MeteoApp

DOCUMENTO DI ANALISI

L'applicazione MeteoApp permette di consultare i dati meteorologici di una città.

L'utente può consultare:

- Condizioni meteo correnti (temperatura, umidità e una descrizione)
- Previsioni meteo nei cinque giorni successivi a quello in cui viene effettuata la richiesta.
- Grafico dell'andamento delle principali grandezze (temperatura, umidità e pioggia) durante uno dei cinque giorni previsti (il giorno è selezionabile dall'utente).

Dopo aver inserito il nome di una città e premuto il tasto "Cerca", l'utente vedrà una schermata come questa:

Lo scenario che si prospetta nell'uso di questa applicazione è il seguente:

1. Il sistema mostra nel campo Utente il nome dell'utente
2. L'utente inserisce il nome di una città nel campo Cerca Città
3. IF l'utente preme Cerca
   3.1. IF la città cercata è presente nel sistema
      3.1.1. Il sistema visualizza, nella sezione Meteo attuale: nome città, temperatura, umidità e descrizione
      3.1.2. Il sistema visualizza la sezione Meteo Previsto
      3.1.3. Per ciascuno dei 5 giorni successivi a quello in cui è stata effettuata la ricerca
         3.1.3.1. Il sistema visualizza icona, descrizione, temperatura minima, massima
      3.1.4. Il sistema visualizza nella sezione Grafico il grafico della temperatura del primo Giorno presente in Meteo previsto
      3.1.5. IF l'utente seleziona un Giorno in Meteo Previsto
         3.1.5.1. Il sistema mostra, nella sezione Grafico, l'andamento della temperatura per il Giorno selezionato
         3.1.5.2. IF l'utente preme su Temperatura
            3.1.5.2.1. Il sistema mostra, nella sezione Grafico, l'andamento della temperatura
         3.1.5.3. IF l'utente preme su Pioggia
            3.1.5.3.1. Il sistema mostra, nella sezione Grafico, l'andamento della pioggia
         3.1.5.4. IF l'utente preme su Umidità
            3.1.5.4.1. Il sistema mostra, nella sezione Grafico, l'andamento dell'umidità
      3.1.6. IF l'utente preme sul tasto con il simbolo "+" la città cercata viene inserita nella lista delle città preferite.
4. Il sistema carica nella sezione Città preferite l'elenco delle città preferite dell'utente
5. FOR EACH città in Città preferite
   5.1. Il sistema visualizza il nome della città
6. IF l'utente preme sul nome di una delle città preferite, i dati relativi a quella città vengono mostrati nelle varie sezioni.
7. IF l'utente seleziona una città e preme sul pulsante Rimuovi quella città viene rimossa dalla lista delle città preferite.

Componenti principali di JAVA I/0

File di configurazione locale in XML

All'avvio, l'applicazione legge dal file xml i seguenti parametri:

- Nome dell'utente dell'applicazione
- Nome utente e password dell'archivio
- Indirizzo IP e porta del server di log
- Unità di misura della temperatura e dell'umidità
- Numero massimo di città preferite che l'utente può salvare in una sessione

Cache locale degli input

L'applicazione memorizza in un file binario locale le seguenti informazioni:

- L'ultima città cercata
- Città preferite indicate dall'utente
- Il numero dell'ultima previsione selezionata nella sezione "Meteo previsto"
- La tipologia di grafico che l'utente sta guardando (temperatura, umidità o pioggia)

Archivio

L'archivio dati contiene:

- La lista delle città presenti
- Il meteo attuale per ogni città presente
- Le previsioni per i cinque giorni successivi per ogni città presente

L'archivio viene interrogato ogni volta in cui l'utente cerca una nuova città ed ogni volta in cui seleziona una previsione.

File di Log remoto in XML

L'applicazione invia al server di log una riga per ciascuno dei seguenti eventi:

- Avvio dell'applicazione
- Pressione di uno dei pulsanti (Cerca, +, temperatura, umidita, pioggia, rimuovi)
- Selezione di una previsione nella sezione Meteo Previsto
- Chiusura dell'applicazione

Ogni riga di log inviata al server è composta da: *nome dell'applicazione*, *IP del client*, *data* e *ora* dell'evento ed etichetta associata all'*evento*.

DOCUMENTO DI PROGETTO

Di seguito è riportato un elenco delle classi che compongono il progetto, per ciascuna è riportata una descrizione delle funzionalità e delle responsabilità.

- **ElencoPrevisioniGiornaliere** (estende TableView): si occupa di rappresentare la sezione "Meteo previsto" dell'interfaccia grafica. Ne viene utilizzata un'istanza nella classe principale MeteoApp. Contiene una tabella con la lista delle previsioni meteo per i 5 giorni successivi a quello in cui l'utente effettua la richiesta.

- **ElencoCittaPreferite** (estende TableView): gestisce la visualizzazione della sezione con la lista delle città preferite. Più precisamente contiene una tabella in cui sono elencate tali città.

- **CittaPreferita**: un oggetto di questa classe rappresenta una città preferita all'interno della tabella implementata dalla classe ElencoCittaPreferite.

- **AndamentoOrarioDelParametro** (estende LineChart): questa classe si occupa della visualizzazione della sezione "Grafico". Una sua istanza è presente nella classe MeteoApp (controller).

- **MeteoApp** (estende Application): è la classe principale dell'applicazione. Si occupa di costruire la finestra e di avviare l'applicazione mediante il metodo start. Svolge inoltre il ruolo di controller, cioè riceve tutte le richieste provenienti dagli oggetti dell'interfaccia (campi di testo, pulsanti e tabelle), recupera i dati dalle classi del back-end e restituisce il risultato.

- **PrevisioneOraria:** un oggetto di questa classe rappresenta una previsione meteo riguardante una città su un intervallo di tre ore (Il sito da cui sono stati scaricati i dati fornisce una previsione ogni tre ore). Una previsione oraria è caratterizzata da un istante iniziale, un istante finale, una temperatura minima e massima e altri parametri caratteristici dell'intervallo che rappresenta.

- **PrevisioneGiornaliera:** un oggetto di questa classe è costituito da un insieme di oggetti di classe PrevisioneOraria e rappresenta una previsione meteo su un'intera giornata. La previsione meteo sull'intera giornata viene determinata analizzando le previsioni orarie che la compongono.

- **ParametriConfigurazioneXML** (implementa Serializable): si occupa della lettura, validazione e deserializzazione del file di configurazione xml. Una sua istanza è presente all'interno della classe MeteoApp. La classe è costituita da due metodi, uno che legge ed uno che valida il file di configurazione xml usando un file schema e deserializza il file in un oggetto della stessa classe ConfigXML.

- **LogEventiGUI:** si occupa della costruzione, serializzazione ed invio di file xml al server di log remoto. Una sua istanza è presente nella classe MeteoApp e consente di inviare un file di log ogni volta che si genera un evento in una delle sezioni dell'interfaccia grafica.

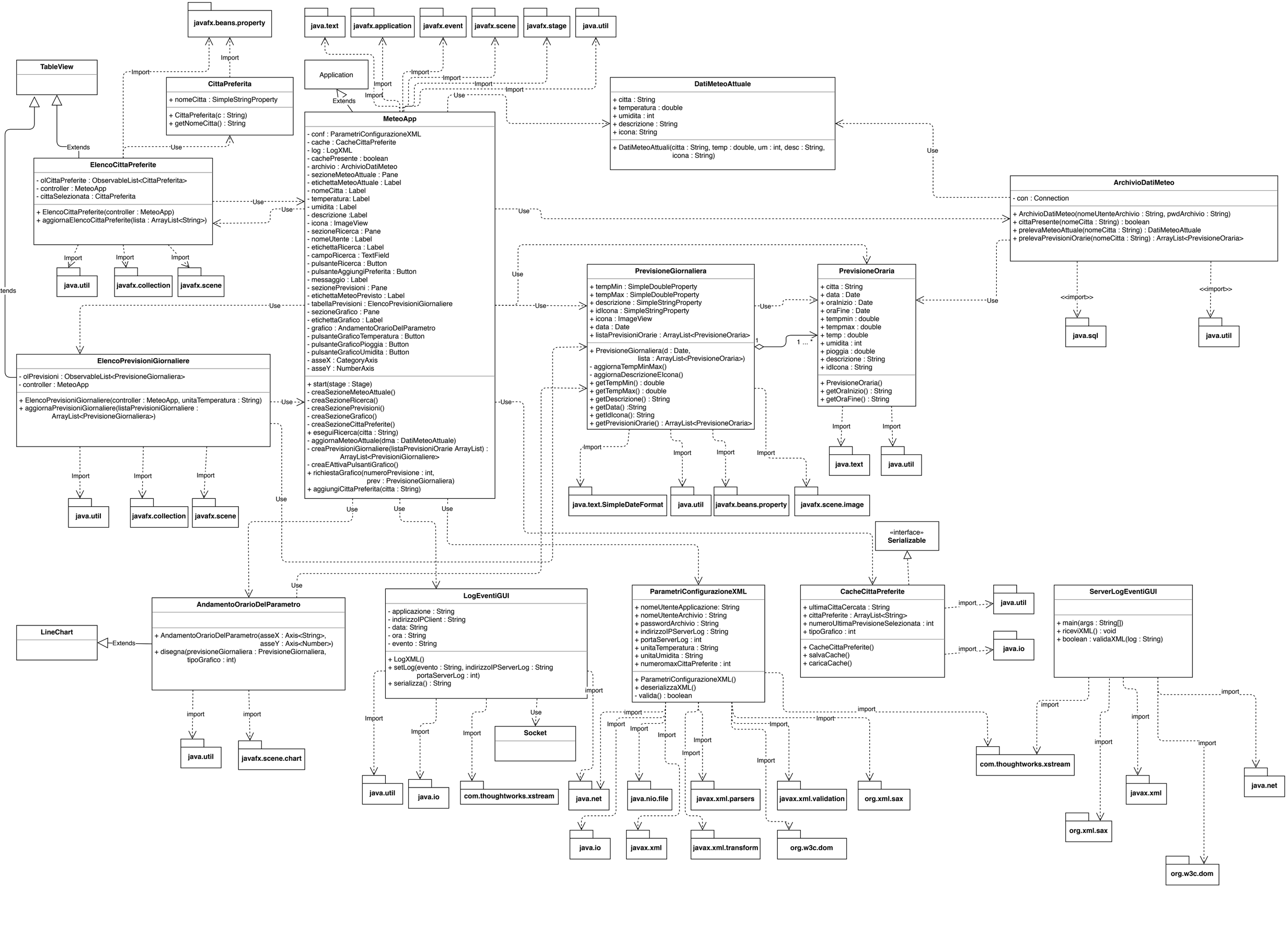
- **CacheCittaPreferite** (implementa Serializable): permette di gestire il file di cache dell'applicazione. Una sua istanza è presente all'interno della classe MeteoApp. Contiene un metodo che permette di salvare i dati nel file di cache ed uno che permette di ripristinare il file salvato precedentemente. Se il file cache non è presente, il metodo caricaCache() stampa un messaggio di errore e restituisce false ma non interrompe l'esecuzione dell'applicazione. Sarà il metodo salvaCache() a creare il file.

- **ArchivioDatiMeteo:** questa classe consente all'applicazione di interfacciarsi con il DBMS. Stabilisce una connessione ed esegue le interrogazioni richieste dalla classe MeteoApp. Ha un metodo che permette di verificare la presenza di una città all'interno dell'archivio ed altri due metodi per la selezione di tutti i dati meteorologici relativi ad una città.

- **ServerLogEventiGUI:** rappresenta il server di ricezione dei file di log XML. Ogni volta che si verifica un evento nella GUI, riceve un file XML con informazioni riguardanti l'evento. Lo valida e lo archivia in un file di log generale.

- **DatiMeteoAttuale:** memorizza i dati riguardanti il meteo attuale su una citta. Viene utilizzato per lo scambio di informazioni tra la classe ArchivioDatiMeteo e la classe principale MeteoApp.

Nella pagina seguente è riportato il **diagramma delle classi**.


javafx.beans.property
java.text
javafx.application
javafx.event
javafx.scene
javafx.stage
java.util
TableView
CittaPreferita
+ nomeCitta : SimpleStringProperty
+ CittaPreferita(c : String)
+ getNomeCitta() : String
Application
Import
Extends
Use
ElencoCittaPreferite
- olCittaPreferite : ObservableList<CittaPreferita>
- controller : MeteoApp
- cittaSelezionata : CittaPreferita
+ ElencoCittaPreferite(controller : MeteoApp)
+ aggiornaElencoCittaPreferite(lista : ArrayList<String>)
java.util
javafx.collection
javafx.scene
MeteoApp
- conf : ParametriConfigurazioneXML
- cache : CacheCittaPreferite
- log : LogXML
- cachePresente : boolean
- archivio : ArchivioDatiMeteo
- sezioneMeteoAttuale : Pane
- etichettaMeteoAttuale : Label
- nomeCitta : Label
- temperatura: Label
- umidita : Label
- descrizione :Label
- icona : ImageView
- sezioneRicerca : Pane
- nomeUtente : Label
- etichettaRicerca : Label
- campoRicerca : TextField
- pulsanteRicerca : Button
- pulsanteAggiungiPreferita : Button
- messaggio : Label
- sezionePrevisioni : Pane
- etichettaMeteoPrevisto : Label
- tabellaPrevisioni : ElencoPrevisioniGiornaliere
- sezioneGrafico : Pane
- etichettaGrafico : Label
- grafico : AndamentoOrarioDelParametro
- pulsanteGraficoTemperatura : Button
- pulsanteGraficoPioggia : Button
- pulsanteGraficoUmidita : Button
- asseX : CategoryAxis
- asseY : NumberAxis
+ start(stage : Stage)
- creaSezioneMeteoAttuale()
- creaSezioneRicerca()
- creaSezionePrevisioni()
- creaSezioneGrafico()
- creaSezioneCittaPreferite()
+ eseguiRicerca(citta : String)
- aggiornaMeteoAttuale(dma : DatiMeteoAttuale)
- creaPrevisioniGiornaliere(listaPrevisioniOrarie ArrayList) : ArrayList<PrevisioniGiornaliere>
- creaEAttivaPulsantiGrafico()
+ richiestaGrafico(numeroPrevisione : int, prev : PrevisioneGiornaliera)
+ aggiungiCittaPreferita(citta : String)
DatiMeteoAttuale
+ citta : String
+ temperatura : double
+ umidita : int
+ descrizione : String
+ icona: String
+ DatiMeteoAttuali(citta : String, temp : double, um : int, desc : String, icona : String)
ArchivioDatiMeteo
- con : Connection
+ ArchivioDatiMeteo(nomeUtenteArchivio : String, pwdArchivio : String)
+ cittaPresente(nomeCitta : String) : boolean
+ prelevaMeteoAttuale(nomeCitta : String) : DatiMeteoAttuale
+ prelevaPrevisioniOrarie(nomeCitta : String) : ArrayList<PrevisioneOraria>
<<import>>
java.sql
java.util
ElencoPrevisioniGiornaliere
- olPrevisioni : ObservableList<PrevisioneGiornaliera>
- controller : MeteoApp
+ ElencoPrevisioniGiornaliere(controller : MeteoApp, unitaTemperatura : String)
+ aggiornaPrevisioniGiornaliere(listaPrevisioniGiornaliere : ArrayList<PrevisioneGiornaliera>)
PrevisioneGiornaliera
+ tempMin : SimpleDoubleProperty
+ tempMax : SimpleDoubleProperty
+ descrizione : SimpleStringProperty
+ idIcona : SimpleStringProperty
+ icona : ImageView
+ data : Date
+ listaPrevisioniOrarie : ArrayList<PrevisioneOraria>
+ PrevisioneGiornaliera(d : Date, lista : ArrayList<PrevisioneOraria>)
- aggiornaTempMinMax()
- aggiornaDescrizioneEIcona()
+ getTempMin() : double
+ getTempMax() : double
+ getDescrizione() : String
+ getData() :String
+ getIdIcona(): String
+ getPrevisioniOrarie() : ArrayList<PrevisioneOraria>
1
1 ... *
PrevisioneOraria
+ citta : String
+ data : Date
+ oraInizio : Date
+ oraFine : Date
+ tempmin : double
+ tempmax : double
+ temp : double
+ umidita : int
+ pioggia : double
+ descrizione : String
+ idIcona : String
+ PrevisioneOraria()
+ getOraInizio() : String
+ getOraFine() : String
java.text
java.util
java.text.SimpleDateFormat
java.util
javafx.beans.property
javafx.scene.image
java.util
javafx.collection
javafx.scene
«interface»
Serializable
AndamentoOrarioDelParametro
+ AndamentoOrarioDelParametro(asseX : Axis<String>, asseY : Axis<Number>)
+ disegna(previsioneGiornaliera : PrevisioneGiornaliera, tipoGrafico : int)
LineChart
Extends
import
java.util
javafx.scene.chart
LogEventiGUI
- applicazione : String
- indirizzoIPClient : String
- data: String
- ora : String
- evento : String
+ LogXML()
+ setLog(evento : String, indirizzoIPServerLog : String portaServerLog : int)
+ serializza() : String
Socket
java.util
java.io
com.thoughtworks.xstream
ParametriConfigurazioneXML
+ nomeUtenteApplicazione: String
+ nomeUtenteArchivio : String
+ passwordArchivio : String
+ indirizzoIPServerLog : String
+ portaServerLog : int
+ unitaTemperatura : String
+ unitaUmidita : String
+ numeromaxCittaPreferite : int
+ ParametriConfigurazioneXML()
+ deserializzaXML()
- valida() : boolean
java.net
java.nio.file
javax.xml.parsers
javax.xml.validation
org.xml.sax
java.io
javax.xml
javax.xml.transform
org.w3c.dom
CacheCittaPreferite
+ ultimaCittaCercata : String
+ cittaPreferite : ArrayList<String>
+ numeroUltimaPrevisioneSelezionata : int
+ tipoGrafico : int
+ CacheCittaPreferite()
+ salvaCache()
+ caricaCache()
java.util
java.io
ServerLogEventiGUI
+ main(args : String[])
+ riceviXML() : void
+ boolean : validaXML(log : String)
com.thoughtworks.xstream
org.xml.sax
javax.xml
java.net
org.w3c.dom

1) All'avvio dell'applicazione viene caricata l'interfaccia grafica, che si presenta nel modo seguente:

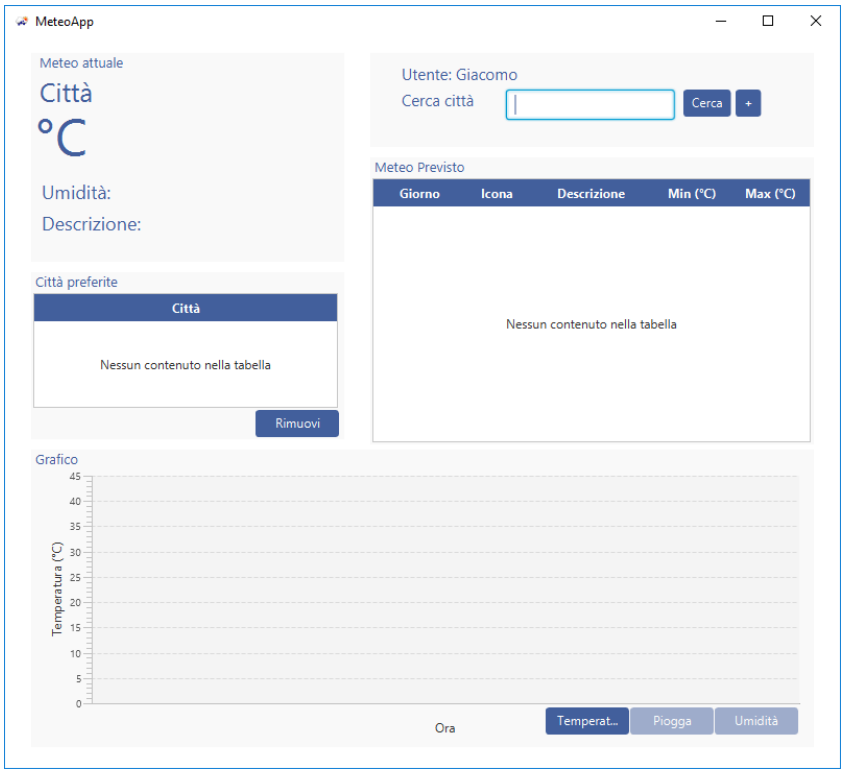


Figura 1

2) L'utente inserisce all'interno del campo "Cerca città" il nome della città interessata dopodiché preme il tasto "cerca".

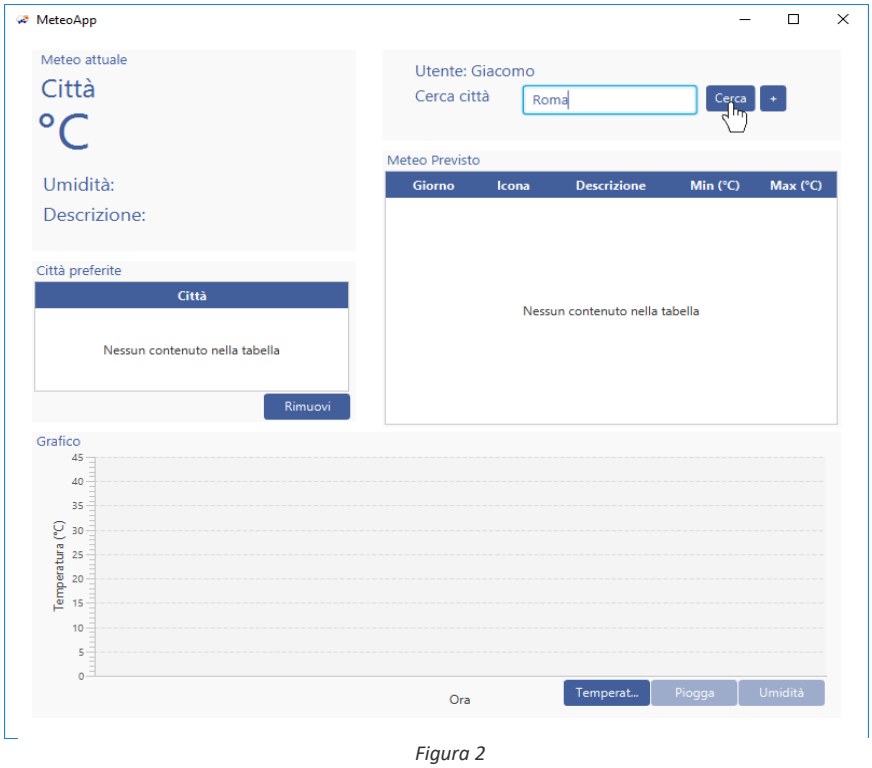


Figura 2

3) Se la città cercata è presente nell'archivio allora l'utente visualizza le sue condizioni meteo attuali nella sezione "Meteo attuale" e una lista di previsioni meteo nella sezione "Meteo previsto" (figura 3)

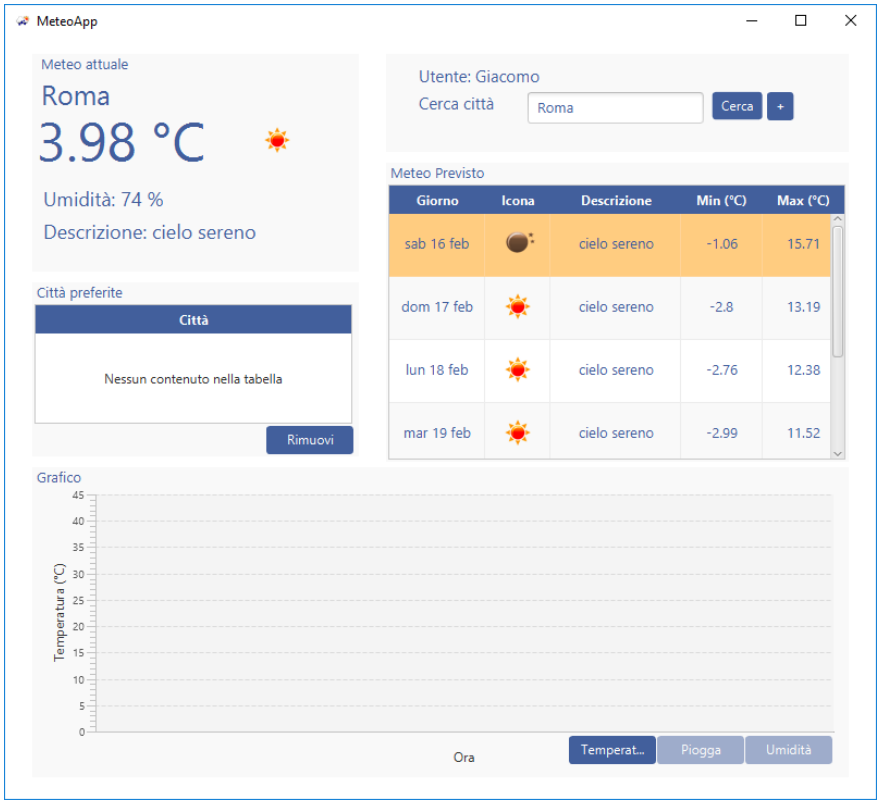


Figura 3

Se invece la città cercata non è presente nell'archivio allora viene mostrato un messaggio di errore (figura 4)

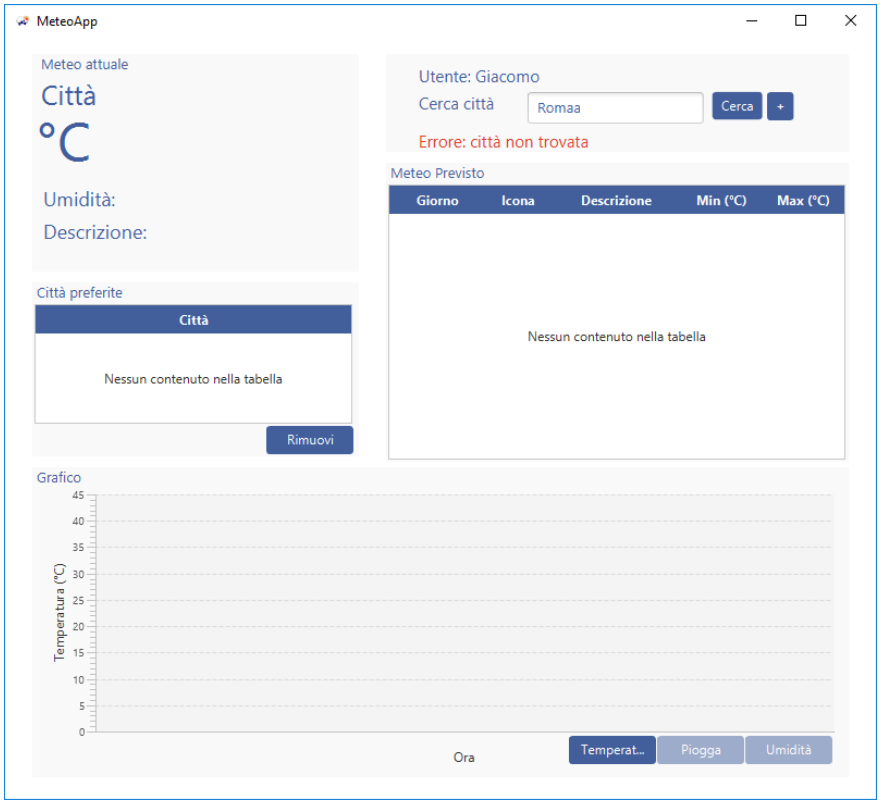


Figura 4

A questo punto l'utente può consultare le previsioni all'interno della sezione Meteo previsto e selezionandone una può visualizzare un grafico giornaliero (di temperatura, pioggia o umidità) nella sezione sottostante (figura 5)

Figura 5

Una volta selezionata una previsione, l'utente può scegliere il tipo di grafico da visualizzare cliccando sui tasti (Temperatura, Pioggia e Umidità) posti in basso a destra (figure 6 e 7)

Figura 6

Figura 7

L’utente può decidere di inserire una città tra le sue città preferite cliccando sul tasto “+” che si trova accanto al tasto “Cerca”. Durante una sessione, l’utente può indicare un numero massimo di città preferite. Il numero massimo viene prelevato dal file di configurazione xml. Quando l’utente clicca sul tasto “+”, la città, se non è già presente, viene salvata all’interno della lista delle città preferite. Tale lista viene mostrata nella sezione “Città preferite”.

Figura 8 – Inserimento città preferita

Figura 9 – Numero massimo di città preferite raggiunto

6) L’utente può accedere direttamente ai dati meteo di una città preferita cliccando sul suo nome all’interno della sezione “Città preferite” come mostrato in figura 11.

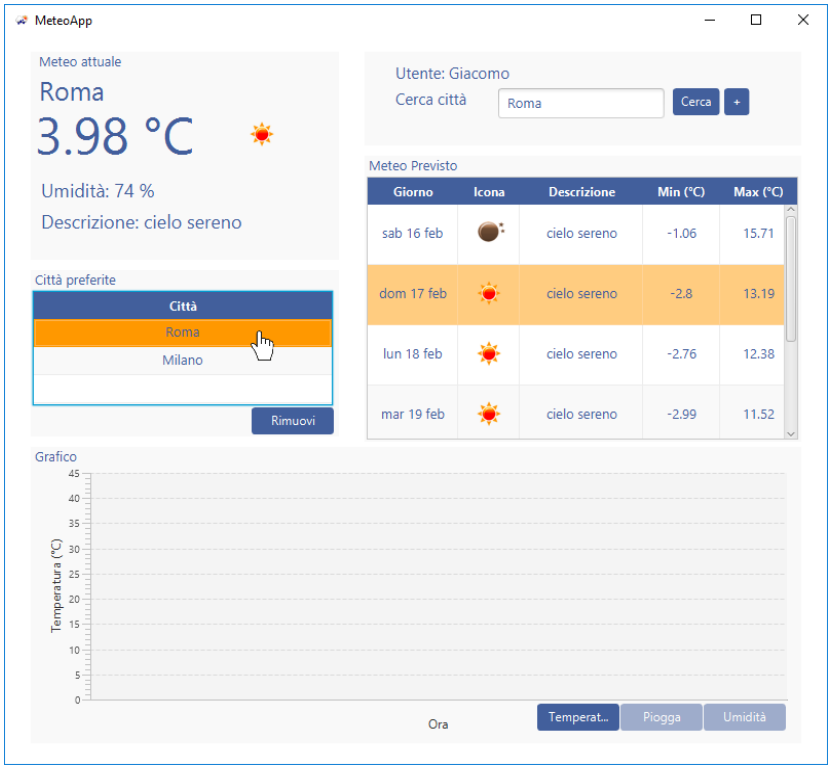


Figura 10 – Accesso rapido alle città preferite

7) L’utente può rimuovere una città preferita selezionando il nome della città all’interno della sezione “Città preferite” e premendo il tasto “Rimuovi”.

Figura 11 – Rimozione città preferita

Quando l'utente chiude l'applicazione, la classe CacheCittaPreferite si occupa di salvare in un file binario l'ultima città cercata, l'ultima previsione selezionata, il tipo di grafico visualizzato e la lista delle città preferite dell'utente. Alla riapertura dell'applicazione la classe CacheCittaPreferite ripristina le informazioni salvate e l'utente visualizzerà nel campo di ricerca il nome dell'ultima città cercata e nella sezione "Città preferite" le sue città preferite (se ce ne sono) come mostrato in figura 12

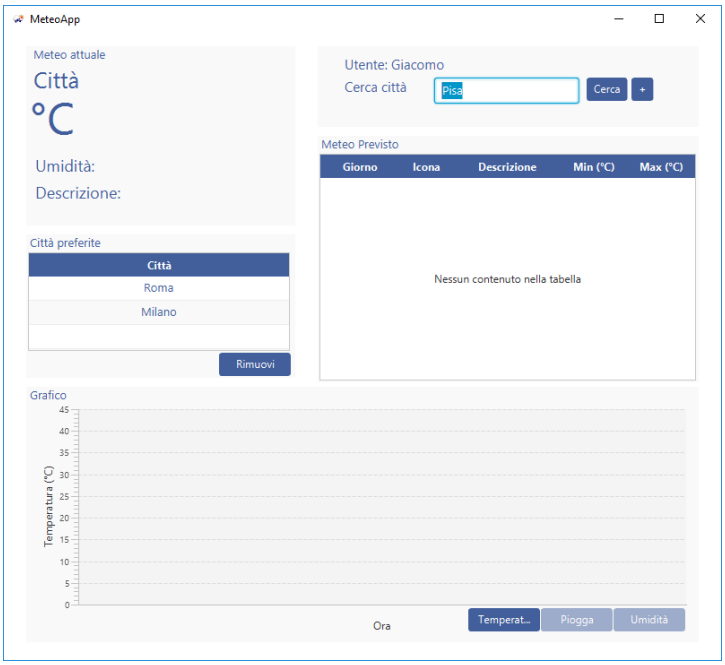


Figura 12 – Avvio applicazione con dati ripristinati dalla cache

Gli eventi che si verificano all'interno dell'applicazione (avvio, terminazione, pressione pulsanti, …) vengono accompagnati dall'invio ad un server di log (classe ServerLogEventiGUI) di un file xml tramite la classe LogEventiGUI. Il server di log è continuamente in ascolto. Ogni volta che arriva un nuovo dato, lo riceve, lo valida e se la validazione ha esito positivo, stampa a video un messaggio di conferma e lo aggiunge ad un file di log complessivo chiamato log.xml.

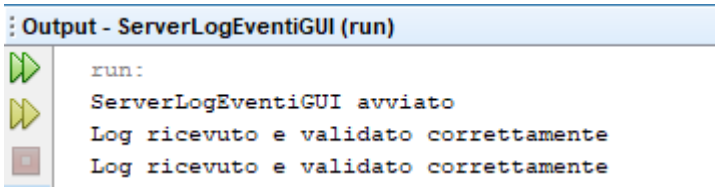


Figura 13 – Console del server di log

Schermate di un editor di testo riguardanti il file di log (log.xml) generato dal server di log ed il file schema (schemaLog.xsd) usato per la validazione.

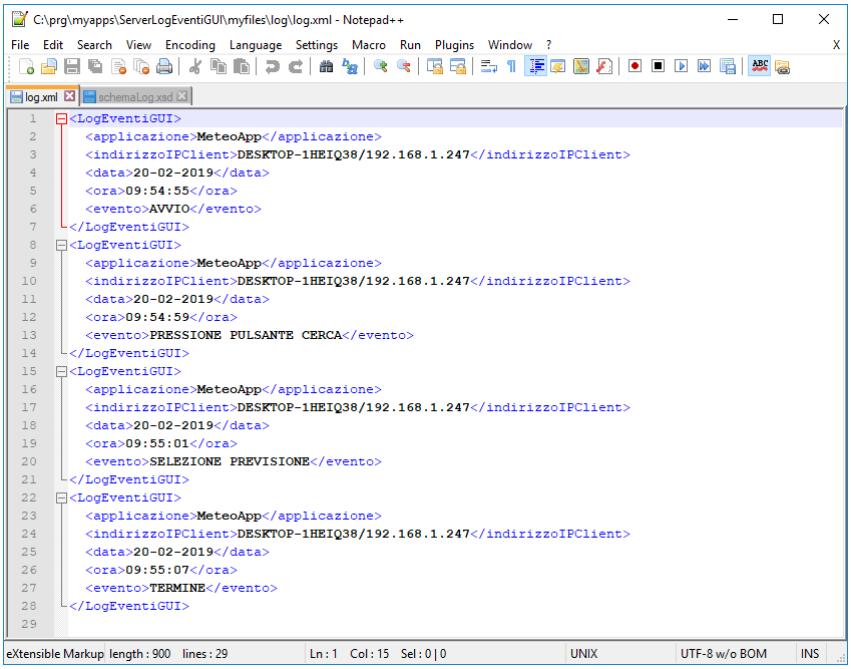


Figura 14 – File log.xml (schermata editor notepad++)

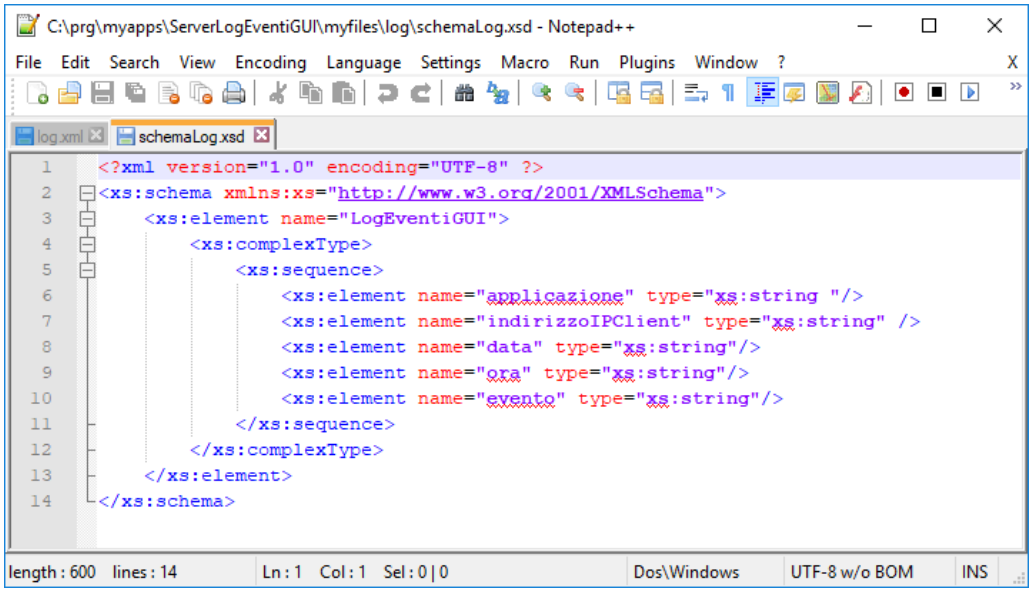


Figura 15 – File schemaLog.xsd (schermata editor notepad++)

Di seguito riporto due schermate raffiguranti il file di dump del database (archiviometeoapp) utilizzato dall'applicazione per memorizzare le condizioni meteo attuali e le previsioni meteo.

dump_archiviometeoapp.sql

```
CREATE DATABASE  IF NOT EXISTS `archiviometeoapp` /*!40100 DEFAULT CHARACTER SET latin1 */;
USE `archiviometeoapp`;
-- MySQL dump 10.13  Distrib 5.6.17, for Win32 (x86)
--
-- Host: 127.0.0.1    Database: archiviometeoapp
-- ------------------------------------------------------
-- Server version   5.6.21

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `meteoattuale`
--

DROP TABLE IF EXISTS `meteoattuale`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `meteoattuale` (
  `citta` varchar(100) NOT NULL,
  `temperatura` double DEFAULT NULL,
  `umidita` int(11) DEFAULT NULL,
  `descrizione` varchar(45) DEFAULT NULL,
  `icona` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`citta`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `meteoattuale`
--

LOCK TABLES `meteoattuale` WRITE;
/*!40000 ALTER TABLE `meteoattuale` DISABLE KEYS */;
INSERT INTO `meteoattuale` VALUES ('Andria',5,75,'poche nuvole','02d'),('Arezzo',2.94,69,'cielo sereno','01d'),('
/*!40000 ALTER TABLE `meteoattuale` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `previsioni`
--
```

Figura 16 – File dump archiviometeoapp.sql (pagina 1, schermata editor notepad++)

```

DROP TABLE IF EXISTS `previsioni`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `previsioni` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `citta` varchar(45) DEFAULT NULL,
  `inizio` varchar(45) DEFAULT NULL,
  `fine` varchar(45) DEFAULT NULL,
  `tempmin` double DEFAULT NULL,
  `tempmax` double DEFAULT NULL,
  `temp` double DEFAULT NULL,
  `umidita` int(11) DEFAULT NULL,
  `pioggia` double DEFAULT NULL,
  `descrizione` varchar(45) DEFAULT NULL,
  `icona` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1777 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `previsioni`
--

LOCK TABLES `previsioni` WRITE;
/*!40000 ALTER TABLE `previsioni` DISABLE KEYS */;
INSERT INTO `previsioni` VALUES (1,'Roma','2019-02-16T06:00:00','2019-02-16T09:00:00',7.94,11.72,11.72,65,0.01,'p
/*!40000 ALTER TABLE `previsioni` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2019-02-20 12:39:54

```

Figura 17 – File dump archiviometeoapp.sql (pagina 2, schermata editor notepad++)

Di seguito riporto due figure che mettono in evidenza il contenuto delle due tabelle del database “archiviometeoapp” contenenti rispettivamente le condizioni meteo attuali (“meteoattuale”) e le previsioni meteo orarie di ogni città (“previsioni”).

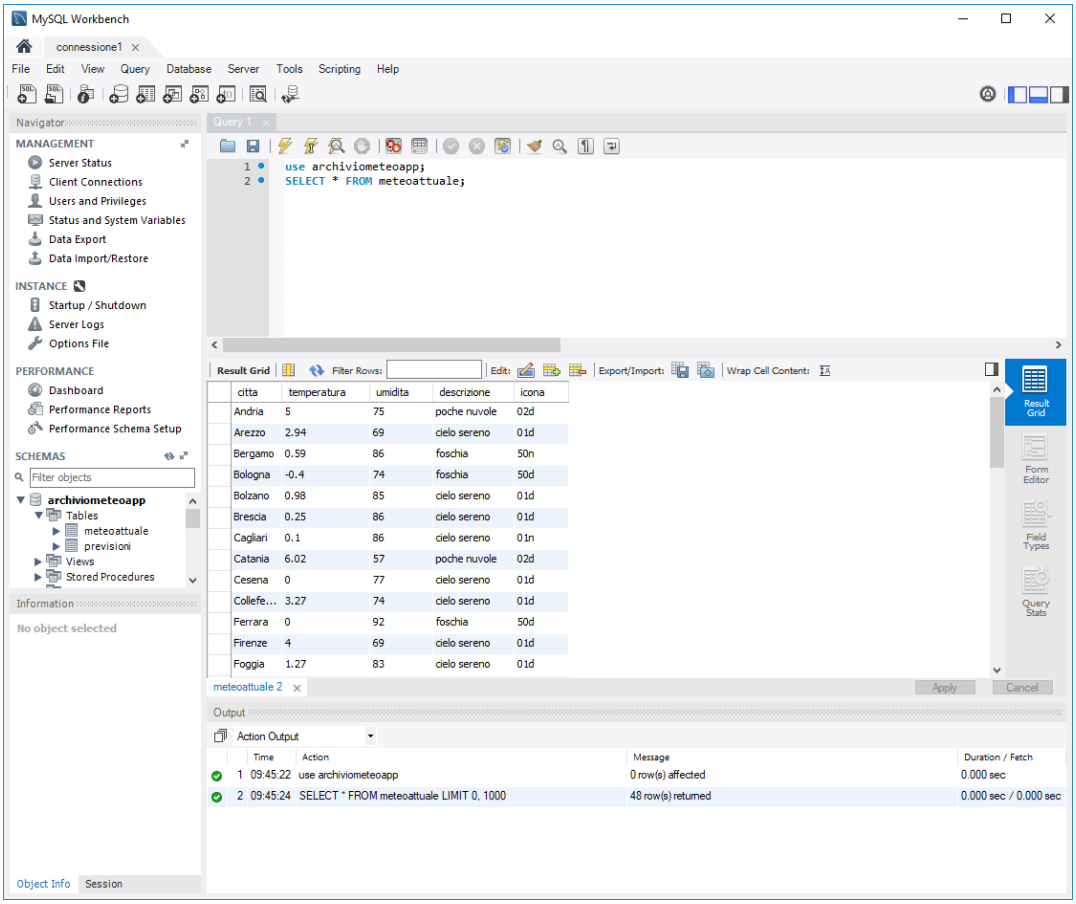


Figura 18 – Interrogazione tabella “meteoattuale” (schermata MySQL Client)

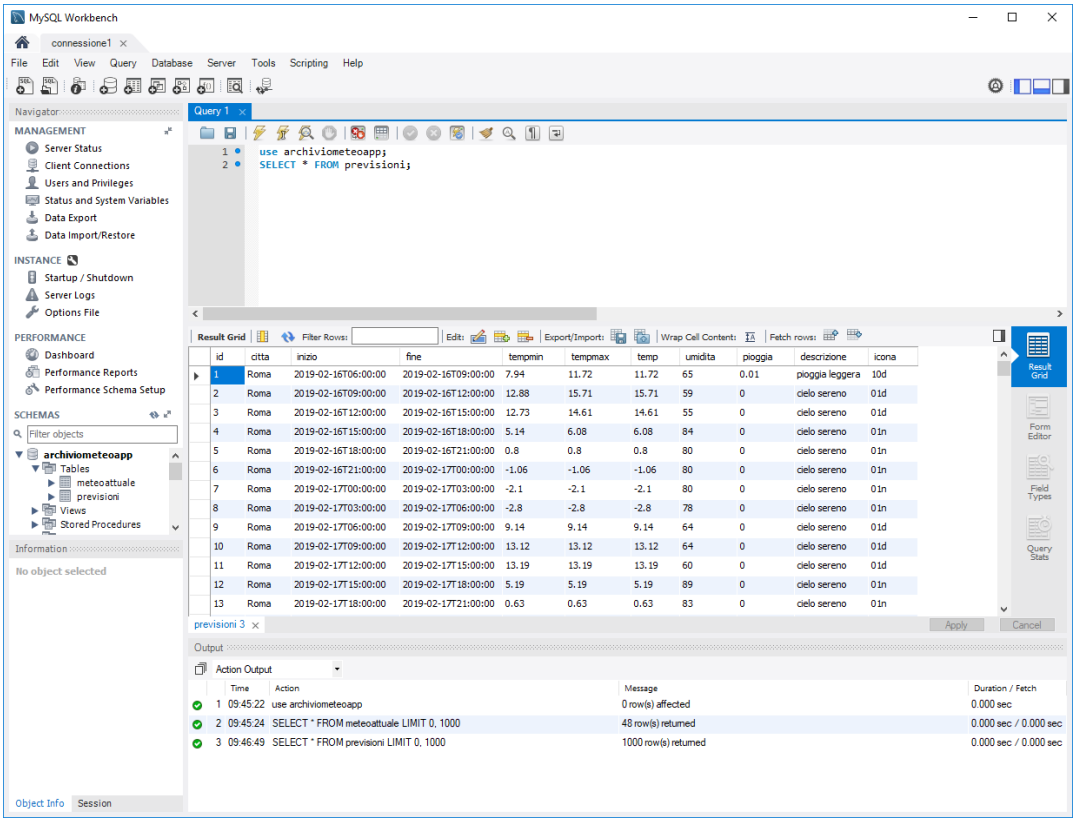


Figura 19 – Interrogazione tabella “previsioni” (schermata MySQL Client)

Alcune informazioni utilizzate nell'applicazione come: nome utente, indirizzo IP e porta del server di log, unità di misura ed il numero massimo di città preferite specificabili vengono memorizzate in un file di configurazione xml e gestite dalla classe ParametriConfigurazioneXML. Di seguito riporto due schermate contenenti la struttura del file di configurazione (fileConfig.xml) e quella del file schema utilizzato nella validazione (schemaConfig.xsd)

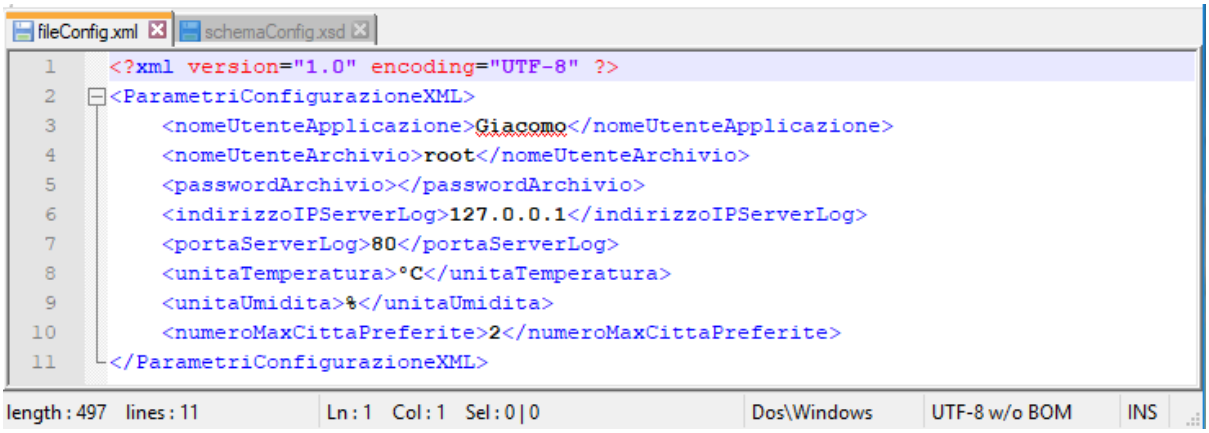


Figura 21 – File di configurazione "fileConfig.xml" (schermata editor notepad++)

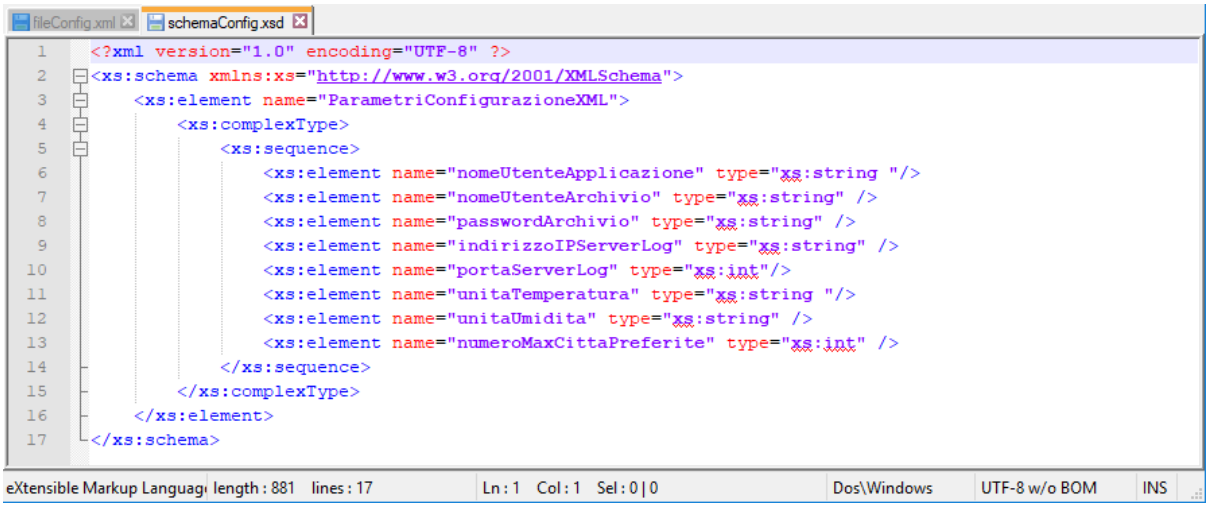


Figura 22 – File schema "schemaConfig.xml" per la validazione del file configurazione (schermata editor notepad++)

Studente: Giacomo Furia

Matricola: 533976

Pisa, 24-02-2019