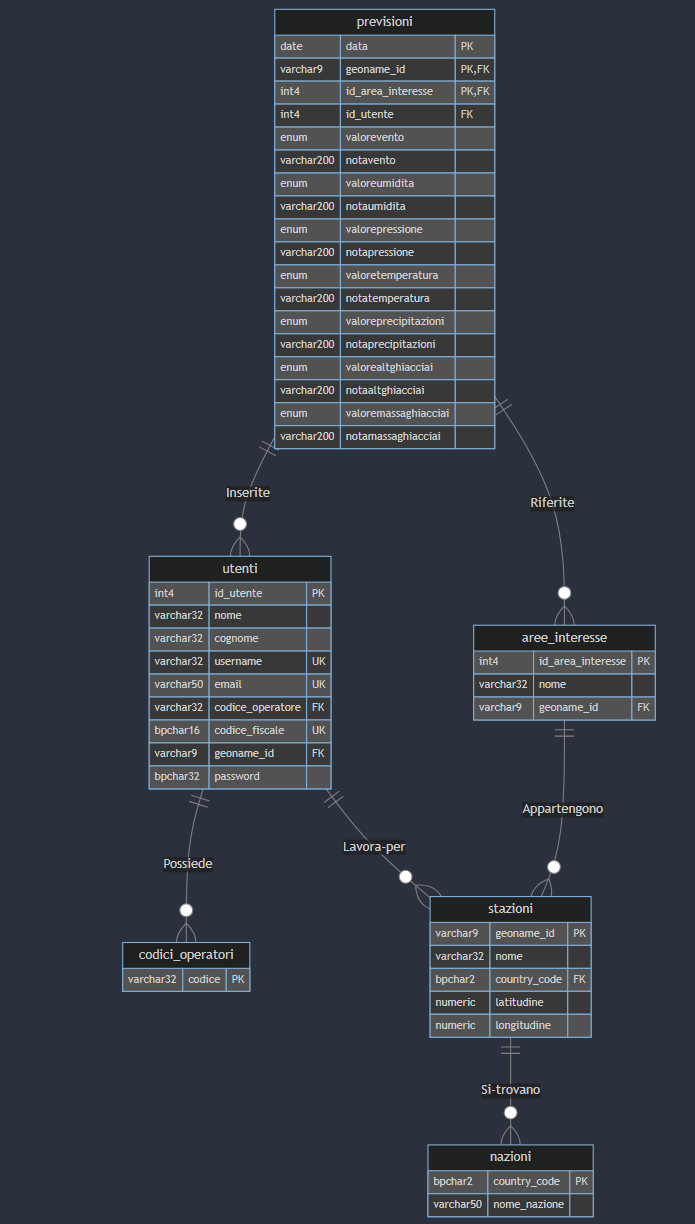
* Connessione al DB;
* DBInterface;
* Server con RMI;
* Modificare homepage per RMI;
* Adattamento codice client;
* Adattamento grafica;

## Suddivisione dei lavori

Come prima cosa ci siamo suddivisi in due gruppi composti da due persone, per poter lavorare sia singolarmente sul codice ma anche insieme tramite la tecnica del “pair programming”, non abbiamo assegnato una parte di codice ad un team, ma aggiornavamo le milestone per completarle come indicato precedentemente. Tutti i componenti hanno lavorato sia nella parte client che in quella server per velocizzare il raggiungimento delle milestone.

# Progettazione del database

## Schema concettuale



### Vincoli d’integrità

|  |  |  |  |
| --- | --- | --- | --- |
| Tabella | Nome Vincolo | Tipo di Vincolo | Dettagli |
| codici\_operatori | codici\_operatori\_key | CHIAVE PRIMARIA | Chiave primaria su codice |
| nazioni | nazioni\_pkey | CHIAVE PRIMARIA | Chiave primaria su country\_code |
| stazioni | stazioni\_pkey, stazioni\_country\_code\_fkey | CHIAVE PRIMARIA, CHIAVE ESTERNA | Chiave primaria su geoname\_id, Chiave esterna su country\_code che fa riferimento a nazioni(country\_code) |
| utenti | utenti\_pkey, utenti\_unique, utenti\_unique\_1, utenti\_unique\_2, utenti\_codice\_operatore\_fkey, utenti\_geoname\_id\_fkey | CHIAVE PRIMARIA, UNIQUE, CHIAVE ESTERNA | Chiave primaria su id\_utente, UNIQUE su username, UNIQUE su email, UNIQUE su codice\_fiscale, Chiave esterna su codice\_operatore che fa riferimento a codici\_operatori(codice), Chiave esterna su geoname\_id che fa riferimento a stazioni(geoname\_id) |
| aree\_interesse | area\_interesse\_pkey, area\_interesse\_geoname\_id\_fkey | CHIAVE PRIMARIA, CHIAVE ESTERNA | Chiave primaria su (id\_area\_interesse, geoname\_id), Chiave esterna su geoname\_id che fa riferimento a stazioni(geoname\_id) |
| previsioni | previsioni\_pkey, previsioni\_geoname\_id\_id\_area\_interesse\_fkey, previsioni\_id\_utente\_utenti\_id\_utente\_fkey | CHIAVE PRIMARIA, CHIAVE ESTERNA | Chiave primaria su (data, geoname\_id, id\_area\_interesse), Chiave esterna su (geoname\_id, id\_area\_interesse) che fa riferimento a aree\_interesse(geoname\_id, id\_area\_interesse), Chiave esterna su id\_utente che fa riferimento a utenti(id\_utente) |

### Scelte Progettuali

Nella tabella codici\_operatori sono presenti i codici degli operatori autorizzati a iscriversi come utenti amministratori al servizio di Climate Monitoring. Ogni tupla nella tabella rappresenta un codice univoco che può essere assegnato ad un solo utente. Per garantire la sicurezza e il controllo, l'inserimento dei codici operatore nel database non può essere effettuato né dal servizio client né dal servizio server. Abbiamo progettato il processo di registrazione in modo che sia preceduto da un passaggio gestito direttamente dagli amministratori di Climate Monitoring tramite un servizio web dedicato.

## Progettazione Logica

Tabelle:

* codici\_operatori:
  + Contiene i codici degli operatori.
  + **Attributi**:
    - codice(VARCHAR(32)): Chiave primaria.
  + **Vincoli**:
    - codici\_operatori\_pkey: Chiave primaria su codice.
* nazioni:
  + Contiene tutte le nazioni del mondo.
  + **Attributi**:
    - country\_code(CHAR(2)): Chiave primaria.
    - nome\_nazione(VARCHAR(50)): Nome della nazione.
  + **Vincoli**:
    - nazione\_pkey: Chiave primaria su country\_code.
* stazioni:
  + Contiene le stazioni con le loro coordinate geografiche.
  + **Attributi**:
    - geoname\_id (VARCHAR(9)): Chiave primaria.
    - nome (VARCHAR(32)): Nome della stazione.
    - country\_code (BPCHAR(2)): Codice della nazione (chiave esterna).
    - latitudine (NUMERIC(8, 5)): Latitudine della stazione.
    - longitudine (NUMERIC(8, 5)): Longitudine della stazione.
  + **Vincoli**:
    - stazioni\_pkey: Chiave primaria su geoname\_id.
    - stazioni\_country\_code\_fkey: Chiave esterna su country\_code che fa riferimento a nazioni(country\_code).
* utenti:
  + Contiene gli utenti del sistema.
  + **Attributi**:
    - id\_utente (INT4): Chiave primaria, generata automaticamente.
    - nome (VARCHAR(32)): Nome dell'utente.
    - cognome (VARCHAR(32)): Cognome dell'utente.
    - username (VARCHAR(32)): Nome utente, deve essere unico.
    - email (VARCHAR(50)): Email, deve essere unica.
    - codice\_operatore (VARCHAR(32)): Codice operatore (chiave esterna).
    - codice\_fiscale (BPCHAR(16)): Codice fiscale, deve essere unico.
    - geoname\_id (VARCHAR(9)): ID della stazione (chiave esterna).
    - password (BPCHAR(32)): Password.
  + **Vincoli**:
    - utenti\_pkey: Chiave primaria su id\_utente.
    - utenti\_unique: UNIQUE su username.
    - utenti\_unique\_1: UNIQUE su email.
    - utenti\_unique\_2: UNIQUE su codice\_fiscale.
    - utenti\_codice\_operatore\_fkey: Chiave esterna su codice\_operatore che fa riferimento a codici\_operatori(codice).
    - utenti\_geoname\_id\_fkey: Chiave esterna su geoname\_id che fa riferimento a stazioni(geoname\_id).
* aree\_interesse:
  + Contiene le aree di interesse.
  + **Attributi**:
    - id\_area\_interesse (INT4): Chiave primaria (parte 1).
    - nome (VARCHAR(32)): Nome dell'area di interesse.
    - geoname\_id (VARCHAR(9)): ID della stazione (chiave primaria parte 2 e chiave esterna).
  + **Vincoli**:
    - area\_interesse\_pkey: Chiave primaria composta su id\_area\_interesse e geoname\_id.
    - area\_interesse\_geoname\_id\_fkey: Chiave esterna su geoname\_id che fa riferimento a stazioni(geoname\_id).
* previsioni:
  + Contiene le previsioni meteorologiche inserite dagli utenti.
  + **Attributi**:
    - data (DATE): Chiave primaria (parte 1).
    - geoname\_id (VARCHAR(9)): ID della stazione (chiave primaria parte 2 e chiave esterna).
    - id\_area\_interesse (INT4): ID dell'area di interesse (chiave primaria parte 3 e chiave esterna).
    - id\_utente (INT4): ID dell'utente che ha inserito la previsione (chiave esterna).
    - valorevento, valoreumidita, valorepressione, valoretemperatura, valoreprecipitazioni, valorealtghiacciai, valoremassaghiacciai (enum\_valore): Valori delle previsioni.
    - notavento, notaumidita, notapressione, notatemperatura, notaprecipitazioni, notaaltghiacciai, notamassaghiacciai (VARCHAR(200)): Note delle previsioni.
  + **Vincoli**:
    - previsioni\_pkey: Chiave primaria composta su data, geoname\_id e id\_area\_interesse.
    - previsioni\_geoname\_id\_id\_area\_interesse\_fkey: Chiave esterna composta su geoname\_id e id\_area\_interesse che fa riferimento a aree\_interesse(geoname\_id, id\_area\_interesse).
    - previsioni\_id\_utente\_utenti\_id\_utente\_fkey: Chiave esterna su id\_utente che fa riferimento a utenti(id\_utente).

Strutture dati:

* enum\_previsioni:
  + È un tipo ENUM che può assumere un valore di tipo char, i possibili valori sono i numeri da 1 a 5 estremi inclusi.
  + Viene utilizzato per semplificare il salvataggio delle informazioni nella tabella delle previsioni.

Funzioni:

* before\_insert\_area\_interesse:
  + Questa funzione viene richiamata dal dbms prima di aggiungere ogni area di interesse.
  + Ritorna un valore intero MAX\_ID, che sta per il l’id massimo trovato per le aree di interesse con un determinato geoname\_id passato come parametro.
  + Questo ci permette di non avere due aree di interesse, con lo stesso id\_area\_interesse, sotto la stessa stazione (stesso geoname\_id)
  + Dopo la creazione della funzione deve essere creato anche un TRIGGER, il quale ogni volta che proviamo ad aggiungere una nuova area, prima di farlo richiama la funzione prima citata.

## Progettazione pratica

### Creazione database

Per la creazione del database abbiamo utilizzato il servizio di PostgreSQL 16.3.

Passaggi per la creazione del database:

* Installare PostgreSQL e PgAdmin(opzionale) dal [sito ufficiale](https://www.postgresql.org/download/).
* Creazione del database:
  + Creare il database con il seguente comando:
    - CREATE DATABASE nome\_del\_database;
* Creazione di strutture dati, tabelle, vincoli e funzioni (il DDL si trova nella sezione successiva)
* Importazione dei dati:
  + Obbligatoria per quanto riguarda la tabella codici\_operatori
  + Opzionale per quanto riguarda tutti gli altri dati.

### Creazione tabelle

CREATE TYPE public.enum\_valore AS ENUM ('1', '2', '3', '4', '5');

CREATE TABLE public.codici\_operatori (

codice varchar(32) NOT NULL,

CONSTRAINT codici\_operatori\_pkey PRIMARY KEY (codice));

CREATE TABLE public.nazioni (

country\_code bpchar(2) NOT NULL,

nome\_nazione varchar(50) NULL,

CONSTRAINT nazioni\_pkey PRIMARY KEY (country\_code));

CREATE TABLE public.stazioni (

geoname\_id varchar(9) NOT NULL,

nome varchar(32) NOT NULL,

country\_code bpchar(2) NOT NULL,

latitudine numeric(8, 5) NOT NULL,

longitudine numeric(8, 5) NOT NULL,

CONSTRAINT stazioni\_pkey PRIMARY KEY (geoname\_id),

CONSTRAINT stazioni\_country\_code\_fkey FOREIGN KEY (country\_code) REFERENCES public.nazioni(country\_code) ON DELETE RESTRICT ON UPDATE CASCADE);

CREATE TABLE public.utenti (

id\_utente int4 GENERATED ALWAYS AS IDENTITY( INCREMENT BY 1 MINVALUE 1 MAXVALUE 2147483647 START 1 CACHE 1 NO CYCLE) NOT NULL,

nome varchar(32) NOT NULL,

cognome varchar(32) NOT NULL,

username varchar(32) NOT NULL,

email varchar(50) NOT NULL,

codice\_operatore varchar(32) NOT NULL,

codice\_fiscale bpchar(16) NOT NULL,

geoname\_id varchar(9) NOT NULL,

"password" bpchar(32) NOT NULL,

CONSTRAINT utenti\_pkey PRIMARY KEY (id\_utente),

CONSTRAINT utenti\_unique UNIQUE (username),

CONSTRAINT utenti\_unique\_1 UNIQUE (email),

CONSTRAINT utenti\_unique\_2 UNIQUE (codice\_fiscale),

CONSTRAINT utenti\_codice\_operatore\_fkey FOREIGN KEY (codice\_operatore) REFERENCES public.codici\_operatori(codice) ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT utenti\_geoname\_id\_fkey FOREIGN KEY (geoname\_id) REFERENCES public.stazioni(geoname\_id) ON DELETE RESTRICT ON UPDATE CASCADE);

CREATE TABLE public.aree\_interesse (

id\_area\_interesse int4 NOT NULL,

nome varchar(32) DEFAULT NULL::character varying NULL,

geoname\_id varchar(9) NOT NULL,

CONSTRAINT area\_interesse\_pkey PRIMARY KEY (id\_area\_interesse, geoname\_id),

CONSTRAINT area\_interesse\_geoname\_id\_fkey FOREIGN KEY (geoname\_id) REFERENCES public.stazioni(geoname\_id) ON DELETE RESTRICT ON UPDATE CASCADE);

CREATE OR REPLACE FUNCTION public.before\_insert\_area\_interesse()

RETURNS trigger

LANGUAGE plpgsql

AS $function$

DECLARE

max\_id INT;

BEGIN

SELECT COALESCE(MAX(id\_area\_interesse), 0) INTO max\_id FROM aree\_interesse WHERE geoname\_id = NEW.geoname\_id;

NEW.id\_area\_interesse := max\_id + 1;

RETURN NEW;

END;

$function$

;

create trigger before\_insert\_area\_interesse before

insert

on

public.aree\_interesse for each row execute function before\_insert\_area\_interesse();

CREATE TABLE public.previsioni (

"data" date NOT NULL,

geoname\_id varchar(9) NOT NULL,

id\_area\_interesse int4 NOT NULL,

id\_utente int4 NOT NULL,

valorevento public."enum\_valore" NULL,

notavento varchar(200) DEFAULT NULL::character varying NULL,

valoreumidita public."enum\_valore" NULL,

notaumidita varchar(200) DEFAULT NULL::character varying NULL,

valorepressione public."enum\_valore" NULL,

notapressione varchar(200) DEFAULT NULL::character varying NULL,

valoretemperatura public."enum\_valore" NULL,

notatemperatura varchar(200) DEFAULT NULL::character varying NULL,

valoreprecipitazioni public."enum\_valore" NULL,

notaprecipitazioni varchar(200) DEFAULT NULL::character varying NULL,

valorealtghiacciai public."enum\_valore" NULL,

notaaltghiacciai varchar(200) DEFAULT NULL::character varying NULL,

valoremassaghiacciai public."enum\_valore" NULL,

notamassaghiacciai varchar(200) DEFAULT NULL::character varying NULL,

CONSTRAINT previsioni\_pkey PRIMARY KEY (data, geoname\_id, id\_area\_interesse),

CONSTRAINT previsioni\_geoname\_id\_id\_area\_interesse\_fkey FOREIGN KEY (geoname\_id,id\_area\_interesse) REFERENCES <?>() ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT previsioni\_id\_utente\_utenti\_id\_utente\_fkey FOREIGN KEY (id\_utente) REFERENCES public.utenti(id\_utente) ON DELETE RESTRICT ON UPDATE CASCADE);

CREATE INDEX fki\_f ON public.previsioni USING btree (id\_utente);

CREATE INDEX fki\_utenti\_fk ON public.previsioni USING btree (id\_utente);

# Progettazione del software

## Struttura del sistema

Le classi del progetto si suddividono in tre categorie; quelle relative all’interfaccia grafica o client (che chiameremo “*classi grafiche*”), quelle che elaborano i diversi dati (che chiameremo “*classi logiche*”) e le classi relative al server.

### Classi Logiche

* JAreaInteresse
* JCoordinate
* JNazione
* JPrevisioni
* JStazione
* JUser
* DatiCondivisi

### Interfaccia

* DBInterface

### Classi Server

* DatabaseConnection
* Server

### Classi Grafiche

* admin\_panel
* creaStazione
* homepage
* infoStazione
* login
* mostraPrevisioni
* registrazione

Presenteremo adesso le diverse classi iniziando con quelle “Logiche”, successivamente quelle “Grafiche” ed infine quelle relative al server, descrivendone il loro compito e i metodi principali di ogni classe.

### Immagine che contiene testo, schermata, diagramma, Parallelo Descrizione generata automaticamenteClass Diagram

### Sequence Diagram

…..

### Use case

Immagine che contiene testo, diagramma, schermata, cerchio

Descrizione generata automaticamente

# Implementazione del sotfware

# Classi logiche

## JAreaInteresse

JAreaIntresse è la classe che gestisce le diverse aree di interesse. Contiene tutti gli attributi richiesti dalla specifica i più significativi sono:

* *id\_area*, di tipo Integer, che è il codice identificativo dell’area d’interesse;
* *geoname\_id*, di tipo Integer, che è il codice identificativo dell'area d'interesse;

*id\_area* è la chiave primaria nella relazione *AreaInteresse.*

Nella classe inoltre sono presenti i metodi get e set per ogni attributo ed anche il metodo toString

Gli attributi della classe sono:

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| Integer | id\_area | Codice identificativo dell'area d'interesse |
| Integer | geoname\_id | Codice della stazione metereologica a cui fa appartenenza l'area d'interesse. |
| String | nome | Nome dell’area d’interesse |

Nella classe non sono presenti metodi particolari oltre i classici operatori, get/set e il toString.

## JCoordinate

La classe JCoordinate e di conseguenza l’omonimo oggetto, ha lo scopo di facilitare la gestione dei punti cardinale utilizzate varie volte nel corso del progetto. Contiene tutti gli attributi:

* *lat*, di tipo Float, che è la coordinata geografica che indica la distanza dall’equatore al punto in cui ti trovi;
* *lon*, di tipo Float, che è la coordinata geografica che indica la distanza dal meridiano di Greenwich al punto in cui ti trovi;

Nella classe inoltre sono presenti i metodi get e set per ogni attributo ed anche il metodo toString

Gli attributi della classe sono:

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| Float | lat | Coordinata geografica che indica la distanza dall’equatore al punto in cui ti trovi |
| Float | lon | Coordinata geografica che indica la distanza dal meridiano di Greenwich al punto in cui ti trovi |

I metodi principali della classe sono

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome metodo | Parametri | Ritorna | Descrizione | Complessità |
| sonoCoordinate | String txt:  stringa che contiene le possibili coordinate | boolean | controlla che la stringa passata sia effettivamente una coordinata | O(n), qui la complessità varia a seconda della lunghezza |

## JStazione

JStazione è la classe che si occupa della gestione delle stazioni metereologiche. Contiene tutti gli attributi richiesti dalla specifica i più significativi sono:

* *geoname\_id*, di tipo Integer, che è il codice identificativo dell'area d'interesse;
* *country\_code*, di tipo String, che è il codice della nazione dove si trova la stazione metereologica;

*geoname\_id* è la chiave primaria nella relazione *Stazioni.*

Nella classe inoltre sono presenti i metodi get e set per ogni attributo ed anche il metodo toString

Gli attributi della classe sono:

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| Integer | geoname\_id | Codice identificativo della cittá dove si trova la stazione metereologica. |
| String | nome | Nome della cittá dove si trova la stazione metereologica |
| String | country\_code | Codice della nazione dove si trova la stazione metereologica |
| String | country | Nome della nazione dove si trova la stazione metereologica |
| JCoordinate | coordinate | Coordinate della città dove si trova la stazione metereologica |

Nella classe non sono presenti metodi particolari oltre i classici operatori, get/set e il toString.

## JPrevisioni

JPrevisioni è la classe che si occupa della creazione e gestione delle previsioni inserite dagli operatori ambientali. Contiene tutti gli attributi richiesti dalla specifica i più significativi sono:

* *id\_area*, di tipo Integer, che è il codice identificativo dell'area d'interesse;
* *id\_centro*, di tipo Integer, che è il codice della stazione metereologica a cui fa appartenenza l'area d'interesse;
* *data*, di tipo Date, che indica la data del giorno in cui viene registrata la previsione

Questi tre attributi formano la chiave composta all’interno del database per la relazione Previsioni, con *id\_area* ed *id\_centro* che sono chiavi esterne per le rispettive relazioni.

Nella classe inoltre sono presenti i metodi get e set per ogni attributo ed anche il metodo toString

Gli attributi della classe sono:

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| Integer | id\_area | Codice identificativo dell'area d'interesse |
| Integer | id\_centro | Codice della stazione metereologica a cui fa appartenenza l'area d'interesse. |
| Date | data | Indica la data in cui è stata inserita la previsione |
| String | username | Indica l’operatore che ha inserito la previsione |
| Integer | vVento | Velocità del vento (km/h), suddivisa in fasce |
| Integer | pUmidita | Percentuale di Umidità, suddivisa in fasce |
| Integer | pressione | Livello della pressione espressa in hPa, suddivisa in fasce |
| Integer | temperatura | Livello della temperatura, espressa in °C, suddivisa in fasce |
| Integer | precipitazioni | Quantità di pioggia caduta, espressa in mm, espressa in fasce |
| Integer | aGhiacciai | Altitudine dei ghiacciai espressa in m, suddivisa in fasce |
| Integer | mGhiacciai | Massa dei ghiacciai espressa in kg, suddivisa in fasce |
| String | nVento | Note, facoltative, che aggiungo informazioni al valore vVento |
| String | nUmidita | Note, facoltative, che aggiungo informazioni al valore pUmidita |
| String | nPressione | Note, facoltative, che aggiungo informazioni al valore pressione |
| String | nTemperatura | Note, facoltative, che aggiungo informazioni al valore temperatura |
| String | nPrecipitazioni | Note, facoltative, che aggiungo informazioni al valore precipitazioni |
| String | nAGhiacciai | Note, facoltative, che aggiungo informazioni al valore aGhiacciai |
| String | nMGhiacciai | Note, facoltative, che aggiungo informazioni al valore mGhiacciai |

Nella classe non sono presenti metodi particolari oltre i classici operatori, get/set e il toString.

## JUser

JUser è la classe che si occupa della gestione degli operatori ambientali, contiene tutti gli attributi relativi all’utente, i più rilevanti sono:

* *id,* di tipo String, che è un codice univoco che identifica l’user
* *cf,* di tipo String, che viene generato automaticamente tramite l’uso di una libreria
* *geoname\_id,* di tipo Integer, che indica la stazione metereologica a cui è associato l'operatore
* *id\_areaselezionata**,* di tipo int, che indica l'ultima area di interesse di cui l'operatore ha aggiunto una previsione

Nella classe inoltre sono presenti i metodi get e set per ogni attributo ed anche il metodo toString Gli attributi della classe sono:

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| String | username | Indica il nome con cui l’operatore inserirà le previsioni e accederà al proprio account |
| String | nome | Nome dell’utente |
| String | cognome | Cognome dell’utente |
| String | id | Codice univoco che identifica l’user |
| String | password | sequenza di caratteri alfanumerici utilizzata per accedere in modo esclusivo |
| String | cf | Codice fiscale dell’utente |
| String | mail | Mail dell’utente |
| Integer | geoname\_id | Indica la stazione metereologica a cui è associato l'operatore |
| int | id\_areaselezionata | Indica l'ultima area di interesse di cui l'operatore ha aggiunto una previsione |

Nella classe non sono presenti metodi particolari oltre i classici operatori, get/set e il toString.

## JNazioni

JNazioni è la classe che si occupa della gestione delle nazioni contiene tutti gli attributi relativi alle nazioni, i più rilevanti sono:

* nome\_nazione, di tipo String, rappresenta il nome della nazione
* country\_code, di tipo String, rappresenta il codice della nazione

Nella classe inoltre sono presenti i metodi get e set per ogni attributo ed anche il metodo toString

Gli attributi della classe sono:

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| String | nome\_nazione | Indica il nome della nazione |
| String | country\_code | Indica il codice della nazione |

## DatiCondivisi

La classe DatiCondivisi implementa il Design Pattern, Singleton, che fornisce un'istanza condivisa per la gestione delle interazioni con il database.

Questa classe garantisce che esista una sola istanza di DatiCondivisi nell'intera applicazione.

Gli attributi presenti sono:

* instance, di tipo DatiCondivisi, statica, implementazione del Singleton
* gestore\_db, di tipo DBInterface, gestore del database

Il metodo presente è il getInstance, che permette di restituire l’istanza già creata, oppure ne crea una nuova se non esiste

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| Static DatiCondivisi | instance | Indica l’istanza creata |
| DBInterface | gestore\_db | Indica il gestore del database |
| Static DatiCondivisi | getInstance() | Ritorna l’istanza DatiCondivisi se esiste, altrimenti la crea |

# Interfaccia

## DBInterface

La classe DBInterface viene utilizzata nella comunicazione tra client e server e descrive tutti i metodi necessari alle varie operazioni (consultare il javadoc per maggiori informazioni), estende la classe Remote per gestire la maggior parte delle eccezioni.

|  |  |  |
| --- | --- | --- |
| Nome metodo | Parametri | Ritorna |
| loadAree\_interesse | String filtro\_nome  JCoordinate filtro\_coordinate  int filtro\_raggio  String id\_stazione  int id\_area | ArrayList<JAreaInteresse> |
| loadPrevisioni | String geoname\_id  int id\_area\_interesse, boolean dateFromFilter, Date dateFilter | ArrayList<JPrevisioni> |
| loadStazioni | String filtro\_id  String filtro\_nome  JCoordinate coordinate  Int raggio | ArrayList<JStazione> |
| loadNazioni |  | ArrayList<JNazione> |
| getStationGeonameIdfromWS | String cityName | ArrayList<JStazione> |
| chechkCodiceOperatore | String codice\_operatore | boolean |
| checkCodiceOperatoreUsed | String codice\_operatore | boolean |
| checkUserAlreadyExistsByUsername | String username | String |
| checkExistStazione | String geoname\_id  String nome | boolean |
| AddStazione | String geoname\_id  String nome  String country\_code  JCoordinate coordinate | boolean |
| AddUser | String nome  String cognome  String password  Integer geoname\_id  String codiceOperatore | String |
| AddAreaInteresse | String nome  String geoname\_id | boolean |
| checkAreeInteresse | String nome  String geoname\_id | boolean |
| AddPrevisione | Date data  Integer id\_area  String id\_centro  Integer username  String vVento  String pUmidita  String pressione  String temperatura  String precipitazioni  String aGhiacciai  String mGhiacciai  String nVento  String nUmidita  String nPRessione  String nTemperatura  String nPrecipitazioni  String nAGhiacciai  String nMGhiacciai | boolean |
| editPrevisione | Date data  Integer id\_area  String id\_centro  Integer username  String vVento  String pUmidita  String pressione  String temperatura  String precipitazioni  String aGhiacciai  String mGhiacciai  String nVento  String nUmidita  String nPRessione  String nTemperatura  String nPrecipitazioni  String nAGhiacciai  String nMGhiacciai | boolean |
| removePrevisione | Date data, Integer id\_area, String id\_centro | boolean |
| getUser | String user, String pass | boolean |

# Classi server

## DatabaseConnection

La classe DatabaseConnection, ha lo scopo di creare e gestire la connessione con il database, per questo l’unico attributo della classe è “*conn*” di tipo *Connection* ed appunto rappresenta la connessione al database. Inoltre, fornisce alcuni metodi che facilitano la scrittura delle query per la classe server, i metodi sono i seguenti:

* *executeQuery*, permette di eseguire una query all’interno database
* *executeUpdate*, permette di eseguire una INSERT, UPDATE o DELETE all’interno del database

Gli attributi della classe sono:

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| Connection | conn | Connesione al db |

I metodi principali, oltre al costruttore che stabilisce la connessione, della classe sono:

|  |  |  |  |
| --- | --- | --- | --- |
| Nome metodo | Parametri | Ritorna | Descrizione |
| executeQuery | String query  Object[] params  boolean hasParams | ResultSet | Esegue la query |
| executeUpdate | String query  Object[] params  boolean hasParams | int | Esegue l’update |

## Server

La classe Server implementa l’interfaccia “DBInterface” inoltre estende la classe UnicastRemoteObject per la gestione di RMI. Ha un unico attributo denominato “*db*” di tipo *DatabaseConnection* che rappresenta la connessione al database che viene utilizzata durante le operazioni di accesso ai dati. I metodi che riservano un minimo di attenzione sono:

* il *main* che Chiede le credenziali per l’accesso al db, e genera il registro e lo stub per gestire RMI.
* *checkCodiceOperatore* e *checkCodiceOperatoreUsed*, due metodi simili il primo verifica se il codice operatore esiste nel database mentre il secondo controlla se il codice è già in uso. Come precedentemente specificato il codice operatore è univoco per il singolo operatore e viene fornito da un’entità terza esclusivamente agli operatori climatici (questo fa si che un cittadino non possa registrarsi)

Gli attributi della classe sono:

|  |  |  |
| --- | --- | --- |
| Tipo | Nome attributo | Descrizione |
| DatabaseConnection | db | Connessione al database utilizzata per le operazioni di accesso ai dati |

I metodi principali della classe sono:

|  |  |
| --- | --- |
| Metodo | Descrizione/Scelte implementative |
| main | Chiede le credenziali per l’accesso al db, e genera il registro e lo stub per gestire RMI. |
| getStationGeonameIdfromWS | Utilizza le API fornite da GeoNames per prelevare la lista delle stazione |
| checkCodiceOperatore  checkCodiceOperatoreUsed | Controlla se esiste già un utente con il codice operatore che si vuole utilizzare per la registrazione. Il codice operatore è univoco per il singolo operatore e viene fornito da un’entità terza esclusivamente agli operatori climatici (questo fa si che un cittadino non possa reigstrarsi) |

# Classi grafiche

## admin\_panel

L’admin\_panel ha lo scopo di permettere una gestione delle aree di interesse e le sue previsioni. Permette di aggiungere, modificare ed eliminare le previsioni, oltre che aggiungere nuove aree di interesse.

L’utente potrà selezionare da una lista tutte le aree di interesse associate alla stazione metereologica a cui è associato e successivamente potrà modificare i valori tramite delle *JComboBox* ed aggiungere delle note nelle *JTextField*. Si potranno aggiungere le previsioni solo dal giorno corrente fino ai quattordici giorni successivi.

Inoltre, sempre in questa finestra visualizzare le previsioni già inserite per l’area d’interesse selezionata.

Permettiamo l’inserimento di previsioni dal giorno in cui si sta inserendo la previsione ai 14 giorni successivi per evitare un problema di sovrappopolamento e ad ogni area d’interesse può essere aggiunta una sola previsione per giorno, mentre non riteniamo utile poter inserire previsioni per mesi passati.

Questa classe grafiche fa uso di 3 metodi synchronized principali:

* removePrevisione()
* editPrevisione()
* addPrevisione

Il Database è costituito già da una serie di controlli che permettono l’atomicità e la gestione delle transazioni/lock, ma all’interno di questi metodi abbiamo voluto gestire l’acceso concorrenziale alle risorse condivise per evitare problemi di accesso simultaneo

## creaStazione

CreaStazione ha l’obiettivo di permettere l’inserimento di un nuovo centro di monitoraggio ad un utente, sia in fase di registrazione sia dopo aver effettuato l’accesso.

L’utente potrà inserire i dati relativi al centro metereologico all’interno di *JTextField* o *JComboBox* e successivamente schiacciando un *JButton*, aggiungerà la nuova stazione nel database.

La pagina creaStazione può essere aperta solamente dopo aver fatto il login oppure in fase di registrazione dopo aver inserito un codice operatore valido.

All’interno di questa schermata, una volta che verrà eseguita la ricerca di una stazione, tramite l’apposito campo, i caratteri verranno inviati al server, che si occuperà attraverso delle chiamate API di GeoNames, di fornire una lista di Stazioni ( Città ) che rispettano la ricerca

DI seguito la chiamata:

String url = "http://api.geonames.org/searchJSON?lang=en&username=gexebit147&maxRows=8&name\_equals=" + URLEncoder.*encode*(cityName, "UTF-8");

La url è composta dalle seguenti opzion:

1. BasePath:<http://api.geonames.org/searchJSON>?
   1. La base dell’url per richiamare l’API, formato JSON
2. lang
   1. Parametro utilizzato restituisce il json formatto in lingua inglese
3. username
   1. Parametro utilizzato per autenticarsi e utilizzare l’API
4. maxRows
   1. Parametro utilizzato per limitare i record restituiti
5. name\_equals
   1. Parametro utilizzato per ricercare il nome della Stazione, utilizziamo un metodo per codificare il valore di cityName nel formato della URL

## homepage

homepage è la pagina inziale del progetto per questo ha diverse funzionalità inserite al suo interno.

La principale è quella di fruire a chiunque, le varie stazioni ed aree d’interesse, quest’ultime sono visualizzate in una *JTable*, tramite delle *JTextField* sarà possibile effettuare una ricerca sia per il nome di un’area o di una stazione, la ricerca non è case sensitive e funzionerà anche con parte del nome restituendo tutti record contenete quella parte. È presente anche una ricerca basata sulle coordinate geografiche; in questo caso verranno restituiti i risultati più vicini alle coordinate fornite, in un raggio di 20 km.

## infoStazione

infoStazione ha come scopo quello di mostrare ad un qualsiasi utente tutti i dati di una stazione metereologica da loro scelta, mostrando anche tutte le aree d’interesse a lei associata.

## login

La schermata login ha come scopo quello di gestire il login e rimandare alla schermata di registrazione di un utente.

Permette ad un utente, già registrato, di autenticarsi dopo i dovuti controlli, nella sua area riservata

L’utente inserirà i suoi dati nelle *JTextField*.

## mostraPrevisioni

La schermata mostraPrevisioni permette di visualizzare tutte le previsioni per una determinata area d’interesse per uno specifico giorno. L’utente selezionerà la data da un *JCalendar* e vedrà i dati nella *JTable* a fianco. Si potranno visualizzare le date dal giorno corrente fino alle due settimane successive. Si possono visualizzare le previsioni dal giorno corrente fino a due settimane succesive.

## registrazione

La schermata registrazione consenti di inserire un nuovo operatore ambientale ad un utente; questa opzione non è disponibile a chiunque, bensì per potersi registrare bisognerà essere in possesso dal codice operatore fornito dalla propria istituzione che usufruisce del progetto come software di terza parte.

L’utente potrà inserire i dati relativi a sé stesso all’interno di *JTextField* o *JComboBox* e successivamente schiacciando un *JButton*, si inserirà il nuovo utente nel file relativo.

# Implementazione del database

## Operazioni di Insert

## Query

## Delete

## Progettazione Logica

# Errori noti

Nella finestra “homepage” per far sì che la ricerca per nome vada a buon fine bisogna scrivere il nome corretto compresi gli accenti, se quest’ultimi non sono presenti verrà restituita una tupla vuota che non farà nulla.

# Sitografia

## codice-fiscale-java-master - matteocastagnaro

<https://github.com/matteocastagnaro/codice-fiscale-java/blob/master/jar/codice-fiscale.jar?raw=true>

## JCalendar - Kai Tödter

<https://toedter.com/jcalendar/>

## JDK 17

[https://www.oracle.com/it/java/technologies/downloads/#java17](https://www.oracle.com/it/java/technologies/downloads/%23java17)

## PostgresSQL

<https://www.postgresql.org/docs/current/index.html>

## PostgresSQL Driver

<https://jdbc.postgresql.org/changelogs/2024-03-14-42.7.3-release/>

## pgAdmin 4

<https://www.pgadmin.org/download/pgadmin-4-windows/>

## DBeaver

<https://dbeaver.io/>

## GSON

<https://repo.maven.apache.org/maven2/com/google/code/gson/gson/2.8.8/>

## GeoNames

<https://www.geonames.org/>

## IntelliJ

<https://www.jetbrains.com/idea/>

## Apache NetBeans

<https://netbeans.apache.org/front/main/index.html>

## Clever Cloud

<https://www.clever-cloud.com/>

## GanttProject

<https://www.ganttproject.biz/>