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](#_Toc168738568)

[Strumenti - linguaggi – SO utilizzati 3](#_Toc168738569)

[Librerie esterne ed API 4](#_Toc168738570)

[Codice fiscale 4](#_Toc168738571)

[JCalendar 4](#_Toc168738572)

[GSON 4](#_Toc168738573)

[POSTGRESQL 5](#_Toc168738574)

[AbsoluteLayout 5](#_Toc168738575)

[Geonames API 5](#_Toc168738576)

[Processo di sviluppo 5](#_Toc168738577)

[Principali milestone 5](#_Toc168738578)

[Suddivisione dei lavori 6](#_Toc168738579)

[Progettazione del database 7](#_Toc168738580)

[Schema concettuale 7](#_Toc168738581)

[Vincoli d’integrità 8](#_Toc168738582)

[Scelte Progettuali 8](#_Toc168738583)

[Progettazione Logica 8](#_Toc168738584)

[Traduzione 8](#_Toc168738585)

[Progettazione pratica 8](#_Toc168738586)

[Creazione database 8](#_Toc168738587)

[Creazione tabelle 8](#_Toc168738588)

[Progettazione del software 8](#_Toc168738589)

[Struttura del sistema 8](#_Toc168738590)

[Classi Logiche 8](#_Toc168738591)

[Interfaccia 8](#_Toc168738592)

[Classi Server 8](#_Toc168738593)

[Classi Grafiche 9](#_Toc168738594)

[Class Diagram 10](#_Toc168738595)

[Sequence Diagram 10](#_Toc168738596)

[Implementazione del sotfware 11](#_Toc168738597)

[Classi logiche 11](#_Toc168738598)

[JAreaInteresse 11](#_Toc168738599)

[JCoordinate 11](#_Toc168738600)

[JStazione 11](#_Toc168738601)

[JPrevisioni 11](#_Toc168738602)

[JUser 11](#_Toc168738603)

[JNazioni 12](#_Toc168738604)

[DatiCondivisi 12](#_Toc168738605)

[Interfaccia 12](#_Toc168738606)

[DBInterface 12](#_Toc168738607)

[Classi server 14](#_Toc168738608)

[DatabaseConnection 14](#_Toc168738609)

[Server 14](#_Toc168738610)

[Classi grafiche 15](#_Toc168738611)

[admin\_panel 15](#_Toc168738612)

[Complessità 15](#_Toc168738613)

[creaStazione 15](#_Toc168738614)

[Complessità 16](#_Toc168738615)

[homepage 16](#_Toc168738616)

[Complessità 16](#_Toc168738617)

[infoStazione 16](#_Toc168738618)

[Complessità 16](#_Toc168738619)

[login 16](#_Toc168738620)

[Complessità 17](#_Toc168738621)

[mostraPrevisioni 17](#_Toc168738622)

[Complessità 17](#_Toc168738623)

[registrazione 17](#_Toc168738624)

[Complessità 17](#_Toc168738625)

[Implementazione del database 18](#_Toc168738626)

[Operazioni di Insert 18](#_Toc168738627)

[Query 18](#_Toc168738628)

[Delete 18](#_Toc168738629)

[Progettazione Logica 18](#_Toc168738630)

[Errori noti 18](#_Toc168738631)

[Sitografia 18](#_Toc168738632)

[codice-fiscale-java-master - matteocastagnaro 18](#_Toc168738633)

[JCalendar - Kai Tödter 18](#_Toc168738634)

[JDK 17 18](#_Toc168738635)

[PostgresSQL 18](#_Toc168738636)

[PostgresSQL Driver 18](#_Toc168738637)

[pgAdmin 4 18](#_Toc168738638)

[DBever 19](#_Toc168738639)

[GSON 19](#_Toc168738640)

[GeoNames 19](#_Toc168738641)

[IntelliJ 19](#_Toc168738642)

[Apache NetBeans 19](#_Toc168738643)

# 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 e aggiungerle.

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

## Strumenti - linguaggi – SO utilizzati

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

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

Linguaggi utilizzati:

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

Librerie esterne:

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

API:

* GeoNames

IDE:

* IntelliJ
* Apache Netbeans
* 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 come se fossero locali, 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.

L'applicazione lato client e server è scritta in Java, con l'utilizzo di SQL per le operazioni di database e JSON per la manipolazione e lo scambio di dati strutturati.

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

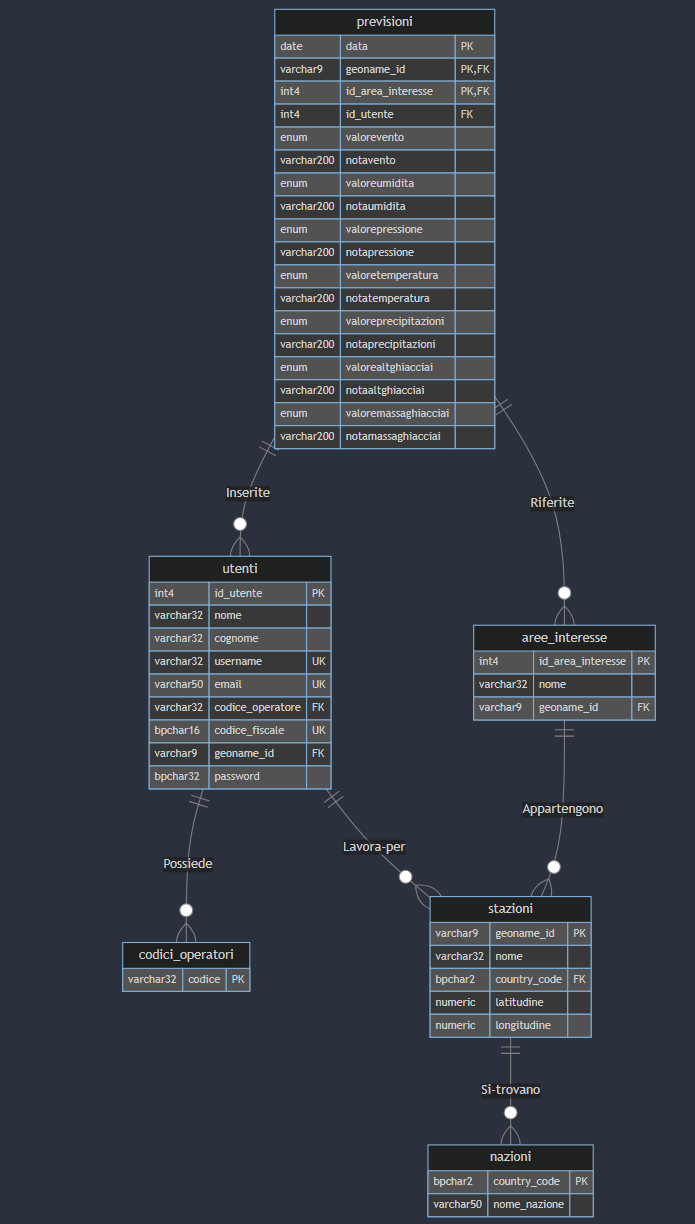
* Connessione al DB;
* DBInterface;
* Server con RMI;
* Modificare homepage per RMI;
* Adattamento codice client;
* Adattamento grafica;

## 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

La relazione codici\_operatori è stata introdotta per far si che venisse rispettata la condizione che solo gli operatori metereologici (**non ricordo il nome giusto**) potessero registrarsi nel sistema per poi permettergli di gestire le stazioni, le aree d’interesse e le previsioni. Nella nostra implementazione abbiamo ipotizzato che un operatore richieda ad un’entità terza il codice che poi gli viene fornito per registrarsi trovandolo già inserito nel db.

### Scelte Progettuali

## Progettazione Logica

### Traduzione

## Progettazione pratica

### Creazione database

### Creazione tabelle

# 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

…..

# 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

Per la descrizione delle classi grafiche non entreremo nel dettaglio dei metodi, perché parte del progetto facoltativa, bensì descriveremo l’obbiettivo di ogni classe e la complessità generale di essa.

## admin\_panel

admin\_panel ha come scopo quello di poter far inserire una nuova previsione.

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 si potranno aggiungere delle nuove aree d’interesse e 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.

### Complessità

Le complessità presenti nei metodi di questa classe sono principalmente O(1), l’unica eccezione è il metodo aggiornaTabella() che dovendo scorrere la lista di tutte le previsioni avrà una complessità O(n+m) che dipendono dalla dimensione del file in cui sono salvate le previsioni e dalla dimensione della lista.

(**togliamo le complessità o le inseriamo generiche anche nelle classi del server e quelle logiche)**

## creaStazione

creaStazione ha come scopo quello di poter far inserire 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.

### Complessità

Le complessità presenti nei metodi di questa classe sono principalmente O(1), l’unica eccezione è il metodo che salva la stazione nel file dove la complessità sarà O(n) con n che indica la quantità di caratteri nella stringa passata.

## 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 basta sulle coordinate geografiche; in questo caso verranno dati i risultati più vicini alle coordinate fornite (in un raggio di 20 km).

### Complessità

Le complessità presenti nei metodi di questa classe possono variare in due casi, nel caso migliore sono O(n), in quei metodi dove bisogna aggiornare l’elemento grafico della tabella con n che è la dimensione della lista. Mentre il caso peggiore è di O(n+m) dove bisogna non solo aggiornare la tabella ma anche filtrare i campi voluti.

## 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.

### Complessità

Le complessità presenti nei metodi di questa classe sono principalmente O(1), l’unica eccezione è il metodo che inizializza i componenti dove viene richiamata il metodo *getAllAreaInteresse()*  la quale complessità è O(n + m), dipende dalla dimensione del file delle aree di interesse e dalle dimensioni della lista.

## login

login ha come scopo quello di far accedere un utente già registrato nella sua area riservata con i dovuti controlli.

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

### Complessità

Le complessità presenti nei metodi di questa classe sono principalmente O(n+m), perché i metodi caricheranno i file su cui sono salvati gli utenti in una lista su cui poi avverranno i controlli.

## mostraPrevisioni

mostraPrevisioni ha come scopo quello 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 in cui si è fino a due settimane.

### Complessità

Le complessità presenti nei metodi di questa classe sono principalmente O(n+m), perché i metodi caricheranno i file su cui sono salvati gli utenti in una lista su cui poi avverranno i controlli.

## registrazione

registrazione ha come scopo quello di poter far 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.

### Complessità

Le complessità presenti nei metodi di questa classe sono al massimo O(n+m+k), perché nei vari metodi richiamati i casi peggiori sono quelli dei metodi *registraUtente()* e *creaListaStazioni()* che hanno la complessità sopra citata, quest’ultima è dovuta dai file e dalle liste che vanno visualizzate.

# 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/>

## DBever

<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>